Imperial College London
Department of Computing

Inductive Logic Programming Using Bounded Hypothesis Space

Duangtida Athakravi

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, September 2015

Abstract

Inductive Logic Programming (ILP) systems apply inductive learning to an inductive learning task by deriving a hypothesis which explains the given examples. Applying ILP systems to real applications poses many challenges as they require large search space, noise is present in the learning task, and in domains such as software engineering hypotheses are required to satisfy domain specific syntactic constraints. ILP systems use language biases to define the hypothesis space, and learning can be seen as a search within the defined hypothesis space. Past systems apply search heuristics to traverse across a large hypothesis space. This is unsuitable for systems implemented using Answer Set Programming (ASP), for which scalability is a constraint as the hypothesis space will need to be grounded by the ASP solver prior to solving the learning task, making them unable to solve large learning tasks.

This work explores how to learn using bounded hypothesis spaces and iterative refinement. Hypotheses that explain all examples are learnt by refining smaller partial hypotheses. This improves the scalability of ASP based systems as the learning task is split into multiple smaller manageable refinement tasks.

The thesis presents how syntactic integrity constraints on the hypothesis space can be used to strengthen hypothesis selection criteria, removing hypotheses with undesirable structure. The notion of *constraint-driven bias* is introduced, where hypotheses are required to be acceptable with respect to the given meta-level integrity constraints.

Building upon the ILP system ASPAL, the system RASPAL which learns through iterative hypothesis refinement is implemented. RASPAL's algorithm is proven, under certain assumptions, to be complete and consistent. Both systems have been applied to a case study in learning user's behaviours from data collected from their mobile usage. This demonstrates their capability for learning with noise, and the difference in their efficiency. Constraint-driven bias has been implemented for both systems, and applied to a task in specification revision, and in learning stratified programs.

Declaration of Originality

I declare that this thesis is my own work. Works by others which were referenced by this thesis have either been acknowledged in the text, or included as an entry in the bibliography.

Copyright Declaration

Acknowledgements

I am deeply grateful to my supervisors Dr. Alessandra Russo and Dr. Krysia Broda. I am indebted to them for having introduced me to Inductive Logic Programming, for organising the financial support for my study, and, most importantly, for their continued support and guidance throughout my PhD. This work relies heavily on ASPAL and I appreciate the help its creator, Domenico Corapi, has given me with his system. I would like to also express my gratitude to the members of the SPIKE research group, in particular to Dalal Alrajeh, whose support and collaboration was vital to many parts of this work, and Mark Law for his collaboration. I am grateful to Professor Ken Satoh of the National Institute of Informatics (NII), for his collaboration, and the many ideas and insights into work related to my research. I am also thankful to the other staff and students of NII who have made me feel welcome during my stay in Japan.

I acknowledge the financial support of the EPSRC which has partially funded this work.

Finally, I wish to thank my family and friends for their great support and encouragements.

Dedication

To my grandfather, Ard Athakravi.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Within the field of psychology learning is described in [Mye99] as *"A relative permanent change in an organism's behaviour due to experience"*. Learning is a crucial ability that allows living organisms to adapt to their environments by acquiring and applying knowledge from past experiences, making it a fascination to many fields including philosophy, psychology, and biology (neuroscience). *Machine learning* [Mit97] is the study of the automation of learning, with computer programs in place of organisms as the learner. Instead of a typical program whose purpose is to tell a machine what actions it should take, a learning program attempts to learn new information from the supplied examples. The examples act as new experiences for the program, and the learnt information is how the program adapts to them.

Induction is a form of reasoning which generates new knowledge by generalising observed examples, thus generating the concept that supports the observations. Within machine learning *Inductive Logic Programming* (ILP) [MD94] is a field that combines the expressive representation of logic programming with inductive reasoning. ILP systems learn new concepts from positively and negatively labelled examples. More specifically, learning through ILP involves supplying the learner with an inductive learning task composed of the observed examples, a set of grounded terms, of the concept being learnt, a *background knowledge*, and a *language bias* which is a specification of what clauses, called *rules*, can be included in the *hypotheses*, a set of rules that are output by the system as possible explanations for the examples. The *solution*

output by the learner is a hypothesis which, together with the background knowledge, logically implies the examples. Essentially, learning in ILP systems can be viewed as a search problem for the solution within the space of all hypotheses called the *hypothesis space*.

Example 1. *Consider the background knowledge:*

$$B = \left\{ \begin{array}{l} even(0). \\ num(0). \\ num(s(0)). \\ num(s(s(0))). \\ num(s(s(s(0)))). \\ num(s(s(s(s(0))))). \\ succ(X, s(X)) \leftarrow num(X), num(s(X)). \end{array} \right\}$$

B contains existing knowledge about some numbers, the successor relationship between them, and that 0 is an even number. Now suppose B is to be used for learning the concept for even and odd numbers from the following positive examples E^+ and negative examples E^-:

$$E^+ = \left\{ \begin{array}{l} even(s(s(s(s(0))))). \\ odd(s(s(s(s(s(0)))))). \end{array} \right\} \quad E^- = \left\{ \begin{array}{l} even(s(0)). \\ even(s(s(s(0)))). \\ odd(0). \\ odd(s(s(s(s(0))))). \end{array} \right\}$$

The solution to the learning task can be found by searching through the hypothesis space $\mathcal{P}(R_M)$:

$$\mathcal{P}(R_M) = \left\{ \begin{array}{lll} \{odd(X). & even(X). & \} \\ \{even(X) \leftarrow succ(Y, X). & odd(X) \leftarrow succ(Y, X), even(X). & \} \\ \{odd(X) \leftarrow not\ even(X). & even(X) \leftarrow succ(Y, X), succ(Z, Y), even(Z). & \} \end{array} \right\}$$

The combination of B and any hypothesis in $\mathcal{P}(R_M)$ would make all positive examples true. Note that "not" in $\{even(X) \leftarrow succ(Y, X), succ(Z, Y), even(Z).; odd(X) \leftarrow not\ even(X).\}$ represents negation as failure (will be covered in Section 2.4). Of the three hypotheses in $\mathcal{P}(R_M)$ only $\{even(X) \leftarrow succ(Y, X), succ(Z, Y), even(Z).; odd(X) \leftarrow not\ even(X).\}$ would not imply any of the negative examples making it the solution of the learning task.

There have been recent works in ILP which address the problem of how to compute the solution of a learning task; in particular systems such as XHAIL [Ray09] and TAL [CRL10] (and ASPAL [CRL11]) have contributed in extending ILP systems to learning inductive tasks with non-monotonic logic. That is, a logic paradigm where adding new knowledge to an existing theory may invalidate previously implied outcomes. Of the mentioned systems XHAIL and ASPAL have been implemented in Answer Set Programming (ASP). ASP is a form of declarative programming based on stable model semantics [GL88], with the focus of generating stable models called *answer sets*, a set of ground atoms, corresponding to models of the logic program. The idea behind stable model semantics is that a set of atoms S can be seen as a partial evaluation of a logic program P, and can be used to simplify P. The simplification process creates a program P^S referred to as the *reduct* of P. S is a stable model of P if and only if it is the least Hebrand model [vEK76] of P^S.

Example 2. *Consider the program:*

$$P = \left\{ \; a \leftarrow not\ b. \; \right\}$$

Suppose $S_1 = \{a\}$*, then the reduct is* $\{a.\}$ *since* b *is not in* S_1*, so the condition not* b *must hold. The least Hebrand model of the reduct is* $\{a\}$*, making* S_1 *a stable model of* P*.*

Suppose $S_2 = \{b\}$*, then the reduct is* $\emptyset$ *since* b *is in* S_2*. The least Hebrand model of the reduct is* $\emptyset$*, and* S_2 *is not a stable model of* P*.*

Answer sets of ASP programs are typically computed using *grounder* and *solver* programs. The grounder takes the given program and replaces all variables in the program with all possible constants that can be used in its place, creating a grounded program. The solver then uses the grounded program to find its answer sets. The exact technique used for solving the grounded program is different from system to system with many ASP solvers using Boolean constraint solving techniques [GKNS07]. What makes ASP an appealing platform for implementing non-monotonic learning systems is that, not only does it has a stable model semantics designed for non-monotonic reasoning, but also that, unlike SLD-resolution (Selective Linear Definite clause

resolution) based approaches such as Prolog [Bra01], rules ordering and recursion (for a finite domain) are inconsequential in ASP. For instance, the program $\{a \leftarrow not\ a.\}$ is not satisfiable in ASP as it has no answer sets, but in Prolog the query $\leftarrow a$ cannot finitely terminate as $a \leftarrow not\ a$ will loop upon itself. For ILP systems being able to ignore loops within the learnt theory makes it simpler to learn theories with recursion, as rules with or without recursive definitions can be treated in the same manner.

1.1 Motivation

Past works in ASP based ILP systems, ILP systems implemented using ASP, have focused on formalising learning algorithms that are complete and sound, and gave little consideration to making their approaches practical for real world applications. This work presents methods for tackling obstacles that are encountered when applying ILP systems. The obstacles that we will focus on are (i) the scalability issue of ASP based ILP systems as a result of the grounder having to ground the entire hypothesis space; (ii) adapting non-monotonic ILP systems to solve learning tasks with noise within the observed examples; and (iii) giving users more control over the hypothesis search so that it is not directed purely by the systems' internal search heuristics.

As pointed out above, ASP programs are grounded before they are solved. This creates a scalability issue for ASP based systems as not only could the background knowledge be large, but the high combinatorial complexity of the hypotheses could also lead to an immense hypothesis space, all of which will need to be grounded. This grounding process has a major effect on the efficiency of the ILP systems, and at times could turn a theoretically solvable learning task into an unsolvable one. The grounding problem is especially relevant to non-monotonic learning as all examples will need to be taken into account. The entire hypothesis space will need to be grounded, as opposed to only the hypothesis spaces relevant to subsets of examples, and the larger hypothesis space leads to larger grounding of the ASP program. Using iterative learning, this work tackle the problem of handling large ASP grounding by breaking the learning process into a sequence of smaller programs.

In [Cor11] it has been suggested that the problem of non-monotonic learner's scalability could be mitigated for ASPAL by allowing the system to learn only rules with very small size, and then iteratively increasing the rule size until a solution is found. Although this iterative method does ensure that the solution is found using the smallest possible hypothesis space, this method does not address the scenario where that smallest possible hypothesis space is still too large for the ILP system to handle. In addition to this, noise also makes it difficult to apply ILP systems to real world applications. There has been past work in making ILP systems able to handle learning tasks with noise, such as Aleph [Sri07] and HYPER/N [OB10] that both use noise threshold to allow hypotheses that cover some noisy examples to be used as solutions to the learning task. However, both systems are for monotonic ILP and not non-monotonic ILP.

When searching through the hypothesis space, ILP systems prefer a systematic search over a naive one as it makes the search more efficient. The systematic search is often carried out by the use of a *scoring function* which assigns a numerical score to a hypothesis, and the learner is directed towards hypotheses with the best score. For instance, when Occam's Razor is incorporated into the search heuristics, the size of the hypothesis is included in the score and the learner will give higher preference to hypotheses with smaller size. However, in some application domains the score directed search may not find the most appropriate hypothesis, as the user's hypothesis selection criteria is not reflected by the scoring function. ILP systems such as DIALOGS [Fle97], Metagol [ML13], and SERIES [WO91] have all demonstrated that the structure of the hypothesis can be exploited for directing the search for the hypothesis. In addition to this, there have been several recent works in ILP which incorporate meta-level information as a part of their learning algorithm. Examples of such meta-level information used in ILP systems include the clausal graph in SOLAR [IFKN10], the meta-interpreter of Metagol [ML13], and the rule encoding of TAL [CRL10] and ASPAL [CRL11]. The use of meta-level information as the core mechanism for learning makes it easy for the users of the ILP systems to have direct interaction with the syntactic information of the hypothesis, creating an opportunity for the users to enhance the selection criteria of the systems with their expertise in the learning domain.

Example 3. *Consider Example 1. Since even and odd are related concepts, it would be rea-*

sonable for the user to expect a solution to have the following form:

$$even(X) \leftarrow odd(Y), P(X,Y).$$
$$odd(X) \leftarrow even(Y), P(X,Y).$$

where $P(X,Y)$ is a place-holder for a relation on X and Y. If the above template is given as part of the learning task, it can act as a guide for the learner so that rules within the hypothesis space that do not have similar structure to one of the template's rules can be dismissed.

This thesis aims to show how learning using bounded hypothesis space could address the scalability problem described above, and control the search with additional domain specific information. The thesis presents two methods for binding the hypothesis space. The first method is a learning algorithm that combines existing ILP techniques with *hypothesis refinement* for solving inductive learning tasks. Hypothesis refinement is used as the core mechanism for the algorithm, and a solution to the learning task is found by iteratively refining a hypothesis while keeping a low maximum rule size. This makes it possible for a large solution to be found by searching through multiple smaller hypothesis spaces than what would be required should it be learnt without using hypothesis refinement. The algorithm makes ASP based ILP systems more efficient as the learning task can be solved using smaller ASP programs, resulting in the ASP grounder producing smaller grounded programs.

Example 4. *Let h be the hypothesis:*

$$even(X) \leftarrow succ(Y,X), succ(Z,Y), even(Z).$$

$$odd(X) \leftarrow not\ even(X).$$

from Example 1. The hypothesis space that contains h will also contain other hypotheses that have rules with 3 or fewer body literals. By using hypothesis refinement h could potentially be found by first finding $\{even(X).; odd(X).\}$, then iteratively adding one or two body literals to them by search through a hypothesis space where rules have at most two body literals. More specifically, $\{even(X).; odd(X).\}$ could be improved to $\{even(X) \leftarrow succ(Y,X).;\ odd(X) \leftarrow$

not $even(X).\}$*, which can then be improved to* $\{even(X) \leftarrow succ(Y, X)$, $succ(Z, Y)$, $even(Z).$; $odd(X) \leftarrow$ *not* $even(X).\}$ *Using this iterative learning method, the maximum size of the hypothesis space require to solve the task can be reduced, consequently resulting in smaller grounded ASP programs.*

This algorithm will be presented for both learning tasks in both cases of not noisy and noisy examples.

The second method for binding the hypothesis space presented in the thesis is a general framework for incorporating meta-level integrity constraints into ILP systems. This is introduced through the notion of *constraint-driven bias*, where the inductive task is extended to include the constraints defined using syntactic meta-level information of the hypothesis and the background knowledge. These constraints act as additional criteria for solutions to the learning task, as not only must the solution explain the examples, but it must also satisfy the constraints. Such constraints give an additional avenue for users to control the hypothesis search.

1.2 Contributions

The thesis has two main contributions:

Learning through hypothesis refinement. We introduce a novel learning algorithm that combines state of the art ASP based ILP techniques ASPAL [CRL11] with theory revision [Wro96]. The learning approach includes a method for selecting the best partial hypothesis and refinements using a scoring function. This algorithm is implemented as the ILP system *RASPAL*, and is shown to be complete for noiseless learning tasks without empty hypotheses. A modified version of the RASPAL algorithm is also presented, demonstrating how a *noise threshold* can be imposed as the termination condition for the algorithm, enabling it to handle learning tasks with noise.

RASPAL is evaluated through comparison against ASPAL and HYPER [Bra99], another ILP system which uses hypothesis refinement, and by its performance when used with different

ASP solvers. The evaluation of RASPAL against HYPER and ASPAL was conducted using three examples that highlight the strength and weakness of each system. The three examples that were used were monotonic learning tasks as HYPER is a monotonic ILP system. The result shows that RASPAL can solve a wider range of learning tasks compared to HYPER and ASPAL, both of which cannot solve one of the learning tasks. The capacity of RASPAL in learning with noise is evaluated alongside a modified version of ASPAL using a case study of learning user's mobile behaviour, which shows that to solve the same task ASPAL would have to use a grounded ASP program five times the size of RASPAL's grounded ASP programs. Details of both evaluations can be found in Section 7.1.

Constraint-driven bias. We present a framework for defining integrity constraints over the hypothesis space as a mechanism for controlling the search for the hypothesis. A set of abstract primitives for defining these constraints, together with their interpretation and formal semantics is defined, giving the user the ability to specify both the desirable and undesirable hypothesis structures. The thesis shows how the constraints can be implemented in ASPAL and RASPAL, and application of constraint-driven bias is demonstrated through two case studies; a learning task for stratified normal programs, and learning task in the automated revision of software specifications. In the first case study we recorded the number of solutions that were output when the constraints was applied compared to when there were no constraints. The time taken to compute all solutions with and without constraints were also recorded. The results show that in some cases, the time taken to solve the task can be reduced by more than half using the constraints. The second case study shows how the constraints can be used in an application for selecting the most appropriate solution that would otherwise be considered non-optimal.

1.3 Publications

The following publications are drawn from the work presented in this thesis:

[ACBR13] Duangtida Athakravi, Dalal Alrajeh, Krysia Broda, Alessandra Russo, and Ken Satoh. Inductive Learning using Constraint-driven Bias. In Proceedings of the 24th

International Conference on Inductive Logic Programming, ILP 2014, Nancy, France, September 14-16, 2014.

The paper describes the constraint-driven bias framework and its implementation in ASPAL, and is the basis for Chapter 5 and Chapter 6.

[AAB+14] Duangtida Athakravi, Domenico Corapi, Krysia Broda, and Alessandra Russo. Learning Through Hypothesis Refinement Using Answer Set Programming. In Proceedings of the 23rd International Conference, ILP 2013, Rio de Janeiro, Brazil, August 28-30, 2013.

The paper presents the hypothesis refinement learning approach, and Chapter 4 delves into this work in more detail.

The foundation for some of the work in thesis prior to the PhD is from the following publication:

[ACR+12] Duangtida Athakravi, Domenico Corapi, Alessandra Russo, Marina De Vos, Julian A. Padget, and Ken Satoh. Handling Change in Normative Specifications. In Proceedings of the 10th International Workshop on Declarative Agent Languages and Technologies, DALT 2012, Valencia, Spain, June 4, 2012, Revised Selected Papers.

In this work an ASP program is used for generating relevant literals, which are literals whose truth values are used for determining whether some hypotheses should be accepted or rejected.

1.4 Structure

The remainder of the thesis is organised as follows:

Chapter 2 summaries the notation and preliminary background on logic programming that are used throughout the thesis.

Chapter 3 describes the different approaches in ILP, and discusses in detail the ILP systems that are most relevant to the thesis. The chapter also discusses how a theory revision task can be translated into an inductive learning task.

Chapter 4 discusses the limitation of ASPAL, and presents our learning algorithm, RASPAL, based on hypothesis refinement. The chapter also contains the proof for the completeness of the algorithm, describes how to adapt RASPAL to learning tasks with noise, and discusses other works that are related to RASPAL.

Chapter 5 presents the constraint-driven bias framework, the set of primitives $\mathcal{L}_\mathcal{C}$, and gives the formal definition for their interpretation. The chapter also discusses other works related to constraint-driven bias.

Chapter 6 describes how constraint-driven bias can be implemented in ASPAL and RASPAL. The chapter gives a translation for primitives in $\mathcal{L}_\mathcal{C}$ into ASP constraints, and proves the correctness of the translation.

Chapter 7 evaluates RASPAL through comparison against ASPAL and HYPER, as well as its performance with different ASP solvers, and presents two case studies for the applications of constraint-driven bias.

Chapter 8 concludes the work by summarising the thesis and discusses possible directions for future work.

Appendix A contains the full description of the inductive learning tasks that are used in Chapter 7.

Chapter 2

Background

This chapter summaries the notation used throughout the thesis and gives the required background knowledge on first-order logic and logic programming.

2.1 Notations

The notation $\{e_1, \ldots, e_n\}$ denotes a set of n objects where $n \geq 1$, and $\{e|exp(e)\}$ is a set composed of objects that satisfy the expression $exp(e)$. The symbol $\emptyset$ denotes an empty set. Let S be a set, its cardinality is represented by $|S|$, and its power set is represented by $\mathcal{P}(S)$. A tuple with $n \geq 1$ elements is expressed as $\langle e_1, \ldots, e_n \rangle$ or as $(e_1, \ldots, e_n)$. A list with $n \geq 1$ elements is represented by $[e_1, \ldots, e_n]$ or $e_1, \ldots, e_n$.

2.2 First-order logic

Syntax

Firstly we define the symbols that make up the language $\mathcal{L}$.

Definition 2.1 (Language $\mathcal{L}$)**.** *The symbols in language $\mathcal{L}$ comprise of:*

- *Logical symbols: '$\forall$' (all), '$\exists$' (exists), '$\neg$' (negation), '$\wedge$' (and), '$\vee$' (or), '$\leftarrow$' (implies), '$\leftrightarrow$' (equal), '$\top$' (true), '$\bot$' (false)*
- *Punctuation symbols: ',', '(', ')'*
- *Variable symbols: 'X', 'Y', 'Rid', 'L_h', 'L_b', ...*
- *Predicate symbols: 'p', 'q', 'extension', 'del', 'is_rule', 'in_rule', ...*
- *Function symbols: 'f'*

The logical symbols have a fixed meaning for each symbol, while the other symbols can be used to form structures as follows:

- *Term*: A variable X or a constant c (a term with arity 0) or a function $f(t_1, \ldots, t_n)$ where its arity $n \geq 0$ and $t_1, \ldots, t_n$ are terms
- *Atom*: A predicate $p(t_1, \ldots, t_n)$ with arity ≥ 0 and $t_1, \ldots, t_n$ are terms
- *Formula*: An atom, truth $\top$ and falsity $\bot$. If F and G are formulae then $(\forall X)F$ (universal), $(\exists X)F$ (existential), $\neg F$ (negation), $(F \wedge G)$ (conjunction), $(F \vee G)$ (disjunction), $(F \leftarrow G)$ (implication), and $(F \leftrightarrow G)$ (equivalence) are also formulae.
- *Literal*: An atom α or its negation $\neg\alpha$

For readability, standard binding conventions will be used so that brackets can be removed from unambiguous formulae. The order of the binding precedence from the highest to the lowest is: $\forall$, $\exists$, $\neg$, $\wedge$, $\vee$, $\leftarrow$, $\leftrightarrow$.

The formulae $(\forall X)F$ and $(\exists X)F$ denote that each occurrence of the variable X is *bound* in F as it is under the scope of the quantifier $\forall$ or $\exists$. Occurrences of variables not under the scope of any quantifier are referred to as *unbound* or *free* variables. Formulae with only bound variables are *closed*, while others with one or more free variables are *open*.

A substitution is a transformation of variables to terms and is denoted by a set of bindings $\theta = \{X_1/t_1, \ldots, X_n/t_n, \}$ where each X_i are distinct variables and t_i is different from X_i. The

application of a substitution θ to a formula F is denoted by $F\theta$, and is obtained by replacing all free occurrences of X_i by the corresponding t_i. F' is an *instance* of F if it can be obtained from F via some substitution θ ($F' = F\theta$).

The notation p/n will be used to denote a predicate p with n arity (and similarly f/m for a function with m arity). In this thesis some predicates such as $rule/1$, $delete/2$ and $extension/2$ will be used with specific meanings: the encoding of a rule, the deletion of a literal within a theory, and the extension of a formula with the theory by one or more literals respectively.

2.2.1 Semantics

The meaning of a first-order logic formula is given by relating it to some domain $\mathcal{D}$. This is achieved through the use of an assignment function h which relates each variable symbol to an element of $\mathcal{D}$, and a mapping I that relates each constant c, function f/m and predicate symbol p/n of $\mathcal{L}$ to a corresponding constant c^I in $\mathcal{D}$, function f^I from $\mathcal{D}^n \to \mathcal{D}$ or predicate p^I representing some relationship in $\mathcal{D}^n$ respectively.

Definition 2.2 (Evaluating a term)**.** *The value of a term $\upsilon_{I,h}(t)$ under I and h is defined as follows:*

- *For a constant c: $\upsilon_{I,h}(c) = c^I$*
- *For a variable X: $\upsilon_{I,h}(X) = h(X)$*
- *For a function f/m: $\upsilon_{I,h}(f(t_1, \ldots, t_m)) = f^I(\upsilon_{I,h}(t_1), \ldots, \upsilon_{I,h}(t_m))$*

Definition 2.3 (Evaluating a formula)**.** *Formula F (and G) can be evaluated to a truth value as follows:*

- $I \vDash_h \top$
- $I \vDash_h p(t_1, \ldots, t_n)$ *if and only if* $p^I(\upsilon_{I,h}(t_1), \ldots, \upsilon_{I,h}(t_n)) \in \mathcal{D}^n$
- $I \vDash_h (\forall X)F$ *if and only if* $I \vDash_h F\{X/e\}$ *for all* $e \in \mathcal{D}$

- $I \models_h (\exists X)F$ *if and only if* $I \models_h F\{X/e\}$ *for some* $e \in \mathcal{D}$
- $I \models_h \neg F$ *if and only if* $I \not\models_h F$
- $I \models_h F \wedge G$ *if and only if* $I \models_h F$ *and* $I \models_h G$
- $I \models_h F \vee G$ *if and only if* $I \models_h F$ *or* $I \models_h G$
- $I \models_h F \leftarrow G$ *if and only if* $I \models_h F$ *or* $I \not\models_h G$
- $I \models_h F \leftrightarrow G$ *if and only if* $I \models_h F \wedge G$ *or* $I \models_h \neg F \wedge \neg G$

If F is true under I for all possible assignments by h, then I is a *model* of F, denoted by $I \models F$, and F is a consequence of I. A formula is *satisfiable* if it has at least one model. Similarly, a *theory* T (a set of formulae) is satisfiable if there exists an interpretation M in which all formulae in the theory is satisfiable.

2.3 Logic Programming

Logic Programming handles the computation of logical formulae by deriving a logical conclusion from some given assertions or query. A logic program is made up of the following types of clauses (also called rules) where variables in clauses are implicitly universally quantified.:

1. Clause: Formula of the form $A \vee \neg B_1 \vee \cdots \vee \neg B_n$, where $n \geq 1$ and A, B_1, ..., B_n are atoms. It is denoted as a *definite* rule $A \leftarrow B_1, \ldots, B_n$. The left hand side of the rule A is called the *head* of the rule, while the right hand side $B_1, \ldots, B_n$ is called the *body* of the rule.
2. Fact: A rule without any body written as A.
3. Query or goal: Formula of the form $\neg A_1 \vee \cdots \vee \neg A_n$, where $n \geq 1$, and is also written as $\bot \leftarrow A_1, \ldots, A_n$.

A set of a such clauses makes up a logic program, and to remove ambiguity, clauses are separated by a full stop '.' Given a rule $r = A \leftarrow B_1, \ldots, B_n$ we will use $head(r)$ to refer to the head of the rule, and $body(r)$ to refer to the list of all of its body conditions.

To infer whether an atom A is true within the context of a given program P ($P \vDash A$) in a top down manner is done through *resolution* [Rob65]. This is an inference rule where contrasting literals in different clauses implies a new clause (for example $A \vee B$ and $\neg A \vee C$ implies $B \vee C$). In Prolog, the technique used is SLD-resolution (Selective Linear Definite clause resolution) where in order to determine if $P \vDash A_1 \wedge \cdots \wedge A_n$ for some atoms $A_1 \wedge \cdots \wedge A_n$, they are negated ($\neg A_1 \vee \cdots \vee \neg A_n$) and used with P in a chain of unifications and resolutions to find the empty clause ($\perp$) which shows that $P \vDash A_1 \wedge \cdots \wedge A_n$ holds.

Example 5. *Suppose P contain the clauses:*
$fly(X) \leftarrow pigeon(X)$.
$pigeon(alex)$.

To determine if $P \vDash fly(alex)$, firstly $fly(alex)$ is negated. By unifying X in the rule $fly(X) \leftarrow pigeon(X)$ for alex, it can be resolved with the $\neg fly(alex)$ to give $\neg pigeon(alex)$, which contradicts with $pigeon(alex)$ in P, and would derive $\perp$.

2.4 Negation as Failure and Non-monotonic Logic Programming

In monotonic logic, the theory can be freely extended while still preserving all of the implications of the original theory. Hence, for a theory T and a formulae F and G, if $T \vDash F$ then $T \cup \{G\} \vDash F$. While monotonic logic is suitable for reasoning when complete information of the domain is available, it is often impossible or infeasible to have complete knowledge of a domain or a situation. Negation as failure (NAF) [Cla77] is useful for reasoning with incomplete information, and complete information is assumed through closed world assumption. With classical negation $\neg A$, where A is an atom, conveys that A is *known to be false*, whereas with NAF *not* A expresses

A is *assumed to be false if it is not known to be true.* Under NAF, an atom is assumed to be false if it is not a derivable implication of a theory. This means that should $T \nvDash A$, for some theory T and literal A, then *not* A is assumed to be true in T, and similarly if $T \nvDash \neg A$ then *not* $\neg A$ is assumed to be true. Note that '*not*' is used instead of '$\neg$' to distinguish NAF from classical negation. Using NAF, negative information does not have to be explicitly stated in the theory.

Example 6. *Consider a small domain consisting of three people: Allen, Beth and Clive, and that the complete knowledge of the parent relationship between them is:*

$$T = \left\{ \begin{array}{lll} parent(allen, beth). & \neg parent(beth, beth). & \neg parent(clive, beth). \\ \neg parent(allen, allen). & \neg parent(beth, allen). & \neg parent(clive, allen). \\ \neg parent(allen, clive). & parent(beth, clive). & \neg parent(clive, clive). \end{array} \right\}$$

Using negation as failure we can assume that someone is not the parent of another person if it is not explicitly stated that they are. Under negation as failure, the theory T can be reduced to:

$$T' = \left\{ parent(allen, beth). \quad parent(beth, clive). \right\}$$

Any other instances of the parent relationship in T' are assumed to be false under NAF (for example $T' \vDash$ not $parent(allen, allen)$).

With NAF, the logic is no longer monotonic as addition of information can invalidate previously implied information.

Example 7. *Consider the following theory:*

$$T = \left\{ \begin{array}{l} like(john, X) \leftarrow food(X), not\ spicy(X). \\ food(curry). \\ food(stew). \end{array} \right\}$$

The rule for $like/2$ in T states that John likes all foods as long as they are not spicy. Therefore, $T \vDash like(john, curry)$ and $T \vDash like(john, stew)$. However, if $spicy(curry)$ is added to T then $T \cup \{spicy(curry)\} \nvDash like(john, curry)$.

Logic where consequences of a theory are not retained with the arrival of new information are called *non-monotonic logic*, and a theory with negation as failure is called a non-monotonic theory or a normal theory.

Non-monotonic Logic Programming (also referred to as normal logic programming) is an extension of definite logic programming, where negation as failure can be used as part of the program. Thus, normal form rule $A \leftarrow B_1, \ldots, B_i, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_j$, where $A, B_1, \ldots, B_i, C_1, \ldots, C_j$ are all atoms, is used in place of definite rule.

2.5 Answer Set Programming

Answer Set Programming is a form of declarative programming aimed to be used for representation and reasoning tasks based on stable model semantics [GL88] designed for handling non-monotonic programs. Problems are solved by grounding the program, and then solving the grounded program to find its minimal models. Solving the grounded program involves finding its stable models, which are also referred to as answer sets, and are the outputs of ASP systems.

Definition 2.4 (Stable Model)**.** *Given a grounded normal logic program P, and a set of atoms X. The definite program P^X called the reduct is obtained from P by the following steps:*

1. *Removing all rules in the P with not B in its body where $B \in X$*
2. *Removing all negated body literals not B in the remaining rules in P*

As P^X is a definite program, a program without any negative literals, it must have a minimal model. If X coincides with any minimal model of P^X then it is a stable model of P.

Definition 2.5 (Entailment in answer set semantics)**.** *Given a program P and a literal L, L is entailed by P ($P \models L$) if and only if for all stable model S of P: $L \in S$. Similarly, if there exists a stable model S of P: $L \notin S$, then $P \not\models L$. If it is neither the case that $P \models L$ nor $P \not\models L$, then L is unknown with respect to P.*

The language used in ASP is called *AnsProlog* [Bar03] (Answer Set Programming in Logic).

Definition 2.6 (AnsProlog Program)**.** *An AnsProlog program is a finite set of rules of the form:*

$$L_0 \vee \cdots \vee L_k \leftarrow L_{k+1}, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n.$$

where L_i are literals (or $\perp$ if $k = 0$), and $n \geq m \geq k \geq 0$.

AnsProlog rules can therefore take the forms of (where each L_i is a literal):

Rule	$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n.$
Fact	$L_0.$
Integrity Constraint	$\perp \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n.$

The first form corresponds to the logic rule $A \leftarrow L_1 \wedge \cdots \wedge L_n$, while the second is a rule without any body literals. Lastly, the integrity is a rule with $\perp$ as its head, thus $L_1 \wedge \cdots \wedge L_n$ must be false for the constraint to hold.

Choice rule is a common feature in ASP systems. In [Bar03] it is described that selecting only one element from a set using AnsProlog can be achieved by the rules:

$$not\_chosen(X) \leftarrow chosen(Y), X \neq Y.$$
$$chosen(X) \leftarrow can\_choose(X), not\ not\_chosen(X).$$

In many ASP systems the same effect can be achieved using aggregates. Aggregates are constraints for stating that elements from a given set can be included in the computed answer set. Aggregates can be in the head or the body of a rule and usually specified as:

$$m\ \{L_1 : \overline{C_1}, \ldots, L_k : \overline{C_n}\}\ n$$

where L_i are literals, $\overline{C_i}$ are (possibly empty) type conditions for variables in the literal, $k \geq 0$, $m \geq 0$, and $n \geq 0$[1]. When used in the head of a rule, the aggregate indicates that, should all

[1] In the ASP system Clingo [GKK+11] if m or n is omitted then it is assumed that to be the lower or upper bound respectively.

of the rule's body conditions be in the answer set, then the answer set will contain at least m number of elements in $\{L_1, \ldots, L_k\}$ but no more than n. When used in the body of a rule it represents a body condition that will be satisfied if the number of elements in $\{L_1, \ldots, L_k\}$ that are in the answer set of at least m but not more than n. For example, the aggregate $1\{a, b, c\}3$ when used as the head of a rule such as $1\{a, b, c\}3 \leftarrow d$ indicates that if d holds then $a \vee b \vee c$ holds, whereas if it was used in the body as $d \leftarrow 1\{a, b, c\}3$ would mean that should $a \vee b \vee c$ hold then d also holds. Similarly $2\{a, b, c\}3 \leftarrow d$ indicates that if d holds then either one of $a \wedge (b \vee c)$, $b \wedge (a \vee c)$, or $c \wedge (a \vee b)$ holds, whereas $d \leftarrow 2\{a, b, c\}3$ indicate that should at least 2 of a, b and c hold then d also holds. Lastly, $3\{a, b, c\}3 \leftarrow d$ indicates that if d holds then a, b and c must all hold, and $d \leftarrow 3\{a, b, c\}3$ indicates that if all of a, b, and b hold then d also holds. Note that the earlier AnsProlog choice rule can be represented by the aggregate $1\ \{chosen(X) : can\_choose(X)\}\ 1$.

Depending on the ASP system used to solve the program, an ASP program may contain other expressions as well. In the following section we will describe the system-dependent features that are relevant to this work.

2.5.1 Clingo

Clingo [GKK+11] is an ASP system containing both the grounder and solver (as opposed to using separate programs for the two tasks). Clingo is the system that we have used to implement our ILP system. Note that this thesis uses Clingo 3, from which the description of its features will be based.

Clingo allows for optimisation statements in ASP programs for defining the criteria of the optimal answer set. It is specified as:

$$opt[L_1 = w_1@p_1, \ldots, L_n = w_n@p_n].$$

where *opt* can be either $\#maximise$ or $\#minimise$, each L_i is a literal, $n \geq 1$, and each w_i and p_i are integers. The statement indicates that each literal L_i has an associated weight w_i

and priority p_i, with higher priority given to literals or rules with higher value of p_i. When more than one optimisation statement is used, Clingo automatically gives the n^{th} optimisation statement priority n (all literals in n^{th} statements are given priority n if they are not explicitly given). Thus, later optimisation statements will have higher priority than the earlier ones. Note that both the weight and priority can be omitted. The weight of 1 will be assumed for any omitted weight, and literals with omitted priorities will be given priority according to the order of the statement it is in. The optimisation statement sum the weight for the literals $L_1, \ldots, L_n$ that are in the answer set, and if it is a maximisation statement it will consider the answer set with the highest sum the optimal answer set (or the one with the lowest sum for the minimal). When there are multiple priorities, literals with higher priorities are optimised before those with lower priorities.

Example 8. *Consider the following ASP program:*

$$P = \left\{ \begin{array}{l} \{a, b, c, d, e, f\}\ 4. \\ \#maximise[a = 3@1, b = 2@2, c = 1@3, d = 4@3, e = 5@2, f = 6@1]. \end{array} \right\}$$

Clingo will search for answer sets that maximise the weighted sum of c and d, as they have the highest priority, in this case this will be all answer sets that include both c and d. From the answer sets that maximise c and d, Clingo will then search for those that would also maximise the weighted sum of b and e, which will be the answer set $\{b, c, d, e\}$*. Lastly, from the answer sets that maximise the weighted sum of b, c, d, and e Clingo will search for those that would also maximise a and f, however since b, c, d, and e only have one answer set that optimise them,* $\{b, c, d, e\}$ *will be returned as the optimal answer set.*

Consider another program.

$$P' = \left\{ \begin{array}{l} \{a, b, c, d, e, f\}\ 4. \\ \#maximise[a = 3, b = 2, c = 1, d = 4, e = 5, f = 6]. \end{array} \right\}$$

The maximisation statement is the first optimisation statement in the program, so all literals in it are assigned equal priority of 1. Clingo will attempt to find the answer set that optimise the weighted sum of all literals in the statement and will return the answer set $\{a, d, e, f\}$*.*

2.5.2 DLV

DLV [LPF+06] is a knowledge representation and reasoning system which supports answer sets that we use as part of our evaluation in Chapter 7. In this section we will note some of the differences between DLV's features and those mentioned in Section 2.5 and Section 2.5.1.

DLV's programs can have disjunctive rules; however, exactly one literal in the disjunctive rule will be in an answer set. For example, the rule $p(1) \vee p(2) \vee p(3)$ will always return exactly one of the three literals. Simulating disjunction that allows more than one literal to be true in DLV can be achieved by adding in fresh predicates to represent $\top$, and adding a constraint that not all of these new predicates can be true. For instance the DLV rules:

$$
\begin{aligned}
&p(1) \vee t(1).\\
&p(2) \vee t(2).\\
&p(3) \vee t(3).\\
&\bot \leftarrow t(1), t(2), t(3).
\end{aligned}
$$

where $t/1$ is a completely new predicate in the program, would allow for $p(1)$, $p(2)$, and $p(3)$ to be included in same answer set.

DLV also allows aggregates to be used but they are different to aggregates previously described in Section 2.5. In DLV aggregates are used exclusively in the body of a rule and are accompanied by aggregate functions $\#count$, $\#sum$, $\#times$, $\#min$, and $\#max$. All of the aggregate functions operate over a given set to return a numerical value. For example, the following DLV rule:

$$\bot \leftarrow \#count\{X : p(X)\} > 3.$$

expresses that at most 2 instance of $p/1$ can be in the answer set.

Optimisation can be expressed in DLV using weak constraints. Unlike typical integrity constraints, weak constraints tells the solver that the constraint should be satisfied if possible.

Optimal answer sets are ones that violate the least amount of weak constraints. Weak constraints are defined as

$$:\sim L_0, \ldots, L_n.[w : p]$$

where L_i are literals, w is the weight of the constraint and p is the priority of the constraint (both weight and priority can be omitted). For instance the DLV rule:

$$:\sim p(X).$$

would convey that an optimal answer set contains the fewest instances of $p/1$.

2.6 Abductive Logic Programming

Abduction was first formally introduced by Charles S. Pierce, and is the inference of a plausible explanation for some observation such that the presence of the explanation implies the observation. To take into account the multiple possible solutions, abductive logic programming is better associated with Gilbert Harman's definition of "The inference to the best explanation for some observed events".

Example 9. *Suppose we know the following:*
$sneeze \leftarrow pollen\_allergy.$
$sneeze \leftarrow cold.$
$winter.$
$\bot \leftarrow winter, pollen\_allergy.$

Suppose we want to find the explanation for sneeze (what will be required to make it true). While both pollen allergy and having a cold are plausible explanations for the observation of someone sneezing, only having a cold satisfies the constraint that pollen allergy is not had during winter.

In [KKT92], abduction in logic can be described as a task involving a background theory T, and an observation G. To solve the task a set of clauses Δ is found such that $T \cup \Delta \models G$ and $T \cup \Delta$ is consistent, with Δ being limited to a set of predefined set of atoms called *abducibles*.

Definition 2.7 (Abductive Task)**.** *An abductive task is a tuple $\langle T, A, G \rangle$, where T is the normal background theory, A is a set of all grounded instances of the abducible atoms, and G is the goal of the task. The solution to an abductive task is a set of the abducible atoms Δ such that $\Delta \subseteq A$, and $T \cup \Delta \models G$. Integrity constraint IC can be added to the task such that the abduced solution is consistent with the integrity constraint $T \cup \Delta \cup IC \not\models \bot$.*

Using the above definition, Example 9 can be expressed as an abductive task $\langle T, A, G \rangle$ where

$$T = \left\{ \begin{array}{l} sneeze \leftarrow pollen\_allergy. \\ sneeze \leftarrow cold. \\ winter. \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} pollen\_allergy, \\ cold \end{array} \right\}$$

$$G = \left\{ sneeze. \right\}$$

The task also has an integrity constraint

$$IC = \left\{ \bot \leftarrow winter, pollen\_allergy. \right\}$$

As discussed previously, the the solution of the task is $\Delta = \{cold\}$.

Abductive Logic Programming is an extension of logic programming where a logic program is given an abductive task as input, and the purpose of the program is to compute the solution to the abductive task.

2.6.1 Abduction in Answer Set Programming

An abductive task can be encoded into an ASP program by expressing the goal of the task as an integrity constraint, and the abducibles as an aggregate [IS00]. For instance, we can solve the

task from Example 9 using an ASP program. The goal of the task (of finding an explanation of the sneeze) can be converted into the integrity constraint:

$$\bot \leftarrow not\ sneeze.$$

which will force all answer sets to contain *sneeze*. The abducibles are converted into the aggregate:

$$0\ \{pollen\_allergy, cold\}\ 2.$$

which conveys that any subset of $\{pollen\_allergy, cold\}$ can be in the answer set. The background theory and integrity constraints of the original task remain the same. The full task from Example 9 can be encoded as the following ASP program:

$$P = \left\{ \begin{array}{l} sneeze \leftarrow pollen\_allergy. \\ sneeze \leftarrow cold. \\ winter. \\ \bot \leftarrow winter, pollen\_allergy. \\ \bot \leftarrow not\ sneeze. \\ 0\ \{pollen\_allergy, cold\}\ 2. \end{array} \right\}$$

By solving P, the answer set $S = \{winter, cold, sneeze\}$ can be found, the solution to the abductive task is found by taking the intersection of S and the abducibles $\{pollen\_allergy, cold\}$ which will result in $\{cold\}$.

Definition 2.8 (Abductive Task in ASP)**.** *Given an abductive task $\langle T, A, G\rangle$ where T is the normal background theory, A is a set of all grounded instances of the abducible atoms, and G is the goal of the task. It can be solved using an ASP program $P = T \cup G' \cup A'$ where G' is the constraint $\{\bot \leftarrow not\ goal.; goal \leftarrow G.\}$ and A' is an aggregate $\{0\ \{a|a \in A\}\ |A|.\}$. Let S be an answer set of P, the solution to the abductive task is acquired from intersection of S and A $(S \cap A)$.*

Non-monotonic ILP systems such as XHAIL and ASPAL, both of which will be discussed in the next chapter, use abduction as an integral part of their learning algorithm in order to handle

learning with negation as failure, and for learning from all examples in the learning task in a single computation.

Chapter 3

Inductive Logic Programming

This chapter presents the background information on Inductive Logic Programming (ILP). An overview of types of ILP systems is given, focusing on those that are most relevant to the thesis.

ILP is a field in machine learning that combines formal knowledge representation with inductive reasoning. It involves learning logical rules which will explain some given examples by extending the known background knowledge. An inductive learning task typically consists of three ingredients: the background knowledge of the learning domain; observed examples of the learning objective; and the language bias which defines the language for constructing hypotheses. The aim is to find a solution to the inductive learning task from the hypothesis space defined by the language bias. The solution is a hypothesis which together with the background knowledge entails the examples. Table 3.1 gives an example of some ILP systems and how they would be categorised according to the type and monotonicity of their learning approach. Of these systems TAL, HYPER and Metagol will be discussed in more detail later in this chapter as they are the most relevant systems for this work.

There are many types of language bias that ILP systems use such as production field in CF-Induction [Ino04] and mode declaration, which was first introduced in [Mug95]. For this work we will be using TAL [CRL10] and ASPAL's [CRL11] mode declarations which are defined as follows.

Learning Approach	Monotonic	Non-monotonic
Bottom-up	Progol 5 [Mug95] Imparo [KBR09] HAIL [RBR03]	XHAIL [Ray09]
Top-down	Hyper [Bra99] TopLog [MSTn08] FOIL [QCj95]	ICN [MV95]
Meta-level	Causal graph [IFKN10] Metagol [ML13]	TAL [CRL10]

Table 3.1: Past ILP systems grouped by the learning approach they use, and whether they are monotonic or non-monotonic

Definition 3.1 (Mode declaration)**.** *A mode declaration can either be a head mode declaration* $modeh(s)$ *or a body mode declaration* $modeb(s)$*. It takes a predicate schema* s *as its only argument, where the schema is a ground atom with* $+t_1$ *input variable,* $-t_2$ *output variable, and* $\#t_3$ *constant placemarkers of some type* t_1*,* t_2 *or* t_3 *as its arguments.*

Definition 3.2 (Compatibility with mode declarations)**.** *A rule is said to be* compatible *with a set of mode declarations* M *if and only if:*

- *Its head literal has the same predicate as a schema* s *for some* $modeh(s)$ *in* M*, and its arguments matches the placemarkers in the matched schema (with respect to their types and whether they should be constants or variables).*
- *Each of its body literal has as predicate a schema* s *for some* $modeb(s)$ *in* M*, and all arguments are correctly typed and satisfy the following conditions:*
  - *All input variables in a body literal appear as input or output variables in literals preceding it*
  - *All output variables in a body literal have not appeared in literals preceding it*
  - *All variables with the same name are of the same type*

Consequently, a hypothesis is said to be compatible with a set of mode declarations M if all of its rules are compatible with M.

Example 10. *Given the following mode declarations:*

$$M = \left\{ \begin{array}{l} modeh(grandparent(+person, +person)). \\ modeb(parent(+person, +person)). \\ modeb(parent(+person, -person)). \end{array} \right\}$$

The following rules are compatible with M:

- $grandparent(X, Y)$. The rule is compatible with the head mode declaration $modeh(grandparent(+person, +person))$ with appropriate variable arguments.
- $grandparent(X, Y) \leftarrow parent(X, Y)$. The body literal has the same predicate as the schema of mode declaration $modeb(parent(+person, +person))$, and its arguments are correctly linked to variables of type $person$ in the head literal.
- $grandparent(X, Y) \leftarrow parent(X, Z), parent(Z, Y)$. The first body literal $parent(X, Z)$ is compatible with the mode declarations $modeb(parent(+person, -person))$, and the arguments are correctly linked as variable Z has not appeared in any literals preceding it. The second body literal $parent(Z, Y)$ is compatible with the mode declaration $modeb(parent(+person, +person))$.

The following rules are not compatible with M:

- $parent(X, Y)$. There is no head mode declaration with predicate $parent/2$ as schema.
- $grandparent(X, Y) \leftarrow parent(Z, Y)$. There is no body mode declaration for $parent/2$ where the first argument is an output variable placemarker.

In Inductive Logic Programming [MD94] a set of rules, called the hypothesis H, is learned from a set of rules called background knowledge B, a set of observed positive and negative examples E^+ and E^-, represented by a set of grounded facts, and some language bias M such as mode declarations. The learned hypothesis is a set of rules compatible with the language bias such that the positive examples become derivable once the hypothesis is added to the background knowledge (3.1), while the negative examples are not derivable (3.2).

$$B \cup H \models \bigwedge_{e \in E^+} e \tag{3.1}$$

$$\forall e \in E^- : B \cup H \not\models e \tag{3.2}$$

We say that a hypothesis H *c*overs an example e if $B \cup H \models e$, and a solution to an inductive learning task is a hypothesis that covers all positive examples and none of the negative examples. The full definitions of an inductive task will be given throughout the chapter as their definitions differ depending on the setting such as whether it is a monotonic inductive task (Definition 3.3) or a brave (non-monotonic) one (Definition 3.9). A variation of the inductive task that incoporates meta-level information as the means for solving the task (Definition 3.6) will also be given. In later chapters we will also discuss our extensions of the inductive task; inductive task with noise (Definition 4.10), and inductive task with constraint-driven bias (Definition 5.4).

3.1 Monotonic Inductive Logic Programming

Early ILP systems such as CIGOL [MB88] focus on learning monotonic theories from the positive examples. Taking advantage of this, rules in the solution are typically are learnt one by one from a selected positive example and then their union forms a solution to the learning task. Negative examples are used in a post-process to check that none of them is covered by each learnt rule before it is added to the accumulating solution. This has become the characteristic of most monotonic ILP systems such as Aleph [Sri07], and Progol [Mug95]. The monotonic inductive task is more formally defined as follows.

Definition 3.3 (Monotonic Inductive Task)**.** *An inductive task is a tuple* $\langle E^+, E^-, B, M \rangle$*, where* E^+ *and* E^- *are sets of ground atoms representing positive and negative examples respectively,* B *is a definite program called the background knowledge, and* M *is a set of mode declarations. A solution to an inductive task is a set of definite rules, a hypothesis* H*, such that (i)* H *is compatible with* M*; (ii)* $B \cup H \models \bigwedge_{e \in E^+} e$*; and (iii)* $\forall e \in E^- : B \cup H \not\models e$*.*

Example 11. *Given the following ILP task:*

$$B = \left\{ \begin{array}{l} person(jim). \\ person(ivan). \\ person(kim). \\ parent(jim, ivan). \\ parent(ivan, kim). \end{array} \right\} \qquad M = \left\{ \begin{array}{l} modeh(grandparent(+person, +person)). \\ modeb(parent(+person, +person)). \\ modeb(parent(+person, -person)). \end{array} \right\}$$

$$E^+ = \left\{ \; grandparent(jim, kim). \; \right\} \quad E^- = \left\{ \begin{array}{l} grandparent(jim, ivan). \\ grandparent(ivan, kim). \end{array} \right\}$$

A solution is the rule $grandparent(X, Y) \leftarrow parent(X, Z), parent(Z, Y)$ *as adding it to* B *would make it cover all positive examples and none of the negative ones.*

In addition to monotonicity, ILP systems can be categorised according to their learning approaches and how they search for the hypothesis. For this they can be divided into three different categories *bottom-up*, *top-down*, and *meta-level* learning. Bottom-up ILP systems use inverse entailment, generating a hypothesis that is the most specific explanation to the examples and then generalising it to acquire its solution. Top-down ILP systems search in the reverse direction and generate an overly general hypothesis that explains all positive examples, and possibly some negative examples, and then specialise it to remove the negative examples covered. Lastly, meta-level ILP systems is a subcategory of both bottom-up and top-down approaches that reasons at the meta-level instead of the object level.

Bottom-up monotonic ILP

Bottom up monotonic ILP systems compute hypotheses by finding the *bottom set*, generalising it, and then adding it to the learning task's hypothesis. This idea was proposed in Progol [Mug95] and is defined for learning tasks where examples can be definite clauses. As a result of this, Skolemisation is applied to examples to ground them. For learning tasks with only grounded examples, Skolemisation is not needed. The observation made in [Mug95] is based on the requirement of the inductive task that all positive examples are covered by the hypothesis

$B \cup H \models E^+$. Thus it must also be the case that $B \cup \overline{E^+} \models \neg H$, where $\overline{E^+}$ is the complement of E^+, a (possibly Skolemised) disjunction of literals acquired by negating each clause in E^+ and Skolemising its variables.

Definition 3.4 (Bottom Set)**.** *Given background knowledge B and a positive example $e \in E^+$. The bottom set $\neg L$ is a set of ground atoms such that $B \cup \overline{e} \models \neg L$ where $\overline{e}$ is the complement of e.*

The bottom set aids the search by restricting the search space to include only clauses that generalise it. The notion of generality used by ILP systems is θ-subsumption [NdW97].

Definition 3.5 (θ-subsumption)**.** *For two clauses A and B, A θ-subsumes B if and only if there exists a substitution θ such that $A\theta \subseteq B$.*

The bottom set used to derive a set of clauses h such that h subsumes L, contain no Skolem constants (if ungrounded examples are used), and does not cover any negative examples. As the hypothesis being learnt is monotonic, this process can be repeated for each positive example to find a clause that covers it. All such clauses are then unified to produce a solution to the learning task.

Top-down monotonic ILP

Top-down monotonic ILP systems, for example TopLog [MSTn08], compute hypotheses by searching from the general to specific. They use the language bias to constrain the search space of the hypothesis as a declarative theory, then search through it to find the best hypothesis that explains the given examples. In TopLog this is done through the *top theory* $\top$, a theory from which the hypothesis is derived. Similarly to bottom-up monotonic ILP, the search is carried out by learning clauses h, each of which explains one or more examples $e \in E^+$ such that $B \cup h \models e$ and $\top \models h$. The solution to the learning task is then found by combining all such clauses. Note that TopLog's top theory contains a set of "non-terminal" predicate symbols which are used to link body predicates together. For example, the clause $p(X) \leftarrow q(X)$ is entailed by the clauses:

$p(X) \leftarrow \$body(X).$

$\$body(X).$

$\$body(X) \leftarrow q(X).$

in the top theory, with non-terminal predicate $body/1$. The clause $p(X) \leftarrow q(X)$ would be learnt by searching top theory, and learning $h_\top = \{p(X) \leftarrow \$body(X).; \$body(X) \leftarrow q(X).\}$. These clauses in $h_\top$ are then reordered to remove the non-terminal predicate $\$body(X)$, resulting in $\{p(X) \leftarrow q(X).\}$

Meta-level monotonic ILP

ILP systems that use meta-level learning approaches abstract an ILP task into some form of meta-level representation. An additional meta-theory is often used to represent the hypothesis space or to help with interpreting its meta-level information.

Definition 3.6 (Meta-level inductive task)**.** *Given an inductive task $\langle E^+, E^-, B, M\rangle$, some transformations τ_1, τ_2, and τ_3 are applied to the task, converting it into a meta level-representation. With some additional meta theory T_M, the solution H to the task is found by finding $\tau_1(H)$ such that $\tau_2(B) \cup T_M \cup \tau_1(H)$ covers all elements of $\tau_3(E^+)$ and no elements from $\tau_3(E^-)$. To acquire H the inverse transformation $\tau_1^{-1}/1$ is applied, giving $H = \tau_1^{-1}(\tau_1(H))$.*

Definition 3.6 is the most general form of a meta-level ILP task and contain separate transformation functions ($\tau_1/1$, $\tau_2/1$ and $\tau_3/1$). In practice, most systems apply the same transformation to the background knowledge, hypothesis, and examples of the inductive learning task. For example, DIALOGS [Fle97] transform the inductive learning task into second-order logic and then use clause templates to synthesise its solution. Similarly, Metagol [ML13] also transform everything into second-order logic. Currently, meta-level ILP system use at most two different transformations such as in TAL [CRL10] which use the identity transformation for its background knowledge and examples, and another transformation for its hypotheses. TAL will be discussed in more detail in Section 3.2.

Most meta-level ILP systems, such as Metagol, apply abductive reasoning to find the meta-level representation of the solution, which is then transformed to its object level representation to give the solution to the original task. On the other hand, DIALOGS use a combination of inductive and abductive reasoning to find its solution. DIALOGS uses abduciton to generate examples for uninstantiated predicates in its clause templates, and these are then generalised to produce the definition of those predicates. For the rest of this section we will discuss Metagol in more detail as its learning approach is most relevant for Chapter 5 of this work.

Metagol [ML13] is a monotonic system that fully transform its task into the meta-level representation where each rule in the background knowledge is replaced by a predicate, for example $meta_a$, $meta_b$, etc., that describes its structure, and whose arguments are the original predicate names and constants used in the original rule. For instance, the fact $grandparent(jim, kim)$ can be transformed into $meta_a(grandparent, jim, kim)$, whereas the rule $grandparent(X, Y) \leftarrow parent(X, Z)$, $parent(Z, Y)$ can be transformed into $meta_b(grandparent, parent, parent)$. Note that if there is another rule with a different structure such as $grandparent(X, Y) \leftarrow parent(Y, Z)$, $parent(Z, X)$, it must be transformed into another meta predicate such as $meta_c(grandparent, parent, parent)$. Effectively each differently named *meta* represents a different rule structure. With respect to Definition 3.6, Metagol applies the same transformation τ to its examples, background knowledge and hypothesis.

Metagol searches for a hypothesis using a *meta-interpreter*, a logic program that describes the acceptable structure of the hypothesis. For example, for hypotheses where the predicates have up to two arguments the following theory structure can be used:

$P(a, b).$
$P(X, Y) \leftarrow Q(X, Y).$
$P(X, Y) \leftarrow Q(X, Z), R(Z, Y).$

The above rules represent a theory where P, Q and R are predicate names, X, Y and Z are variable arguments, and a and b are constant arguments. Let $predicate2/1$ be the type for predicate names that have two arguments, and let $object/1$ be the type for constant argument. The above theory can then be transformed into the following a meta-interpreter:

$$
\begin{aligned}
1: \quad & prove(P,X,Y) \leftarrow meta1(P,X,Y).\\
2: \quad & prove(P,X,Y) \leftarrow predicate2(Q), meta2(P,Q), prove(Q,X,Y).\\
3: \quad & prove(P,X,Y) \leftarrow predicate2(Q), predicate2(R), meta3(P,Q,R), object(Z),\\
& \qquad\qquad prove(Q,X,Z), prove(R,Z,Y).
\end{aligned}
$$

In the above theory $meta1/3$, $meta2/2$, and $meta3/3$ are used to represent the rule structures from the meta-interpreter. $meta1(p,a,b)$ represents a predicate p with two constant arguments $p(a,b)$, while $meta2(p,q)$ represents a rule with the predicate $p/2$ in the head and $q/2$ in the body $p(X,Y) \leftarrow q(X,Y)$, and lastly $meta3(p,q,r)$ represents the rule $p(X,Y) \leftarrow q(X,Z), r(Z,Y)$. Suppose we want to use the above meta-interpreter with Example 11, firstly the background knowledge of the learning task is transformed into the following meta representation:

$$
\tau(B) = \left\{
\begin{array}{l}
object(jim).\\
object(ivan).\\
object(kim).\\
predicate2(parent).\\
meta1(parent,jim,ivan).\\
meta1(parent,ivan,kim).
\end{array}
\right\}
$$

The task is then solved as an abductive task $\langle T, A, G\rangle$ where the background theory T is the union of the meta-interpreter and $\tau(B)$, the abducibles A are $meta2/2$ and $meta3/3$ atoms, and the goal is the query $\leftarrow prove(grandparent, jim, kim)$. Note that the negative examples are not used as part of the goal of the abductive task, but are instead used for checking that the union of T and the abduced atoms does not entail any negative examples. When solving the abductive task, line 1 of the meta-interpreter will fail as there does not exist $meta1(grandparent, jim, kim)$ in $\tau(B)$. Line 2 of the meta-interpreter could only learn the rule $meta2(grandparent, parent)$ representing the rule $grandparent(X,Y) \leftarrow parent(X,Y)$ which will not be learnt as it does not cover $grandparent(jim, kim)$. The learner will solve the learning task by applying line 3 of the meta-interpreter and abduce $meta3(grandparent, parent, parent)$ representing the rule $grandparent(X,Y) \leftarrow parent(X,Z), parent(Z,Y)$. This succeeds as it

would solve the query and not cover any negative examples, and would not be reject by a post-process check for negative examples coverage.

3.2 Non-monotonic Inductive Logic Programming

Learning a non-monotonic theory and learning using a non-monotonic theory presents two obstacles. The first obstacle is that the monotonic approach of iteratively learning from one selected example can lead to incorrect solution as the consequences of a theory may change as rules are added to it. The second obstacle is that there can be multiple models to a non-monotonic theory and when the background is extended with the learned solution, the notion of examples coverage or (not-)entailment of examples may have two different semantic meanings, namely brave and cautious semantics [SI09]. Let us consider the first problem using the following example.

Example 12. *Given the following learning task:*

$$B = \left\{ \begin{array}{l} p(X) \leftarrow q(X), not\ r(X). \\ s(c). \\ char(a). \\ char(b). \\ char(c). \end{array} \right\} M = \left\{ \begin{array}{l} modeh(q(+char)). \\ modeh(r(+char)). \\ modeb(s(+char)). \end{array} \right\} E^+ = \left\{ \begin{array}{l} p(a). \\ r(c). \end{array} \right\}$$

Using monotonic ILP learning convention, in order to find the solution of the inductive learning task we would consider the first example $p(a)$ and learn a hypothesis h_1 that covers it. The most concise h_1 would be $\{q(X).\}$, which is added to the background knowledge. As h_1 does not cover the other example $r(c)$, the next step is to learn another hypothesis h_2 that covers $r(c)$, leading us to $\{r(X).\}$. Separately, h_1 and h_2 cover all positive examples, however adding h_2 to $B \cup h_1$ makes it no longer covers the first example. Although the learning task is solvable through the hypothesis $\{q(X).; r(X) \leftarrow s(X).\}$ which can only be found if the learner learns from all examples together, as opposed to learning from only a subset of the examples.

As all examples must be considered when the background is a normal logic program, it would be

sensible for the positive E^+ and negative E^- examples of an ILP task to be combined together as $E = \{e_1^+, \ldots, e_n^+, not\ e_1^-, \ldots, not\ e_m^-\}$, where each $e_i^+ \in E^+$ and $e_i^- \in E^-$.

For the second obstacle, the presence of negation as failure in the background and hypothesis means that the normal theory may have many models. For instance, consider the following example from [SI09]:

Example 13. *Given the following learning task:*

$$B = \left\{ \begin{array}{ll} person(adam). & mountain(X) \leftarrow not\ sea(X), person(X). \\ person(nancy). & couple(adam, nancy). \\ person(bob). & couple(bob, jane). \\ person(jane). & \bot \leftarrow couple(X,Y), sea(X), mountain(Y). \\ sea(X) \leftarrow not\ mountain(X), & \bot \leftarrow couple(X,Y), sea(Y), mountain(X). \\ \qquad person(X). & \end{array} \right\}$$

$$E^+ = \left\{ \begin{array}{l} tanned(adam). \\ tanned(nancy). \end{array} \right\} \qquad E^- = \left\{ \begin{array}{l} tanned(bob). \\ tanned(jane). \end{array} \right\}$$

$$M = \left\{ \begin{array}{ll} modeh(tanned(+person)). & modeb(sea(+person)). \\ modeb(mountain(+person)). & modeb(not\ sea(+person)). \\ modeb(not\ mountain(+person)) & \end{array} \right\}$$

The background knowledge B contains an integrity constraint to ensure that every couple can exclusively go on holiday at the sea or the mountain. The constraints, and the normal rules in B, creates four possible different consequences where both couples go to the same location, or where one couple goes to the sea while the other goes to the mountain. For instance, even if *adam* could go to the sea, it is not classically entailed by the background $B \not\models sea(adam)$. Thus, while it is reasonable to consider the hypothesis $H = \{tanned(X) \leftarrow sea(X),\ \text{not}\ mountain(X)\}$ as a solution to the task, not all models of $B \cup H$ will explain all positive examples $B \cup H \not\models \bigwedge_{e \in E^+} e$. $tanned(adam)$ and $tanned(nancy)$ will only be true in the models in which they go to the sea, and similarly $tanned(bob)$ and $tanned(jane)$ will not hold only if they go to the mountain

instead.

To be able to meaningfully reason with a non-monotonic theory we will use the operator $\models_{\mathcal{B}}$ to indicate brave entailment [SI09] and take into account the difference in example coverage.

Definition 3.7 (Cautious Entailment)**.** *A formula F is cautiously entailed by theory T ($T \models_{\mathcal{C}} F$) if F holds in* all *models of T.*

Definition 3.8 (Brave Entailment)**.** *A formula F is bravely entailed by theory T ($T \models_{\mathcal{B}} F$) if F holds in* at least one *model of T.*

Under brave entailment $H = \{tanned(X) \leftarrow sea(X),\ \text{not } mountain(X)\}$ can be considered as a valid hypothesis of the learning task in Example 13 as there exists a model of $B \cup H$ where $tanned(adam)$ and $tanned(nancy)$ holds, and $tanned(bob)$ and $tanned(jane)$ do not.

The non-monotonicity requires a different notion of inductive task as the previously defined coverage criteria (3.1) and (3.2) for monotonic ILP is not suitable for it. The coverage criteria for non-monotonic inductive task is as follows:

$$B \cup H \models_{\mathcal{B}} E \tag{3.3}$$

Note that (3.3) is the example coverage criteria that we will be using in this work. Other systems such as ILASP [LRB14] divides its positive and negative examples and use weaker coverage criteria which requires each positive example to be covered by a model that are not necessary the same one as models that cover other examples. So while ILASP requires for all examples must be covered, each positive example can be covered by a different model.

The non-monotonicity also changes the input of the learning task so that instead of having examples in separate set, they are combined into a single set with negation added for negative examples. These examples and coverage criteria gives us the brave inductive learning task following the definition given in [SI09]:

Definition 3.9 (Brave Inductive Learning Task)**.** *A brave inductive task is a tuple $\langle E, B, M \rangle$, where E is the set of ground examples, B is the background knowledge expressed as a normal*

logic program, and M is a set of mode declarations. A solution to a brave inductive task is a set of normal clauses, a hypothesis H compatible with M, where at least one model of $B \cup H$ implies all examples in E ($B \cup H \models_{\mathcal{B}} E$).

Similar to monotonic ILP, non-monotonic ILP can also be divided into the subcategories according to their learning algorithm.

Bottom-up non-monotonic ILP

XHAIL [Ray09], an extension of HAIL [RBR03], is an ILP system that uses abductive learning for computing its hypothesis. Rather than using the bottom set, XHAIL uses a *kernel set*, a theory of the most specific rules for the hypothesis. It solves an ILP task using three separate phases: (i) abductive phase; (ii) deductive phase; and (iii) inductive phase. All phases are solved as an abductive task. The abductive phase finds all possible ground head literals for rules in the hypothesis according to the given mode bias. The deductive phase finds the body literals for the head learnt in the previous phase with respect to the given bias and linking of the variables, creating a kernel set. This is transformed into a pseudo kernel set, where ground terms are replaced by variables. In the last phase the hypothesis of the learning task is computed from the pseudo kernel set such that only vital body literals for distinguishing the negative examples from the positive examples are retained while other are discarded. The output of the algorithm is the most compressed hypothesis that bravely covers the examples.

Top-down and meta-level non-monotonic ILP

TAL (Top-directed Abductive Learning) [CRL10] is an ILP system that solves an inductive learning task by converting it into an equivalent abductive learning task (ALP). While it may have been inspired by another top-down system TopLog [MSTn08], its approach of fully translating the learning task into an abductive one allows for normal theories to be used in both the background knowledge and the learnt hypothesis, as well as for multiple clauses to be learnt.

TAL's transformation of an inductive task $\langle E, B, M\rangle$ into an abductive task relies upon two features: (i) its hypothesis representation; and (ii) a theory called the *top theory* that encodes the hypothesis space. For the hypothesis representation TAL labels each mode declaration and uses this label and lists of variable argument index and constant variables for encoding each hypothesis as an atom. For example consider the following mode declarations labelled by $\{m_1, m_2, m_3, m_4\}$:

$$M = \left\{ \begin{array}{l} m1 : modeh(grandparent(+person, +person)). \\ m2 : modeb(parent(+person, +person)). \\ m3 : modeb(parent(+person, -person)). \\ m4 : modeb(parent(\#person, +person)). \end{array} \right\}$$

Each mode declaration can then be represented by a tuple (L, Os, Is) where L is its label, Os is the list of constant arguments, and Is is the list of input variables according to their order of appearance. A list of such tuples is used to represent a clause, and a set of such list is then used to represent the hypothesis. For instance, from the above mode declarations the hypothesis $\{grandparent(X, Y) \leftarrow parent(X, Z), parent(Z, Y)\}$ is represented by the list $[[(m1, [], []), (m3, [], [1]), (m2, [], [3, 2])]]$. The first element of each tuple indicate the mode declaration associated with the literal through its label. The second argument of each tuple is a list of constant arguments in each literal. Thus hypotheses with constants such as $\{grandparent(X, Y) \leftarrow parent(ivan, Z)\}$ will be represented by the list $[[(m1, [], []), (m4, [ivan], [])]]$. Lastly, the third element of each tuple is a list of indexes indicating what previously encountered variables the input variables of a literal is linked to. For instance in $(m3, [], [1])$ the list $[1]$ indicates that the input variable in $modeb(parent(+person, -person))$ is linked to the first variable X of the rule, making the tuple represents the literal $parent(X, Z)$. Similarly, the list $[3, 2]$ in $(m2, [], [3, 2])$ corresponds to the variable Z and Y respectively. Note that the linking of variables are captured by the indexing of the variables according to the order they appear in the rule. Thus head literals do not have any links as its variables do not appear in any previous literals.

The second component of TAL's transformation is the top theory $\top$. For a schema s in a mode declaration, let $s*$ be the schema s with variables replacing its placemarkers, $types(s*)$ be the conjunction of type constraints of variables in $s*$, $id(s*)$ be the label of mode declaration of $s*$, $ground(s*)$ be the list of constant arguments in $s*$, $inputs(s*)$ be the list of input variables in $s*$, $outputs(s*)$ be the list of output variables in $s*$, and $link\_variables/3$ be a function that given the inputs variables of $s*$ and the list of variables in previous literals will return the index of literals unifiable with input variables in $s*$. The top theory is derived from the mode declarations as follows:

- For each $modeh(s) \in M$ the following clause is added to $\top$:

$$s* \leftarrow types(s*), prule(RId, [(id(s*), ground(s*), [])]), rule\_id(RId),$$
$$body(RId, inputs(s*), [(id(s*), ground(s*), [])]).$$

- The following clause is in $\top$:

$$body(RId, \_, Rule) \leftarrow rule(RId, Rule).$$

- For each $modeb(s) \in M$ the following clause is added to $\top$:

$$\begin{aligned} body(RId, Inputs, Rule) \leftarrow & types(s*), s*, \\ & link\_variables(inputs(s*), Inputs, Links), \\ & append(Rule, [(id(s*), ground(s*), Links)], NewRule), \\ & append(Inputs, outputs(s*), Outputs), \\ & prule(RId, NewRule), \\ & body(RId, Outputs, NewRule). \end{aligned}$$

Effectively the first clause starts the process of building a rule by encoding the head literal and calling $body/3$ to search for possible body literals that can be added to it. The second rule is the base case for $body/3$ which will return a rule through the abducible $rule/2$ once the goal of

the abductive task is met. The third rule is the recursive case for *body* that adds a body literal to the rule and collects output variables to pass onto subsequent body literals. The first and third rule contains another abducible predicate $prule/2$ which abduces a partial rule that can be used as part of a heuristic based search strategies.

For instance, consider Example 11. The top theory for this task is:

$$\top = \left\{ \begin{array}{l} grandparent(X,Y) \leftarrow person(X), person(Y), prule(RId, [(m1, [], [])]), \\ \quad rule\_id(RId), body(Rid, [X,Y], [(m1, [], [])]). \\ body(RId, \_, Rule) \leftarrow rule(RId, Rule). \\ body(RId, Inputs, Rule) \leftarrow person(X), person(Y), parent(X,Y), \\ \quad link\_variables([X,Y], Inputs, Links), append(Rule, [(m2, [], Links)], NewRule), \\ \quad append(Inputs, [], Outputs), prule(RId, NewRule), body(RId, Outputs, NewRule). \\ body(RId, Inputs, Rule) \leftarrow person(X), person(Y), parent(X,Y), \\ \quad link\_variables([X], Inputs, Links), append(Rule, [(m3, [], Links)], NewRule), \\ \quad append(Inputs, [Y], Outputs), prule(RId, NewRule), body(RId, Outputs, NewRule). \end{array} \right\}$$

The definition of the ALP task used by TAL is a modification from Definition 2.7, and has been defined in [CRL10] as follows:

Definition 3.10 (TAL Abductive Task)**.** *An abductive learning task is defined as* $\langle g, T, A, I \rangle$ *where* g *is a grounded goal,* T *is a normal logic program,* A *is a set of abducible facts, and* I *is the set of integrity constraints. The solution of the abductive task is* $\Delta \subseteq A$ *such that* $T \cup \Delta$ *is consistent,* $T \cup \Delta \models_{\mathcal{B}} g$ *and* $T \cup \Delta \models_{\mathcal{B}} I$*.*

Using the top theory $\top$, given an inductive task $\langle E, B, M \rangle$, TAL will transform it to an abductive task $\langle E, B \cup \top, A, \emptyset \rangle$ where A is set of $prule/2$ and $rule/2$ instances, representing partial rules and full rules. The solution of the original inductive task is acquired by transforming the abduced $rule/2$ instances into its clausal representation. On the other hand, abduced $prule/2$ are not transformed back into clausal representation.

With respect to Example 11 and the above top theory, the solution for the abductive task is

the following set:

$$\left\{ \begin{array}{l} rule(1,[(m1,[],[]),(m3,[],[1]),(m2,[],[3,2])]), \\ prule(1,[(m1,[],[])]), \\ prule(1,[(m1,[],[]),(m3,[],[1])]), \\ prule(1,[(m1,[],[]),(m3,[],[1]),(m2,[],[3,2])]) \end{array} \right\}$$

The abduced $prule(1,[(m1,[],[])])$ corresponds to the partial rule $grandparent(X,Y)$, while $prule(1,[(m1,[],[]),(m3,[],[1])])$ corresponds to $grandparent(X,Y) \leftarrow parent(X,Z)$. Both $rule(1,[(m1,[],[]),(m3,[],[1]),(m2,[],[3,2])])$ and $prule(1,[(m1,[],[]),(m3,[],[1]),(m2,[],[3,2])])$ corresponds to the rule $grandparent(X,Y) \leftarrow parent(X,Z), parent(Z,Y)$ which is both a full and partial rule, and is the solution to the learning task.

TAL can be considered as being a partial meta-level approach as although it uses meta-level representation for its hypothesis, the reasoning is still conducted at the object level. Thus with respect to Definition 3.6 both τ_2 and τ_3 are the identity transformation, τ_1 is TAL's hypothesis representation, and the meta-theory T_M is the top theory $\top$. As TAL's meta-theory is automatically constructed from the mode declarations of the learning task, its inductive task needs not be as specific as systems such as Metagol where explicit rule structures need to be given.

3.3 ILP for Theory Revision

Theory revision involves taking an existing theory and improving it in some manner. As described in [Wro96] this can be seen in two forms: (i) *theory refinement* where a theory is revised to make it more complete or correct by modifying the existing rules to change its model; and (ii) *theory reconstruction* where a theory is revised to make it more efficient or easier to understand by rewriting it in a different way while preserving the same models. The type of theory revision we are interested in is theory refinement where an existing theory is modified in some way so that it covers some given examples.

Revision of a task is done by applying some revision operations, for example extending or deleting a rule, to the given theory, producing a revised version of it. Each theory revision system has an associated set R of all revision operators it can use. A subset $\rho \subseteq R$ is then used for actual revision of a given task. We will use the symbol $\otimes$ to indicate the application of ρ on a theory T as $\rho \otimes T$. Our definition of a theory revision task extends the one given in [Wro96] to include non-revisable theory, and is as follows.

Definition 3.11 (Theory Revision Task)**.** *A theory revision task is a tuple* $\langle T_N \cup T_R, E^+, E^- \rangle$ *where* T_N *is the non-revisable definite theory and* T_R *is the initial revisable definite theory and* E^+ *and* E^- *is the set of positive and negative examples. Let* R *be a set of revision operators, a solution to the revision task is a revised theory* T'_R *such that (i)* $T'_R = \rho \otimes T_R$*; (ii)* $T_N \cup T'_R$ *covers all positive examples:* $T_N \cup T'_R \models E^+$*; and (iii)* $T_N \cup T'_R$ *does not cover any negative examples:* $\forall e \in E^- : T' \not\models e$*.*

Notice that the above requirement for the solution of the revision task is similar to that of an monotonic ILP task in that they both require the coverage of all positive examples and none of the negative examples. Examples of ILP system with theory revision includes HYPER [Bra99] and the enhancement of FORTE [RM95] by the bottom set [DPZ09]. HYPER is a system that iteratively modify its hypothesis, and we will describe in more detail later in this section as, like our system RASPAL, it is a top-down ILP system that use hypothesis refinement.

For non-monotonic ILP the following generalised definition of a revision task [Wro96] can be used where the examples are replaced by a set of integrity constraints to be satisfied by the revised theory.

Definition 3.12 (Generalised Theory Revision Task)**.** *A theory revision task is a tuple* $\langle T_N \cup T_R, IC \rangle$*, where* T_N *is the non-revisable theory,* T_R *is the initial theory and* IC *is the set of integrity constraints such that* $T_N \cup T_R$ *does not satisfy* IC*. The solution to the revision task is the revised theory* T'_R *such that (i)* T' *is generated from the smallest set of revision operations; and (ii)* $T_N \cup T'_R$ *satisfies all integrity constraints in* IC*.*

Note that the revision operations and the notion of the best revised theory may vary with the

learning systems.

HYPER

HYPER (Hypothesis Refiner) [Bra99] is a monotonic ILP system that refines a hypothesis until it covers all positive examples and none of the negated ones. This is achieved by first generating an overly general hypothesis (a hypothesis that covers all positive examples) using the given mode declarations, and then iteratively refine it so that it does not cover negative examples. The over general hypotheses are ones that cover all positive examples, and any hypotheses that do not cover all positive examples are discarded by the system. This is due to how HYPER refines its hypothesis which will only remove any covered negative examples, but not increase the coverage of positive examples. As it does not use integrity constraints to guide its refinement, its revision task follows that of Definition 3.11, and the refinement operations used by HYPER are as follows:

- Unifying two variables in a clause
- Refining a variable into a background term such as replacing a variable L with $[H|B]$
- Adding a body literal to a clause

At each iteration, the refined hypothesis H is evaluated using a scoring function which defines its cost as: $Cost(H) = w_1 * Size(H) + w_2 * NegCover(H)$ where w_1 and w_2 are weights, $Size(H)$ is the size of the hypothesis and $NegCover(H)$ is the number of negative examples covered by the hypothesis. The size of a hypothesis is calculated as: $Size(H) = k_1 * \#literal(H) + k_2 * \#variables(H)$ where k_1 and k_2 are weights, $\#literal(H)$ is the number of literals in H, and $\#variables(H)$ is the number of variables in H. The learner tries to minimise the overall cost of the hypothesis, leading it to refine its hypotheses into the most compact hypothesis that will cover the fewest negative examples. This makes HYPER favours its first two refinement operations over the last one of adding a body literal, thus favouring the refinement that will produce a theory most similar to its original theory.

Example 14. *Given the following learning task:*

$$B = \left\{ \begin{array}{ll} person(grace). & female(kim). \\ person(helen). & male(ivan). \\ person(ivan). & male(jim). \\ person(jim). & parent(jim, ivan). \\ person(kim). & parent(grace, ivan). \\ female(grace). & parent(ivan, kim). \\ female(helen). & parent(helen, kim). \end{array} \right\}$$

$$E^+ = \left\{ \begin{array}{l} father(jim, ivan). \\ father(ivan, kim). \end{array} \right\} \qquad E^- = \left\{ \begin{array}{l} father(grace, ivan). \\ father(ivan, jim). \end{array} \right\}$$

$$M = \left\{ \begin{array}{l} modeh(parent(+person, +person)). \\ modeb(male(+person)). \\ modeb(female(+person)). \end{array} \right\}$$

Let $k_1 = w_2 = 10$ and $w_1 = k_2 = 1$. The overgeneral hypothesis that explains all examples is $\{father(X, Y).\}$ and its cost is 32. Consider HYPER's three refinement operations. Applying the first operation of variable unification to acquire $\{father(X, X)\}$ will improve the score as both negative examples will no longer be covered, however it would no longer cover any positive examples, making it an inappropriate refinement operation. For the second operation there isn't a way to refine variables in $father(X, Y)$ into another term such that all positive examples are still covered as there is only one instance of $father(X, Y)$ in the hypothesis, thus the operation cannot be applied. This leaves only the third operation of adding a literal to the hypothesis. For the third refinement operation the mode declarations $parent/2$ or $male/1$ can be used to extend a rule. There are two extensions that would remove negative examples coverage and still retain the positive examples coverage: (i) $\{father(X, Y) \leftarrow male(X).\}$ and (ii) $\{father(X, Y) \leftarrow parent(X, Y).\}$. Each one of these refinements will remove one negative example coverage and add a body literal, thus retaining the cost of 32. Further refinement of either of them will result in finding the hypothesis $\{father(X, Y) \leftarrow parent(X, Y), male(X).\}$ Note that HYPER

does not require that the refinement strictly improves the cost of the hypothesis and will also consider refinements that retain the cost of the current hypothesis.

Suppose HYPER is applied to Example 11, then it would first generate the hypothesis $\{grandparent(X, Y)\}$ as this is the most general hypothesis that covers all the positive example. However it would be stuck in the next iteration as both $parent(X, Z)$ and $parent(Z, Y)$ are needed in order to remove any negative examples covered and improve its cost. However, HYPER could only add one body literal to a clause it will not be able to improve the hypothesis.

3.4 ASPAL

ASPAL (ASP Abductive Learning) [CRL11], an Answer Set Programming (ASP) implementation of TAL, is the most relevant ILP system for this work. Recall that TAL is implemented in Prolog and uses its list structure to iteratively build rules. However, ASP does not have such a structure, and using list-like structure in ASP is inefficient and often impossible as the solver completely grounds the program before solving it. Consider using TAL's top theory in ASP; not only would there be a lot of grounding needed for variable linkage alone, every possible body literal link would also need to be grounded regardless of where it is a permutation of another existing link. In contrast to each clause in the top theory corresponding to a mode declaration schema, each clause in ASPAL's top theory corresponds to a rule in the hypothesis space. Furthermore the body literals can be ordered to remove redundancy. Note that this would work best in the absence of output variables as they will make the ordering matter due to TAL's hypothesis representation. Consider the rule $grandparent(X, Y) \leftarrow parent(X, Z), parent(Z, Y)$. Using TAL's hypothesis representation $parent(X, Z)$ must have appeared in the rule before $parent(Z, Y)$ as it takes the output variable Z as its input. Thus, while ordering can be used to reduce the hypothesis space, in the presence of output variables literals with output will need to be placed at the beginning of the rule, and multiple permutations of literals with outputs may be required to represent the entire hypothesis space. For instance, consider the mode declarations $modeb(p(+int, -int))$ and $modeb(q(+int, -int))$. If $p/2$ is placed before $q/2$, such

as $p(X,Y), q(Y,Z)$, then in the hypothesis encoding $q/2$ can include a variable index pointing to the output of p. However, as $p/2$ is in front of $q/2$ then in the encoding it cannot refer to the output of $q/2$.

Similar to TAL, ASPAL also transforms an inductive task into an abductive one. However, the two main components, the hypothesis representation and the top theory, have to be adapted for ASP. For the hypothesis representation, as each clause in the top theory corresponds to a rule, there is no longer a need to build up the rules in the hypothesis literal by literal.

Definition 3.13 (ASPAL rule encoding)**.** *Each rule* $r = h \leftarrow b_1, \ldots, b_n$ *in the hypothesis space can be encoded* $enc(r)$ *as* $rule(id(r), (C_1, \ldots, C_n))$*, where* $id(r)$ *is a tuple of the form* $(m_h, m_1, l_{1,1}, \ldots, l_{1,p_1}, \ldots, m_n, l_{n,1}, \ldots, l_{n,p_n})$*, where* m_h *is the label of the mode declaration* h*,* m_i *is the identifier of the mode declaration for body literal* b_i*, and for each* m_i*, the elements* $l_{i,1}, \ldots, l_{i,p_i}$ *are indexes denoting the variables in the rule, counting from start of the rule, which the input variable of the literal* b_i *are linked to.* $(C_1, \ldots, C_n)$ *is the tuple of constant arguments in* r*.*

Note that when a rule has no constant arguments $\$e$ is used in place of $(C_1, \ldots, C_n)$ to indicate the empty tuple of constant.

For example, suppose we use the following mode declarations from Example 11.
$m1 : modeh(grandparent(+person, +person))$.
$m2 : modeb(parent(+person, +person))$.
$m3 : modeb(parent(+person, -person))$.

Then the encoding for the rule $r = grandparent(X,Y) \leftarrow parent(X,Z), parent(Z,Y)$ will be $rule((m1, m3, 1, m2, 3, 2), \$e)$ where $id(r)$ is the tuple $(m1, m3, 1, m2, 3, 2)$ and $\$e$ is the empty tuple of constants.

Unlike TAL, ASPAL's rules are not learnt recursively. Instead each clause in the top theory acts like a "switch". For each rule $r = h \leftarrow b_1, \ldots, b_n$ in the hypothesis space the following rule is in ASPAL's top theory: $h \leftarrow b_1, \ldots, b_n, enc(r)$. By abducing $enc(r)$ a rule is "activated" and is included in the hypothesis.

Definition 3.14 (Encoding of Top-theory)**.** *Let R_M be the set of rules that makes up the hypothesis space constructed from the given mode declarations M of the learning task. For each rule $r = h \leftarrow b_1, \ldots, b_n$ in R_M, the rule $h \leftarrow b_1, \ldots, b_n, enc(r)$ is in ASPAL's* top-theory $\top$*.*

We have already mentioned that the top theory of ASPAL has one clause per rule in the hypothesis space. Note however that this correspondence is not one to one as constant arguments are only instantiated when grounding the top theory. Secondly the variables in a predicate head h do not link to any variable occurring before them in the rule so there is no sequence $l_{0,1}, \ldots, l_{0,p_0}$ after m_0 in the representation. The structure of $rule/2$ makes this translation bijective, allowing automatic inverse translation of abduced atoms to related rule hypothesis.

With ASPAL's top theory $\top$ and rule encoding [Cor11], given an inductive task $\langle E, B, M\rangle$, ASPAL transforms it into an abductive task $\langle g, B \cup \top, A^\top, \emptyset\rangle$ where g is the set of clauses $\{\bot \leftarrow not\ examples.; examples \leftarrow \bigwedge e \in E.\}$ and $A^\top$ is the set of $rule/2$ atoms. The solution of the original inductive task is acquired by transforming the abduced $rule/2$ instances into its clausal representation

Definition 3.15 (Decoding the Abducibles)**.** *Let R_M be set rules that makes up the hypothesis space of a learning task. The set $A^\top = \{rule(id(r), (C_1, \ldots, C_n)) \mid r \in R_M\}$ is called* abducibles*. Given a subset $\Delta \subseteq A^\top$, for each $rule(id(r), (c_1, \ldots, c_n)) \in \Delta$ its inverse transformation $enc^{-1}(rule(id(r), (c_1, \ldots, c_n)))$ is the rule $r = h \leftarrow b_1, \ldots, b_m$ with each constant variable of r being replaced by $c \in (c_1, \ldots, c_n)$ in order of their appearances.*

Definition 3.16 (Learning Task Transformation)**.** *Given an ILP task $\langle E, B, M\rangle$, let $\top$ be the top theory representing the rules R_M in the hypothesis space of the task, $A^\top$ be the set of abducibles, and let g be the set of rules $\{examples \leftarrow \bigwedge e \in E.; \bot \leftarrow not\ examples.\}$. The original inductive task can be solved through an abductive task $\langle \emptyset, B \cup \top \cup g, A, \emptyset\rangle$. The solution to the abductive task is $\Delta \subseteq A^\top$ such that $B \cup \top \cup g \cup \Delta$ has a stable model. The solution to the original inductive learning task is the hypothesis H where $H = \{enc^{-1}(d) | d \in \Delta\}$.*

Example 15. *Given the following learning task:*

$E = \{has\_cold(alice), not\ has\_cold(bob).\}$

$$B = \left\{ \begin{array}{l} person(alice). \\ person(bob). \\ condition(blister). \\ condition(sore\_throat). \\ symptom(alice, sore\_throat). \\ symptom(bob, blister). \end{array} \right\}$$

$$M = \left\{ \begin{array}{l} m_1 : modeh(has\_cold(+person)). \\ m_2 : modeb(symptom(+person, \#condition)). \end{array} \right\}$$

The rules within the hypothesis space that have at most two body literals are:

$$R_M = \left\{ \begin{array}{l} has\_cold(X). \\ has\_cold(X) \leftarrow symptom(X, Y). \\ has\_cold(X) \leftarrow symptom(X, Y), symptom(X, Z). \end{array} \right\}$$

These rules in R_M would correspond to the top theory:

$$\top = \left\{ \begin{array}{l} has\_cold(X) \leftarrow rule((m_1), \$e). \\ has\_cold(X) \leftarrow symptom(X, Y), rule((m_1, m_2, 1), (Y)). \\ has\_cold(X) \leftarrow symptom(X, Y), symptom(X, Z), rule((m_1, m_2, 1, m_2, 1), (Y, Z)). \end{array} \right\}$$

The minimal solution to the learning task is the subset $\Delta = \{rule((m_1, m_2, 1), (sore\_throat))\}$ corresponding to the rule $has\_cold(X) \leftarrow symptom(X, sore\_throat)$. Although not minimal $\Delta' = \{rule((m_1, m_2, 1, m_2, 1), (sore\_throat, sore\_throat))\}$ which is the rule $has\_cold(X) \leftarrow symptom(X, sore\_throat)$, $symptom(X, sore\_throat)$ and the union $\Delta \cup \Delta'$ would also cover all examples.

3.4.1 ASPAL for Theory Revision

Previous work [CRV+11] has demonstrated how ASPAL can be applied to theory revision task. This is achieved by transforming a revision task into an equivalent ILP task. We will use this mechanism of performing theory revision through an inductive task in the next chapter to make non-monotonic ILP systems able to learn iteratively.

The revision operations ASPAL uses are as follows.

Definition 3.17 (Revision operation)**.** *The refinement operations that ASPAL can apply on a partial hypothesis are: (i) adding a new rule; (ii) adding body literals to an existing rule; (iii) deleting an existing rule; and (iv) deleting body literals from existing rules.*

Given an initial theory T, ASPAL converts it into a revisable theory $r(T)$ using the following transformation.

Definition 3.18 (Revisable theory)**.** *For each rule r given by $h_i \leftarrow b_{i,1}, \ldots, b_{i,n}$ in a given theory T, let $vars(r)$ be the list of all variables in r and $vars(b_{i,j})$ be the list of all variables in a body literal $b_{i,j}$. The following clauses are added to the revisable form $r(T)$:*

- $h_i \leftarrow try(i, 1, vars(b_{i,1})), \ldots, try(i, n, vars(b_{i,n})), extension(i, vars(r_i))$
- $try(i, j, vars(b_{i,j})) \leftarrow b_{i,j}, not\ delete(i, j)$, for each $try(i, j, vars(b_{i,j}))$
- $try(i, j, vars(b_{i,j})) \leftarrow delete(i, j)$, for each $try(i, j, vars(b_{i,j}))$

The above revisable theory has three utility predicates $try/3$, $extension/2$ and $delete/2$. The $try/3$ clauses test if a body literal $b_{i,j}$ in the ith clause is needed in the revised clause. If it is no longer relevant, then the corresponding $delete(i, j)$ is learnt, indicating that it can be removed. The $extension/2$ literal added to each rule indicates whether a rule is retained in the refined theory, and in the case where it is retained, whether additional body literals are added to the rule.

The mode declaration M for the learning task consists of head mode declarations for new rules and the utility predicates $extensions/2$ and $delete/2$. It also consists of body mode declarations for any body literals that can be used to revise the given theory.

Definition 3.19 (Change Transaction)**.** *Let T be a theory. A change transaction C is a set of revision operations, clauses with extension/2 as head literals and delete/2 facts, that can be applied to a theory to refine it into a revised theory $C \otimes T = T'$.*

Definition 3.20 (Revision task in ILP)**.** *Given a theory revision task* $\langle T_N \cup T_R, IC \rangle$*, it can be solved through an ILP task* $\langle \emptyset, T_N \cup r(T_R) \cup IC, M \rangle$ *where* $r(T_R)$ *is the revisable form of* T_R *and* M *is the set of mode declarations for revision operations applicable to* $r(T_R)$*. The solution to the revision task is a revised theory* $T'_R = C \otimes r(T_R)$ *and* T'_R *satisfies* IC*, where the change transaction* C *is the solution to the inductive task that is compatible with* M*.*

The change transactions that can be learnt is made up of a possibly empty set of *delete*/2 facts, clauses with head predicate *extension*/2 and body literals using predicates from body mode declarations in M, and other clauses compatible with M. They correspond to the revision operators of ASPAL. The refined theory T' is generated from the original theory T and the learned change transactions C by the following steps:

- Each clause $r_{new} \in C$ that are not *extension* nor *delete*, with $r_{new} \notin T$, r_{new} is added to T'.
- For each pair: $r_i \leftarrow b_1, \ldots, b_n \in T$ and $extension(r_i, vars(r_i)) \leftarrow b_{n+1}, \ldots, b_m$ in C, the clause $r_i \leftarrow b_1, \ldots, b_n, b_{n+1}, \ldots, b_m$ is added to T'.
- Each clause in T that does not have a corresponding *extension*/2 clause in C is not added to T'.
- For each $delete(i, j)$ in C, the body literal $b_{i,j}$ is removed from clause r_i in T'.

Example 16. *Consider extending Example 14 with the following revisable theory:*

$$T_R = \left\{ \; father(X, Y) \leftarrow female(Y). \; \right\}$$

This can be transformed into a revisable rule represented by the following theory, and added to the background knowledge of the inductive task, making the background comprise of a non-revisable *and a* revisable *part.*

$$r(T_R) = \left\{ \begin{array}{l} father(X, Y) \leftarrow try(1, 1, vars(Y)), extension(1, vars(X, Y)). \\ try(1, 1, vars(Y)) \leftarrow female(Y), notdelete(1, 1). \\ try(1, 1, vars(Y)) \leftarrow delete(1, 1). \end{array} \right\}$$

The examples for the learning task can be expressed as ASPAL's examples constraint, and used as the integrity constraint for the revision task:

$$IC = \left\{ \begin{array}{l} \bot \leftarrow not\ examples. \\ examples \leftarrow grandparent(jim, kim), not\ grandparent(jim, ivan), \\ \qquad\qquad not\ grandparent(ivan, kim). \end{array} \right\}$$

For this example we want to show how the revisable theory $r(T)$ and $delete/2$ and $extension/2$ atoms are used to revise it. Thus, we will use the following mode declarations which only have head declarations for $delete/2$ and $extension/2$ and so does not allow new rules to be learnt:

$$M = \left\{ \begin{array}{l} modeh(extension(1, vars(+person, +person))). \\ modeh(delete(1, 1)). \\ modeb(male(+person)). \\ modeb(female(+person)). \end{array} \right\}$$

There are many change transactions that could be learnt to refine this theory:

- $C_1 = \left\{ \begin{array}{l} extension(1, vars(X, Y)). \\ delete(1, 1). \end{array} \right\}$
- $C_2 = \left\{ \begin{array}{l} extension(1, vars(X, Y)) \leftarrow male(X). \\ delete(1, 1). \end{array} \right\}$
- $C_3 = \left\{ \begin{array}{l} extension(1, vars(X, Y)) \leftarrow parent(X, Y). \\ delete(1, 1). \end{array} \right\}$
- $C_4 = \left\{ \begin{array}{l} extension(1, vars(X, Y)) \leftarrow parent(X, Y). \\ extension(1, vars(X, Y)) \leftarrow male(X). \\ delete(1, 1). \end{array} \right\}$
- $C_5 = \left\{ \begin{array}{l} extension(1, vars(X, Y)) \leftarrow parent(X, Y), male(X). \\ delete(1, 1). \end{array} \right\}$

Applying these change transactions will produce the following revised theories:

- $C_1 \otimes r(T) = \left\{ \; father(X,Y). \; \right\}$
- $C_2 \otimes r(T) = \left\{ \; father(X,Y) \leftarrow male(X). \; \right\}$
- $C_3 \otimes r(T) = \left\{ \; father(X,Y) \leftarrow parent(X,Y). \; \right\}$
- $C_4 \otimes r(T) = \left\{ \begin{array}{l} father(X,Y) \leftarrow parent(X,Y). \\ father(X,Y) \leftarrow male(X). \end{array} \right\}$
- $C_5 \otimes r(T) = \left\{ \; father(X,Y) \leftarrow parent(X,Y), male(X). \; \right\}$

In all change transactions $delete(1,1)$ *will remove* $female(Y)$ *from* $father(X,Y) \leftarrow female(Y)$. *The change transaction* C_1 *will only remove* $female(Y)$*, thus making the revised theory cover all positive and negative examples. The change transaction* C_2 *and* C_3 *remove* $female(Y)$ *and add, respectively,* $male(X)$ *and* $parent(X,Y)$*, making the revised theory cover all positive examples but also one negative example in each case. The change transaction* C_4 *produces a revised theory with the same example coverage as* C_2 *and* C_3*, but shows that a change transaction can include more than one way of revising a rule in* $r(T)$ *and applying the change transaction will create two versions of revised rules. The change transaction* C_5 *is the solution to the learning task as its revised theory will satisfies the integrity constraint* IC.

3.5 Discussion

This chapter gave an overview of differences in ILP systems in terms of their monotonicity and learning approaches. In addition to this some systems were highlighted and described in detail as they are relevant to subsequent chapters of this work. While recent works in non-monotonic ILP systems have increased the class of inductive learning tasks ILP can be applied to, the combinatorial complexity of the hypotheses often makes the learning inefficient. Prolog based systems such as TAL can mitigate this problem as a hypothesis is built by gradually adding head and body literals to it. However, for ASP based systems such as ASPAL this is a major

problem as all hypotheses in the hypothesis space are considered at the same time. This means that while in theory ASPAL is complete, in practical application there are learning tasks that it cannot solve as the search space becomes too large to handle. In the Chapter 4 we will discuss how theory revision method described in this chapter can be used for controlling the size of the hypothesis space in order to make the learning of large learning tasks more scalable.

Chapter 4

Learning through Hypothesis Refinement

Recall that ASPAL pre-processes the learning task to find all possible rule structures in its hypothesis space to construct its top theory. This method of pre-constructing the rules is used to remove utility predicates in order to reduce the grounding of its ASP program. Despite this, ASPAL can still produce extremely large grounded programs when used on learning problems of significant size. The size of the ungrounded program can be estimated using the upper bound on the number of skeletal rules (ungrounded rules compatible with the mode declarations) in the ASPAL top theory. In [CRL11] the formula for this upper bound was given as:

$$|\top| < |M_h| \times (|M_b| \times (max_o \times d_{max})^{max_i})^{d_{max}} \tag{4.1}$$

where M_h is the number of head mode declarations in the learning task, M_b is the number of body mode declarations, max_o and max_i is the largest number of output and input variables in the body mode declarations respectively, and d_{max} is the maximum number of body literals in a rule. A flaw in this formula is that if there is no output variables in the body mode declarations this upper bound will be 0, a vast underestimation. In addition to this the above formula only consider rules with exactly d_{max} number of conditions and disregard rules that have fewer of body literals than the maximum. These observations made us update the formula

to the following:

$$|\top| \leq \sum_{d=0}^{d_{max}} |M_h| \times (|M_b| \times (max_i^h + max_o \times (d-1))^{max_i})^d \tag{4.2}$$

where M_h is the number of head mode declarations in the learning task, M_b is the number of body mode declarations, max_o and max_i is the largest number of output and input variables in the body mode declarations respectively, d_{max} is the maximum number of body literals in a rule, and max_i^h is the largest number of input variables in all head mode declarations. The updated formula sums the maximum number of rules for rules of different lengths up to those with maximum number of body conditions. The parameter max_i^h has been included as variables in a body literal can either be from input variables in the head literal or output variables in other body literals, giving $max_i^h + max_o \times (d-1)$ as the total number of variables that can serve as a inputs to a body literal.

Even though this formula of the upper bound does not take into account body literals ordering, the type constraints or variable linkage constraints of the mode declarations, it is still clear that even for a small learning task, the size of $\top$ increases exponentially with an increase in the maximum number of body literals in a rule d_{max}. Note that as this is the ungrounded rules, if the learning task also has a large domain, the size of the grounded theory can easily become large enough to be unsolvable, as the ASP solver runs out of memory.

In Equation 4 the parameter that affects the upper bound the most, as well as being the only parameter not dependent on the learning task, is the maximum number of body literals in the rule d_{max}. Using this observation we address the scalability problem of ASPAL by limiting d_{max}, thus restricting the size of its top theory.

This chapter describes a learning approach that uses hypothesis refinement, an idea that was raised in[Cor11] as response to the above scalability problem experienced by ASP based systems such as ASPAL. The aim of learning through hypothesis refinement in ASP is to make such systems more scalable by dividing the learning of a large hypothesis into smaller and more manageable refinements. Thus, instead of learning a large hypothesis in a single computation,

a partial hypothesis is learnt and iteratively refined until it becomes a solution to the learning task.

RASPAL (Refinement ASP Abductive Learning) is the learning algorithm which given an inductive learning task, and a limit on the maximum length of rules in the hypothesis, will find a solution to the learning task using iterative theory refinement. Instead of directly learning a solution to a learning task, RASPAL learns a *partial* hypothesis and then repeatedly refines it until it becomes a solution to the task. In the remaining of this chapter we will present how partial hypotheses can be learnt and the criteria for comparing them against one another. We will also be extending ASPAL's method for solving revision task, previously described in Section 3.4.1, to make it applicable to refining partial hypotheses. The methods for learning and refining partial hypotheses will be used in RASPAL's algorithm, which will be presented for learning tasks without noise, as well as a modified version for learning tasks with noise.

A partial hypothesis is a hypothesis that does not cover all positive examples, or covers some negative examples.

Definition 4.1 (Partial hypothesis)**.** *A set of rules H is a partial hypothesis of an inductive learning task $\langle E, B, M\rangle$ if it is compatible with M and $B \cup H \not\models_{\mathcal{B}} E$.*

RASPAL's learning method can be separated into two phases: (i) the initial learning phase where an initial partial hypothesis is learnt; and (ii) the iterative refinement phase where the partial hypothesis is improved until it becomes a solution.

4.1 Learning a partial hypothesis

Recall that in ASPAL the examples are used to create the following integrity constraint on the answer sets such that only solutions to the learning task are found.

$$examples \leftarrow e_1, \ldots, e_n, not\ e_{n+1}, \ldots, not\ e_m.$$
$$\bot \leftarrow not\ examples.$$

This constraint prevents any partial hypotheses from being learnt. However, simply removing it would not be beneficial as it would allow all hypotheses in the search space to be learnt. In order to find a suitable hypothesis for refinement we need to be able to compare each hypothesis in the search space with one another. To achieve this, we introduce a scoring scheme the partial hypotheses for directing the search towards the optimal partial hypothesis.

Definition 4.2 (Scoring partial hypotheses)**.** *Let $\langle E, B, M\rangle$ be an ILP task and let H be a partial hypothesis in the learning task's hypothesis space $\mathcal{P}(R_M)$. The score of H is the tuple $score(H) = \langle cover_{e^+}(H), cover_{e^-}(H), length(H)\rangle$, where $cover_{e^+}(H)$ is the number of positive examples covered by H, $cover_{e^-}(H)$ is the number of negative examples covered by H, and $length(H)$ is the total number of literals in H.*

Definition 4.3 (Comparing partial hypotheses)**.** *Let $\langle E, B, M\rangle$ be an ILP task and let H and H' be two (partial) hypotheses in the hypothesis space $\mathcal{P}(R_M)$. H is* better *than H', denoted $score(H) > score(H')$ if and only if one of the following cases applies:*

- $cover_{e^+}(H) > cover_{e^+}(H')$,
- $cover_{e^+}(H) = cover_{e^+}(H') \wedge cover_{e^-}(H) < cover_{e^-}(H')$,
- $cover_{e^+}(H) = cover_{e^+}(H') \wedge cover_{e^-}(H) = cover_{e^-}(H') \wedge length(H) < length(H')$

The scoring scheme is used to direct the search towards the optimal partial hypothesis.

Definition 4.4 (Optimal hypothesis)**.** *Given a hypothesis space $\mathcal{P}(R_M)$, the hypothesis H_{opt} is the optimal hypothesis of $\mathcal{P}(R_M)$ if and only if for all other hypothesis H in $\mathcal{P}(R_M)$: $score(H_{opt}) \geq score(H)$.*

The above scoring scheme differs from the more conventional scoring mechanisms used in ILP. Our choice in scoring scheme is the result of an in-depth comparative analysis of what scoring function would perform better. Specifically, we have considered two other scoring functions: $sc_1 : \langle cover_{e^+}(H) - cover_{e^-}(H), length(H)\rangle$ and $sc_2 : cover_{e^+}(H) - cover_{e^-}(H) - length(H)$.

The scoring scheme sc_1 emphasises the coverage of examples over the hypothesis size. It has an advantage of having smaller range of score values, from $-|not\ e \in E|$ to $|e \in E|$ as opposed

to 0 to $|e \in E| \times |not\ e \in E|$ in our scoring scheme, which could help reduce the search for the optimal hypothesis. However, it is too general and would result in learned revisions that are indistinguishable from each other. For instance, suppose we have two partial hypotheses of equal size s, one covers none of the positive nor negative examples, and the other covers one positive and one negative example. Using the scoring scheme sc_1 both hypotheses would have the same score of $\langle 0, s\rangle$; however with our scoring scheme, the former would have the score $\langle 0, 0, s\rangle$, while the latter would have the score $\langle 1, 1, s\rangle$. The difference between the two scores makes our scoring scheme able to differentiate a wider range of hypotheses.

For the scoring scheme sc_2, we found it to be unsuitable as it would give too much emphasis to the hypothesis size, which is not the most relevant for discriminating among potential revisions that are required to be solutions to the learning task. Consider the following example.

Example 17. *Given the following noiseless background knowledge and examples:*

$$B = \left\{ \begin{array}{ll} even(0). & num(s(s(s(0)))). \\ num(0). & num(s(s(s(s(0))))). \\ num(s(0)). & num(s(s(s(s(0)))))). \\ num(s(s(0))). & succ(X, s(X)) \leftarrow num(X), num(s(X)). \end{array} \right\}$$

$$E = \left\{ \begin{array}{lll} even(s(s(s(s(0))))), & odd(s(s(s(s(0)))))), & not\ odd(0), \\ not\ even(s(0)), & not\ odd(s(s(s(s(0))))), & not\ even(s(s(s(0))))). \end{array} \right\}$$

Consider the following partial hypotheses:

- $H_1 = \emptyset$
- $H_2 = \left\{ \begin{array}{l} even(X). \\ odd(X). \end{array} \right\}$
- $H_3 = \left\{ \begin{array}{l} even(X) \leftarrow succ(Y, X). \\ odd(X) \leftarrow succ(Y, X), not\ odd(Y). \end{array} \right\}$

Partial Hypothesis	RASPAL scoring scheme	sc_1	sc_2
H_1	$\langle 0, 0, 0 \rangle$	$\langle 0, 0 \rangle$	0
H_2	$\langle 2, 4, 2 \rangle$	$\langle -2, 2 \rangle$	-4
H_3	$\langle 2, 2, 5 \rangle$	$\langle 0, 5 \rangle$	-5
H_4	$\langle 2, 0, 6 \rangle$	$\langle 2, 6 \rangle$	-4

Table 4.1: Score of partial hypotheses in Example 17 using RASPAL's scoring scheme $\langle$ $cover_{e^+}(H)$, $cover_{e^-}(H)$, $length(H)$ $\rangle$, sc_1 : $\langle$ $cover_{e^+}(H) - cover_{e^-}(H)$, $length(H)$ $\rangle$ and $sc2$: $cover_{e^+}(H) - cover_{e^-}(H) - length(H)$

- $H_4 = \left\{ \begin{array}{l} even(X) \leftarrow succ(Y, X), not\ even(Y). \\ odd(X) \leftarrow succ(Y, X), even(Y). \end{array} \right\}$

Table 4.1 shows that our scoring scheme would be able to assign an unique score to all partial hypotheses, giving a clear preference of H_4 over H_3, H_3 over H_2, and H_2 over H_1. For sc_1, which would maximise the first element of the tuple and minimise the second element, its highest preference is still given to H_4, but the empty hypothesis H_1 is preferred over H_2 and H_3. As we want to find a partial hypothesis for further refinements, we would want H_2 and H_3 to be preferred over H_1. Lastly, for sc_2, which aims at maximising its score, it is clear that the $length(H)$ would need to be scaled down to lessen its impact on the scoring scheme. However, the exact weight assigned to it will be dependent on the learning task, making it not as general as RASPAL's scoring scheme.

This initial phase of RASPAL can be summarised as a learning task.

Definition 4.5 (Learning Initial Partial Hypothesis)**.** *Let $\langle E, B, M \rangle$ be an inductive task and i a limit on rule length. A set of rules H is an optimal partial hypothesis to the learning task if and only if (i) H is compatible with M; (ii) for all rules r in H: $length(r) \leq i$; (iii) for all other partial hypothesis H' with rule length not exceeding i: $score(H) \geq score(H')$*

4.2 Refining a partial hypothesis

At each iteration RASPAL tries to improve the partial hypothesis by learning a *change transaction*, a set of refinement operations that can be applied on the current partial hypothesis. This is done through the revision task $\langle B \cup H, IC \rangle$ where B is the background knowledge of the original learning task, H is a partial hypothesis and IC is the goal $\{examples \leftarrow \bigwedge e \in E.; \bot \leftarrow not\ examples.\}$ constructed from the examples of the original learning task. However, unlike the typical revision task, RASPAL does not require the revised hypothesis to be a solution, but only to be the best *improvement* of the previous partial hypothesis. RASPAL's revision task uses its scoring mechanism and a weaker form of constraint in order to select the refinement that would produce a revised hypothesis with the highest score. A length limit is placed on the learned partial hypotheses and refinements, and the solution to the learning task is learnt by either finding a solution within the length limit or by applying one or more refinements to a partial hypothesis.

Definition 4.6 (RASPAL refinement task)**.** *Given an inductive learning task $\langle E, B, M \rangle$ and a partial hypothesis H, an improved partial hypothesis H' can be found through a hypothesis refinement task $\langle B \cup H, IC \rangle$ and a given rule length limit i. The constraint IC for the task is that the score of the revised hypothesis H' must be better than the score for the current partial hypothesis $\{\bot \leftarrow score(H') \leq score(H).\}$ and that the learnt refinement cannot add more than i body literals to a rule.*

Definition 4.7 (RASPAL refinement task)**.** *RASPAL refinement task $\langle B \cup H, IC \rangle$ for the inductive task $\langle E, B, M \rangle$ can be solved through another ILP task $\langle \emptyset, B \cup r(H) \cup IC, M \cup M_\Delta \rangle$ where $r(H)$ is the revisable form of H and M_Δ is the set of mode declarations for revision operations applicable to $r(H)$. The solution to the inductive task is a change transaction C such that (i) C is compatible with $M \cup M_\Delta$; (ii) $score(H') > score(H)$ where $H' = C \otimes r(H)$; (iii) there does not exist another change transaction C' compatible with $M \cup M_\Delta$ such that $score(C' \otimes H) > score(H')$.*

This revision task is applied iteratively to a partial hypothesis until it becomes a solution of the original inductive task. In the cases where a partial hypothesis cannot be improved, as the

algorithm is operating under the assumption that the given task does not have noise, RASPAL is recalled with i incremented by 1. Note that as ASPAL is complete, i cannot be incremented indefinitely as RASPAL's initial iteration will eventually be the same as using ASPAL to solve the initially given learning task.

Each refinement task of RASPAL uses the scoring scheme as selection criteria for the learnt change transaction. The scoring scheme is dependent on the size of the refined hypothesis which differs from the conventional selection criteria of revision tasks where preference is given to the change transaction with the least number of revision operations. To reflect this, the weight given to each revision operation in RASPAL's top theory is the difference in the size of the original partial hypothesis and the revised partial hypothesis as a result of applying the revision operation. Each $extension/2$ clause in the top theory is given a weight equal to the number of body literals it contains, and each $delete/2$ fact is given the weight -1.

The selection criteria based on the size of the revised hypothesis makes ASPAL's revisable theory in Definition 3.18 inappropriate for RASPAL's refinement tasks. ASPAL's revisable theory allows $delete/2$ facts to be learnt for any revisable rule even if it is deleted from the revised theory by the learnt change transaction. By giving each $delete/2$ fact a negative score and trying to minimise the overall sum of these weight, the $delete/2$ facts in the top theory become overvalued compared to other learnable revision operations. The revisable theory in Definition 3.18 is modified as follows so that it can be used for RASPAL.

Definition 4.8 (RASPAL Revisable theory)**.** *For each rule r given by $h_i \leftarrow b_{i,1}, \ldots, b_{i,n}$ in a given theory T, let $vars(r)$ be the list of all variables in r and $vars(b_{i,j})$ be the list of all variables in a body literal $b_{i,j}$. The following clauses are added to the revisable form $r(T)$:*

- $h_i \leftarrow try(i, 1, vars(b_{i,1})), \ldots, try(i, n, vars(b_{i,n})), extension(i, vars(r_i))$
- $try(i, j, vars(b_{i,j})) \leftarrow b_{i,j}, not\ delete(i, j)$, for each $try(i, j, vars(b_{i,j}))$
- $try(i, j, vars(b_{i,j})) \leftarrow delete(i, j)$, for each $try(i, j, vars(b_{i,j}))$
- $\bot \leftarrow delete(i, j), not\ extension(i, vars(r_i))$, for each $delete(i, j)$

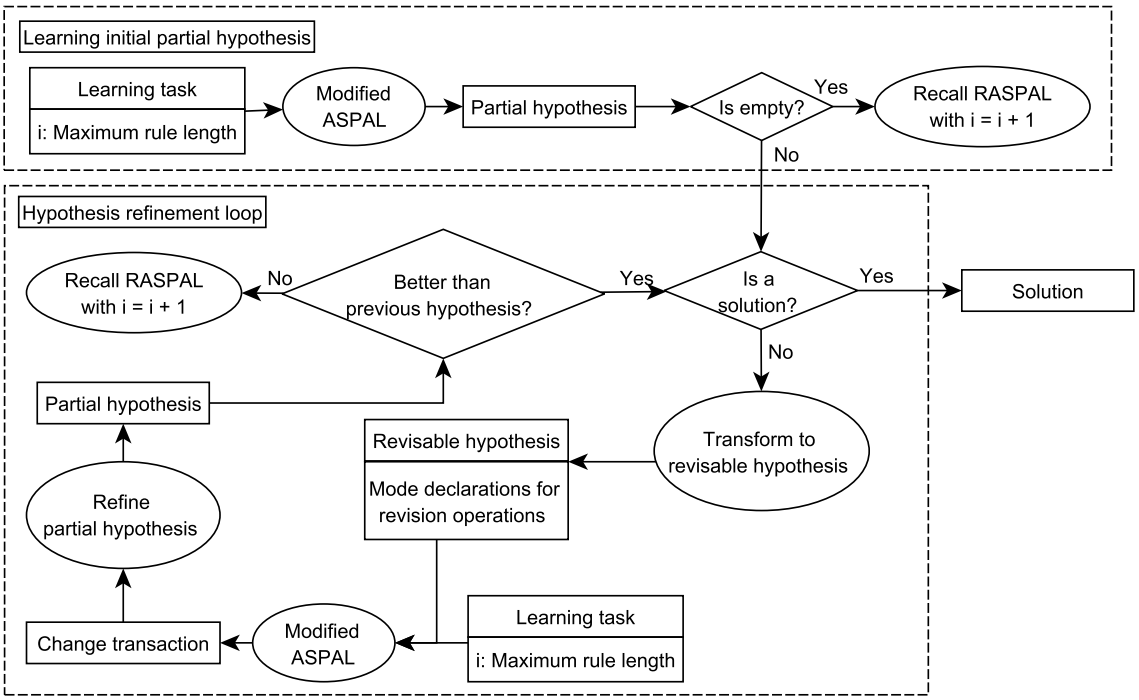

Figure 4.1: RASPAL learning process

The above definition of revisable theory impose a constraint on each $delete(i, j)$, such that it can only be learnt if the corresponding $extension(i, vars(r_i))$ is also learnt. This means that delete facts are only learnt for rules that are retained in the revised theory.

Figure 4.1 shows how the two phases are used together in RASPAL. Both phases use a modified version of ASPAL to learn the optimal partial hypothesis, for finding the initial partial hypothesis in the first phase and for finding the change transaction that would result in the highest scoring refined partial hypothesis in the second phase. The algorithm assumes that there exists a non-empty solution to the given learning task. Thus in both phases RASPAL can call itself with rule length limit i increased by 1. In the first phase i will be incremented if the optimal partial hypothesis is empty as the second phase would lack a partial hypothesis to use for refinement. In the second phase RASPAL can recall itself should the refined hypothesis not have a higher score than the previous one. Again, as the algorithm assumes that there exists a non-empty solution, the lack of improvement from the refinement suggests that the current

value of i is too restrictive to find the solution.

4.2.1 Algorithm

Algorithm 1 RASPAL(P, i)
Require: $P = \langle E, B, M \rangle$
Output: $\langle Hypothesis, Score \rangle$, a solution to P and its score

```
1:  let ⟨Hypothesis, Score⟩ =FindOptimalHypothesis(P, i)
2:  if Hypothesis == ∅ then                          ▷ Empty hypothesis found
3:      return RASPAL(P, i + 1)
4:  loop
5:      if Score ≥ ⟨|{e|e ∈ E}|, 0, +∞⟩ then          ▷ Solution found
6:          return ⟨Hypothesis, Score⟩
7:      ⟨Hypothesis, Score_new⟩ =RefineHypothesis(Hypothesis, P, i + 1)
8:      if Score_new ≤ Score then                    ▷ Score does not improve
9:          return RASPAL(P, i + 1)
10:     Score = Score_new
```

The RASPAL main learning algorithm is shown in Algorithm 1. It takes as input an inductive learning task P and a limit i on the length of the rules that could be learnt at each iteration. The function FindOptimalHypothesis is the modified ASPAL algorithm for finding the optimal partial hypothesis with rule length limited to i. Note that while for this work we have used ASPAL, the framework of RASPAL could be used with other learning algorithms. Lines 1-3 is the first phase of RASPAL where the inital partial hypothesis is found and checked to ensure that it is not empty. If an empty hypothesis is found RASPAL recalls itself with i incremented by 1.

The loop in lines 4-10 is the second phase of RASPAL where the algorithm checks if the partial hypothesis is a solution to the learning task in which case it is returned to the user. Otherwise the partial hypothesis is refined using the RefineHypothesis function. The new hypothesis and its score is checked against the current hypothesis, and if the new score does not improve RASPAL is recalled with i incremented by 1. Otherwise the score is updated and the algorithm returns to the beginning of the loop where the new partial hypothesis is checked to see if it is a solution to the learning task.

Algorithm 2 RefineHypothesis($Hypothesis, P, i$)
Require: $P = \langle E, B, M \rangle$
Output: $\langle Hypothesis_{new}, Score_{new} \rangle$, a refinement of $Hypothesis$ of and its score

1: $P' = \langle \emptyset, B \cup r(Hypothesis), M \cup M_\Delta \rangle$ = MakeRevisionTask($P, Hypothesis$)
2: $\langle Changes, Score_{new} \rangle$ = FindOptimalHypothesis($P', i + 1$)
3: $Hypothesis_{new}$ = ApplyRefinement($Hypothesis, Changes$)
4: **return** $\langle Hypothesis_{new}, Score_{new} \rangle$

Algorithm 2 shows the function RefineHypothesis. Given a partial $Hypothesis$, the ILP task P, and the clause length limit i, the algorithm returns a refined $Hypothesis$ and its score. This is done by creating a theory revision task P' using MakeRevisionTask function which transforms P and $Hypothesis$ into another inductive learning task for hypothesis refinement P' as described in Definition 4.6. Specifically, the conversion is done by generating the revisable hypothesis $r(Hypothesis)$ from the given partial hypothesis. The set of mode declaration is extended to include the revision operators M_Δ. The function FindOptimalHypothesis is then used to learn the change transaction $Changes$, with associated score $Score_{new}$, as solution to the revision task. The change transaction is then applied to the current hypothesis using the ApplyRefinement function. This results in the revised hypothesis $Hypothesis_{new}$ which is returned together with its score. Note that, at line 2 in Algorithm 2, the function FindOptimalHypothesis is called with $i + 1$. The increase in i is because for RefineHypothesis to extend a clause in the current hypothesis up to i literals, it must be able to learn an extension clause with i body literals, thus having length $i + 1$. Furthermore, the integrity constraint IC, that the refined hypothesis is an improvement of the previous one, for the hypothesis refinement task is not imposed in the learning program, but by the check in line 8 of Algorithm 1 that the refined hypothesis does have better score than the previous partial hypothesis.

Example 18. *Consider the following task of learning the concept of even and odd numbers:*

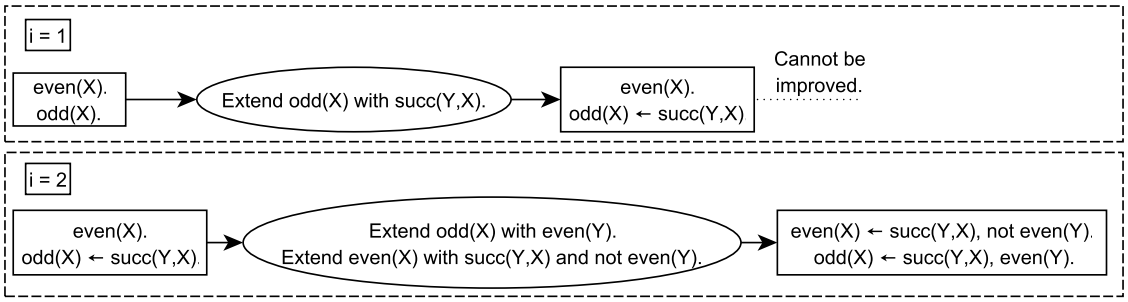

Figure 4.2: Using RASPAL to learn even and odd with 1 as initial value of i.

$$B = \left\{ \begin{array}{l} even(0). \\ num(0). \\ num(s(0)). \\ num(s(s(0))). \\ num(s(s(s(0)))). \\ num(s(s(s(s(0))))). \\ num(s(s(s(s(s(0)))))). \\ succ(X, s(X)) \leftarrow num(X), num(s(X)). \end{array} \right.$$

$$M = \left\{ \begin{array}{l} modeh(odd(+num)). \\ modeh(even(+num)). \\ modeb(not\ even(+num)). \\ modeb(even(+num)). \\ modeb(succ(-num, +num)). \end{array} \right\}$$

$$E = \left\{ \begin{array}{ll} even(s(s(s(s(0))))), & not\ even(s(0)), \\ odd(s(s(s(s(s(0)))))), & not\ odd(s(s(s(s(0))))), \\ not\ odd(0), & not\ even(s(s(s(0)))) \end{array} \right\}$$

Figure 4.2 shows the partial hypotheses found by RASPAL when solving the learning task. RASPAL is first called with $i = 1$. It manages to find the initial partial hypothesis $\{even(X).; odd(X).\}$ and refines it to produce $\{even(X).; odd(X) \leftarrow succ(Y, X).\}$, removing one negative example coverage. However, with i at 1 it cannot find further improvements for the partial hypothesis. RASPAL is recalled with i increased to 2. The last hypothesis for the previous i value is relearned, but this time it can be further revised, into $\{even(X) \leftarrow succ(Y, X), not\ even(Y).; odd(X) \leftarrow succ(Y, X), even(Y).\}$, which is a solution to the learning task.

4.2.2 Property

The following theorem shows the completeness of our RASPAL approach. The proof builds upon the completeness of ASPAL's rule encoding, which is used in FindOptimalHypothesis and RefineHypothesis.

Theorem 4.1 (Completeness of RASPAL)**.** *Let $P = \langle E, B, M \rangle$ be an inductive task and let $\mathcal{H}$ be a set of solutions of P such that $\mathcal{H}$ is a non-empty set containing no empty solution of P. Then* RASPAL(P, i_{input}) *returns a tuple $\langle H, S \rangle$ for a lower bound on $i_{input} > 0$, where H is a solution $H \in \mathcal{H}$, and S is the score of H.*

Proof. As $\mathcal{H}$ is not empty and contains no empty solution, let I_{max} be the maximum length of all rules in all solutions in $\mathcal{H}$, and let L be the maximum solution size in $\mathcal{H}$. Using well founded induction on $I_{max} - i_0$ we show that for all $i_0 > 0$, RASPAL(P, i_0) will return a tuple $\langle H, S \rangle$ where $H \in \mathcal{H}$ and S is the score of H. Let i_0 be the input of RASPAL's initial call RASPAL(P, i_0).

Inductive hypothesis: For all j such that $I_{max} - j < I_{max} - i_0$, RASPAL$(P, j)$ will return a tuple a tuple $\langle H, S \rangle$ for some $H \in \mathcal{H}$ and S is the score of H.

Case 1: $(I_{max} - i_0 \leq 0)$. Since a solution with maximum rule length equal to I_{max} exists and FindOptimalHypothesis is assumed to be complete, then at line 1 of Algorithm 1 the function FindOptimalHypothesis will find a non-empty solution, making the condition in line 2 fail. As a non-empty solution has been found, it will be returned by line 6 of Algorithm 1.

Case 2: $(I_{max} - i_0 > 0)$. RASPAL could find a solution through 3 different paths.

1. A non-empty solution with maximum rule length less than I_{max} is found at line 1 of Algorithm 1. The same as for Case 1, this solution will be return by line 6 of Algorithm 1.

2. In line 1 of Algorithm 1 a non-empty partial hypothesis is found and either line 3 or line 9 of Algorithm 1 is executed, resulting in RASPAL$(P, i_0 + 1)$ being called. In this case the

inductive hypothesis can be applied as $I_{max}-(i_0+1) < I_{max}-i_0$, thus RASPAL$(P, i_0+1)$ will return a solution and its score.

3. In line 1 of Algorithm 1 a non-empty partial hypothesis is found and both line 3 and line 9 of Algorithm 1 are not executed. This means that in the loop in lines 4-10 the refinements for the partial hypothesis which improve its score are found. There can only be a finite number of refinements applied to the partial hypothesis, after which a solution will be found (and returned by line 6) thus the score can only improve a finite number of times. For RefineHypothesis to be repeatedly called, each time it is called the score of the revised hypothesis has to be better than the previous hypothesis. The score can improve by increasing the the number of positive example covered, decreasing the number of negative examples covered, or decreasing the size of the hypothesis without worsening the examples coverage. This can continue until a solution is eventually found with score $\langle|\{e|e \in E\}|, 0, K\rangle$, where $K \geq L$, at which point the loop in lines 4-10 will terminate and a solution and its score will be returned.

□

As the completeness of RASPAL is dependent on the completeness of the internal learning algorithm it uses, at its worst (i equal to the maximum number of body literals in a solution) it would be as complete as the internal algorithm.

4.3 Learning with noise

Algorithm 1 assumes that a solution to the given learning task exists. However, noise is a natural part of real world problems. Noise can be introduced into the learning tasks in many ways, such as errors introduced when collecting or recording information, or outliers in the data collected. For this work we are concerned with noise in the examples of the learning task. In the presence of noise we take an approach similar to Aleph [Sri07] that the learner should aim to find a hypothesis with example coverage within a certain noise threshold.

Definition 4.9 (Noise Threshold)**.** *A noise threshold over a set of examples E is a pair of positive integers $\langle e^+_{min}, e^-_{max}\rangle$, where $e^+_{min} \leq |\{e|e \in E\}|$ and $e^-_{max} \leq |\{e|not\ e \in E\}|$.*

A hypothesis will cover the examples within the noise threshold if it covers at least e^+_{min} number of positive examples and at most e^-_{max} number of negative examples. A threshold is used, as opposed to trying to maximise or minimise respectively the number of positive and negative examples, to avoid over-fitting the hypothesis to the noisy examples. The threshold can be used to define a learning task with noise.

Definition 4.10. *[Brave Inductive Task with Noise] A brave inductive learning task with noise data is a tuple $\langle E, B, M, N\rangle$ where E is set of grounded examples, B is a background theory expressed as a normal logic program, M is a set of mode declarations, and $N = \langle e^+_{min}, e^-_{max}\rangle$ is a noise threshold, represented as a pair of integers. A solution to the learning task is a hypothesis H, a set of normal clauses, such that (i) H is compatible with M; and (ii) for at least one model of $B \cup H$ the following conditions hold:*

1. $|\{e \mid B \cup H \models_{\mathcal{B}} e,\ e \in E\}| \geq e^+_{min}$
2. $|\{e \mid B \cup H \models_{\mathcal{B}} e,\ not\ e \in E\}| \leq e^-_{max}$

Using this definition of a learning task, tasks without noisy examples can be learnt by setting N equal to $\langle|\{e|e \in E\}|, 0\rangle$, capturing the case where the hypothesis must covers all positive examples and none of the negative examples.

Algorithm 3 RASPAL$_N$ is an adaptation of Algorithm 1 that takes as its input a learning task with noise. The terminate condition in line 5 is changed so that any partial hypothesis with a score better or equal to $\langle e^+_{min}, e^-_{max}, +\infty\rangle$ is considered solution to the learning task. Consider Example 18, the learner could solve the task with $i = 1$ and accept $\{even(X).; odd(X) \leftarrow succ(Y, X)\}$ as a solution to the task should the noise threshold be set to $\langle 2, 3\rangle$ as it cover the two positive examples and three negative examples. Note that since RefineHypothesis uses FindOptimalHypothesis to find the best refinement for the given partial hypothesis, any refined hypothesis used in the test in line 5 will be the best scoring hypothesis for the previous

Algorithm 3 RASPAL$_N(P, i)$
Require: $P = \langle E, B, M, \langle e^+_{min}, e^-_{max} \rangle \rangle$
Output: $\langle Hypothesis, Score \rangle$, a solution to P and its score

```
1: let ⟨Hypothesis, Score⟩ =FindOptimalHypothesis(P, i)
2: if Hypothesis == ∅ then                          ▷ Empty hypothesis found
3:     return RASPAL_N(P, i + 1)
4: loop
5:     if Score ≥ ⟨e⁺_min, e⁻_max, +∞⟩ then          ▷ Solution found
6:         return ⟨Hypothesis, Score⟩
7:     ⟨Hypothesis, Score_new⟩ =RefineHypothesis(Hypothesis, P, i + 1)
8:     if Score_new ≤ Score then                    ▷ Score does not improve
9:         return RASPAL_N(P, i + 1)
10:    Score = Score_new
```

iteration. Therefore the returned solution and score are not necessary the first hypothesis found which satisfies the noise threshold, as better scoring hypothesis can be returned provided it can be found in the same iteration of the loop in RASPAL$_N$.

4.4 Note on RASPAL Implementation

RASPAL's hypothesis scoring scheme is implemented by removing the ASPAL's constraint on examples coverage and replacing it with the following Clingo's optimisation statements:

$$\#minimise[r_1 = weight(r_1), \ldots, r_k = weight(r_k)].$$
$$\#minimise[e_1, \ldots, e_n].$$
$$\#maximise[e_{n_1}, \ldots, e_m].$$

where $r_1, \ldots, e_k$ are all rules in set of rules R_M compatible with the learning task's mode declarations, $e, \ldots, e_n$ are positive examples, $e, \ldots, e_m$ are negative examples, and $weight(r)$ is $length(r)$ if the head of the rule is not a revision operation, $length(r) - 1$ if the rule is an extension operation, or -1 if the rule is a deletion operation.

Clingo will output a list of pairs of answer set and its score from the least to the most optimised, with one answer set per score (to find all answer sets for each score the option "`--opt-all`" must be used when calling Clingo). In order to find the highest scoring partial hypothesis or

change transaction, RASPAL finds the answer set with highest score and discard the other answer sets output by Clingo.

4.5 Related Work and Discussion

There are three other works that are most relevant to RASPAL. They are ASPAL [CRL11], HYPER [Bra99], and ILED [KAP14].

RASPAL's learning algorithm uses ASPAL as a black box for solving an ILP task. However, this could potentially be substituted by another ILP algorithm, as long as it is a non-monotonic system. Although ASPAL has been shown in [CRV$^+$11] to be able to solve revision tasks, theory revision itself is not an integral part of the learning algorithm.

Both HYPER and RASPAL are ILP systems that use theory refinement as part of their learning algorithm. However, unlike HYPER, RASPAL is a non-monotonic system and it uses different revision operations. While Hyper only allows for one body literal to be added when refining a hypothesis, RASPAL's limit is more flexible as this is dependent on the value of i given. In addition to this RASPAL allows for completely new rules to be added to the partial hypothesis, and for a rule to be revised in multiple ways by a change transaction. RASPAL does not explicitly have variables unification or refinement as part of its revision operations as mode declarations are general enough for specifying such changes if they are needed. For instance, suppose an integer X in a body literal $p(X)$ of a rule $q(X,Y) \leftarrow p(X)$ could be refined into by changing it into $s(Y)$, then this effect can be achieved by having mode declarations $modeb(p(+int))$ and $modeb(p(s(+int)))$. This would allow RASPAL to revise the rule by deleting $p(X)$ and extending it with $p(s(Y))$.

ILED is a recently proposed incremental learning system based on XHAIL [Ray09]. It has been designed to address the scalability problem of learning from continuously collected real life temporal data. Like our work it uses hypothesis refinement and abductive reasoning for learning, and is capable of learning nonmonotonic clauses. However, unlike RASPAL, which uses theory revision as its learning mechanism, ILED uses revision for processing new knowledge

and incorporating it into previous learnt concepts. Most of ILED's revision operations are the same as RASPAL'S apart from body literals deletion which ILED does not use. Arguably, the effect of deleting body literals can be achieved by learning a new rule to replace the old one. This approach is not as suitable for RASPAL which attempts to learn large hypotheses by limiting the maximum rule length, thus at each iteration it would not be possible to learn a rule with length exceeding the limit.

There have not been a lot of work on using ILP for learning tasks with noise, and those that exist are for monotonic ILP. We have briefly mentioned that Alpeh [Sri07] uses a noise threshold. Another way Aleph could tolerate noise is by setting an upper bound on the search time. Other examples include LIME [MS97] which used Bayesian heuristics for finding a hypothesis that is a most probable solution, and HYPER/N [OB10], an ILP system based on HYPER, where the user can specify an approximate number of examples that are noise thus changing the criteria for terminating the search, as well as set the maximum number of iterations at which point the search is forced to terminate and the current partial hypothesis returned to the user. All of the approaches mentioned require that some estimate of noise be given to the system prior to the search.

Another approach for handling noise in the examples by non-monotonic systems is by allowing it to learn exceptions, which will effectively let the learner find a complete and consistent solution by categorising which examples are exceptions to the solution. For instance, in Examples 18, the following mode declarations:

$$modeh(even\_exp(\#num)). \quad modeb(not\ even\_exp(+num)).$$
$$modeh(odd\_exp(\#num)). \quad mode_(not\ odd\_exp(+num)).$$

could be added to the learning task. These mode declarations would allow the learner to solve the learning task by defining $even/1$ as $even(X) \leftarrow not\ even\_exp(X)$, and $odd/1$ as $odd(X) \leftarrow not\ odd\_exp(X)$, as well as learning the exceptions to those rules as a collection of $even\_exp/1$ and $odd\_exp/1$ facts. Changes to the scoring scheme will need to be made in order to discourage the learner to output solutions with many $even\_exp/1$ and $odd\_exp/1$ facts. This could be achieved by giving higher preference to solutions with lowest number of head literals,

or minimal number of *even_exp*/1 and *odd_exp*/1 facts. However, this method may lead to the learner over-fitting the hypothesis to the examples.

In Chapter 7, we will evaluate the performance of RASPAL compared to ASPAL when used with a large learning task, as well as an example for using RASPAL for learning tasks with noise.

Chapter 5

Constraint-Driven Bias

In Chapter 3 we have described how different ILP systems can be grouped into top-down and bottom-up systems. Furthermore, within both of these categories there are meta-level systems that solve the inductive task using meta-level information. Like in [EFLP03] where meta-level information was used for reasoning about rules' structures, meta-level ILP systems could provide mechanisms for controlling the search over the hypothesis space. In this chapter we propose a concept of learning using constraint-driven bias, whereby the constraints over the meta-level representation of a hypothesis space could be specified, and used to direct the search towards a specific class of hypotheses that would otherwise not be computed using conventional ILP heuristics such as Occam's razor. Specifically, in this chapter we present a general language and primitive predicates that can be used to express constraints over language bias, and define the type of declarative heuristics that we can capture with this language. Later, in Chapter 6, we will demonstrate how this notion of constraint-driven bias can be integrated into both ASPAL and our RASPAL learning approaches, thus extending the capabilities of meta-level ILP systems with structured declarative bias.

Example 19. *Consider the following learning task:*

$$E = \left\{ \begin{array}{ll} even(s(s(0))). & not\ even(s(s(s(0)))). \\ odd(s(s(s(0)))). & not\ odd(0). \\ odd(s(s(s(s(s(0)))))). & not\ odd(s(s(s(s(0))))). \\ not\ even(s(0)). & \end{array} \right\}$$

$$B = \left\{ \begin{array}{ll} even(0). & num(s(s(s(s(0))))). \\ num(0). & num(s(s(s(s(s(s(0))))))). \\ num(s(0)). & succ(X, s(X)) \leftarrow \\ num(s(s(0))). & \quad num(X), num(s(X)). \\ num(s(s(s(0)))). & \end{array} \right\}$$

$$M = \left\{ \begin{array}{ll} modeh(even(+num)), & modeh(odd(+num)), \\ modeb(even(+num)), & modeb(not\ even(+num)), \\ modeb(not\ odd(+num)), & modeb(succ(-num, +num)) \end{array} \right\}$$

The most concise solution is the non-stratified (see Definition 7.1 for stratified programs) $H_1 = \{even(X) \leftarrow not\ odd(X).; odd(X) \leftarrow not\ even(X).\}$ since there exists a model which satisfies the examples. Hence, by Occam's Razor other solutions such as $H_2 = \{even(X) \leftarrow succ(Y, X), not\ even(Y).; odd(X) \leftarrow succ(Y, X), even(Y).\}$, which is more informative and satisfiable by all models, will be less preferred to H_1 as they are less compressed than H_1. We would like to be able to specify to the learner that certain hypotheses, for instance H_1, should not be considered as solutions to the given learning task despite being included in the hypothesis space, making the learner direct the search towards solutions that are more preferred in the given problem domain; hence outputting H_2 as the most concise solution of the task in the above example.

5.1 Constraint-Driven Learning

To characterise the class of (preferred) hypothesis described in Example 19, we use the notion of *acceptable hypothesis* adapted from acceptable revision in theory revision [GRR10]. Informally, an acceptable hypothesis is a hypothesis that is not only compatible with the given language

bias and covers the examples of a learning task, but it also satisfies domain-dependent integrity constraints over the language bias, given by the user as part of the learning task. The type of constraints which are of interest to us are syntactic constraints over the hypothesis space. To convey this syntactic information we introduce a set $\mathcal{L}_\mathcal{C}$ of eight primitives for expressing domain-dependent constraints. These primitives assume that there exists a labelling scheme $\lambda/1$ for the mode declarations such that for each given mode declaration m has a label l_m. Whichever labelling scheme the system uses should be made known to the user. For a rule $h \leftarrow b_1, \ldots, b_n$ in the hypothesis space we will use the notation $\lambda(h)$ as a shorthand for the label of head mode declaration that h is compatible with, and similarly $\lambda(b_i)$ as a shorthand for the label of body mode declaration that b_i is compatible with.

We assume in this chapter that the label for each mode declaration is the predicate name and the constant arguments that appear in the mode declaration itself. If the literal is negated then its label is prefixed with *not_*, so *not* $p(X)$ will have the label *not_p*. For example, the mode declaration $modeh(has\_cold(+person))$ does not include any constant argument and will have the label $has\_cold(\$e)$, where $\$e$ is a special constant to indicate the absence of place-holders for constants. Whereas $modeb(symptom(+person, \#condition))$ does have a constant and will have the label $symptom(X)$ where X is a place-holder variable for a constant of type *condition*.

5.1.1 Primitives of $\mathcal{L}_\mathcal{C}$

All primitives in $\mathcal{L}_\mathcal{C}$ can have either variables (denoted by L_h and L_b) or constants (denoted by l_h, l_b and m) as their arguments. A variable may be of type head label variable (L_h), and may be unified with a label of a head literal in a compatible rule, or of type body label variable (L_b) that unifies with a label of a body literal in a compatible rule. A constant is either a label l_h or l_b, referring to a specific head or body literal label respectively, or an integer m. For simplicity, we assume that constraints are pre-processed such that all variable labels are partially grounded with all possible labels such that the predicate names in the labels are grounded, but the constant arguments in the labels are left ungrounded, producing multiple versions of the constraint. The grounded primitive in $\mathcal{L}_\mathcal{C}$ and their meanings are as follows:

1. $in\_some\_rule(l_h, l_b)$ – the solution must include a rule with head literal labelled l_h and body including the literal labelled l_b

2. $in\_all\_rule(l_h, l_b)$ – all rules in the solution with head literal labelled l_h must have a body literal labelled l_b

3. $in\_same\_rule(l_h, l_{b_1}, l_{b_2})$ – if the solution contain a rule with head literal labelled l_h and body literal labelled l_{b_1} (or l_{b_2}), then it must also contain a body literal labelled l_{b_1} (or l_{b_2})

4. $in\_diff\_rule(l_{h_1}, l_{b_1}, l_{h_2}, l_{b_2})$ – in the solution, body literals labelled l_{b_1} and l_{b_2} must appear in two different rules, with heads labelled l_{h_1} and l_{h_2} respectively

5. $max\_body(l_h, m)$ – in the solution rules with head literal labelled l_h have at most x body literals

6. $min\_body(l_h, m)$ – in the solution rules with head literal labelled l_h have at least x body literals

7. $max\_head(l_h, m)$ – the solution must include at most x rules with head labelled l_h

8. $min\_head(l_h, m)$ – the solution must include at least x rules with head labelled l_h

Definition 5.1 (Domain-dependent Constraint)**.** *Let B be a background knowledge expressed in a language $\mathcal{L}_\mathcal{B}$ and M a set of mode declarations. A domain-dependent constraint in $\mathcal{L}_\mathcal{C}$ is $\bot \leftarrow C, \overline{C_B}$ where C is a primitive of $\mathcal{L}_\mathcal{C}$ or its negated form and $\overline{C_B}$ is a conjunction of literals from the language $\mathcal{L}_\mathcal{B}$.*

Example 20. Suppose we have a background knowledge of different animal species and properties of each one. To learn a rule for classifying amphibians, a constraint such as

$$\bot \leftarrow not\ in\_some\_rule(amphibian, not\_exception),\ animal(X),\ not\ amphibian(X)$$

could be used to force the learned rules to include a negated exception if there are species in the background that are not amphibians. On the other hand if all rules must include exceptions then the constraint

$$\bot \leftarrow not\ in\_all\_rule(L_h, exception)$$

can be used. This will be interpreted by grounding the constraint with all possible values for L_h, creating multiple versions of the constraint, such as $\bot \leftarrow not\ in\_all\_rule(amphibian, exception)$ and $\bot \leftarrow not\ in\_all\_rule(mammal, exception)$, for each possible head literal label.

We use the notation $[\mathcal{P}(R_M)]_{ic}$ to denote the set of hypotheses in the hypothesis space $\mathcal{P}(R_M)$ that satisfies an integrity constraint ic. The following definition states what it means for a hypothesis, compatible with a given mode declaration M, to satisfy a domain-dependent constraint.

Definition 5.2 (Satisfiability of Domain-dependent Constraint)**.** *Let B be a background knowledge, H a set of normal clauses, $\lambda/1$ be a labelling scheme, and $\bot \leftarrow C, \overline{C_B}$ a domain-dependent constraint ic. Then H* satisfies *ic, denoted $H \in [\mathcal{P}(R_M)]_{ic}$, if and only if either $B \cup H \not\models \overline{C_B}$ or one of the following cases holds:*

1. *$in\_some\_rule(l_h, l_b)$*
   - *$C = not\ in\_some\_rule(l_h, l_b)$ and there exists $r \in H$ such that $l_h = \lambda(head(r))$ and $b \in body(r)$ and $\lambda(b) = l_b$*
   - *$C = in\_some\_rule(l_h, l_b)$ and there does not exist $r \in H$ such that $l_h = \lambda(head(r))$ and $b \in body(r)$ and $\lambda(b) = l_b$*
2. *$in\_all\_rule(l_h, l_b)$*
   - *$C = not\ in\_all\_rule(l_h, l_b)$ and either there is no rule $r \in H$ such that $l_h = \lambda(head(r))$ or for all $r \in H$ such that $l_h = \lambda(head(r))$ it is the case that $b \in body(r)$ and $\lambda(b) = l_b$*
   - *$C = in\_all\_rule(l_h, l_b)$ and there exists $r \in H$ such that $l_h = head(r)$ and does not exist $b \in body(r)$ such that $\lambda(b) = l_b$*
3. *$in\_same\_rule(l_h, l_{b_1}, l_{b_2})$*
   - *$C = not\ in\_same\_rule(l_h, l_{b_1}, l_{b_2})$ and either there is no $r \in H$ such that $l_h = \lambda(head(r))$ and does not exist $b_1 \in body(r)$ and $l_{b_1} = \lambda(b_1)$ nor $b_2 \in body(r)$ and*

$l_{b_2} = \lambda(b_2)$, *or for all* $r \in H$ *such that* $l_h = \lambda(head(r))$ *if there exists* $b_1 \in body(r)$ *where* $l_{b_1} = \lambda(b_1)$ *then there exists* $b_2 \in body(r)$ *where* $l_{b_2} = \lambda(b_2)$ *(or if there exists* $b_2 \in body(r)$ *where* $l_{b_2} = \lambda(b_2)$ *then there exists* $b_1 \in body(r)$ *where* $l_{b_1} = \lambda(b_1)$*)*

- $C = in\_same\_rule(l_h, l_{b_1}, l_{b_2})$ *and there exists* $r \in H$ *such that* $l_h = head(r)$ *and does not exist* $b_1, b_2 \in body(r)$ *where* $l_{b_1} = \lambda(b_1)$ *and* $l_{b_2} = \lambda(b_2)$

4. $in\_diff\_rule(l_{h_1}, l_{b_1}, l_{h_2}, l_{b_2})$

- $C = not\ in\_diff\_rule(l_{h_1}, l_{b_1}, l_{h_2}, l_{b_2})$ *and there exists* $r_1 \in H$ *such that* $l_{h_1} = \lambda(head(r_1))$ *and* $b_1 \in body(r_1)$ *where* $\lambda(b_1) = l_{b_1}$*, and there must also exists* $r_2 \in H$ *such that* $r_1 \neq r_2$*,* $l_{h_2} = \lambda(head(r_2))$ *and* $b_2 \in body(r_2)$ *where* $\lambda(b_2) = l_{b_2}$*.*
- $C = in\_diff\_rule(l_{h_1}, l_{b_1}, l_{h_2}, l_{b_2})$ *and for all* $r_1 \in H$ *such that* $l_{h_1} = \lambda(head(r_1))$ *and* $b_1 \in body(r_1)$ *where* $l_{b_1} = \lambda(b_1)$*, does not exist* $r_2 \in H$ *such that* $r_1 \neq r_2$*,* $l_{h_2} = \lambda(head(r_2))$*, and* $b_2 \in body(r_2)$ *where* $l_{b_2} = \lambda(b_2)$*.*

5. $max\_body(l_h, m)$

- $C = not\ max\_body(l_h, m)$ *and for all* $r \in H$ *if* $l_h = \lambda(head(r))$ *then* $|body(r)| \leq m$
- $C = max\_body(l_h, m)$ *and there exists a rule* $r \in H$ *with* $l_h = \lambda(head(r))$ *such that* $|body(r)| > m$

6. $min\_body(l_h, m)$

- $C = not\ min\_body(l_h, m)$ *and for all* $r \in H$ *if* $l_h = \lambda(head(r))$ *then* $|body(r)| \geq m$
- $C = min\_body(l_h, m)$ *and there exists a rule* $r \in H$ *with* $l_h = \lambda(head(r))$ *such that* $|body(r)| < m$

7. $max\_head(l_h, m)$

- $C = not\ max\_head(l_h, m)$ *and for* $R = \{r | r \in H$ *and* $l_h = \lambda(head(r))\}$ *then* $|R| \leq m$
- $C = max\_head(l_h, m)$ *and for* $R = \{r | r \in H$ *and* $l_h = \lambda(head(r))\}$ *then* $|R| > m$

8. $min\_head(l_h, m)$

- $C = not\ min\_head(l_h, m)\ and\ for\ R = \{r | r \in H\ and\ l_h = \lambda(head(r))\}\ then\ |R| \geq m$
- $C = min\_head(l_h, m)\ and\ for\ R = \{r | r \in H\ and\ l_h = \lambda(head(r))\}\ then\ |R| < m$

In addition to what primitive is used in the constraint, the class of acceptable hypotheses is also dependent on whether the arguments in those primitives are constant or variable labels. The following are some examples of how having variable head and body labels effects the constants.

- $\bot \leftarrow not\ in\_some\_rule(l_h, L_b)$. The constraint is grounded, creating many versions of it with different groundings for L_b. The constraint ensures that rules with head literal labelled l_h have appeared with all body literals, possibly in multiple rules.

- $\bot \leftarrow in\_some\_rule(L_h, L_b)$. The grounding of this constraint creates an *in_some_rule*/2 constraint for each pair of head literal label and body literal label, resulting in no rules being able to contain any body literal. Acceptable hypotheses for the constraint can only contain facts.

- $\bot \leftarrow not\ in\_all\_rule(l_h, L_b)$. If a rule has the head literal labelled l_h then it contain all body literals, as L_b makes the constraint be grounded by all possible body literal label.

- $\bot \leftarrow not\ in\_all\_rule(L_h, L_b)$. Every rule in the hypothesis must contain all body literals as *in_all_rule*/2 is applied to all pairs of head literal label and body literal label, or there are no rules in the hypothesis.

- $\bot \leftarrow in\_all\_rule(l_h, L_b)$. For every possible body literal label there exists at least one rule with head literal labelled l_h that does not contain its corresponding body literal.

- $\bot \leftarrow not\ in\_same\_rule(l_h, L_{b_1}, L_{b_2})$. Rules in the hypothesis with head literal labelled l_h either does not contain any body literals, or it contains all body literals.

- $\bot \leftarrow not\ in\_diff\_rule(l_{h_1}, L_{b_1}, l_{h_2}, L_{b_2})$. Either rules with head literal labelled l_{h_1} and l_{h_2} do not have body literals, or their body literals are mutually exclusive to each other.

Equivalences

The primitives that we have introduced are not all orthogonal to each other. In this section we will show some equivalences that we have identified, as well as give counter examples for pairs that are false equivalences. Firstly we will consider equivalent pair of constraints.

1. Opposing cardinality constraints such as $\bot \leftarrow not\ max\_head(l_h, m)$ and $\bot \leftarrow min\_head(l_h, m+1)$.

   - From its definition, for a hypothesis H to satisfy the constraint $\bot \leftarrow not\ max\_head(l_h, m)$, it must be true that for $R = \{r | r \in H \text{ and } l_h = head(r)\}$ the $|R| \leq m$. In other words, there can be at most m rules in the hypothesis with head literal labelled l_h.
   - Similarly for a hypothesis H to satisfy the constraint $\bot \leftarrow min\_head(l_h, m+1)$, it must be true that for $R = \{r | r \in H \text{ and } l_h = head(r)\}$ the $|R| < m+1$. This is the same as quantifying that $|R| \leq m$. Therefore there can be at most m rules in the hypothesis with head literal labelled l_h.

     Conditions for hypothesis H to satisfy any one of the constraints are reducible to $R = \{r | r \in H \text{ and } l_h = head(r)\}$ and $|R| \leq m$, showing that they are equivalent to each other.

2. Equivalences between sets of constraints. For example $\{\bot \leftarrow not\ in\_some\_rule(l_h, l_b); \bot \leftarrow not\ in\_all\_rule(l_h, l_b)\}$ and $\{\bot \leftarrow not\ min\_head(l_h, 1); \bot \leftarrow not\ in\_all\_rule(l_h, l_b)\}$.

   Both of these constraints use $not\ in\_all\_rule(l_h, l_b)$ for one of their constraint, which will either produce solutions with no rules with head literal labelled l_h, or it could produce solutions where all rules with head literal labelled l_h will also contain a body literal labelled l_b. The constraint $\bot \leftarrow not\ in\_some\_rule(l_h, l_b)$ serves the same purpose as $\bot \leftarrow not\ min\_head(l_h, 1)$ as they force the solution to contain at least one rule with head literal labelled l_h. Combining either of these with $not\ in\_all\_rule(l_h, l_b)$, any solution

which does not contain any rule with head labelled l_h is no longer an acceptable solution. Therefore, the acceptable solutions for both sets of constraints are sets of rules which contain at least one rule with head literal labelled l_h and for all of these rule, they will have at least one body literal labelled l_b.

Lastly, we will give counter example for false equivalence of some pairs of primitives that might intuitively appear to be equivalent to one another:

1. Pairs of primitives that seem to be opposite to each other such as $\bot \leftarrow not\ in\_some\_rule(l_h, l_b)$ and $\bot \leftarrow in\_all\_rule(l_h, l_b)$.
   The constraint $\bot \leftarrow not\ in\_some\_rule(l_h, l_b)$ requires for the hypothesis to contain at least one rule with head literal matching the label l_h and for that rule to contain a body literal that matches the label l_b. The constraint $\bot \leftarrow in\_all\_rule(l_h, l_b)$ requires for the hypothesis to contain at least one rule with head literal matching the label l_h and for at least one of these rule to not contain any body literal that matches the label l_b. While there are some overlap between hypotheses for the two, the hypotheses such as $\{h \leftarrow b\}$ where $\lambda(h) = l_h$ and $\lambda(b) = l_b$ will satisfy $\bot \leftarrow not\ in\_some\_rule(l_h, l_b)$ but not $\bot \leftarrow in\_all\_rule(l_h, l_b)$.

2. The pair $\bot \leftarrow not\ in\_same\_rule(l_h, l_{b_1}, l_{b_2})$ and $\bot \leftarrow in\_diff\_rule(l_h, l_{b_1}, l_h, l_{b_2})$.
   The constraint $\bot \leftarrow not\ in\_same\_rule(l_h, l_{b_1}, l_{b_2})$ requires acceptable hypothesis to not only have rules with head labelled l_h that does not have body literals labelled both l_{b_1} or l_{b_2}. On the other hand $\bot \leftarrow in\_diff\_rule(l_h, l_{b_1}, l_h, l_{b_2})$ requires the existence of a rule with head labelled l_h containing either body labelled l_{b_1} (or l_{b_2}) but lacks another rule with the same head label and containing body labelled l_{b_2} (or l_{b_1}). Thus the empty hypothesis $\emptyset$ will satisfy the first constrain, but not the second one.

3. $\bot \leftarrow not\ in\_all\_rule(L_h, L_b)$ and $\bot \leftarrow not\ in\_same\_rule(L_h, L_{b_1}, L_{b_2})$

   - Consider $\bot \leftarrow not\ in\_all\_rule(L_h, L_b)$, the definition for $\bot \leftarrow not\ in\_all\_rule(l_h, l_b)$ is that either there is no rule $r \in H$ such that $l_h = \lambda(head(r))$ or for all $r \in H$ such that $l_h = \lambda(head(r))$ it is the case that $b \in body(r)$ and $\lambda(b) = l_b$. Replacing the

arguments by the variables L_h and L_b makes it so that the constraints is satisfiable by a hypothesis that:

- Contains no rules as L_h unifies with every possible head labels or;
- All rules in the hypothesis contain all possible body literals.

Thus either the hypothesis is completely empty, or every rules in the hypothesis contain all body literals.

- Now consider $\bot \leftarrow not\ in\_same\_rule(L_h, L_{b_1}, L_{b_2})$, the definition of $\bot \leftarrow not\ in\_same\_rule(l_h, l_{b_1}, l_{b_2})$ is that either there are no $r \in H$ with $l_h = \lambda(head(r))$ and only one of $b_1 \in body(r)$ or $l_{b_2} = \lambda(b_1)$ or $b_2 \in body(r) \wedge l_{b_2} = \lambda(b_2)$, or for all $r \in H$ such that $l_h = \lambda(head(r))$ if $b_1 \in body(r)$ and $l_{b_2} = \lambda(b_1)$, then $b_2 \in body(r)$ and $l_{b_2} = \lambda(b_2)$, and vice versa. Replacing the arguments by the variables L_h, L_{b_1} L_{b_2} makes it so that the constraints is satisfiable by a hypothesis that:
  - Contains no rules or;
  - For every rule r with head literal labelled l_h, r is either a fact or it contains all body literals.

This gives us the counter example hypothesis $\{h\}$ which would satisfy $\bot \leftarrow not\ in\_same\_rule(L_h, L_{b_1}, L_{b_2})$ but not $\bot \leftarrow not\ in\_all\_rule(L_h, L_b)$.

5.1.2 Constraint-Driven Learning Task

Having defined the condition for a hypothesis to satisfy a domain-dependent constraint, we can now introduce the notion of hypothesis space that satisfies the constraints.

Definition 5.3 (Constraint Biased Search Space)**.** *The constraint biased search space for a given set IC of domain-dependent constraints, denoted* $[\mathcal{P}(R_M)]_{IC}$*, is the intersection of the constraint biased search spaces with respect to each domain-dependent constraint in IC:*

$$[\mathcal{P}(R_M)]_{IC} = \bigcap_{ic \in IC} [\mathcal{P}(R_M)]_{ic}$$

We can now define our notion of learning task with constraint-driven bias and acceptable solutions.

Definition 5.4 (Inductive Learning Task with Constraint-driven Bias)**.** *An inductive learning task with constraint-driven bias is a tuple* $\langle B, M, E, IC\rangle$ *where* B *is background knowledge,* M *is a set of mode declarations,* E *is a set of examples and* IC *is a set of domain-dependent constraints.* H *is an* acceptable solution *if and only if (i)* $B \cup H \models_{\mathcal{B}} \bigwedge_{e\in E} e$*; and (ii)* $H \in [\mathcal{P}(R_M)]_{IC}$.

Consider the even and odd example at the beginning of this chapter. The primitive $in\_all\_rule/2$ can be used to specify the domain-dependent constraint $\bot \leftarrow not\ in\_all\_rule(L_h, succ(\$e))$, requiring that all rules in the hypothesis contains a body literal labelled $succ(\$e)$. The constraint will rule out the hypothesis $H_1 = \{even(X) \leftarrow not\ odd(X); odd(X) \leftarrow not\ even(X)\}$, making the hypothesis $H_2 = \{even(X) \leftarrow succ(Y, X),\ not\ even(Y).;\ odd(X) \leftarrow succ(Y, X),\ even(Y).\}$ become the more preferred solution in the absence of H_1.

5.2 Related Work

The notion of meta-level top theory for reasoning about hypotheses and for expressing constraints on answer sets is related to the approach in [EFLP03] where meta-level information is used to express constraints and preferences. Somewhat related to our approach is the work in [IDN13] where knowledge is represented as causal graphs and constraints are added by specifying impossible connections between nodes. Similarly to the work in [IDN13], Metagol [ML13] handles the problem of learning through a meta-interpreter top theories by lifting the entire learning task into a second-order representation and making use of a meta-interpreter – theory of allowable rule formats – for restricting the hypothesis space. Our work differs from these two approaches in that we generally use meta-level representation only to represent the hypothesis space and constraints, rather than the entire background knowledge and examples.

More closely related to our work is the notion of production fields for enforcing constraints on the hypothesis [Ino04]: a tuple $\langle L, C\rangle$ could be expressed where L is a literal and C is the

condition to be satisfied for L to be included in the hypothesis. As far as we are aware, in past works, the condition in a production field has mainly been used to specify conditions like length of the clause or depth of a term. Our approach is able to enforce similar constraints for body declarations using the primitives $max\_body/2$ and $min\_body/2$. Furthermore, as shown in our case studies in Section 7.2 more complex constraints across different clauses can also be imposed using our approach. Other more loosely related approaches that have made use of enhanced language bias to apply additional constraints to the hypothesis space include [BR98] and [BR14]. They both add a cardinality constraint to each mode declaration to specify the minimal or maximal number of times the mode declaration can be used in the hypothesis. Past work that use integrity constraints in ILP includes [JB96], which focuses on how to check for constraint satisfiability, and with respect to meta-level information only uses arguments' types. This differs from our work which directly reasons on the structure of the hypothesis, and allows for the constraints to be defined over relationships between different rules.

Chapter 6

Learning Systems with Constraint-Driven Bias

The previous chapter has defined a language for expressing domain-dependent constraints for an ILP task. In this chapter we present their implementation in ASPAL and RASPAL. There are three main factors to consider when implementing constraint-driven bias: i) how to represent meta-level information about rules; ii) how to generate the meta-level information when the rules are learnt; and iii) how to use this information to represent constraints defined using primitives from $\mathcal{L}_{\mathcal{C}}$.

6.1 Meta-level information

ASPAL's rule encoding already assigns a unique label for each mode declaration, and uses it to condense all information about a rule into a single atom. However, these labels are unsuitable for defining constraints as they are arbitrary constants internally supplied by the system to uniquely identify each mode declaration, of which the user has no knowledge of. For users to be able to use the constraints they must be able to refer to specific mode declarations in the constraints, which requires that the user know the specific mode declaration. This requires for the user to know the labelling scheme used by the system. Furthermore, this means scheme

should be meaningful to the user, for instance that it should take into account the constant arguments of the labelled literal.

In this chapter, we will still use L_h and L_b to denote variables representing the labels of head and body literals, but to represent our labelling scheme more accurately $l_h(\overline{C_h})$ with $\overline{v(C_h)}$ and $l_b(\overline{C_b})$ with $\overline{v(C_b)}$ will be used to represent specific head and body labels. $\overline{C_h}$ and $\overline{C_b}$ are vectors of argument variables in the literals, and $\overline{v(C)}$ represents the vector of the types of the constant arguments when there are place-holders for constant arguments in the label.

Example 21. *Consider the following mode declarations used in Example 15*

$$M = \left\{ \begin{array}{l} modeh(has\_cold(+person)). \\ modeb(symptom(+person, \#condition)). \end{array} \right\}$$

Suppose the following background is given:

$$B = \left\{ \begin{array}{l} person(alex). \\ person(beth). \\ condition(cough). \\ condition(fever). \\ condition(blister). \\ symptom(alex, blister). \\ symptom(alex, cough). \\ symptom(beth, fever). \\ symptom(beth, cough). \end{array} \right\}$$

The labels for this set of mode declarations are $has\_cold(\$e)$, and $symptom(X)$ with type $condition(X)$. By having the constant argument as part of the label, specific constraints such as $\bot \leftarrow not\ in\_all\_rule(has\_cold(\$e), symptom(fever))$ can be defined by the user. This constraint will only allow solutions where all rules with the head literal $has\_cold(P)$ will have the body literal $symptom(P, fever)$. Should the labels only use predicate names, it would make these constraints impossible to specify.

We propose a set of fresh predicates for representing the meta-level information of rules. These predicates are $is\_rule(Rid, L_h)$, and $in\_rule(Rid, L_h, L_b, I)$ where Rid is a unique identifier, L_h is the label of the rule's head literal, L_b is the label of a body of the rule, and I is the index of the body literal's position in the rule. The unique identifier Rid of each rule is the predicate $rid(k, \overline{C})$ where k is an automatically generated number to uniquely representing each rule template in ASPAL's top theory, and $\overline{C}$ is the tuple of constant arguments in the rule.

The predicates $is\_rule/2$ is used to indicate that a rule with head literal labelled L_h is in the solution, and $in\_rule/4$ indicates that a rule with head literal labelled L_h, body literal labelled L_b at position I is in the solution. Hence, given a rule in the hypothesis $r = h \leftarrow b_1, \ldots, b_n$ that contains $\overline{c}$ constants, and has the unique number k, the associated meta-level information $\mu(r)$ is the set of ground instances $\{in\_rule(rid(k, \overline{c}), l_h(\overline{c_h}), l_{b_1}(\overline{c_{b_1}}), 1), \ldots, in\_rule(rid(k, \overline{c}), l_h(\overline{c_h}), l_{b_n}(\overline{c_{b_n}}), n), is\_rule(rid(k, \overline{c}), l_h(\overline{c_h}))\}$.

Definition 6.1. *Given a hypothesis H, its meta-level information, denoted by $meta(H)$, is the union of the meta-level information of its rules $meta(H) = \bigcup_{r \in H} \mu(r)$.*

6.2 Extending Learning Systems with Meta-information

ASPAL

As shown in [Cor11] and captured by Algorithm 4, at the highest level ASPAL's algorithm consists of three steps: i) constructing the top-theory from the given mode declarations; ii) abducing the rule encodings of the hypothesis; and iii) post-processing the encodings to turn them into rules. To extend ASPAL's top theory with meta-level information, we add an additional two steps in Algorithm 4 before line 3. These create an extended theory $\top_M$ to deduce the meta-level information of hypotheses as they are learnt, and translate the domain dependent constraints into ASP integrity constraints later discussed in Section 6.3. This modification is shown in Algorithm 5, which uses Algorithm 6 for generating an extended top theory $\top_M$.

Algorithm 4 ASPAL(E, B, M)
Require: E examples; B background theory; M mode declarations
Output: H hypothesis

1: $\langle \top, A^{\top} \rangle$ =TOP-THEORY(B, M)
2: ϵ =EXAMPLE-PRE-PROCESS(E)
3: Δ = ABDUCE$(B \cup \top \cup \epsilon, A, \emptyset)$
4: H =POST-PROCESS(Δ, M)

Algorithm 5 ASPAL$_C(E, B, M, IC)$
Require: E examples; B background theory; M mode declarations; IC domain dependent integrity constraints
Output: H hypothesis

1: $\langle \top, A^{\top} \rangle$ =TOP-THEORY(B, M)
2: ϵ =EXAMPLE-PRE-PROCESS(E)
3: $\top_M$ =META-PRE-PROCESS$(\top, M)$
4: $t(IC)$ =TRANSLATE(IC)
5: Δ = ABDUCE$(B \cup \top_M \cup \epsilon, A^{\top}, t(IC))$
6: H =POST-PROCESS(Δ, M)

Algorithm 6 META-PRE-PROCESS$(\top, M)$
Require: $\top$ ASPAL top theory; M mode declarations
Output: $\top_M$ extended top theory

1: $\mathcal{M} = \emptyset$
2: **for each** $h \leftarrow b_1, \ldots, b_n, \mathit{rule}(\mathit{id}(h \leftarrow b_1, \ldots, b_n), \bar{C}) \in \top$ with identifier k **do**
3: $\quad \mathcal{M} = \mathcal{M} \cup \{n+1\ \{\mathit{is\_rule}(\mathit{rid}(k, \bar{C}), l_h(\bar{C_h})), \mathit{in\_rule}(\mathit{rid}(k, \bar{C}), l_h(\bar{C_h}), l_{b_1}(\bar{C_{b_1}}), 1), \ldots,$
4: $\quad\quad \mathit{in\_rule}(\mathit{rid}(k, \bar{C}), l_h(\bar{C_h}), l_{b_n}(\bar{C_{b_n}}), n)\}\ n+1 \leftarrow \mathit{rule}(\mathit{id}(h \leftarrow b_1, \ldots, b_n), \bar{C}).\}$
5: **return** $\top \cup \mathcal{M}$

META-PRE-PROCESS derives $\top_M$ from $\top \cup \mathcal{M}$ where $\top$ is the ASPAL top-theory and the theory $\mathcal{M}$ includes for each rule $h \leftarrow b_1, \ldots, b_n, rule(id(h \leftarrow b_1, \ldots, b_n), \bar{C})$ in $\top$, the following rule, which relates each rule compatible with the learning task's mode declaration with its meta-level information.

$$n+1\ \{is\_rule(rid(k,\bar{C}), l_h(\bar{C}_h)), in\_rule(rid(k,\bar{C}), l_h(\bar{C}_h), l_{b_1}(\bar{C}_{b_1}), 1),$$
$$\ldots, in\_rule(rid(k,\bar{C}), l_h(\bar{C}_h), l_{b_n}(\bar{C}_{b_n}), n)\}\ n+1 \leftarrow rule(id(h \leftarrow b_1, \ldots, b_n), \bar{C}).$$

During the computation of a hypothesis, if the encoding $rule(id(h \leftarrow b_1, \ldots, b_n), \bar{C})$ of a rule r is abduced, then the above rule in $\mathcal{M}$ will enforce that the associated meta-level information of r to be also inferred if r satisfies the domain-dependent constraints.

Example 22. *Consider the following rules in the hypothesis space and top theory in Example 15.*

$$R_M = \left\{ \begin{array}{l} has\_cold(X). \\ has\_cold(X) \leftarrow symptom(X,Y). \\ has\_cold(X) \leftarrow symptom(X,Y), symptom(X,Z). \end{array} \right\}$$

$$\top = \left\{ \begin{array}{l} has\_cold(X) \leftarrow rule((m_1), \$e). \\ has\_cold(X) \leftarrow symptom(X,Y), rule((m_1, m_2, 1), (Y)). \\ has\_cold(X) \leftarrow symptom(X,Y), symptom(X,Z), rule((m_1, m_2, 1, m_2, 1), (Y,Z)). \end{array} \right\}$$

The theory $\mathcal{M}$ corresponding to the above rules in the hypothesis space is:

$$\mathcal{M} = \left\{ \begin{array}{l} 1\{is\_rule(rid(1,(\$e)), has\_cold(\$e))\}1 \leftarrow rule((m_1), \$e). \\ 2\{is\_rule(rid(2,(Y)), has\_cold(\$e)), \\ \quad in\_rule(rid(2,(Y)), has\_cold(\$e), symptom(Y), 1)\}2 \leftarrow rule((m_1, m_2, 1), (Y)). \\ 3\{is\_rule(rid(3,(Y,Z)), has\_cold(\$e)), \\ \quad in\_rule(rid(3,(Y,Z)), has\_cold(\$e), symptom(Y), 1), \\ \quad in\_rule(rid(3,(Y,Z)), has\_cold(\$e), symptom(Z), 2)\}3 \\ \quad \leftarrow rule((m_1, m_2, 1, m2, 1), (Y,Z)). \end{array} \right\}$$

We can show that the meta-level information derived by $\mathcal{M}$ for each set of ASPAL rule encoding Δ is unique. As a consequence of this, and with a labelling scheme that is able to differentiate all the mode declarations, the meta-representation of a hypothesis can be used to identify each individual literal in the hypothesis.

Lemma 1. *Given a learning task with constraint-driven bias $\langle B, M, E, IC\rangle$, let $\top_M = \top \cup \mathcal{M}$ be its extended top theory where $\top$ is ASPAL's top theory for the learning task $\langle B, M, E\rangle$, and $\mathcal{M}$ is the theory for deriving meta-level information of rules in the hypothesis space. For each hypothesis H compatible with M, let Δ be ASPAL's encoding of H. Then, $\mathcal{M} \cup \Delta$ has a unique answer set which is the set $\Delta \cup meta(H)$.*

Proof. Let P be the grounded program of $\mathcal{M} \cup \Delta$. For a set X of grounded atoms, the reduct P^X is the following program.

$$P^X = \left\{ \begin{array}{l} d. \text{ (for each } d \in \Delta) \\ n+1 \ \{is\_rule(rid(k,\bar{c}), l_h(\bar{c_h})), in\_rule(rid(k,\bar{c}), l_h(\bar{c_h}), l_{b_1}(\bar{c_{b_1}}), 1), \ \ldots, \\ \quad in\_rule(rid(k,\bar{c}), l_h(\bar{c_h}), l_{b_n}(\bar{c_{b_n}}), n) \ \} \ \ n+1 \leftarrow rule(id(h \leftarrow b_1, \ldots, b_n), \bar{c}). \\ \quad \text{(for each } rule(id(h \leftarrow b_1, \ldots, b_n), \bar{c})) \end{array} \right\}$$

Consider the aggregate:

$$n+1 \ \{is\_rule(rid(k,\bar{c}), l_h(\bar{c_h})), in\_rule(rid(k,\bar{c}), l_h(\bar{c_h}), l_{b_1}(\bar{c_{b_1}}), 1), \\ \ldots, in\_rule(rid(k,\bar{c}), l_h(\bar{c_h}), l_{b_n}(\bar{c_{b_n}}), n)\} \ n+1 \leftarrow rule(id(h \leftarrow b_1, \ldots, b_n), \bar{c}).$$

which is defined over a set of exactly $n+1$ elements. The aggregate's limits cause all elements in the set to be in the answer set should $rule(id(h \leftarrow b_1, \ldots, b_n), \bar{c})$ be in the answer set. The above aggregate can be expanded into the following definite clauses.

$is\_rule(rid(k,\bar{c}), l_h(\bar{c_h})) \leftarrow rule(id(h \leftarrow b_1, \ldots, b_n), \bar{c}).$

$in\_rule(rid(k,\bar{c}), l_h(\bar{c_h}), l_{b_1}(\bar{c_{b_1}}), 1) \leftarrow rule(id(h \leftarrow b_1, \ldots, b_n), \bar{c}).$

$\ldots$

$in\_rule(rid(k,\bar{c}), l_h(\bar{c_h}), l_{b_n}(\bar{c_{b_n}}), n) \leftarrow rule(id(h \leftarrow b_1, \ldots, b_n), \bar{c}).$

The above rules show that for any rule encoding $enc(r) = rule(id(h \leftarrow b_1, \ldots, b_n), \bar{c})$, if $enc(r)$ is in the model of P^X, then its meta-level representation $\mu(r)$ must also be included the model. Each $d \in \Delta$ is a grounded $rule/2$ atom, and is used with its corresponding aggregate in P^X to derive $\mu(r)$. Since the model of P^X must include Δ, and each $enc(r) \in \Delta$ requires that $\mu(r)$ is also part of the model, the minimal model of P^X is $\Delta \cup \bigcup_{enc(r) \in \Delta} \mu(r)$ which is equivalent to $\Delta \cup meta(H)$.

Suppose there is another answer set S of $\mathcal{M} \cup \Delta$ such that $S \neq \Delta \cup meta(H)$. As Δ is a set of ground atoms, and $\mathcal{M}$ deduces meta-level information as a set, $S = \Delta \cup Q$ where Q is a set of meta-level representations made up from $\mu(r)$ for some rule r such that r is compatible to M. If $Q \neq meta(H)$ then there must exists a rule r such that $\mu(r) \subseteq Q$ but $\mu(r) \nsubseteq meta(H)$, or $\mu(r) \nsubseteq Q$ but $\mu(r) \subseteq meta(H)$. $meta(H)$ contains all meta-level information derived from Δ, so Q must contain all of its elements $meta(H) \subset Q$, and there does not exist a rule r such that $\mu(r) \nsubseteq Q$ but $\mu(r) \subseteq meta(H)$. If $\mu(r)$ is a subset of Q that is not also a subset of $meta(H)$, then there must also be $enc(r)$ in S, and more specifically in Δ as Q does not contain any ASPAL rule encoding. However, all elements of Δ are in $\Delta \cup meta(H)$, so $enc(r)$ must also be in $\Delta \cup meta(H)$, and the answer set S cannot exist. □

RASPAL

ASPAL's representation of meta-level information can also be used in RASPAL. Furthermore, its method for generating meta-level information can also be used for newly learned rules using the original learning task's mode declarations. However, additional consideration is needed when generating meta-level information of the revisable partial hypothesis and revision operators in RASPAL. As the revisable hypothesis and revision operators are used together to represent the revised rules, their meta-level information should be linked together. This means that meta-level information of a revisable rule should be generated only when it is included in the revised hypothesis, and what meta-level is added or removed from it depends on the revision operations applied to the revisable hypothesis.

To make RASPAL generate meta-level information of a rule at the same time it is learnt, we

need to consider the following three revision operations it uses:

1. *Learn a new rule*: The meta-level information can be generated using the same method as ASPAL.

2. *Extend an existing rule with new body literals*: The extension operation learnt should have the same head label as the rule it is extending in its meta-level representation. No meta-level information should be generated for its head literal representing the operation ($extension/2$), and all body literals should be linked to the head literal of the revisable rule. The meta-level information of the revisable hypothesis should be generated alongside the meta-level information of the extension operation.

3. *Delete an existing body literal from a rule*: All extension of the same revisable rule should not include meta-level information of the deleted body literal.

4. *Delete an existing rule*: No meta-level information of the revisable rule should be generated.

Example 23. Given the revisable rules:

$$r(T) = \left\{ \begin{array}{l} father(X,Y) \leftarrow try(1,1,vars(Y)), extension(1,vars(X,Y)). \\ try(1,1,vars(Y)) \leftarrow female(Y), not\ delete(1,1). \\ try(1,1,vars(Y)) \leftarrow delete(1,1). \\ \bot \leftarrow delete(1,1), 0\{extension(1,vars(X,Y))\}0. \end{array} \right\}$$

Suppose the following three alternative change transactions can be learnt:

- $C_1 = \left\{ \begin{array}{l} extension(1,vars(X,Y)) \leftarrow parent(X,Y). \end{array} \right\}$
- $C_2 = \left\{ \begin{array}{l} extension(1,vars(X,Y)). \\ delete(1,1). \end{array} \right\}$
- $C_3 = \emptyset$

Let i and j be the unique identifiers of revision operations. The meta-level information that should be generated with each change transaction is:

- $meta(C_1) = \left\{ \begin{array}{l} is\_rule(rid(i,(\$e)), father(\$e)), \\ in\_rule(rid(i,(\$e)), father(\$e), female(\$e), p1), \\ in\_rule(rid(i,(\$e)), father(\$e), parent(\$e), 1) \end{array} \right\}$

  Meta-level information of the revisable rule are generated as no deletions are learnt. Information of the additional body literal is added as $in\_rule(rid(i,(\$e)),\ father(\$e),\ parent(\$e), 1)$. The identifier i of the extension is used as identifier in all meta-level information, indicating that they all represent the same revised rule. The index $p1$ is used as the index of the old body literal to differentiate it from the ones added by the extension.

- $meta(C_2) = \{is\_rule(rid(j,(\$e)), father(\$e))\}$

  The head meta-level information is generated but the information for the body literal $female(Y)$ is not as it has been deleted by the change transaction.

- $meta(C_3) = \emptyset$

  No meta-level information for the revisable rule is generated.

To tie the meta-level information of a revisable rule to its revision operations we will use an auxiliary predicate $partial\_meta(RID, i, l_h(\overline{C}_h))$ whose arguments are the identifier RID of an extension operation, the index i of the revisable rule, a label l_h of the head literal of the revisable rule, and the list $\overline{C}_h$ of the constant arguments in that head literal. The revisable theory needs to be redefined to take meta-level information into account.

Definition 6.2 (Revisable theory with meta-level information)**.** *For each clause $h_i \leftarrow b_{i,1}, \ldots, b_{i,n}$ in a given theory T, let $vars(r)$ be the list of all variables in a rule r and $vars(b)$ be the list of all variables in a body literal b. The following clauses are in its revisable form $r_M(T)$:*

- $h_i \leftarrow try(i, 1, vars(b_{i,1})), \ldots, try(i, n, vars(b_{i,n})),\ extension(i, vars(r_i))$.
- $is\_rule(RID, l_{h_i}(\bar{C}_{h_i})) \leftarrow partial\_meta(RID, i, l_{h_i}(\bar{C}_{h_i})), extension(i, vars(r_i))$.
- $try(i, j, vars(b_{i,j})) \leftarrow b_{i,j}, not\ delete(i, j)$. (for each $try(i, j, vars(b_{i,j}))$)

- $try(i, j, vars(b_{i,j})) \leftarrow delete(i, j)$. (for each $try(i, j, vars(b_{i,j}))$)
- $\bot \leftarrow delete(i, j), \{extension(i, vars(r_i))\}0$. (for each $delete(i, j)$)
- $in\_rule(RID, l_{h_i}(\bar{C_{h_i}}), l_{b_{i,j}}(\bar{C_{b_{i,j}}}), p_j) \leftarrow partial\_meta(RID, i, l_{h_i}(\bar{C_{h_i}})),$
  $extension(i, vars(r_i)), not\ delete(i, j)$. (for each $delete(i, j)$)

where p_j is the index j prefixed by p.

The difference between $r(T)$ and $r_M(T)$ is that $r_M(T)$ will generate $is\_rule/2$ for its revisable rule should the corresponding $partial\_meta/3$ be generated and $extension/2$ be learnt. Should multiple extensions of a rule be learnt then it can still generate $is\_rule/2$ with different RID as the different extensions would generate $partial\_meta/3$ with different identifications. $r_M(T)$ will also generate $in\_rule/4$ for each body literal in a revisable rule should the corresponding $partial\_meta/3$ be generated, $extension/2$ for the rule be learnt, and no $delete/2$ for that literal be learnt. Again, the use of $partial\_meta/3$ makes it possible for the $in\_rule/4$ to be generated for multiple revised rules. The function MakeRevisionTask used in Algorithm 2 is refined into MakeRevisionTask$_R$ which can generate meta-level information of the revisable hypothesis.

The meta-level information of both extension and delete rules in the top theory $\top$ cannot be generated using the same method as the one used for generating meta-level information of newly learnt rules in $\top$. The theory $\mathcal{M}$ from Section 6.2 needs to be modified so that revision operations are taken into account. Each rule with head literal $delete(i, j)$ in $\top$ does not generate any meta-level information, so $\mathcal{M}$ should not generate any meta-level information for each delete operation's rule encoding. Let $extension(i, vars(r_i))$ be an extension operation on the revisable rule $r = h \leftarrow b_1, \ldots, b_n$. For each $extension(i, vars(r)) \leftarrow b_1, \ldots, b_n$, $rule(id(extension(i, vars(r)) \leftarrow b_1, \ldots, b_n), \bar{C})$ in $\top$, the following rule is automatically added to $\mathcal{M}$.

$$
\begin{aligned}
n + 1\{&partial\_meta(rid(k, \bar{C}), i, l_h(\bar{C_h})), in\_rule(rid(k, \bar{C}), l_h(\bar{C_h}), l_{b_1}(\bar{C_{b_1}}), 1),\\
&\ldots, in\_rule(rid(k, \bar{C}), l_h(\bar{C_h}), l_{b_n}(\bar{C_{b_n}}), n)\}n + 1\\
&\leftarrow rule(id(extension(i, vars(r)) \leftarrow b_1, \ldots, b_n), \bar{C}).
\end{aligned}
$$

The theory $\mathcal{M}$ is used in Algorithm 7, defined below, which is variant of Algorithm 6 such that meta-level information of revision operations can be added, and will used to make ASPAL able to generate different meta-level information for rules that are extension operations.

Algorithm 7 META-PRE-PROCESS$_R(\top, M)$
Require: $\top$ ASPAL top theory; M mode declarations
Output: $\top_M$ extended top theory

1: $\mathcal{M} = \emptyset$
2: **for each** $h \leftarrow b_1, \ldots, b_n, rule(id(h \leftarrow b_1, \ldots, b_n), \bar{C}) \in \top$ with identifier k and h is not a revision operation **do**
3: $\quad \mathcal{M} = \mathcal{M} \cup \{n+1\{is\_rule(rid(k, \bar{C}), l_h(\bar{C_h})), in\_rule(rid(k, \bar{C}), l_h(\bar{C_h}), l_{b_1}(\bar{C_{b_1}}), 1), \ldots,$
4: $\quad\quad in\_rule(rid(k, \bar{C}), l_h(\bar{C_h}), l_{b_n}(\bar{C_{b_n}}), n)\}n+1 \leftarrow rule(id(h \leftarrow b_1, \ldots, b_n), \bar{C})\}$
5: **for each** $extension(i, vars(r)) \leftarrow b_1, \ldots, b_n, rule(id(extension(i, vars(r)) \leftarrow b_1, \ldots, b_n), \bar{C}) \in \top$ with corresponding revisable rule $h \leftarrow b_{p1}, \ldots, b_{pm}$ and identifier k **do**
6: $\quad \mathcal{M} = \mathcal{M} \cup \{n+1\{partial\_meta(rid(k, \bar{C}), i, l_h(\bar{C_h})), in\_rule(i(\bar{C}), l_h(\bar{C_h}), l_{b_1}(\bar{C_{b_1}}), 1), \ldots,$
7: $\quad\quad in\_rule(i(\bar{C}), l_h(\bar{C_h}), l_{b_n}(\bar{C_{b_n}}), n)\}n + 1 \leftarrow rule(id(extension(i, vars(r)) \leftarrow b_1, \ldots, b_n), \bar{C})\}$
8: **return** $\top \cup \mathcal{M}$

Example 24. Given the partial hypothesis $H = \{father(X, Y) \leftarrow female(Y).\}$. Its revisable representation with meta-level information is:

$$r_M(H) = \left\{ \begin{array}{l} father(X, Y) \leftarrow try(1, 1, vars(Y)), extension(1, vars(X, Y)). \\ is\_rule(ID, father(\$e)) \leftarrow partial\_meta(ID, 1, father(\$e)), \\ \quad extension(1, vars(X, Y)). \\ try(1, 1, vars(Y)) \leftarrow female(Y), not\ delete(1, 1). \\ try(1, 1, vars(Y)) \leftarrow delete(1, 1). \\ \bot \leftarrow delete(1, 1), 0\{extension(1, vars(X, Y))\}0. \\ in\_rule(ID, father(\$e), female(\$), p1) \leftarrow partial\_meta(ID, 1, father(\$e)), \\ \quad extension(1, vars(X, Y)), not\ delete(1, 1). \end{array} \right\}$$

Suppose this is used in RASPAL's refinement step with the following mode declarations:

$$r_M(H) = \left\{ \begin{array}{l} m_1 : modeh(father(+person, +person)). \\ m_2 : modeb(male(+person)). \\ m_3 : modeb(female(+person)). \\ m_4 : modeb(parent(+person, +person)). \\ m_5 : modeh(extension(1, vars(+person, +person))). \\ m_6 : modeb(delete(1, 1)). \end{array} \right\}$$

The top theory $\top$ would contain the rules:

$$\top = \left\{ \begin{array}{l} \ldots \\ father(X, Y) \leftarrow male(X), rule((m_1, m_2, 1), \$e). \\ \ldots \\ extension(1, vars(X, Y)) \leftarrow parent(X, Y), rule((m_5, m_4), \$e). \\ \ldots \\ delete(1, 2) \leftarrow rule((m_6), \$e). \\ \ldots \end{array} \right\}$$

Let i, j and k be identifiers, then $\mathcal{M}$ for the top theory would contain the rules:

$$\mathcal{M} = \left\{ \begin{array}{l} \ldots \\ 2\{is\_rule(rid(i, (\$e)), father(\$e)), in\_rule(rid(i, (\$e)), father(\$e), male(\$e))\}2 \\ \qquad \leftarrow rule((m_1, m_2, 1), \$e). \\ \ldots \\ 2\{partial\_meta(rid(j, (\$e)), 1, father(\$e)), in\_rule(rid(j, (\$e)), father(\$e), \\ \qquad parent(\$e))\}2 \leftarrow rule((m_5, m_4), \$e). \\ \ldots \end{array} \right\}$$

Algorithm 7 is in turns used in Algorithm 8 which, in addition to using META-PRE-PROCESS$_R$ and its extended $\mathcal{M}$, will convert IC into $t_{RASPAL}(IC)$, which is added to the background theory of the abductive task. $t_{RASPAL}(IC)$ converts each integrity constraint in IC into its ASP representation, but will convert all ASP constraints into rules. $t_{RASPAL}(IC)$ is defined in Definition 6.3 below.

Definition 6.3. *The set of clauses $t_{RASPAL}(IC)$ is constructed from the set of domain-dependent*

Algorithm 8 ASPAL$_{CR}(E, B, M, IC)$
Require: E examples; B background theory; M mode declarations; IC domain dependent integrity constraints
Output: H hypothesis

1: $\langle \top, A^{\top} \rangle$ =TOP-THEORY(B, M)
2: ϵ =EXAMPLE-PRE-PROCESS(E)
3: $\top_M$ =META-PRE-PROCESS$_R(\top, M)$
4: $t(IC)$ =TRANSLATE(IC)
5: Δ = ABDUCE$(B \cup \top_M \cup \epsilon \cup t_{RASPAL}(IC), A^{\top}, \emptyset)$
6: H =POST-PROCESS(Δ, M)

constraints IC as follows.

- *For each $ic_i \in IC$, let $t(ic_i)$ be its ASP representation, and for each clause c in $t(ic_i)$:*
  - *If $head(c) \neq \bot$ then c is in $t_{RASPAL}(IC)$;*
  - *If $head(c) = \bot$ then $cdb\_ic_i \leftarrow body(c)$ is in $t_{RASPAL}(IC)$;*

Effectively, each cdb_ic_i represents a constraint that is violated as its inclusion in an answer set means that the body literals of an integrity constraint are also in the answer set. This makes it possible to find out the number of violated constraints from the answer set of the abductive task, and incorporate it into RASPAL's scoring scheme.

Definition 6.4 (Scoring partial hypotheses with constraints)**.** *Let $\langle E, B, M, IC \rangle$ be an ILP task with constraint-driven bias and let H be a partial hypothesis in its hypothesis space $\mathcal{P}(R_M)$. The score of H is the tuple $score(H) = \langle cover_{e^+}(H), cover_{e^-}(H), ic(H), length(H) \rangle$, where $cover_{e^+}(H)$ is the number of positive examples covered by H, $cover_{e^-}(H)$ is the number of negative examples covered by H, $ic(H)$ is the number of constraints in IC violated by H, and $length(H)$ is the total number of literals in H.*

Definition 6.5 (Comparing partial hypotheses with constraints)**.** *Let $\langle E, B, M, IC \rangle$ be an ILP task with constraint-driven bias and let H and H' be two (partial) hypotheses in $\mathcal{P}(R_M)$. H is* better *than H', denoted $score(H) > score(H')$ if and only if one of the following cases applies:*
− $cover_{e^+}(H) > cover_{e^+}(H')$,
− $cover_{e^+}(H) = cover_{e^+}(H') \wedge cover_{e^-}(H) < cover_{e^-}(H')$,

– $cover_{e^+}(H) = cover_{e^+}(H') \wedge cover_{e^-}(H) = cover_{e^-}(H') \wedge ic(H) < ic(H')$,

– $cover_{e^+}(H) = cover_{e^+}(H') \wedge cover_{e^-}(H) = cover_{e^-}(H') \wedge ic(H) = ic(H') \wedge length(H) < length(H')$

The modified scoring scheme and Algorithm 8 are used to modify RASPAL's learning algorithm (Algorithm 1), and its hypothesis refinement algorithm (Algorithm 2)

Algorithm 9 RASPAL$_C(P, i)$
Require: $P = \langle E, B, M, IC\rangle$
Output: $\langle Hypothesis, Score\rangle$, a solution to P and its score

```
1: let ⟨Hypothesis, Score⟩ =FindOptimalHypothesis_C(P, i)
2: if Hypothesis == ∅ then                          ▷ Empty hypothesis found
3:     return RASPAL_C(P, i + 1)
4: loop
5:     if Score ≥ ⟨|{e|e ∈ E}|, 0, 0, +∞⟩ then      ▷ Solution found
6:         return ⟨Hypothesis, Score⟩
7:     ⟨Hypothesis, Score_new⟩ =RefineHypothesis_C(Hypothesis, P, i + 1)
8:     if Score_new ≤ Score then                    ▷ Score does not improve
9:         return RASPAL_C(P, i + 1)
10:    Score = Score_new
```

Algorithm 10 RefineHypothesis$_C(Hypothesis, P, i)$
Require: $P = \langle E, B, M, IC\rangle$
Output: $\langle Hypothesis_{new}, Score_{new}\rangle$, a refinement of $Hypothesis$ of and its score

```
1: P' = ⟨∅, B ∪ r_M(Hypothesis), M ∪ M_Δ⟩ = MakeRevisionTask(P, Hypothesis)
2: ⟨Changes, Score_new⟩ = FindOptimalHypothesis_C(P', i + 1)
3: Hypothesis_new = ApplyRefinement(Hypothesis, Changes)
4: return ⟨Hypothesis_new, Score_new⟩
```

Both Algorithms 9 and 10 use the function FindOptimalHypothesis$_C$ rather than FindOptimalHypothesis. FindOptimalHypothesis$_C$ uses the abductive task in Algorithm 8. FindOptimalHypothesis$_C$ also use the following optimisation statements so that ASPAL will search for the optimal hypothesis with respect to Definition 6.5, and another element is added to RASPAL's score as Defined in 6.4.

$$\#minimise[r_1 = weight(r_1), \ldots, r_k = weight(r_k)].$$

$$\#minimise[cdb\_ic_1, \ldots, cdb\_ic_p].$$

$$\#minimise[e_1, \ldots, e_n].$$

$$\#maximise[e_{n_1}, \ldots, e_m].$$

where $r_1, \ldots, e_k$ are all rules in the set of rules R_M compatible with the learning task's mode declarations, $e, \ldots, e_n$ are positive examples, $e, \ldots, e_m$ are negative examples, $cdb\_ic_1, \ldots, cdb\_ic_p$ are the constants that represent constraints $ic_1, \ldots, ic_p$ in IC, and $weight(r)$ is $length(r)$ if the head of the rule is not a revision operation, $length(r) - 1$ if the rule is an extension operation, or -1 if the rule is a deletion operation. The added optimisation statement $\#minimise[cdb\_ic_1, \ldots, cdb\_ic_p]$ will prioritise the minimisation of the violated constraint below the examples coverage, and above the minimisation of the hypothesis size.

Algorithm 9 takes as its input a learning task with constraint-driven bias instead of a typical inductive learning task, and uses the scoring scheme from Definition 6.4. Its termination condition on line 5 reflects the score comparison from Definition 6.5, making it returns only hypotheses that satisfies all constraints in IC. Algorithm 10 has also been modified so that $r_M/1$ is used in place of $r/1$, in order to generate the meta-level information of any extended revisable rules.

Similar to ASPAL, we can show the modified $\mathcal{M}$ will derive a unique set of meta-level representations for solutions learnt by RASPAL. To do this we first reason about the meta-level information of each revised partial hypothesis in the RASPAL refinement loop.

Lemma 2. *Given a RASPAL refinement task expressed as an ILP task $\langle E, B \cup r_M(H), M \cup M_\Delta \rangle$, where E are grounded examples, B is the background knowledge, $r_M(H)$ is the revisable form of partial hypothesis H, M are the mode declarations of the original learning task, and M_Δ are the mode declarations for the refinement operations. Let $\top_M = \top \cup \mathcal{M}$ be its extended top theory, Δ be the ASPAL encoding corresponding to the change transaction C that is the solution of the task, then $\mathcal{M} \cup \Delta \cup r_M(H) \cup C$ has a unique answer set S such that $\Delta \cup meta(C \otimes r_M(H)) \subset S$ and any is_rule/2 and in_rule/4 that are not in $meta(C \otimes r_M(H))$ are not in S.*

Proof. The theory $r_M(H)$ can be partitioned into two programs $r^1_M(H)$ and $r^2_M(H)$, such that $r^1_M(H)$ are parts of $r_M(H)$ with meta-level information and $r^2_M(H)$ are parts of $r_M(H)$ without any meta-level information. Specifically, $r^1_M(H)$ contains only rules of the forms:

- $is\_rule(RID, l_{h_i}(\bar{C_{h_i}})) \leftarrow partial\_meta(RID, i, l_{h_i}(\bar{C_{h_i}})), extension(i, vars(r_i))$.

- $in\_rule(RID, l_{h_i}(\bar{C_{h_i}}), l_{b_{i,j}}(\bar{C_{b_{i,j}}}), p_j) \leftarrow partial\_meta(RID, i, l_{h_i}(\bar{C_{h_i}})),$
  $extension(i, vars(r_i)), not\ delete(i, j)$. *(for each $delete(i, j)$)*

We will use $r^1_M(H)$ for this proof as we are only concerned about the meta-level information in $r_M(H)$. Furthermore, we will use the set C of change transaction derived from $\top \cup \Delta$ in place of $\top \cup \Delta$ to make the proof more readable.

Let P be the grounded program of $\mathcal{M} \cup r^1_M(H) \cup \Delta \cup C$. For a set X of grounded atoms, the reduct P^X is the following program.

$$P^X = \left\{ \begin{array}{l}
\text{1: } d. \text{ (for each } d \in \Delta) \\
\text{2: } c. \text{ (for each } c \in C) \\
\text{3: } is_rule(rid, l_{h_i}(\overline{c_{h_i}})) \leftarrow partial_meta(rid, i, l_{h_i}(\overline{c_{h_i}}), extension(i, vars(r_i))). \\
\quad \text{(for each pair of } rid \text{ and } l_{h_i}(\overline{c_{h_i}}), \text{ and its corresponding } r_i) \\
\text{4: } in_rule(rid, l_{h_i}(\bar{(}c_{h_i})), l_{b_{i,j}}(\bar{(}c_{b_{i,j}})), p_j) \leftarrow partial_meta(rid, i, l_{h_i}(\bar{(}c_{h_i})), \\
\quad extension(i, vars(r_i))). \text{ (for each pair of } rid \text{ and } l_{h_i}(\overline{c_{h_i}}), \text{ and its corresponding} \\
\quad r_i \text{ such that } delete(i, j) \text{ is not in } C) \\
\text{5: } n+1\ \{is_rule(rid(k, \bar{c}), l_h(\bar{c_h})), in_rule(rid(k, \bar{c}), l_h(\bar{c_h}), l_{b_1}(\bar{c_{b_1}}), 1), \ldots, \\
\quad in_rule(rid(k, \bar{c}), l_h(\bar{c_h}), l_{b_n}(\bar{c_{b_n}}), n)\ \}\ \ n+1 \leftarrow rule(id(h \leftarrow b_1, \ldots, b_n), \bar{c}). \\
\quad \text{(for each } h \leftarrow b_1, \ldots, b_n \text{ compatible with } M) \\
\text{6: } n+1\{partial_meta(rid(k, \bar{c}), i, l_h(\bar{c_h})), in_rule(rid(k, \bar{c}), l_h(\bar{c_h}), l_{b_1}(\bar{c_{b_1}}), 1), \ldots, \\
\quad in_rule(rid(k, \bar{c}), l_h(\bar{c_h}), l_{b_n}(\bar{c_{b_n}}), n)\} n+1 \\
\quad \leftarrow rule(id(extension(i, vars(r)) \leftarrow b_1, \ldots, b_n), \bar{c}) \\
\quad \text{(for each } ex \leftarrow b_1, \ldots, b_n \text{ where } ex \text{ is an extension operator)}
\end{array} \right\}$$

The two aggregates in lines 5 and 6 of P^X specify that all elements of the set should be in the model when the rule encoding in their respective body condition is in the model. The minimal model of P^X must include $\Delta \cup C$. As it contains C, From line 6 of P^X, its model will contain meta-level information for all body predicates in the extension operation. Line 6 also makes an instance of $partial\_meta/3$ be included in the answer set for each extension operation.

Each instance of *partial_meta*/3, when combined with rules 3 and 4, will make the meta-level representation of the head and (not deleted) body literals of the revisable rule be included in the model of P^X. In summary, the combination of lines 3, 4 and 5 of P^X, an extension and deletion operations of the same revisable rule will result in $\mu(r) \cup \{partial\_meta(rid(k, \overline{c}, i, l_h(\overline{c_h}))\}$ be included in the model where r is the revised rule with index i in the revisable hypothesis such that its head literal is labelled l_h, it has $\overline{c_h}$ constant arguments, and k is identifier of the extension rule.

Line 5 of P^X is the same aggregate that was used for ASPAL's meta-level representation which we have shown in the proof of Lemma 1 that it will derive the unique meta-level representation for newly learnt rules. The minimal model of P^X is $\Delta \cup C \cup \bigcup_{enc(r) \in \Delta_M} \mu(r) \cup \bigcup_{enc(c) \in \Delta_M} \mu(c \otimes r(H)) \cup Q$ where r is a newly learnt rule, c is a learnt revision operation, and Q is the set of *partial_meta*/3 instances. The combination of $\bigcup_{enc(r) \in \Delta_M} \mu(r) \cup \bigcup_{enc(c) \in \Delta_M} \mu(c \otimes r(H))$ corresponds to the meta-level representation of the revise rules $meta(C \otimes r_M(H))$, thus $\Delta \cup meta(C \otimes r_M(H))$ must be a subset of the model. All *is_rule*/2 and *in_rule*/4 instances in the minimal model of P^X can only be in $\bigcup_{enc(r) \in \Delta_M} \mu(r) \cup \bigcup_{enc(c) \in \Delta_M} \mu(c \otimes r(H))$, and as this corresponds to $meta(C \otimes r_M(H))$, then *is_rule*/2 and *in_rule*/4 that do not describe the refined hypothesis cannot be in the minimal model of P^X. □

Lemma 3. *Given a learning task with constraint-driven bias* $\langle E, B, M, IC \rangle$*. Let* H *be a hypothesis that can be generated by* RASPAL_C*, then* $meta(H)$ *will also be generated by* RASPAL_C*.*

Proof. The hypothesis H can be generated by either Algorithm 8 as RASPAL's initial partial hypothesis, or by the refinement loop in RASPAL_C. For the former, there is no revisable partial hypothesis in the background knowledge nor any extension rules that can be learnt, reducing $\mathcal{M}$ to be the same as ASPAL's version. By Lemma 1 we know that $meta(H)$ must have been generated alongside H. For the latter, if H is the product of RASPAL_C's refinement loop, then we can apply Lemma 2 that $meta(H)$ must have been generated alongside H. □

6.3 Translation of $\mathcal{L}_\mathcal{C}$

Using the meta-level predicates we can translate constraints in IC defined using primitives of $\mathcal{L}_\mathcal{C}$ into an ASP representation. This translation is given in Table 6.1 for each constraint $\leftarrow P_i, \overline{C_{Bi}} \in IC$. The translation in Table 6.1 introduce the new predicate ic_cond_i into the translation of most constraints, and is used as a mechanism for negating multiple literals, and negating literals with unbounded variables. Each ic_cond_i represents a predicate name that is used solely for the corresponding $\leftarrow P_i, \overline{C_{Bi}}$, making the translation of different constraints not overlap when they are in the same program. Note that we assume the labels to all be grounded before their encoding.

Primitive predicate $\leftarrow P_i, \overline{C_{Bi}}$	Declarative semantics of $\leftarrow P_i, \overline{C_{Bi}}$
$\bot \leftarrow not\ in\_some\_rule(l_h(\overline{C_h}), l_b(\overline{C_b})),$ $\overline{v(C_h)}, \overline{v(C_b)}, \overline{C_{Bi}}.$	$ic\_cond_i \leftarrow in\_rule(Rid, l_h(\overline{C_h}), l_b(\overline{C_b}), I),$ $\overline{v(C_h)}, \overline{v(C_b)}.$ $\bot \leftarrow not\ ic\_cond_i, \overline{C_{Bi}}.$
$\bot \leftarrow in\_some\_rule(l_h(\overline{C_h}), l_b(\overline{C_b})),$ $\overline{v(C_h)}, \overline{v(C_b)}, \overline{C_{Bi}}.$	$ic\_cond_i \leftarrow in\_rule(Rid, l_h(\overline{C_h}), l_b(\overline{C_b}), I),$ $\overline{v(C_h)}, \overline{v(C_b)}.$ $\leftarrow ic\_cond_i, \overline{C_{Bi}}.$
$\bot \leftarrow not\ in\_all\_rule(l_h(\overline{C_h}), l_b(\overline{C_b})),$ $\overline{v(C_h)}, \overline{v(C_b)}, \overline{C_{Bi}}.$	$\bot \leftarrow is\_rule(Rid, l_h(\overline{C_h})), \overline{v(C_h)},$ $not\ ic\_cond_i(Rid, l_h(\overline{C_h})), \overline{C_{Bi}}.$ $ic\_cond_i(Rid, l_h(\overline{C_h})) \leftarrow \overline{v(C_h)}, \overline{v(C_b)},$ $in\_rule(Rid, l_h(\overline{C_h}), l_b(\overline{C_b}), I).$
$\bot \leftarrow in\_all\_rule(l_h(\overline{C_h}), l_b(\overline{C_b})),$ $\overline{v(C_h)}, \overline{v(C_b)}, \overline{C_{Bi}}.$	$ic\_cond_{i,1} \leftarrow is\_rule(Rid, l_h(\overline{C_h})), \overline{v(C_h)},$ $not\ ic\_cond_{i,2}(Rid, l_h(\overline{C_h})), \overline{v(C_b)}.$ $ic\_cond_{i,2}(Rid, l_h(\overline{C_h})) \leftarrow \overline{v(C_h)}, \overline{v(C_b)},$ $in\_rule(Rid, l_h(\overline{C_h}), l_b(\overline{C_b}), I).$ $\bot \leftarrow not\ ic\_cond_{i,1}, \overline{C_{Bi}}.$
$\bot \leftarrow not\ in\_same\_rule(l_h(\overline{C_h}), l_{b_1}(\overline{C_{b_1}}), l_{b_2}(\overline{C_{b_2}})),$ $\overline{v(C_h)}, \overline{v(C_{b_1})}, \overline{v(C_{b_2})}, \overline{C_{Bi}}.$	$ic\_cond_{i,1}(Rid, l_h(\overline{C_h})) \leftarrow \overline{v(C_h)}, \overline{v(C_{b_1})},$ $in\_rule(Rid, l_h(\overline{C_h}), l_{b_1}(\overline{C_{b_1}}), I_1).$ $ic\_cond_{i,2}(Rid, l_h(\overline{C_h})) \leftarrow \overline{v(C_h)}, \overline{v(C_{b_2})},$ $in\_rule(Rid, l_h(\overline{C_h}), l_{b_2}(\overline{C_{b_2}}), I_2).$

	$\bot \leftarrow ic\_cond_{i,1}(Rid, l_h(\overline{C_h})),$ $\quad not\ ic\_cond_{i,2}(Rid, l_h(\overline{C_h})),$ $\quad \overline{v(C_h)}, \overline{C_{Bi}}.$ $\bot \leftarrow not\ ic\_cond_{i,1}(Rid, l_h(\overline{C_h})),$ $\quad ic\_cond_{i,2}(Rid, l_h(\overline{C_h})),$ $\quad \overline{v(C_h)}, \overline{C_{Bi}}.$
$\bot \leftarrow in\_same\_rule(l_h(\overline{C_h}), l_{b_1}(\overline{C_{b_1}}), l_{b_2}(\overline{C_{b_2}})),$ $\overline{v(C_h)}, \overline{v(C_{b_1})}, \overline{v(C_{b_2})}, \overline{C_{Bi}}.$	$ic\_cond_{i,1}(Rid, l_h(\overline{C_h})) \leftarrow \overline{v(C_h)}, \overline{v(C_{b_1})},$ $\quad in\_rule(Rid, l_h(\overline{C_h}), l_{b_1}(\overline{C_{b_1}}), I_1).$ $ic\_cond_{i,2}(Rid, l_h(\overline{C_h})) \leftarrow \overline{v(C_h)}, \overline{v(C_{b_2})},$ $\quad in\_rule(Rid, l_h(\overline{C_h}), l_{b_2}(\overline{C_{b_2}}), I_2).$ $\bot \leftarrow ic\_cond_{i,1}(Rid, l_h(\overline{C_h})), \overline{v(C_h)},$ $\quad ic\_cond_{i,2}(Rid, l_h(\overline{C_h})), \overline{C_{Bi}}.$
$\bot \leftarrow not\ in\_diff\_rule(l_{h_1}(\overline{C_{h_1}}), l_{b_1}(\overline{C_{b_1}}),$ $l_{h_2}(\overline{C_{h_2}}), l_{b_2}(\overline{C_{b_2}})),$ $\overline{v(C_{h_1})}, \overline{v(C_{h_2})}, \overline{v(C_{b_1})}, \overline{v(C_{b_2})}, \overline{C_{Bi}}.$	$ic\_cond_i \leftarrow \overline{v(C_{h_1})}, \overline{v(C_{h_2})}, \overline{v(C_{b_1})}, \overline{v(C_{b_2})},$ $\quad in\_rule(Rid_1, l_{h_1}(\overline{C_{h_1}}), l_{b_1}(\overline{C_{b_1}}), I_1),$ $\quad in\_rule(Rid_2, l_{h_2}(\overline{C_{h_2}}), l_{b_1}(\overline{C_{b_2}}), I_2),$ $\quad Rid_1 \neq Rid_2.$ $\bot \leftarrow not\ ic\_cond_i, \overline{C_{Bi}}.$
$\bot \leftarrow in\_diff\_rule(l_{h_1}(\overline{C_{h_1}}), l_{b_1}(\overline{C_{b_1}}),$ $l_{h_2}(\overline{C_{h_2}}), l_{b_2}(\overline{C_{b_2}})),$ $\overline{v(C_{h_1})}, \overline{v(C_{h_2})}, \overline{v(C_{b_1})}, \overline{v(C_{b_2})}, \overline{C_{Bi}}.$	$ic\_cond_{i,1}(Rid) \leftarrow \overline{v(C_{h_1})}, \overline{v(C_{b_1})},$ $\quad in\_rule(Rid, l_{h_1}(\overline{C_{h_1}}), l_{b_1}(\overline{C_{b_1}}), I_1).$ $ic\_cond_{i,2}(Rid) \leftarrow \overline{v(C_{h_2})}, \overline{v(C_{b_2})},$ $\quad in\_rule(Rid, l_{h_2}(\overline{C_{h_2}}), l_{b_2}(\overline{C_{b_2}}), I_2).$ $\bot \leftarrow ic\_cond_{i,1}(Rid_1), ic\_cond_{i,2}(Rid_2),$ $\quad Rid_1 \neq Rid_2, \overline{C_{Bi}}.$
$\bot \leftarrow not\ max\_body(l_h(\overline{C_h}), x), \overline{v(C_h)}, \overline{C_{Bi}}.$	$\bot \leftarrow is\_rule(Rid, l_h(\overline{C_h})), \overline{v(C_h)},$ $\quad x + 1\{in\_rule(Rid, l_h(\overline{C_h}), L_b, I)\}, \overline{C_{Bi}}.$
$\bot \leftarrow max\_body(l_h(\overline{C_h}), x), \overline{v(C_h)}, \overline{C_{Bi}}.$	$ic\_cond_i \leftarrow is\_rule(Rid, l_h(\overline{C_h})), \overline{v(C_h)},$ $\quad x + 1\{in\_rule(Rid, l_h(\overline{C_h}), L_b, I)\}.$ $\bot \leftarrow not\ ic\_cond_i, \overline{C_{Bi}}.$
$\bot \leftarrow not\ min\_body(l_h(\overline{C_h}), x), \overline{v(C_h)}, \overline{C_{Bi}}.$	$\bot \leftarrow is\_rule(Rid, l_h(\overline{C_h})), \overline{v(C_h)},$ $\quad 0\{in\_rule(Rid, l_h(\overline{C_h}), L_b, I)\}x - 1, \overline{C_{Bi}}.$
$\bot \leftarrow min\_body(l_h(\overline{C_h}), x), \overline{v(C_h)}, \overline{C_{Bi}}.$	$ic\_cond_i \leftarrow is\_rule(Rid, l_h(\overline{C_h})), \overline{v(C_h)},$

	$0\{in\_rule(Rid, l_h(\overline{C_h}), L_b, I)\}x - 1.$ $\bot \leftarrow not\ ic\_cond_i, \overline{C_{Bi}}.$
$\bot \leftarrow not\ max\_head(l_h(\overline{C_h}), x), \overline{v(C_h)}, \overline{C_{Bi}}.$	$\bot \leftarrow x + 1\{is\_rule(Rid, l_h(\overline{C_h}))\}, \overline{v(C_h)}, \overline{C_{Bi}}.$
$\bot \leftarrow max\_head(l_h(\overline{C_h}), x), \overline{C_{Bi}}.$	$ic\_cond_i \leftarrow x + 1\{is\_rule(Rid, l_h(\overline{C_h}))\}, \overline{v(C_h)}, .$ $\bot \leftarrow not\ ic\_cond_i, \overline{C_{Bi}}.$
$\bot \leftarrow not\ min\_head(l_h(\overline{C_h}), x), \overline{C_{Bi}}.$	$\bot \leftarrow 0\{is\_rule(Rid, l_h(\overline{C_h}))\}x - 1, \overline{v(C_h)}, \overline{C_{Bi}}.$
$\bot \leftarrow min\_head(l_h(\overline{C_h}), x), \overline{C_{Bi}}.$	$ic\_cond_i \leftarrow 0\{is\_rule(Rid, l_h(\overline{C_h}))\}x - 1, \overline{v(C_h)}.$ $\bot \leftarrow not\ ic\_cond_i, \overline{C_{Bi}}.$

Table 6.1: Declarative semantics of primitives of $\mathcal{L}_\mathcal{C}$

To show the correctness of the translation, we first reason that the set of all hypotheses generated by a learning task with constraint-driven bias is a subset of those generated by the equivalent learning task without the constraints. This ensures that the addition of the meta-constraints will not effect the learnt hypothesis' coverage of the examples. We can then reason translation by translation that the learnt hypothesis satisfies the meta-constraints with respect to Definition 5.2.

Proposition 1. *Given a learning task with constraint-driven bias $LT_1 = \langle B, M, E, IC\rangle$. Let $LT_2 = \langle B, M, E\rangle$ be an equivalent learning task without constraints. If a hypothesis H is an solution to LT_1 generated by* ASPAL$_c$ *, then it is also a solution to LT_2 generated by* ASPAL*.*

Proof. Consider Algorithm 4, the ASP program used to solve the abductive task is $\Pi_1 = B \cup \top \cup \epsilon$ with abducible $A^\top$, where ϵ is the integrity constraint from the examples E. Similarly for Algorithm 5, the ASP program is $\Pi_2 = B \cup \top_M \cup \epsilon \cup t(IC)$ with abducible $A^\top$. The only differences between them would be $\top_M$ and IC. $\mathcal{M}$ derives the meta-representation of H, represented by *is_rule*/2 and *in_rule*/4 which are used as body literals in $t(IC)$ but are in no other parts of the ASP program excluding $\mathcal{M}$ and $t(IC)$. If $IC = \emptyset$ then $t(IC) = \emptyset$. In such case, for every answer set A_1 of Π_1, corresponding to hypothesis H of LT_1, there is an set $A_2 = A_1 \cup meta(H)$ which is an answer set of Π_2. Thus, LT_1 have at most as many hypotheses as LT_2. When $IC \neq \emptyset$, some of these hypotheses can be ruled out by the constraints, but

neither $\mathcal{M}$ nor IC can generate additional hypotheses. This is because both programs shares the same set of abducibles $A^{\top}$ thus can only generate at most the same number of hypotheses, and limited by the same constraint from examples ϵ. □

The next proposition shows that the ASP encoding of an hypothesis from a given constraint biased search space satisfies the ASP encoding of the given domain-dependent constraints. While the following proposition if for ASPAL_c the same reasoning is applicable for RASPAL_c as satisfying a domain dependent constraint in ASPAL_c requires that the corresponding ASP constraints be satisfiable, meaning that the body of the constraint is not satisfiable, where for RASPAL_c the corresponding cdb_ic_i must be unsatisfied which also requires for the same body to not be satisfied.

Proposition 2. *Given a learning task with constraint-driven bias $\langle B, M, E, IC\rangle$, let ϵ be the integrity constraint constructed from its examples. Let $\Pi = B \cup \top_M \cup \epsilon \cup t(IC)$ be the ASP program in ASPAL_c abduction step with abducibles $A^{\top}$, and $h(S) = \{enc^{-1}(s) | s \in S\}$. For each solution H of the learning task, there exists a corresponding answer set A of Π such that $\Delta \cup meta(H) = A \setminus B \setminus \top \setminus \epsilon \setminus t(IC)$ and $h(\Delta) = H$.*

Proof. There are two conditions H must satisfy if it is to be a solution of $\langle B, M, E, IC\rangle$:

1. $B \cup H \vDash_{\mathcal{B}} E$
2. $H \in [\mathcal{P}(R_M)]_{IC}$

The first condition is given by Proposition 1 as H must also be a solution of $\langle B, M, E\rangle$. By Definition 5.3 for the second condition to be satisfied, then it is required that $H \in [\mathcal{P}(R_M)]_{ic}$ for all $ic \in IC$ and for each meta-constraint $ic = \bot \leftarrow C, \overline{C_B}$. As defined in Definition 5.2, $H \in [\mathcal{P}(R_M)]_{ic}$ holds if:

1. $B \cup H \not\models \overline{C_B}$
2. The constraint $\bot \leftarrow C$ holds

In Table 6.1, $\overline{C_B}$ is always conjoined to the transformed constraint in denial form, thus for all answer set A of Π, $\overline{C_B} \notin A$ which satisfies the first condition. For the second condition we'll need consider the different ways that $\bot \leftarrow C$ can be satisfied. Note that we will use rid, rid_1 and rid_2 to indicate some grounded rule identification, i, i_1 and i_2 for some ground body literal index, and will write $l_h(\overline{C_h}) = \lambda(h)$ to indicate that for some θ, $\theta l_h(\overline{C_h}) = \lambda(h)$.

- $C = not\ in\_some\_rule(l_h(\overline{C_h}), l_b(\overline{C_b}))$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if there exists $r \in H$ such that $l_h(\overline{C_h}) = \lambda(head(r))$ and $b \in body(r)$ and $\lambda(b) = l_b(\overline{C_b})$. Consider $t(C)$, for A to be an answer set of Π then $in\_rule(rid, l_h(\overline{C_h}), l_b(\overline{C_b}), i)$ must be in A for some rid and i. By Lemma 1, it must be the case that $h(\Delta)$ has at least a rule with head literal h, and contains body literal b such that $l_h(\overline{C_h}) = \lambda(h)$ and $l_b(\overline{C_b}) = \lambda(b)$.

- $C = not\ in\_all\_rule(l_h(\overline{C_h}), l_b(\overline{C_b}))$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if either there is no rule $r \in H$ such that $l_h(\overline{C_h}) = \lambda(head(r))$ or for all $r \in H$ such that $l_h(\overline{C_h}) = \lambda(head(r))$ it is the case that $b \in body(r)$ and $\lambda(b) = l_b(\overline{C_b})$. Consider $t(C)$, then if $is\_rule(rid, l_h(\overline{C_h})) \in A$ then it must be true that $in\_rule(rid, l_h(\overline{C_h}), l_b(\overline{C_b}), i) \in A$. By Lemma 1, it must be the case that if $h(\Delta)$ has at a rule with head literal h such that $l_h(\overline{C_h}) = \lambda(h)$ then at least one of its body literal is b such that $l_b(\overline{C_b}) = \lambda(b)$.

- $C = not\ in\_same\_rule(l_h(\overline{C_h}), l_{b_1}(\overline{C_{b_1}}), l_{b_2}(\overline{C_{b_2}}))$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if either there are no $r \in H$ with $l_h(\overline{C_h}) = \lambda(head(r))$ and $b_1 \in body(r)$ and $l_{b_1}(\overline{C_{b_1}}) = \lambda(b_1)$ nor $b_2 \in body(r)$ and $l_{b_2}(\overline{C_{b_2}}) = \lambda(b_2)$, or for all $r \in H$ such that $l_h(\overline{C_h}) = \lambda(head(r))$ if $b_1 \in body(r)$ and $l_{b_1}(\overline{C_{b_1}}) = \lambda(b_1)$ then $b_2 \in body(r)$ and $l_{b_2}(\overline{C_{b_2}}) = \lambda(b_2)$, and vice versa. The translation $t(C)$ ensures that if $in\_rule(rid, l_h(\overline{C_h}), l_{b_1}(\overline{C_{b_1}}), i_1) \in A$, then $in\_rule(rid, l_h(\overline{C_h}), l_{b_2}(\overline{C_{b_2}}), i_2) \in A$ and vice versa. This means that by Lemma 1, if $h(\Delta)$ has a rule with head literal h such that $l_h(\overline{C_h}) = \lambda(h)$ and has body literal b_1 where $l_{b_1}(\overline{C_{b_1}}) = \lambda(b_1)$, then it must also has another body literal b_2 such that $l_{b_2}(\overline{C_{b_2}}) = \lambda(b_2)$, and similarly for the reverse. The translation is also satisfiable by having neither any $in\_rule(rid, l_h(\overline{C_h}), l_{b_1}(\overline{C_{b_1}}), i_1)$, nor $in\_rule(rid, l_h(\overline{C_h}), l_{b_2}(\overline{C_{b_2}}), i_2)$ in A, covering the case when neither body literals are in the hypothesis.

- $C = not\ in\_diff\_rule(l_{h_1}(\overline{C_{h_1}}), l_{b_1}(\overline{C_{b_1}}), l_{h_2}(\overline{C_{h_2}}), l_{b_2}(\overline{C_{b_2}}))$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if

and only if there exists $r_1 \in H$ such that $l_{h_1}(\overline{C_{h_1}}) = \lambda(head(r_1))$ and $b_1(\overline{C_{b_1}}) \in body(r_1)$ where $\lambda(b_1) = l_{b_1}(\overline{C_{b_1}})$, and there must also exists $r_2 \in H$ such that $r_1 \neq r_2$, $l_{h_2}(\overline{C_{h_2}}) = \lambda(head(r_2))$ and $b_2(\overline{C_{b_2}}) \in body(r_2)$ where $\lambda(b_2) = l_{b_2}(\overline{C_{b_2}})$. For the ASP translation, the constraint is satisfiable if $ic\_cond \in A$, and this can only be the case if $in\_rule(rid_1, l_{h_1}(\overline{C_{h_1}}), l_{b_1}(\overline{C_{b_1}}), i_1) \in A$ and $in\_rule(rid_2, l_{h_2}(\overline{C_{h_2}}), l_{b_2}(\overline{C_{b_2}}), i_2) \in A$. By Lemma 1, $h(\Delta)$ has a rule with head literal h_1, such that $l_{h_1}(\overline{C_{h_1}}) = \lambda(h_1)$, and has a body literal labelled l_{b_1}, such that $l_{b_1}(\overline{C_{b_1}}) = \lambda(b_1)$, as well as another rule with head literal h_2, such that $l_{h_2}(\overline{C_{h_2}}) = \lambda(h_2)$, and body literal b_2, such that $l_{b_2}(\overline{C_{b_2}}) = \lambda(b_2)$.

- $C = not\ max\_body(l_h(\overline{C_h}), m)$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if for all $r \in H$ if $l_h(\overline{C_h}) = \lambda(head(r))$ then $|body(r)| \leq m$. Consider $t(C)$, the number of grounded $in\_rule(Rid, l_h(\overline{C_h}), L_b, I)$ atom in A cannot be greater than $m + 1$. Thus by Lemma 1, all rules in $h(\Delta)$ with head literal h and $l_h(\overline{C_h}) = \lambda(h)$ cannot have more than m number of body literals.

- $C = not\ min\_body(l_h(\overline{C_h}), m)$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if for all $r \in H$ if $l_h(\overline{C_h}) = \lambda(head(r))$ then $|body(r)| \geq m$. Consider $t(C)$, the number of grounded $in\_rule(Rid, l_h(\overline{C_h}), L_b, I)$ atom in A cannot be from 0 to $m - 1$. Thus by Lemma 1, all rules in $h(\Delta)$ with head h and $l_h(\overline{C_h}) = \lambda(h)$ must have at least m number of body literals.

- $C = not\ max\_head(l_h(\overline{C_h}), m)$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if for $R = \{r | r \in H \text{ and } l_h(\overline{C_h}) = \lambda(head(r))\}$ then $|R| \leq m$. Consider $t(C)$, the number of grounded $in\_rule(Rid, l_h(\overline{C_h}), L_b, I)$ atoms in A cannot be greater than $x + 1$. By Lemma 1, all the number of rules in $h(\Delta)$ with head h and $l_h(\overline{C_h}) = \lambda(h)$ cannot be over m.

- $C = not\ min\_head(l_h(\overline{C_h}), m)$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if for $R = \{r | r \in H \text{ and } l_h(\overline{C_h}) = \lambda(head(r))\}$ then $|R| \geq m$. Consider $t(C)$, the number of grounded $in\_rule(Rid, l_h(\overline{C_h}), L_b, I)$ atoms in A cannot be $m - 1$ or fewer. By Lemma 1, there is at least m number of number of rules in $h(\Delta)$ with head literal h such that $l_h(\overline{C_h}) = \lambda(h)$.

- $C = in\_some\_rule(l_h(\overline{C_h}), l_b(\overline{C_b}))$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if there does not exists $r \in H$ such that $l_h(\overline{C_h}) = \lambda(head(r))$ and $b \in body(R)$ and $\lambda(b) = l_b(\overline{C_b})$. For the

translation $t(C)$ to satisfy the constraint it must be that $ic\_cond \notin A$, and consequently for all Rid and I, $in\_rule(Rid, l_h(\overline{C_h}), l_b(\overline{C_b}), I) \notin A$. By Lemma 1, $h(\Delta)$ does not have any rule with head h and contains body literal b such that $l_h(\overline{C_h}) = \lambda(h)$ and $l_b(\overline{C_b}) = \lambda(b)$.

- $C = in\_all\_rule(l_h(\overline{C_h}), l_b(\overline{C_b}))$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if there exists $r \in H$ such that $l_h(\overline{C_h}) = \lambda(head(r))$ and for all $b \notin body(r)$, $\lambda(b) \neq l_b(\overline{C_b})$. For the translation $t(C)$, to satisfy the constraint ic_cond_1 must be in A, which means that there exists $in\_rule(Rid, l_h(\overline{C_h})) \in A$ such that $ic\_cond_2(Rid, l_h(\overline{C_h})) \notin A$ and consequently $in\_rule(Rid, l_h(\overline{C_h}), l_b(\overline{C_b}), I) \notin A$ for all I. By Lemma 1, $h(\Delta)$ has a rule with head literal h that does not have any body literal b such that $l_h(\overline{C_h}) = \lambda(h)$ and $l_b(\overline{C_b}) = \lambda(b)$.

- $C = in\_same\_rule(l_h(\overline{C_h}), l_{b_1}(\overline{C_{b_1}}), l_{b_2}(\overline{C_{b_2}}))$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if there exists $r \in H$ with $l_h(\overline{C_h}) = head(r)$ that does not have both $b_1 \in body(r)$ such that $\lambda(b_1) = l_{b_1}(\overline{C_{b_1}})$ and $b_2 \in body(r)$ such that $\lambda(b_2) = l_{b_2}(\overline{C_{b_2}})$. Consider $t(C)$, both $ic\_cond_1(Rid, l_h(\overline{C_h}))$ and $iccond_2(Rid, l_h(\overline{C_h}))$ must not be in A for all Rid. This can only be the case if both $in\_rule(Rid, l_h(\overline{C_h}), l_{b_1}(\overline{C_{b_1}}), I_1) \notin A$ and $in\_rule(Rid, l_h(\overline{C_h}), l_{b_2}(\overline{C_{b_2}}), I_1) \notin A$ for all Rid, I_1 and I_2. By Lemma 1, $h(\Delta)$ cannot have a rule with head literal h, such that labelled $l_h(\overline{C_h}) = \lambda(h)$, which have body literals b_1 and b_2 such that $l_{b_1}(\overline{C_{b_1}}) = \lambda(b_1)$ and $l_{b_2}(\overline{C_{b_2}}) = \lambda(b_2)$.

- $C = in\_diff\_rule(l_{h_1}(\overline{C_{h_1}}), l_{b_1}(\overline{C_{b_1}}), l_{h_2}(\overline{C_{h_2}}), l_{b_2}(\overline{C_{b_2}}))$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if for all $r_1 \in H$ such that $l_{h_1}(\overline{C_{h_1}}) = \lambda(head(r_1))$ and $b_1 \in body(r_1)$ where $l_{b_1}(\overline{C_{b_1}}) = \lambda(b_1)$, does not exist $r_2 \in H$ such that $r_1 \neq r_2$, $l_{h_2}(\overline{C_{h_2}}) = \lambda(head(r_2))$, and $b_2 \in body(r_2)$ where $l_{b_2}(\overline{C_{b_2}}) = \lambda(b_2)$. From the translation $t(C)$, for all different Rid_1 and Rid_2, $ic\_cond_1(Rid_1) \notin A$ and $ic\_cond_2(Rid_1) \notin A$. This can only be true if both $in\_rule(Rid_1, l_{h_1}(\overline{C_{h_1}}), l_{b_1}(\overline{C_{b_1}}), I_1)$ and $in\_rule(Rid_2, l_{h_2}(\overline{C_{h_2}}), l_{b_2}(\overline{C_{b_2}}), I_2)$ are not in A for all I_1 and I_2. By Lemma 1, $h(\Delta)$ cannot have a pair of different rules such that one has the head literal h_1 and body literal b_1, such that $l_{h_1}(\overline{C_{h_1}}) = \lambda(h_1)$ and $l_{b_1}(\overline{C_{b_1}}) = \lambda(b_1)$, and the other has head literal h_2 and body literal b_2 such that $l_{h_2}(\overline{C_{h_2}}) = \lambda(h_2)$ and $l_{b_2}(\overline{C_{b_2}}) = \lambda(b_2)$.

- $C = max\_body(l_h(\overline{C_h}), m)$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if for all $r \in H$ if $l_h(\overline{C_h}) = \lambda(head(r))$ then $|body(r)| > m$. Consider $t(C)$, to satisfy the constraint ic_cond

must be in A. There are at least $m+1$ number of grounded $in\_rule(Rid, l_h(\overline{C_h}), L_b, I)$ atom in A. By Lemma 1, $h(\Delta)$ contains a rule with head literal h and $l_h(\overline{C_h}) = \lambda(h)$ with more than m number of body literals.

- $C = min\_body(l_h(\overline{C_h}), m)$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if for all $r \in H$ if $l_h(\overline{C_h}) = head(r)$ then $|body(r)| < m$. Consider $t(C)$, to satisfy the constraint ic_cond must be in A. There are $m-1$ or fewer grounded $in\_rule(Rid, l_h(\overline{C_h}), L_b, I)$ atom A. By Lemma 1, $h(\Delta)$ contains has a rule with head literal h and $l_h(\overline{C_h}) = \lambda(h)$ with less than m number of body literals.

- $C = max\_head(l_h(\overline{C_h}), m)$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if for $R = \{r | r \in H \text{ and } l_h(\overline{C_h}) = head(r)\}$ then $|R| > m$. Consider $t(C)$, to satisfy the constraint ic_cond must be in A. There are at least $m+1$ number of grounded $is\_rule(Rid, l_h(\overline{C_h}))$ atom in A. By Lemma 1, $h(\Delta)$ contains more than m number of rules with head literal h such that $l_h(\overline{C_h}) = \lambda(h)$.

- $C = min\_head(l_h(\overline{C_h}), m)$: by Definition 5.2 $H \in [\mathcal{P}(R_M)]_{ic}$ if and only if for $R = \{r | r \in H \text{ and } l_h(\overline{C_h}) = head(r)\}$ then $|R| < m$. Consider $t(C)$, to satisfy the constraint ic_cond must be in A. For this to be the case, there must be $m-1$ or fewer grounded $is\_rule(Rid, l_h(\overline{C_h}))$ atom in A. By Lemma 1, $h(\Delta)$ contains less than m number of rules with head literal h and $l_h(\overline{C_h}) = \lambda(h)$.

□

This concludes how constraint-driven bias can be implemented in ASPAL and RASPAL. In Section 7.2 we will describe some examples of learning tasks, and provide the solution learnt using constraint-driven bias.

Chapter 7

Application and Evaluation

This chapter presents the evaluation and application of RASPAL and learning with constraint-driven bias. The RASPAL evaluation is shown in terms of its performance with respect to HYPER, another ILP system that learns by hypothesis refinement, and ASPAL, the ILP system whose scalability issue was the motivation of RASPAL. All three systems are compared using learning tasks that highlight the differences between them. In addition, ASPAL and RASPAL are compared on a task for learning a user's behaviour on a mobile phone from data collected from a real user. The mobile example is constructed from data gathered by our research group as the ILP community currently lacks datasets of non-monotonic learning tasks. Two versions of this task are used; the first is a learning task without noise to show the difference in the maximum size of the learning programs of ASPAL and RASPAL, the second is a learning task to show how ASPAL and RASPAL react to noise in the examples. For this version, ASPAL has been modified so that noise threshold can be used as part of its terminating condition. In addition to this, we have also evaluated the performance of ASPAL and RASPAL with respect to two different ASP systems in order to show that the scalability problem is independent to the ASP systems' implementation, making it essential that there are other directions for making learning systems more scalable in ASP.

The second part of this chapter shows applications for learning with constraint-driven bias. The first of these applies constraint-driven bias to learning stratified programs, and shows how the

domain-dependent constraints can help reduce the number of solutions to a such a learning task. The second learning task tackles an open problem in software engineering, namely revision of a goal model. This application shows that minimality of the scoring function, when computing a revised model, is not sufficient for learning the desired solutions. Thus domain-dependent constraints can be used as additional criteria for the solution.

Note that the full description of the learning tasks used in this chapter can be found in Appendix A.

7.1 RASPAL evaluation

Comparison against ASPAL and HYPER on Simple Examples

We first present the comparison of RASPAL against ASPAL and HYPER [Bra99] using simple synthesised examples. ASPAL has been chosen as the scalability issue in ASP based systems was the motivation for RASPAL. The objective is to confirm that RASPAL does manage to solve learning tasks that are unsolvable by ASPAL. HYPER has been chosen for comparison as, like RASPAL, it uses hypothesis refinement as part if its learning algorithm.

As mentioned in Section 3.3, HYPER is a top down ILP system that learns through hypothesis refinement. However, as it is a monotonic system the learning tasks that we have used for this comparison neither have negation in the background knowledge nor in the solution in order to ensure that they are expressible in all three systems. For the same reason, all learning tasks did not include list structure. Note that all experiments were run on a 3.40GHz Intel Core i7 PC. The HYPER code was run on Sicstus Prolog. The HYPER implementation used was from Chapter 19 of [Bra01] and can be found at `http://www.pearsoned.co.uk/highereducation/resources/bratkoprologprogrammingforartificialintelligence3e/`. The weights used in HYPER's scoring were $k_1 = w_2 = 10$ and $w_1 = k_2 = 1$.

Table 7.1 shows the time taken by the three systems to find the solution of each task. For HYPER and RASPAL the number of refinements taken to find the solution was recorded. The

ILP Task	System	Time (s)	Refinements	Rules generated
mother	HYPER	0	2	20
	RASPAL	8×10^{-3}	2	1, 18, 19
	ASPAL	6×10^{-3}		37
nonealike	HYPER	0.610	168	9325
	RASPAL	9.142	4	1, 102, 103, 104, 105
	ASPAL	–	–	–
highroll	HYPER	–	–	–
	RASPAL	3.6×10^{-2}	1	1, 10, 5, 38
	ASPAL	1.4×10^{-2}		19

Table 7.1: Some experimental results of running HYPER, ASPAL and RASPAL.

last column of the table records the number of rules generated by each system. For HYPER this is the number of rules it has generated while searching for the solution, and for RASPAL and ASPAL this is the number of rule templates in the top theory. Note that for RASPAL the number of rules per refinement step is also recorded. This number increases during iterative revisions within the same value of i, and when i is incremented by RASPAL the number will decrease as the Algorithm 1 is restarted. For some tasks such as the *highroll* learning task, RASPAL has longer computation time than ASPAL as it has the additional overhead from the revision. This includes the extra revisable theory within the background knowledge and the additional mode declarations for the revision operators, which in turn cause an increase in the rule space. This explains the high increase in the number of rules in RASPAL between the first two iterations. However, further iterations, such as in the *mother* and *nonealike* learning tasks, often have much smaller increase in rules once the mode declarations for expansions have already been added and i has not been increased.

In the *mother* learning task, the number of rules in RASPAL is smaller than the number in ASPAL. For the *nonealike* experiment, where the expected solution is the rule $nonealike(A, B, C, D, E) \leftarrow diff(A, B), diff(B, C), diff(C, D), diff(D, E)$ the ASPAL search space becomes very big as there are many ways to link the variables using the $diff/2$ predicate. ASPAL eventually hits the memory bound while grounding the program, and no solution is computed. RASPAL (executed with $i = 1$) avoids this problem as the top theory is more compact as learned changes

never have more than one body literal. As for HYPER, many more refinements are computed than RASPAL. This is due to HYPER's scoring function. For instance, if the number of literals and negative examples are weighted equally, a solution that covers one negative example and has three literals has the same score as another that covers three negative examples and has one literal only, assuming they have the same number of variables. Our score schema separates these components avoiding such problems.

In the *highroll* learning task, HYPER cannot learn the solution. This is due to HYPER's refinement operator only adding one body literal per each refinement step. The solution to this learning task is $highroll(A, B) \leftarrow add(A, B, C), greaterThan(C, 7)$, which requires both $add(A, B, C)$ and $greaterThan(C, 7)$ to be added at the same time in order to reduce the negative examples coverage. As adding just one of $add(A, B, C)$ or $greaterThan(C, 7)$ will increase the size of the hypothesis without improving its score, HYPER rejects such a solution a priori.

The results of the three learning tasks show that although RASPAL sometimes takes longer to compute a solution compared to the other two systems, it is the only system that is capable of solving all three learning tasks.

Comparison against ASPAL on real example

Next we compare RASPAL against ASPAL using a real life learning task larger than the toy learning tasks in the previous section. In this task, a mobile user's past behaviours were recorded by an application installed on their phone. The aim of the learning task is to use the recorded behaviour to learn the circumstances in which the user will accept an incoming call. Calls that are accepted have a minimal duration otherwise they are assumed to be rejected by the user. Noise therefore occur in the way calls have been grouped between accepted (positive examples) and rejected (negative examples).

The recorded information includes the time and date a call was received, the length of time the user talked to the caller, the application used by the user when the call was received, and

System	Max program size	Max grounding size
ASPAL ($d = 3$)	540kB	1580MB
RASPAL ($i = 1$)	35kB	296MB

Table 7.2: Maximum size of the grounded programs when learning the task using ASPAL and RASPAL.

information on the status of the phone such as the battery level or whether the screen was on and active. With the large amount of information gathered, there are many possible mode declarations that could be used to learn the concept, amounting to large hypothesis space. The meta-theory of ASPAL easily becomes so large that the solver could not successfully solve the task before it runs out of memory.

The learning task that we used included information of a user's mobile usage pattern from one week of his or her activity. For the first part of this comparison some examples were omitted so that the learning task is without noise. The following is an example solution of the learning task:

$$H = \left\{ \begin{array}{l} accept(A,B,C,D,E,F,G,H,I) \leftarrow \\ \quad user\_is\_using\_app(A,B,googletalk), not\ is\_charging(I), \\ \quad screen\_on(G). \\ accept(A,B,C,D,E,F,G,H,I) \leftarrow \\ \quad high\_battery(F), afternoon(B). \\ accept(A,B,C,D,E,F,G,H,I) \leftarrow \\ \quad user\_is\_using\_app(A,B,googletalk), high\_volume(D), \\ \quad not\ low\_battery(F). \end{array} \right\}$$

The maximum size of the grounded programs produced by Gringo (the grounder used in Clingo) using the ASP program generated by ASPAL and RASPAL is shown in the Table 7.2. ASPAL's grounded program is roughly five times of that of RASPAL, as should be expected when RASPAL can solve the task with i limited to 1 while ASPAL will require all rules with lengths matching those in the solution to be included within its meta theory.

Figure 7.1 shows the maximum size of the grounded program produced by ASPAL and RASPAL

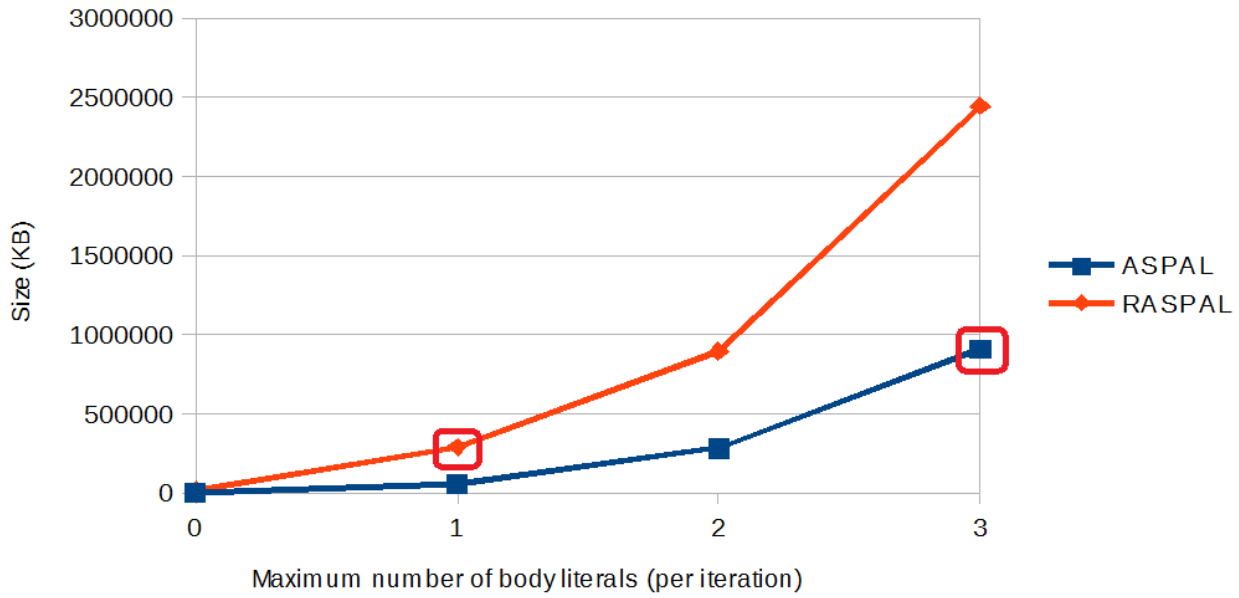

Figure 7.1: Maximum grounding size of ASP programs produced by ASPAL and RASPAL. The squares around the two points in the graph indicate the actual number of body literals required to solve the task by each system.

respectively for the noiseless mobile example with different maximum number of body literals in their top theories. As we have previously mentioned, RASPAL has the additional overhead of mode declarations for its refinement operators. While the graph might suggest that RASPAL produces larger ASP programs to solve the same task, that is not the case. Recall that for this example the task can be solved by RASPAL with just 1 body literal, while ASPAL would require 3 body literals. The advantage of RASPAL is that even if ASPAL need to use a high number of body literals to solve a learning task, RASPAL can solve those tasks with much lower number of body literals.

As the upper bound of the size of the top theory (Equation 4) is more affected by the number of body literals as opposed to the number of mode declarations. RASPAL's theory refinement has led to an increase in the size of top theory due to the increase in the head mode declarations, this is a linear increase. However, what RASPAL achieved by using theory refinement is reducing the increase in the number of body literals, which is responsible for the exponential increase in the size of the top theory.

To apply RASPAL and ASPAL to the learning task without any examples omitted we have modified ASPAL's termination condition so that the noise constraint is included. Recall that ASPAL defines a constraint over the examples to ensure that the abduced solutions cover all the examples. We want to relax this constraint so that the learner can also abduce hypotheses within the noise threshold. This can be achived by replacing its constraint from the examples, $\{\bot \leftarrow not\ examples.; examples \leftarrow \bigwedge e \in E.\}$, with two constraints constructed from the noise threshold $\langle e^+_{min}, e^-_{max}\rangle$, and negative examples E:

$$\bot \leftarrow 0\ \{e \mid e\ \in E\}\ e^+_{min} - 1.$$

$$\bot \leftarrow e^-_{max} + 1\ \{e \mid not\ e \in E\}\ |\{e \mid not\ e \in E\}|.$$

These constraints use the ASP choice rule to specify that the solution to the task cannot cover less than e^+_{min} number of positive examples, and that the number of negative examples covered cannot be more than e^-_{max}.

The mode declarations for the learning task were also changed so that the learner is not learning general rules for all calls, but instead will learn different rules for each different caller. The following is an example of a solution for the learning task where the third argument of each rule is an identification number of an incoming number.

$$H = \left\{ \begin{array}{l} accept(A,B,3,C,D,E,F,G,H) \leftarrow low\_battery(E), screen\_on(F). \\ accept(A,B,4,C,D,E,F,G,H) \leftarrow not\ low\_battery(E). \\ accept(A,B,8,C,D,E,F,G,H). \end{array} \right\}$$

We have conducted a leave-one-out cross validation for both ASPAL and RASPAL, with the cross-validation performed for various different noise thresholds: 10%, 30%, 60% and 80% positive examples coverage, which correspond respectively to different percentage values of e^+_{min}, and 20%, 40%, 60% and 80% negative examples coverage, which correspond to different percentage values of e^-_{max}. For RASPAL we have also considered the cases of 0% and 100% for negative example and 100% for positive examples coverage.

For each noise threshold and for each system we have calculated the *sensitivity* and *specificity* of the learner. The *sensitivity* is the number of correctly classified positive examples divided by

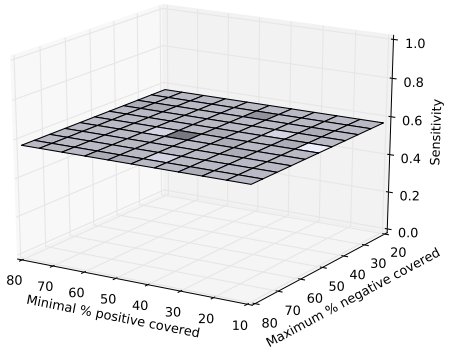

Figure 7.2: ASPAL sensitivity

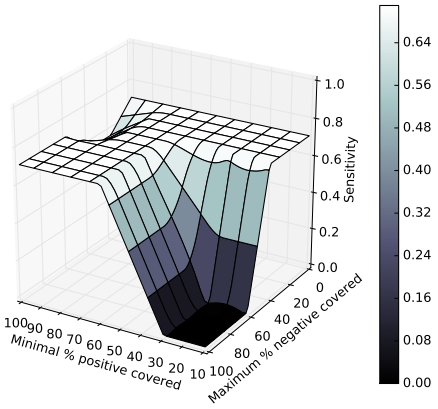

Figure 7.3: RASPAL sensitivity

the total number of positive examples, and the *specificity* is the number of correctly classified negative examples divided by the total number of negative examples. Figures 7.2 and 7.3 show the sensitivity of ASPAL and RASPAL. Figure 7.2 shows that changing the noise threshold has no effect to the sensitivity of ASPAL, as no positive examples were noisy, so by prioritising the coverage of positive examples ASPAL was able to find a solution that cover all of them for all noise thresholds. Figure 7.3 shows that RASPAL's sensitivity increases as the percentage value of e^+_{min} increases (more positive examples covered), or as the e^-_{max} decreases. This is because RASPAL performes iterations of learning, with limited rule length at each iteration, and will iteratively improve a low scoring partial hypothesis, bringing it closer to the desired threshold at each iteration. With low positive and high negative examples coverage it can return the smallest rule that satisfies the noise threshold. In this most extreme case the solution may be a single rule with a head but no body literals, so that any positive examples needing a different head literal would be wrongly classified (see lowest dark corner of the graph in Figure 7.3). Higher positive examples coverage (higher e^+_{min}) will often lead to an increase in the number of iterations needed by the algorithm to find the optimal solution, thus increasing the sensitivity of the solutions learned. Similarly, lowering the negative examples coverage (lower e^-_{max}), directs the learner to compute solutions that cover less negative examples, and therefore may cause further refinements of partial hypothesis, thus increasing the number of iterations and the sensitivity of the optimal solution. Figures 7.4 and 7.5 show the specificity of ASPAL and RASPAL. In both cases, the specificity decreases as the threshold on the positive examples

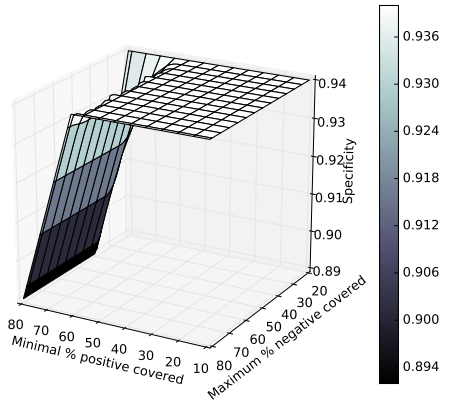

Figure 7.4: ASPAL specificity

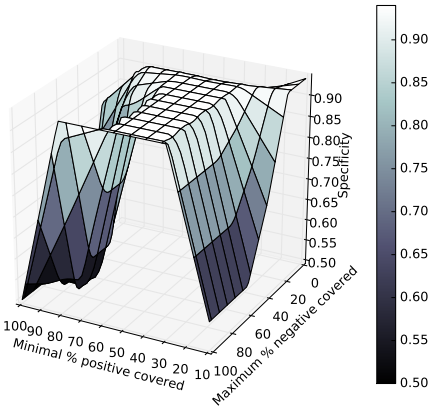

Figure 7.5: RASPAL specificity

coverage increases. This is to be expected due to the learner trying to over fit the solution to the positive examples. In the case of RASPAL, the specificity decreases also at the end of small value of e^+_{min} (number of positive examples required to be covered is small). This is because, due to the low threshold, the learner can return the hypothesis found in the early iteration without having to refine it, thus generating solutions with low scores and specificity. To this end, ASPAL's specificity does not change for the same reason as pointed out above for constant sensitivity value.

Lastly, we have selected some of the toy examples and the mobile example to compare the largest ASP program produced by RASPAL and ASPAL for each task when run on two different ASP solvers Clingo [GKK+11] and DLV [LPF+06]. This has been done to check the compatibility between the solver and the RASPAL top theory, and to show that the grounding problem of the learning program is not exclusive to a specific solver.

Learning Task	ASPAL			RASPAL		
	DLV	Clingo	$\lvert\top\rvert$	DLV	Clingo	$\lvert\top\rvert + r(H)$
odd/even	17.0kB	11.3kB	23	172.9kB	159.2kB	65+5=70
nonealike	-	-	251176	35.7MB	40.5MB	105+10=115
train	932.8kB	518.0kB	118	126.4MB	131.0MB	238+20=258
mobile	-	11.2GB	1200	-	1.9GB	85+18=103

Table 7.3: Maximum size of the ground program when solving a problem using RASPAL and ASPAL, and the number of ungrounded clauses in their top theories. For RASPAL $r(H)$ is the number of clauses in the revisable hypothesis added to the background.

ASPAL and RASPAL already generate ASP programs compatible to Clingo, and the DLV compatible version has been created by modifying these programs by: (i) replacing Clingo's aggregate rules by DLV's disjunctive rules that allows for more than one of the disjunctive element to be true as described in Section 2.5.2; (ii) using aggregate functions instead of the limits on a choice rule for limiting the number of rules in the solution, and defining the constraint on the abducible *delete*/2 literals; and (iii) replacing optimisation statements over examples coverage by soft constraints. While both solvers return the same answer set, when solving a learning task by RASPAL on DLV, additional post-processing is required to find the most optimal partial hypothesis. To compare the sizes of the ground programs in Table 7.3, their sizes were found by making the solvers ground the program but not solve it. Notice that for the *mobile* learning task, the size of the ground programs are noticeably larger than those in Table 7.2. This difference is due to DLV not having the opting for obtaining its internal representation of the grounded program, the results in Table 7.3 are the two systems' "readable" grounded programs (the option "`-instantiate`" for DLV, and "`-t`" for Clingo) as opposed to the internal representation in Table 7.2 from applying Clingo's grounder (Gringo 3 [GKKS11]) on the learning task.

Regarding DLV and Clingo, the results in Table 7.3 show that for smaller learning tasks there is not much difference in the size of the ground programs produced by the solvers. When Clingo cannot ground the program for the *nonealike* task, neither could DLV. Moreover for large problems that can still be ground by Clingo such as the *mobile* task, DLV can neither solve nor ground the ASP programs. For this reason and easier processing due to the optimisation statements in Clingo, we find Clingo to be more suitable for solving our ASP programs.

Regarding the number of ungrounded clauses in the top theory of each learning task, note that for RASPAL the value $r(H)$ denotes the number of clauses of the revisable hypothesis added to the background knowledge. For the learning tasks *odd/even* and *train* both the number of clauses in the top theory and the size of the ground programs of RASPAL are much larger than those of ASPAL. This is because RASPAL has an overhead from the revisable theory and the additional mode declarations used to learn the revision operations, making it less efficient than ASPAL for solving smaller learning tasks. For the *nonealike* learning task, the top theory of

ASPAL is very large as there are many permutations of the variables in the learnable clauses. Similarly, the *mobile* learning tasks also have large ASPAL's top theories and have much large domain knowledge than other tasks. For both of these tasks RASPAL's hypothesis refinement can significantly reduce the size of the program's grounding. This reduction by RASPAL is due to the difference in the maximum clause size of the solution and the minimum value of i required to learn it. In the learning tasks *odd/even* and *train*, which have maximum clause sizes of, respectively, 3 and 4, the minimum values of i required to learn these tasks are 2 and 3 respectively. This is because their solutions' body literals are highly dependent on one another, thus smaller values for i cannot be used. This makes the overhead of refining the partial hypotheses greater than the advantage of reducing the top theory's maximum clause size. On the other hand, for the *nonealike* and *mobile* learning tasks with maximum clause sizes of 5 and 4 respectively, there are less dependencies between the body literals in their solutions, and for both tasks RASPAL was able to find a solution using $i = 1$. The bigger difference between i and maximum clause size allows the learning task to be solved by a much smaller top theory compared to ASPAL.

7.2 Constraint-Driven Bias

We will first demonstrate how constraint-driven bias can be used by applying it for learning stratified programs. This shows how additional meta-level information can be added to the learning task and incorporated into the constraints. The second learning task in this section is for learning revision of a goal model. This gives an example application for constraint-driven bias.

Stratified Programs

Stratified programs are a class of logic programs where restrictions are placed on using negated predicates recursively, thus ensuring that the program has a unique minimal model. Stratified programs are defined in [ABW88] as follows:

Definition 7.1 (Stratified Program)**.** *A program P is* stratified if there exists a partition $P = P_1, \ldots, P_n$ such that for $1 \leq i \leq n$ all the following conditions hold:

1. If a positive body literal p occurs in P_i then its definition, that is all of the body literals in clauses with head literal p, is contained within $\bigcup_{j \leq i} P_j$.
2. If a negative body literal q occurs in P_i then its definition is contained within $\bigcup_{j < i} P_j$.

P_1 can be empty.

Each of these partitions $P_1, \ldots, P_n$ is referred to as a *stratum* of P. For a predicate to be used as a body literal in a stratum it must either be defined in previous strata, or if used positively its definition can also be contained in the same stratum. Note that a literal without any definition can always be used in any strata.

The problem of learning stratified programs appears to be scarcely explored with [Che94] being the only work we know of. In [Che94] the full algorithm for learning stratified programs was not presented, and from the method described the stratified program is generated from a definite program, with negation introduced in order to minimise the size of the learnt solution. Using constraint-driven bias gives a more general method for learning stratified programs, with the constraints being based on the definition of stratified programs. We assume that the given learning task's background knowledge is stratified and we want to compute a solution such that the union of the solution and given background knowledge is stratified and proves the examples.

At the highest level the domain-dependent constraints for guiding the search to stratified programs are as follows:

$\bot \leftarrow in\_some\_rule(H, B), is\_positive(B), stratum(H, L), not\ defined(B, L).$

$\bot \leftarrow in\_some\_rule(H, B), is\_negative(B), stratum(H, L), not\ predefined(B, L).$

$\bot \leftarrow not\ min\_head(H, 1), stratum(H, L), L \neq 1.$

The first constraint checks that positively occurring body literals are defined within the same or lower stratum than the head stratum. The second constraint checks that negatively occurring

body literals are defined in a lower stratum than the head stratum, and the third constraint ensures that literals exclusively used in the body (i.e. are not used in any head) are in the lowest stratum (stratum 1). Auxiliary predicates can be defined to help reasoning about basic concepts related to stratification. These include facts about labels of positive literals using a predicate $is\_positive/1$, labels of negative literals using a predicate $is\_negative/1$, and facts for linking the pair of positive and negative labels of literals using the predicate $negative\_positive\_pair/2$. A notion of strata can be expressed using the ASP choice rule to ensure that each literal in the hypothesis is assigned a unique stratum; the predicate *level* asserts an integer limit on the number of strata allowed in a hypothesis, and the number of levels needed is at most equal to the number of predicates that can appear in the hypothesis:

$$1\{stratum(H, L) : level(L)\}1 \leftarrow is\_positive(H).$$

$$\bot \leftarrow stratum(H, L1), stratum(H, L2), L1 \neq L2.$$

Other key auxiliary predicates are defined below. Predicates $defined$ and $predefined$ are used to check that positive and negative body literals are appropriately defined in a lower or equal stratum to where they are used:

$$defined(B, L1) \leftarrow level(L1), stratum(B, L2), L2 \leq L1.$$

$$predefined(NegB, L1) \leftarrow level(L1), negative\_positive\_pair(NegB, B),$$
$$stratum(B, L2), L2 < L1.$$

We have considered two learning tasks[1] of this kind. The first one has no examples given, and the background included only the definition of the auxiliary predicates, described above, and

[1]Clingo's option `--project` was used when running the ASP programs for solving the examples using the constraints to ensure that each solution is output only once regardless of the number of ways it could be stratified.

Task	Head declaration	Body declaration
1	modeh(p(+arg,+arg)). modeh(q(+arg,+arg)).	modeb(q(+arg,+arg)). modeb(not q(+arg,+arg)). modeb(p(+arg,+arg)). modeb(not p(+arg,+arg)).
2	modeh(p(+arg,+arg)). modeh(q(+arg,+arg)). modeh(a(+arg,+arg)).	modeb(q(+arg,+arg)). modeb(not q(+arg,+arg)). modeb(p(+arg,+arg)). modeb(not p(+arg,+arg)). modeb(a(+arg,+arg)). modeb(not a(+arg,+arg)).

Table 7.4: Mode declarations used in the two learning tasks for testing the stratification constraints. All tasks were run to find solutions with at most two clauses and maximum two body literals per clause.

the mode declaration were defined as shown in Table 7.4. Each learning task was solved for a different range of allowable values for variables of type *int*, from 1 to 5, 1 to 10, and 1 to 20.

Table 7.5 shows the number of unique solutions computed in each instance, with and without the constraints. In Table 7.5 are the recorded results of the case study[2]. For both tasks the time taken to find all solutions using the constraints is smaller than the learning task without constraints. We can conclude that for these experiments that domain-specific constraints help improving the efficiency of the task of learning stratified programs. More importantly, for the non-constrained tasks, the solutions produced are not all stratified, thus further parsing of the results is required to find only the stratified solutions.

The second type of problem we have considered is when we want to learn programs that the union of the program with an existing background knowledge will be stratified. The stratification constraints now have to apply to the union of the given background and a computed solution. To achieve this we extend the meta-level information in the background so that *is_positive*/2 and *is_negative*/2 include an additional argument to take into account how our labelling convention can produce more than one label per predicate if the same predicate can contain different number of constant arguments (from its scheme being used in multiple mode declarations). Our learning task only handles constraints with constant free rules, but the

[2]All tasks were run using the ASP solver Clingo 3 [GKK+11] on a 2.13GHz laptop computer with 4GB memory

Task	Range for `arg`	Without constraints		With constraints	
		No. of solutions	Time (s)	No. of solutions	Time (s)
1	1-5	25126	3.588	8614	1.716
	1-10		9.282		4.820
	1-15		34.788		14.960
2	1-5	302860	219.306	158482	198.433
	1-10		1494.942		1082.694
	1-15		4640.889		2346.723

Table 7.5: Number of solution learnt with and without using the constraints, and the time taken to learn all solutions for when the argument of the rules can be integer within the ranges 1 to 5, 1 to 10 and 1 to 15.

constraint is still applicable to rules with constants.

Consider the following example for learning the concept of even and odd.

$$B = \left\{ \begin{array}{l} even(0). \\ num(0). \\ num(s(0)). \\ num(s(s(0))). \\ num(s(s(s(0)))). \\ num(s(s(s(s(0))))). \\ num(s(s(s(s(s(0)))))). \\ succ(X, s(X)) \leftarrow \\ \quad num(X), num(s(X)). \end{array} \right\} \quad E = \left\{ \begin{array}{l} even(s(s(0))), \\ odd(s(s(s(0)))), \\ odd(s(s(s(s(s(0)))))) \\ not\ even(s(0)), \\ not\ even(s(s(s(0)))), \\ not\ odd(0), \\ not\ odd(s(s(s(s(0))))). \end{array} \right\}$$

$$M = \left\{ \begin{array}{ll} modeh(even(+num)). & modeb(not\ even(+num)). \\ modeh(odd(+num)). & modeb(not\ odd(+num)). \\ modeb(even(+num)). & modeb(succ(-num, +num)). \end{array} \right\}$$

This example already has a rule for $even(0)$ in the background knowledge. Suppose we can "mark" part of the background knowledge so that its meta-level information will also be taken into account when solving the learning task. Then meta-level information of this could be added to the learning task as $\{is\_rule(rid, even(0))\}$ for some unique rule identification number rid. However, this means that there are two labels for $even$, one with the constant argument and other one without it. To handle this the predicates $is\_positive/2$ and $is\_negative/2$ can be

extended as follows:

$$is\_positive(succ, succ(\$e)). \qquad is\_negative(not\_even, not\_even(\$e)).$$
$$is\_positive(even, even(\$e)). \qquad is\_negative(not\_odd, not\_odd(\$e)).$$
$$is\_positive(even, even(X)) \leftarrow num(X). \qquad is\_positive(odd, odd(\$e)).$$

The original domain-dependent constraints are in this case:

$$IC = \left\{ \begin{array}{l} \leftarrow in\_some\_rule(H, B\_label), is\_positive(B\_name, B\_label), \\ \quad stratum(H, L), not\ defined(B\_name, L). \\ \leftarrow in\_some\_rule(H, B\_label), is\_negative(B\_name, B\_label), \\ \quad stratum(H, L), not\ predefined(B\_name, L). \end{array} \right\}$$

Solving for solutions containing at most two rules and at most three body literals, a total of ten solutions can be found including the following:

$$\left\{ \begin{array}{l} odd(A) \leftarrow succ(B, A), succ(C, B), not\ even(C). \\ even(A) \leftarrow succ(B, A), succ(C, B), even(C). \end{array} \right\}$$

$$\left\{ \begin{array}{l} even(A) \leftarrow succ(B, A), succ(C, B), even(C). \\ odd(A) \leftarrow succ(B, A), even(B). \end{array} \right\}$$

$$\left\{ \begin{array}{l} even(A) \leftarrow succ(B, A), succ(C, B), even(C). \\ odd(A) \leftarrow not\ even(A). \end{array} \right\}$$

In order to find only the more succinct solutions, additional constraints could be used. For instance, the constraint $\bot \leftarrow not\ max\_body(odd, 2)$ states that rules defining the concept of $odd/1$ should have a maximum of two body literals. From the three solutions above, the constraint would rule out the following hypothesis.

$$\left\{ \begin{array}{l} odd(A) \leftarrow succ(B, A), succ(C, B), not\ even(C). \\ even(A) \leftarrow succ(B, A),\ succ(C, B), even(C). \end{array} \right\}$$

While the following solutions would remain acceptable by the constraint.

$$\left\{ \begin{array}{l} even(A) \leftarrow succ(B, A), succ(C, B), even(C). \\ odd(A) \leftarrow succ(B, A), even(B). \end{array} \right\}$$

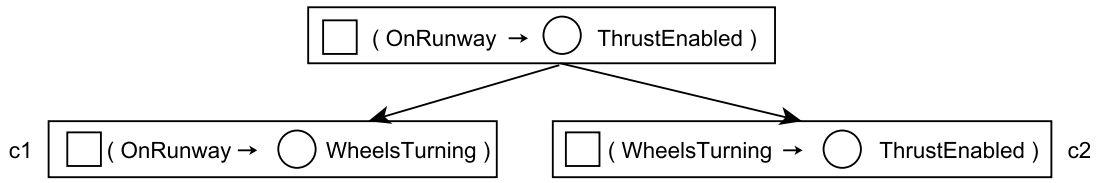

Figure 7.6: Goal model of Flight Control System.

$$\left\{ \begin{array}{l} even(A) \leftarrow succ(B, A), succ(C, B), even(C). \\ odd(A) \leftarrow not\ even(A). \end{array} \right\}$$

Requirements Engineering

For this learning task we show that while ILP systems typically use language bias and heuristics, such as minimal or most compressed solutions [Mug95, Bra99] to search for suitable hypotheses, this may not find the solution that the user desires. In such situations constraint-driven bias can be used to supply additional criteria to the learning task.

In the area of requirements engineering, goal models are directed acyclic graphs for representing links between objectives the software system is expected to meet. Each node in the graph is a goal which can be satisfied by satisfying all of its sub-goals (children nodes). This graph of goal patterns can be expressed in propositional temporal logic [MP92], and their refinement semantics are given in [DvL96]. For example, consider a simplified model of a Flight Control System (FCS):

Figure 7.6 shows how the FCS can be modelled expressing that at all time points if an aeroplane is on the runway then its engine's reverse thrust must be enabled at the next time point. This is then satisfiable by the sub-goal $c1$ that if the plane is on the runway then the plane's wheels must be turning which means that in the next instance its reverse thrust is enabled[3].

The problem with such a model is that if it is incorrect or too weak, it will need to be revised until a complete and correct specification is produced. Consider a case where the runway surface

[3]The symbol $\Box$ means always, $\bigcirc$ means at the next time point, $\rightarrow$ is for logical implication, and *OnRunway*, *ThrustEnabled* and *WheelsTurning* are propositional atoms.

is wet, the plane is moving on the runway, but the wheels are not turning, hence violating the sub-goal the wheels are always turning on the runway. The model shown in Figure 7.6 then needs to be revised to avoid any potential system failures. This is not a trivial task as changes made to one goal will potentially need to be propagated to other related sub-goals or the parent-goal. Using an Event Calculus formalism [KS86], similar to the one presented in [AKvL+12], the sub-goals in the model can be represented by the following theory where the predicates $a\_holds/3$ and $c\_holds/3$ represent the notion that the antecedent and consequent, respectively, of the goal holds at a time point in a scenario.

$a\_holds(c1, T, S) \leftarrow holds\_at(onRunway, T, S).$

$c\_holds(c1, T, S) \leftarrow holds\_at(wheelsTurning, T, S).$

$a\_holds(c2, T, S) \leftarrow holds\_at(wheelsTurning, T, S).$

$c\_holds(c2, T, S) \leftarrow holds\_at(thrustEnabled, T, S).$

Violating scenarios are included in the background knowledge as a series of facts. For example, the scenario s_1 where the aeroplane lands while it is raining, and switching on the wheels, enabling the reverse thrust in the next time instance would add the following facts to the background knowledge.

$$B = \left\{ \begin{array}{ll} happens(rain, 0, s1). & happens(landPlane, 1, s1). \\ happens(switchPulseOn, 1, s1). & happens(enableReverseThrust, 2, s1). \\ ... & \end{array} \right\}$$

The aim of the learning task in this case is to revise the goals by replacing all occurrences of $WheelsTurning$ with $WheelsPulseOn$. However, if ASPAL were used to solve this task without domain-dependent constraints, it would generate the following most compressed solution R'_1, which is also shown in Figure 7.7.

$$R'_1 = \left\{ \begin{array}{l} a\_holds(c1, T, S) \leftarrow holds\_at(onRunway, T, S). \\ c\_holds(c1, T, S). \\ a\_holds(c2, T, S) \leftarrow holds\_at(wheelsPulseOn, T, S). \\ c\_holds(c2, T, S) \leftarrow holds\_at(thrustEnabled, T, S). \end{array} \right\}$$

This does not preserve the refinement semantics of the goal model where the assertion which

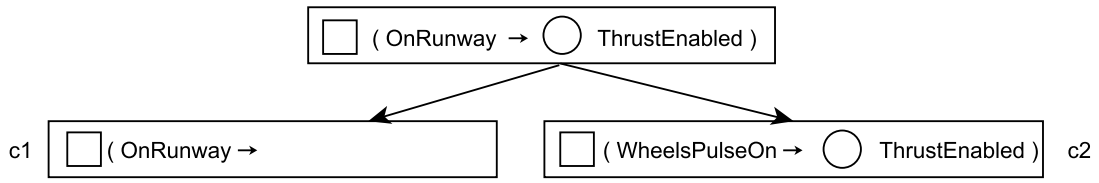

Figure 7.7: Revised model learnt from using ASPAL without constraint

appears in the antecedent of $c2$ must also appear in the consequent of $c1$, and that parent goal holds only if the conjunction of its children holds.

Constraint-driven bias can be applied to learn acceptable revisions of the goal models. An example constraint is that literals appearing in the body of a rule with head literal labelled $a\_holds(C)$ must also appear in the body of a rule with head literal label $c\_holds(C)$. The constraint is as follows.

$$IC = \left\{ \begin{array}{l} \bot \leftarrow in\_diff\_rule(a\_holds(C1), BL1, c\_holds(C2), BL2), \\ \quad right\_child\_of(G, C1), left\_child\_of(G, C2), BL1 \neq BL2, C1 \neq C2. \\ \bot \leftarrow in\_diff\_rule(c\_holds(C1), BL1, a\_holds(C2), BL2), \\ \quad left\_child\_of(G, C1), right\_child\_of(G, C2), BL1 \neq BL2, C1 \neq C2. \\ \bot \leftarrow not\ min\_body(a\_holds(C), 1). \\ \bot \leftarrow not\ min\_body(c\_holds(C), 1). \end{array} \right\}$$

The solution for the learning task with constraint is the following revised theory R'_2 and shown in Figure 7.8, which ensure that the change is propagated throughout the model.

$$R'_2 = \left\{ \begin{array}{l} a\_holds(c1, T, S) \leftarrow holds\_at(onRunway, T, S). \\ c\_holds(c1, T, S) \leftarrow holds\_at(wheelsPulseOn, T, S). \\ a\_holds(c2, T, S) \leftarrow holds\_at(wheelsPulseOn, T, S). \\ c\_holds(c2, T, S) \leftarrow holds\_at(thrustEnabled, T, S). \end{array} \right\}$$

The example have shown how constraint-driven bias can be applied for maintaining consistency of a model in the refinement process as syntactic constraints on the solution.

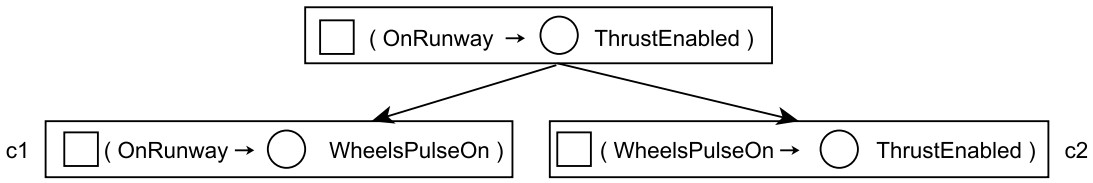

Figure 7.8: Revised model learnt from using ASPAL with constraint.

7.3 Discussion

The learning tasks which we have run shows that HYPER, ASPAL and RASPAL all have tasks which they are more suitable for than the other systems. HYPER is the fastest solver for learning tasks that it could solve, and the learner would apply higher number of alternative refinements compared to RASPAL. However, RASPAL can solve learning tasks that are unsolvable to HYPER. RASPAL's scoring scheme which has three different components can differentiate hypotheses more clearly than HYPER, whose scoring scheme is a single sum. This and the ability to backtrack by deleting rules and literals and add new rules and literal to its partial hypothesis makes it unnecessary for RASPAL to apply more refinements at each iteration.

For ASPAL and RASPAL, the learning tasks show that for smaller tasks ASPAL is the more suitable learner as it does not have RASPAL's overhead from the addition of the revisable partial hypothesis and mode declarations for the revision operators. However, RASPAL's learning algorithm does make a difference in some learning tasks as RASPAL could solve the *nonealike* learning task which ASPAL could not, and it significantly reduce the size of the grounded program for the *mobile* learning task. From this we can conclude that while ASPAL perform better for smaller leaning tasks, RASPAL is more suitable than ASPAL for larger learning tasks provided that they can be learnt by hypothesis refinement. For learning tasks with noise, ASPAL's sensitivity and specificity tends to level off. For RASPAL, should the threshold be too high or too low compared to the number of noisy examples, then its iterative learning tends to over-fit the solution to the examples.

In conclusion, the comparisons show that RASPAL can handle a wider range of learning tasks

than HYPER and ASPAL. It should be noted, however, that RASPAL's learning approach will only be beneficial if the the literals in the solution are not highly dependent on each other. For instance, suppose the longest rule in the solution is $r = h \leftarrow b_1, b_2, b3$, and to improve the examples coverage of the partial hypothesis h all body literals b_1, b_2 and b_3 must be added to it. RASPAL can only successfully refine this rule if it is executed with the rule length limit i set to 3, which will eventually call ASPAL with maximum number of conditions per rule d_{max} set to 3. However, as r is the longest rule in the solution, it can be solved by simply running normal ASPAL with d_{max} set to 3. RASPAL will perform worse than ASPAL, even if they both result in ASPAL being called with the same value for d_{max}, as RASPAL will have the overhead from the revisable theory and additional mode declarations. Thus, while hypothesis refinement can be used to reduce the hypothesis space of many inductive tasks, there is a class of learning tasks where this approach is not applicable.

Comparison of the different ASP solvers shows that overall Clingo is the more suitable choice of ASP system to use with ASPAL and RASPAL as it could solve more of ASPAL and RASPAL's ASP programs than DLV. Furthermore, the size of the ground programs produced by Clingo is nearly always smaller than the ones produced by DLV.

Through two different applications we have shown how constraint-driven bias could be applied. In addition to this, the stratification learning task also showed that the constraint could help prevent many non-stratified hypotheses from being output as solutions, as well as discussing how additional meta-level information could be added to handle multiple labels for the same predicate. The software engineering learning task presents a problem for which minimality constraint cannot be used to successfully solve the task as the minimal revision will not retain the model's structure. Constraint-driven bias is applied so that the revised goal model retains its balanced tree structure. Note that the labelling scheme used in this work does not take function symbols into account as we have not considered applying constraint-driven bias to any learning task which has function symbols in its mode declaration.

Chapter 8

Conclusion

This thesis presented two new learning approaches in the context of Inductive Logic Programming (ILP) based on the reduction of the hypothesis space. The first of theses is *Learning though Hypothesis Refinement* which combines theory revision and abductive learning in Answer Set Programming (ASP). It has been designed for addressing the scalability problem of ASP based ILP systems, by dividing the learning task into smaller and more scalable hypothesis refinement tasks. The learning approach has been implemented as the system RASPAL, which combines hypothesis refinement with the ILP system ASPAL, whose completeness has been shown to follow the completeness of RASPAL. Despite ASPAL being the system used for this work, it could be replaced by other ASP based ILP systems. The transformation of revision into an inductive task is general enough that other non-monotonic ILP systems can be used instead of ASPAL. Overall the evaluations conducted in Chapter 7 show that RASPAL is able to take advantage of hypothesis refinement, applying to large learning tasks to make the computation more scalable.

Using RASPAL's scoring scheme, we have shown how RASPAL (and ASPAL) can be adapted to solve learning tasks with noise by using a noise threshold. We have evaluated both systems on a real life application with noise. The result have shown how the noise threshold affects the number of iterations refinement RASPAL performs, which could also be used as a way for finding approximate hypotheses rather than solutions in situations where performance is valued

over complete accuracy.

The second approach for reducing the side of the hypothesis space is *Constraint-driven Bias*. In this approach, the inductive learning task is extend by a set of domain-dependent constraints, which are designed to be constraints over the bias, and consequently the hypothesis space. A set of abstract predicates is introduced and the formalisation of each predicate is given in Chapter 5. In Chapter 6 we have show how domain-dependent constraints can be translated into ASP constraints and implemented in ASPAL and RASPAL, and provide the correctness of the translation.

8.1 Future Work

With respect to the goal of improving the scalability of ASP based ILP systems without compromising their completeness, RASPAL could be combined with techniques from other works that also tackled the ASP grounding problem. A possibility is [RIB13] where a modular approach to ASP is proposed for reducing the search space of ASP programs by dividing them into linked modules, and using meta-knowledge to manage how the modules will be combined together. The application of this would be for dividing learning tasks that contains large background knowledge, by dividing the background knowledge into modules of closely related concepts. The learner can then fetches only modules that are related to the concept that it is learning. Note that dividing the hypothesis space itself into multiple modules would be a difficult task and could lead to making learning tasks unsolvable. For instance, if different rules in the hypothesis space are learnt using different modules, then it is possible that solutions containing rules that are dependent on one another cannot be learnt.

RASPAL currently uses ASPAL as its core solver, and further improvements to ASPAL will also be of benefit to RASPAL. For instance, parallelising ASPAL will make each iteration of RASPAL more efficient. RASPAL itself could potentially be parallelised for a more through search of the hypothesis space. Recall that RASPAL currently only selects one highest scoring partial hypothesis per iteration. However, there could be many partial hypotheses with the same

score as the one selected. Multiple instances of RASPAL could be run for each of such partial hypotheses as some of them may require less refinement to find the solution of the learning task. In addition to this, RASPAL itself is a wrapper around ASPAL. Further investigation into its learning approach could be carried out by using different ASP-based ILP systems, such as XHAIL, as RASPAL's core learner to see if RASPAL would be able to mitigate their grounding problem.

The constraint-driven bias approach presented in this thesis only allows one primitive of $\mathcal{L}_{\mathcal{C}}$ to be used in a domain-dependent constraint. Exploring how multiple primitives interact in a single constraint could allow for more expressive constraints to be specified. Further applications of constraint-driven bias could be explored to extend those already given in this thesis.

Other possible direction for future work is Predicate Invention [Kra95]. Predicate Invention is the problem of learning a concept that was not in the background knowledge and examples of the inductive learning task. The purpose of learning new predicates can either be for restructuring the knowledge [Fla93], called *reformulation*, or for finding missing concepts that would make previously unsolvable learning task become solvable [MB88], called *bias shift*. A requirement of predicate invention is that predicates should only be invented when they are needed so that the accumulated knowledge will not be filled with useless concepts. In [Sta96] it has been identified that the type of predicates necessary for bias shift are those which have recursive definitions (and facts) as other predicates can be unfolded. Recursive definitions are challenging to learn for many ILP systems and systems in the past have built-in checks specifically for ensuring that recursive rules will terminate (for instance, CHILLIN [ZMK94] will force recursive rules to have a term in its body that is a reduction of a term in the rule's head). As we have noted in Chapter 1, ASP is a great platform for learning recursive rules as cycles in programs do not effect its computation. In [LZB08] a method for inventing predicates called *placeholders* is used. Placeholders are mode declarations containing the schema of the new concept, making its definition be included in the hypothesis space. The problem with this approach is that prior knowledge of the new concept is required. A way of avoiding this problem is to give the learner multiple schemas for the new concept, however, this would unnecessarily increase the hypothesis space. RASPAL's iterative learning could be used to determine when it would be

appropriate to invent new predicates, that is when it fails to find a solution due to it not being able to refine a partial hypothesis, as well as help mitigate the increase in the hypothesis space due to the addition of placeholders.

Appendix A

Learning Tasks

A.1 mother

Given some background on the parenthood ($child(C, P)$ indicates that C is the child of P) and gender of a group of people, learn the relationship $mother(X, Y)$, where X is the mother of Y.

$$E^+ = \left\{ \begin{array}{l} mother(m_1, s_1). \\ mother(m_1, m_2) \end{array} \right\} \quad E^- = \left\{ \begin{array}{ll} mother(s_1, m_3). & mother(s_1, s_3) \\ mother(m_2, m_4). & mother(s_2, s_3) \\ mother(m_1, m_3). & \end{array} \right\}$$

$$B = \left\{ \begin{array}{ll} person(s_1)., \ldots, person(s_5). & person(m_1)., \ldots, person(m_5). \\ male(s_1)., \ldots, male(s_5). & female(m_1)., \ldots, female(m_5). \\ child(s_1, m_1). & child(m_5, m_4). \\ child(s_2, m_1). & child(m_3, s_1). \\ child(m_2, m_1). & child(s_3, s_1). \end{array} \right\}$$

$$M = \left\{ \begin{array}{ll} modeb(male(+person)). & modeh(mother(+person, +person)). \\ modeb(female(+person)). & modeb(child(+person, +person)). \end{array} \right\}$$

This task's objective is to learn the rule $mother(X, Y) \leftarrow child(Y, X), female(X)$. RASPAL and ASPAL were able to output this as their only correct solution.

A.2 nonealike

Dice poker is an adaptation of poker but replacing the card with five die. In this game players roll the five die to see who will have the highest hand and win the round. The objective of the task below is to learn a check to confirm that none of the rolled die have a value on their upside faces. We are assuming face values in $nonealike/5$ are always given sorted from the lowest value to the highest.

$$E^+ = \left\{ \begin{array}{l} nonealike(1,2,3,4,5). \\ nonealike(1,3,4,5,6). \end{array} \right\}$$

$$E^- = \left\{ \begin{array}{ll} nonealike(2,2,3,4,5). & nonealike(1,3,3,4,6). \\ nonealike(1,2,3,3,5). & nonealike(2,3,4,5,5). \end{array} \right\}$$

$$B = \left\{ \begin{array}{ll} face(1). & face(4). \\ face(2). & face(5). \\ face(3). & face(6). \\ eq(X,X) \leftarrow face(X). & \\ diff(X,X) \leftarrow face(X), face(Y), X \neq Y. & \end{array} \right\}$$

$$M = \left\{ \begin{array}{ll} modeh(nonealike(+face,+face,+face,+face,+face)). & \\ modeb(eq(+face,+face)). & modeb(diff(+face,+face)). \end{array} \right\}$$

ASPAL was unable to solve the task while RASPAL returns the solution of:

$nonealike(A,B,C,D,E) \leftarrow diff(A,B), diff(B,C), diff(C,D), diff(D,E).$

A.3 highroll

In the game High & Low, a player guesses whether the sum of two rolled die will be 'high' or 'low' value. A low roll occurs when the sum of the die is less than 7, while a high roll is when the sum is more than 7.

$$E^+ = \left\{ \begin{array}{lll} high(3,5). & high(6,3). & high(6,6). \end{array} \right\}$$

$$E^- = \left\{ \begin{array}{lllll} high(1,1). & high(2,3). & high(4,1). & high(3,3). & high(5,2). \end{array} \right\}$$

$$B = \left\{ \begin{array}{lll} face(1). & face(2). & face(3). \\ face(4). & face(5). & face(6). \\ sum(2). & sum(3). & sum(4) \\ sum(5). & sum(6). & sum(7) \\ sum(8). & sum(9). & sum(10) \\ sum(11). & sum(12). & \\ add(X,Y,Z) \leftarrow face(X), face(Y), Z = X + Y, sum(Z). & & \\ greaterThan(X,Y) \leftarrow sum(X), sum(Y), X > Y. & & \end{array} \right\}$$

$$M = \left\{ \begin{array}{l} modeh(high(+face, +face)). \\ modeb(add(+face, +face, -sum)). \\ modeb(greaterThan(+sum, \#sum)). \end{array} \right\}$$

HYPER was unable to solve this task while both ASPAL and RASPAL were able to output rule for the high roll $high(X,Y) \leftarrow add(X,Y,Z), greaterThan(Z,7)$.

A.4 mobile without noise

This example is from work done by our research group where an application records a mobile user's phone usage behaviour. From this information we extracted a week-long information regarding times when the user accepts or rejects an incoming call. These are used as examples for the learning task.

$$E^+ = \left\{ \begin{array}{l} accept((18,3,2011),(18),3,5,2,-1,1,0,0). \\ accept((19,3,2011),(11),4,7,2,52,0,0,0). \\ accept((19,3,2011),(14),4,7,2,23,0,0,0). \\ accept((19,3,2011),(16),4,7,2,57,0,1,1). \\ accept((20,3,2011),(12),4,7,2,12,1,1,1). \\ accept((20,3,2011),(15),4,7,2,71,1,1,1). \\ accept((20,3,2011),(15),8,7,2,69,0,1,1). \end{array} \right\}$$

$$E^- = \left\{ \begin{array}{l} accept((20,3,2011),(13),1,7,2,37,1,1,1). \\ accept((20,3,2011),(13),1,7,2,39,1,1,1). \\ accept((20,3,2011),(14),1,7,2,44,1,1,1). \\ accept((20,3,2011),(14),1,7,2,44,1,1,1). \\ accept((20,3,2011),(14),6,7,2,46,1,1,1). \\ accept((18,3,2011),(17),-1,5,2,-1,1,1,1). \\ accept((20,3,2011),(13),3,7,2,34,0,1,1). \\ accept((20,3,2011),(13),3,7,2,36,1,1,1). \\ accept((20,3,2011),(14),3,7,2,42,1,1,1). \\ accept((20,3,2011),(14),3,7,2,42,1,1,1). \\ accept((20,3,2011),(14),3,7,2,44,1,1,1). \\ accept((24,3,2011),(21),3,0,0,100,1,1,1). \\ accept((20,3,2011),(14),3,7,2,46,1,1,1). \\ accept((18,3,2011),(18),3,5,2,-1,0,0,0). \\ accept((20,3,2011),(12),4,7,2,10,0,0,0). \\ accept((18,3,2011),(18),7,5,2,-1,1,1,1). \end{array} \right\}$$

The background includes domain information for each argument of *accept*/9, and some additional rules. The background and the mode declarations are as follows:

$$
B = \left\{
\begin{array}{ll}
time(0)., \ldots, time(23). & contact(-1)., \ldots, contact(8). \\
volume(0)., \ldots, volume(7). & vibrator(0)., \ldots, vibrator(2). \\
battery\_level(-1)., \ldots, battery\_level(100). & \\
screen\_brightness(0). & screen\_brightness(1). \\
light\_level(0). & light\_level(1). \\
battery\_charging(0). & battery\_charging(1). \\
high\_volume(X) \leftarrow volume(X), X > 5. & \\
low\_battery(X) \leftarrow battery\_level(X), X \leq 10. & \\
high\_battery(X) \leftarrow battery\_level(X), X > 50. & \\
screen\_on(1). & is\_charging(1). \\
morning(H) \leftarrow time(H), H < 12, H \geq 0. & \\
afternoon(H) \leftarrow time(H), H \geq 12, H < 18. & \\
evening(H) \leftarrow time(H), H \geq 18. & \\
app(googletalk). & app(sms). \\
user\_is\_using\_app((18,3,2011), 18, sms). & \\
user\_is\_using\_app((18,3,2011), 18, googletalk). & \\
user\_is\_using\_app((18,3,2011), 17, googletalk). & \\
user\_is\_using\_app((18,3,2011), 16, googletalk). & \\
user\_is\_using\_app((18,3,2011), 15, googletalk). & \\
user\_is\_using\_app((19,3,2011), 0, googletalk). & \\
user\_is\_using\_app((19,3,2011), 1, googletalk). & \\
user\_is\_using\_app((19,3,2011), 10, googletalk). & \\
user\_is\_using\_app((19,3,2011), 11, googletalk). & \\
user\_is\_using\_app((19,3,2011), 12, googletalk). & \\
user\_is\_using\_app((19,3,2011), 14, googletalk). & \\
user\_is\_using\_app((20,3,2011), 12, googletalk). & \\
user\_is\_using\_app((24,3,2011), 21, googletalk). &
\end{array}
\right\}
$$

$$M = \left\{ \begin{array}{l} modeh(accept(+date, +time, +contact, +volume, +vibrator, \\ \quad +battery\_level, +screen\_brightness, +light\_level, +battery\_charging)). \\ modeb(not\ user\_is\_using\_app(+date, +time, \#app)). \\ modeb(user\_is\_using\_app(+date, +time, \#app)). \\ modeb(high\_volume(+volume)). \\ modeb(not\ high\_volume(+volume)). \\ modeb(low_battery(+battery\_level)). \\ modeb(not\ low_battery(+battery\_level)). \\ modeb(high_battery(+battery\_level)). \\ modeb(not\ high\_battery(+battery\_level)). \\ modeb(screen\_on(+screen\_brightness)). \\ modeb(not\ screen\_on(+screen\_brightness)). \\ modeb(is\_charging(+battery\_charging)). \\ modeb(not\ is\_charging(+battery\_charging)). \\ modeb(morning(+time)). \\ modeb(not\ morning(+time)). \\ modeb(afternoon(+time)). \\ modeb(not\ afternoon(+time)). \\ modeb(evening(+time)). \\ modeb(not\ evening(+time)). \end{array} \right\}$$

The following is one of the possible solution for the learning task, describing three different scenarios in which the user will accept an incoming call:

$$H = \left\{ \begin{array}{l} accept(A, B, C, D, E, F, G, H, I) \leftarrow \\ \quad user\_is\_using\_app(A, B, googletalk), not\ is\_charging(I), \\ \quad screen\_on(G). \\ accept(A, B, C, D, E, F, G, H, I) \leftarrow \\ \quad user\_is\_using\_app(A, B, googletalk), high\_volume(D), \\ \quad not\ low\_battery(F). \\ accept(A, B, C, D, E, F, G, H, I) \leftarrow \\ \quad high\_battery(F), afternoon(B). \end{array} \right\}$$

A.5 mobile with noise

$$E^+ = \left\{ \begin{array}{l} accept((18,3,2011),(18),3,5,2,-1,1,0,0). \\ accept((19,3,2011),(11),4,7,2,52,0,0,0). \\ accept((19,3,2011),(14),4,7,2,23,0,0,0). \\ accept((19,3,2011),(16),4,7,2,57,0,1,1). \\ accept((20,3,2011),(12),4,7,2,12,1,1,1). \\ accept((20,3,2011),(15),4,7,2,71,1,1,1). \\ accept((20,3,2011),(15),8,7,2,69,0,1,1). \end{array} \right\}$$

$$E^- = \left\{ \begin{array}{l} accept((20,3,2011),(13),1,7,2,37,1,1,1). \\ accept((20,3,2011),(13),1,7,2,39,1,1,1). \\ accept((20,3,2011),(14),1,7,2,44,1,1,1). \\ accept((20,3,2011),(14),1,7,2,44,1,1,1). \\ accept((19,3,2011),(12),2,7,2,47,1,0,0). \\ accept((19,3,2011),(14),5,7,2,22,0,1,1). \\ accept((20,3,2011),(14),6,7,2,46,1,1,1). \\ accept((18,3,2011),(17),-1,5,2,-1,1,1,1). \\ accept((20,3,2011),(12),3,7,2,13,1,1,1)). \\ accept((20,3,2011),(13),3,7,2,34,0,1,1)). \\ accept((20,3,2011),(13),3,7,2,36,1,1,1)). \\ accept((20,3,2011),(14),3,7,2,42,1,1,1)). \\ accept((20,3,2011),(14),3,7,2,44,1,1,1)). \\ accept((24,3,2011),(21),3,0,0,100,1,1,1)). \\ accept((20,3,2011),(14),3,7,2,46,1,1,1)). \\ accept((18,3,2011),(18),3,5,2,-1,0,0,0)). \\ accept((20,3,2011),(12),4,7,2,10,0,0,0)). \\ accept((18,3,2011),(18),7,5,2,-1,1,1,1)). \end{array} \right\}$$

The background include domain information for each argument of $accept/9$, and some additional rules. The background and the mode declarations are as follows:

$$B = \left\{ \begin{array}{ll}
time(0)., \ldots, person(23). & contact(-1)., \ldots, contact(8). \\
volume(0)., \ldots, volume(7). & vibrator(0)., \ldots, vibrator(2). \\
battery_level(-1)., \ldots, battery_level(100). & \\
screen_brightness(0). & screen_brightness(1). \\
light_level(0). & light_level(1). \\
battery_charging(0). & battery_charging(1). \\
high_volume(X) \leftarrow volume(X), X > 5. & \\
low_battery(X) \leftarrow battery_level(X), X \leq 10. & \\
high_battery(X) \leftarrow battery_level(X), X > 50. & \\
screen_on(1). & is_charging(1). \\
morning(H) \leftarrow time(H), H < 12, H \geq 0. & \\
afternoon(H) \leftarrow time(H), H \geq 12, H < 18. & \\
evening(H) \leftarrow time(H), H \geq 18. & \\
app(googletalk). & app(sms). \\
user_is_using_app((18,3,2011), 18, sms). & \\
user_is_using_app((18,3,2011), 18, googletalk). & \\
user_is_using_app((18,3,2011), 17, googletalk). & \\
user_is_using_app((18,3,2011), 16, googletalk). & \\
user_is_using_app((18,3,2011), 15, googletalk). & \\
user_is_using_app((19,3,2011), 0, googletalk). & \\
user_is_using_app((19,3,2011), 1, googletalk). & \\
user_is_using_app((19,3,2011), 10, googletalk). & \\
user_is_using_app((19,3,2011), 11, googletalk). & \\
user_is_using_app((19,3,2011), 12, googletalk). & \\
user_is_using_app((19,3,2011), 14, googletalk). & \\
user_is_using_app((20,3,2011), 12, googletalk). & \\
user_is_using_app((24,3,2011), 21, googletalk). &
\end{array} \right\}$$

$$M = \left\{ \begin{array}{l} modeh(accept(+date, +time, \#contact, +volume, +vibrator, \\ \quad +battery\_level, +screen\_brightness, +light\_level, +battery\_charging)). \\ modeb(user\_is\_using\_app(+date, +time, \#app)). \\ modeb(not\ user\_is\_using\_app(+date, +time, \#app)). \\ modeb(high\_volume(+volume)). \\ modeb(not\ high\_volume(+volume)). \\ modeb(low\_battery(+battery\_level)). \\ modeb(not\ low\_battery(+battery\_level)). \\ modeb(high\_battery(+battery\_level)). \\ modeb(not\ high\_battery(+battery\_level)). \\ modeb(screen\_on(+screen\_brightness)). \\ modeb(not\ screen\_on(+screen\_brightness)). \\ modeb(is\_charging(+battery\_charging)). \\ modeb(not\ is\_charging(+battery\_charging)). \\ modeb(morning(+time)). \\ modeb(not\ morning(+time)). \\ modeb(afternoon(+time)). \\ modeb(not\ afternoon(+time)). \\ modeb(evening(+time)). \\ modeb(not\ evening(+time)). \end{array} \right\}$$

The following two solutions were learnt with noise threshold of $\langle 1, 3\rangle$ in ASPAL, and $\langle 7, 0\rangle$ in ASPAL respectively:

$$H_1 = \left\{ \begin{array}{l} accept(A, B, 4, C, D, E, F, G, H) \leftarrow afternoon(B), high\_volume(C). \end{array} \right\}$$

$$H_2 = \left\{ \begin{array}{l} accept(A, B, 3, C, D, E, F, G, H) \leftarrow low\_battery(E), screen\_on(F). \\ accept(A, B, 4, C, D, E, F, G, H) \leftarrow not\ low\_battery(E). \\ accept(A, B, 8, C, D, E, F, G, H). \end{array} \right\}$$

A.6 odd & even

Learn the concept of even and odd numbers using the following task:

$$E^+ = \left\{ \begin{array}{ll} even(0). & even(s(s(s(s(0))))). \\ even(s(s(0))). & \\ odd(s(s(s(0)))). & odd(s(s(s(s(s(0)))))). \end{array} \right\}$$

$$E^- = \left\{ \begin{array}{ll} even(s(0)). & even(s(s(s(0)))). \\ odd(0). & odd(s(s(s(s(0))))). \end{array} \right\}$$

$$B = \left\{ \begin{array}{ll} num(0). & num(s(s(s(0)))). \\ num(s(0)). & num(s(s(s(s(0))))). \\ num(s(s(0))). & num(s(s(s(s(s(0)))))). \\ succ(X, s(X)) \leftarrow num(X), num(s(X)). & \end{array} \right\}$$

$$M = \left\{ \begin{array}{ll} modeh(even(+num)). & modeh(even(\#num)). \\ modeh(odd(+num)). & modeb(even(+num)). \\ modeb(noteven(+num)). & modeb(succ(-num, +num)). \end{array} \right\}$$

The aim of this task is to learn the definition of even and odd in terms of even. Possible solutions for this task include:

$$H_1 = \left\{ \begin{array}{l} even(0). \\ even(A) \leftarrow succ(B, A), not\ even(B). \\ odd(A) \leftarrow succ(B, A), not\ even(A). \end{array} \right\}$$

$$H_2 = \left\{ \begin{array}{l} even(0). \\ even(A) \leftarrow succ(B, A), not\ even(B). \\ odd(A) \leftarrow succ(B, A), even(B). \end{array} \right\}$$

$$H_3 = \left\{ \begin{array}{l} even(0). \\ even(A) \leftarrow succ(B, A), not\ even(B). \\ odd(A) \leftarrow not\ even(A). \end{array} \right\}$$

The ASPAL encoding will return all three of these solutions. For RASPAL with Clingo, if the option "–opt-all" is not used only H_2 is returned. When using RASPAL with DLV, since the same optimisation clause in Clingo cannot be used to only output one answer set per optimisation level, all three solutions will be returned.

A.7 train

This is the train classification challenge set by Michalski. The original data set[1] contains information about ten trains, half of them eastbound train, and the other half are westbound train. We used the following representation for describing each train[2]:

Representation	Meaning
$eastbound(X)$	Train X is an eastbound train
$westbound(X)$	Train X is a westbound train
$has\_car(X, Y)$	Carriage Y is part of train X
$short(X)$	Carriage X is a short carriage
$long(X)$	Carriage X is a long carriage
$shape(X, Y)$	Carriage X is in shape Y
$open\_car(X)$	Carriage X has no roof
$closed(X)$	Carriage X has a roof
$load(X, Y, Z)$	Carriage X has Z number of load with shape Y
$wheels(X, Y)$	Carriage X has Y number of wheels

The following mode declarations were used for this example:

[1] Obtained at `http://ftp.ics.uci.edu/pub/machine-learning-databases/trains/`

[2] Taken from examples at `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/misc/examples.zip`

$$M = \left\{ \begin{array}{ll} modeh(eastbound(+train\_id)). & modeh(westbound(+train\_id)). \\ modeb(short(+car\_id)). & modeb(has\_car(+train\_id, -car\_id)). \\ modeb(closed(+car\_id)). & modeb(shape(+car_id, \#car\_shape)). \\ modeb(long(+car\_id)). & modeb(noteastbound(+train\_id)). \\ modeb(open(+car\_id)). & modeb(wheels(+car_id, \#num\_wheels)). \\ modeb(has\_load(+car\_id, \#load\_shape, \#num\_load)). & \end{array} \right\}$$

The objective of the task is to learn how to classify east and westbound trains. From the examples, the shortest solution to the task is:

$eastbound(X) \leftarrow has\_car(X, Y), short(Y), closed(Y).$
$westbound(X) \leftarrow noteastbound(X).$

Note that when running in the all output mode, ASPAL will also return the following suboptimal solution:

1. $eastbound(X) \leftarrow has\_car(X, Y), short(Y), closed(Y).$
   $westbound(X) \leftarrow not\ eastbound(X), has\_car(X, Y).$
2. $eastbound(X) \leftarrow has\_car(X, Y), short(Y), closed(Y).$
   $westbound(X) \leftarrow noteastbound(X), has\_car(X, Y), has\_car(X, Z).$
3. $eastbound(X) \leftarrow has\_car(X, Y), short(Y), closed(Y).$
   $westbound(X) \leftarrow noteastbound(X), has\_car(X, Y), long(Y).$

A.8 Flight Control System

The aim of this example is to learn how to refine a model of a flight control system such that the refined model retains the syntactic structure of the original model.

The background of the task can be divided into domain domain background D, event calculus axioms A_e, domain pre-conditions and constraints D_c, goal axioms A_g, and scenarios S ($B = D \cup A_e \cup D_c \cup A_g \cup S$).

$$M = \left\{ \begin{array}{l} modeh(holds(\#goal\_exp, +timepoint, +scenario)). \\ modeb(holds\_at(\#usable\_fluent, +timepoint, +scenario)). \\ modeb(not\ holds\_at(\#usable\_fluent, +timepoint, +scenario)). \end{array} \right\}$$

$$D = \left\{ \begin{array}{ll} usable\_fluent(wheelsPulseOn). & usable\_fluent(wheels\_out). \\ usable\_fluent(aqua\_planing). & fluent(onRunway). \\ fluent(thrustEnabled). & fluent(wheelsPulseOn). \\ fluent(wheelsTurning). & fluent(aqua\_planing). \\ fluent(wet\_surface). & fluent(wheels\_out). \\ fluent(wheels\_blocked). & goal\_exp(g). \\ goal\_exp(c2). & goal\_exp(c1). \\ event(land). & event(takeOff). \\ event(park). & event(switchOn). \\ event(switchOff). & event(enable). \\ event(disable). & event(turnWheels). \\ event(stopWheels). & event(dry). \\ scenario(s1). & scenario(s2). \\ timepoint(0). & timepoint(1). \\ timepoint(2). & \\ next(1,0). \quad \ldots & next(11,10). \end{array} \right\}$$

$$
A = \left\{
\begin{array}{l}
clipped(T1, F, T2, S) \leftarrow T1 \leq T, T < T2, \\
\qquad happens(E, T, S), terminates(E, F, T, S). \\
holds\_at(F, T2, S) \leftarrow T1 < T2, happens(E, T1, S), \\
\qquad initiates(E, F, T, S), not\ clipped(T1, F, T2, S). \\
holds\_at(F, T2, S) \leftarrow initially(F, S), not\ clipped(0, F, T2, S). \\
happens(E, T, S) \leftarrow event(E), timepoint(T), scenario(S), \\
\qquad executed(E, T, S), not\ impossible(E, T, S). \\
exists\_next\_timepoint(T1) \leftarrow next(T2, T1). \\
initiates(enable, thrustEnabled, T, S). \\
terminates(disable, thrustEnabled, T, S). \\
initiates(switchOn, wheelsPulseOn, T, S). \\
terminates(switchOff, wheelsPulseOn, T, S). \\
initiates(land, onRunway, T, S). \\
terminates(park, onRunway, T, S). \\
terminates(takeOff, onRunway, T, S). \\
initiates(turnWheels, wheelsTurning, T, S). \\
terminates(stopWheels, wheelsTurning, T, S). \\
initiates(land, aqua\_planing, T, S) \leftarrow holds\_at(wet\_surface, T, S).
\end{array}
\right\}
$$

$$
D_c = \left\{
\begin{array}{l}
impossible(enable, T, S) \leftarrow holds\_at(thrustEnabled, T, S). \\
impossible(disable, T, S) \leftarrow not\ holds\_at(thrustEnabled, T, S). \\
impossible(land, T, S) \leftarrow holds\_at(onRunway, T, S). \\
impossible(park, T, S) \leftarrow not\ holds\_at(onRunway, T, S). \\
impossible(takeOff, T, S) \leftarrow not\ holds\_at(onRunway, T, S). \\
impossible(switchOn, T, S) \leftarrow holds\_at(wheelsPulseOn, T, S). \\
impossible(switchOff, T, S) \leftarrow not\ holds\_at(wheelsPulseOn, T, S). \\
impossible(turnWheels, T, S) \leftarrow holds\_at(wheelsTurning, T, S). \\
impossible(stopWheels, T, S) \leftarrow not\ holds\_at(wheelsTurning, T, S). \\
\leftarrow holds\_at(wheelsTurning, T, S), holds\_at(aqua\_planing, T, S). \\
\leftarrow holds\_at(wheelsTurning, T, S), holds\_at(wheels\_outocked, T, S). \\
\leftarrow holds\_at(wheelsTurning, T, S), not\ holds\_at(wheels\_out, T, S). \\
\leftarrow not\ holds\_at(F, T1, S), holds\_at(F, T1, S). \\
\leftarrow happens(E, T, S), impossible(E, T, S). \\
\leftarrow goal\_exp(G), not\ holds(G, T, S). \\
\leftarrow child\_of(G, C), holds(G, T, S), not\ holds(C, T, S). \\
\leftarrow goal\_exp(G), child\_of(G, G). \\
\leftarrow goal\_exp(G), holds\_non\_vacuously(G, T, S), holds\_vacuously(G, T, S). \\
\leftarrow child\_of(G, C), holds\_non\_vacuously(G, T, S), holds\_vacuously(C, T, S).
\end{array}
\right\}
$$

$$G = \left\{ \begin{array}{l} holds(G,T,S) \leftarrow holds\_non\_vacuously(G,T,S). \\ holds(G,T,S) \leftarrow holds\_vacuously(G,T,S). \\ holds(G,T,S) \leftarrow holds\_at\_end\_of\_time(G,T,S). \\ holds\_vacuously(G,T1,S) \leftarrow not\ antecedent\_holds(G,T1,S). \\ root\_goal(g). \\ antecedent\_holds(g,T1,S) \leftarrow holds\_at(onRunway,T1,S). \\ consequent\_holds(g,T2,S) \leftarrow holds\_at(thrustEnabled,T2,S). \\ holds\_non\_vacuously(g,T1,S) \leftarrow antecedent\_holds(g,T1,S), next(T2,T1), \\ \qquad consequent\_holds(g,T2,S). \\ holds\_at\_end\_of\_time(g,T1,S) \leftarrow antecedent\_holds(g,T1,S), \\ \qquad not\ exists\_next\_timepoint(T1). \\ child\_of(g,c1). \\ child\_of(g,c2). \\ left\_child\_of(g,c1). \\ right\_child\_of(g,c2). \\ holds\_non\_vacuously(c2,T1,S) \leftarrow antecedent\_holds(c2,T1,S), next(T2,T1), \\ \qquad consequent\_holds(c2,T2,S). \\ holds\_at\_end\_of\_time(c2,T1,S) \leftarrow antecedent\_holds(c2,T1,S), \\ \qquad not\ exists\_next\_timepoint(T1). \end{array} \right\}$$

$$S = \left\{ \begin{array}{ll} initially(wet\_surface,0,s1). & initially(wheels\_out,s1). \\ initially(wheels\_out,s2). & \\ happens(land,0,s1). & happens(switchOn,0,s1). \\ happens(enable,1,s1). & happens(land,0,s2). \\ happens(turnWheels,0,s2). & happens(switchOn,0,s2). \\ happens(enable,1,s2). & happens(park,1,s2). \end{array} \right\}$$

The examples are that all the goals holds within the given scenarios, and there are no negative examples.

$$E^+ = \left\{ \begin{array}{ll} holds(g,1,s1). & holds(c1,1,s1). \\ holds(c2,1,s1). & \\ holds(g,2,s2). & holds(c1,2,s2). \\ holds(c2,2,s2). & \end{array} \right\}$$

The revisable rules consist are $antecedent\_holds/3$ and $consequent\_holds/3$ rules that express if a sub-goal is true at certain time and scenario.

$$R = \left\{ \begin{array}{l} revisable((antecedent\_holds(c1,T1,S) \leftarrow holds\_at(onRunway,T1,S)), \\ \quad antecedent\_holds(c1)). \\ revisable((consequent\_holds(c1,T1,S) \leftarrow holds\_at(wheelsTurning,T1,S)), \\ \quad consequent\_holds(c1)). \\ revisable((antecedent\_holds(c2,T1,S) \leftarrow holds\_at(wheelsTurning,T1,S)), \\ \quad antecedent\_holds(c2)). \\ revisable((consequent\_holds(c2,T1,S) \leftarrow holds\_at(thrustEnabled,T1,S)), \\ \quad consequent\_holds(c2)). \end{array} \right\}$$

$$IC = \left\{ \begin{array}{l} \leftarrow in\_rule(antecedent\_holds(C1),Body), right\_child\_of(G,C1), \\ \quad left\_child\_of(G,C2), C1 \neq C2, not\ in\_rule(consequent\_holds(C2),Body). \end{array} \right\}$$

The aim of the task is to revise the the rules in R into R', replacing the conditions of $consequent\_holds(c1,T1,S)$ and $antecedent\_holds(c2,T1,S)$.

$$R' = \left\{ \begin{array}{l} antecedent\_holds(c1,T1,S) \leftarrow holds\_at(onRunway,T1,S). \\ consequent\_holds(c1,T1,S) \leftarrow holds\_at(wheelsPulseOn,T1,S). \\ antecedent\_holds(c2,T1,S) \leftarrow holds\_at(wheelsPulseOn,T1,S). \\ consequent\_holds(c2,T1,S) \leftarrow holds\_at(thrustEnabled,T1,S). \end{array} \right\}$$

Bibliography

[AAB+14] Duangtida Athakravi, Dalal Alrajeh, Krysia Broda, Alessandra Russo, and Ken Satoh. Inductive learning using constraint-driven bias. In Jesse Davis and Jan Ramon, editors, *Inductive Logic Programming - 24th International Conference, ILP 2014, Nancy, France, September 14-16, 2014, Revised Selected Papers*, volume 9046 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2014.

[ABW88] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter Towards a Theory of Declarative Knowledge, pages 89–148. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[ACBR13] Duangtida Athakravi, Domenico Corapi, Krysia Broda, and Alessandra Russo. Learning through hypothesis refinement using answer set programming. In Gerson Zaverucha, Vítor Santos Costa, and Aline Paes, editors, *Inductive Logic Programming - 23rd International Conference, ILP 2013, Rio de Janeiro, Brazil, August 28-30, 2013, Revised Selected Papers*, volume 8812 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2013.

[ACR+12] Duangtida Athakravi, Domenico Corapi, Alessandra Russo, Marina De Vos, Julian A. Padget, and Ken Satoh. Handling change in normative specifications. In Matteo Baldoni, Louise A. Dennis, Viviana Mascardi, and Wamberto Weber Vasconcelos, editors, *Declarative Agent Languages and Technologies X - 10th International Workshop, DALT 2012, Valencia, Spain, June 4, 2012, Revised Selected*

Papers, volume 7784 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2012.

[AKvL+12] Dalal Alrajeh, Jeff Kramer, Axel van Lamsweerde, Alessandra Russo, and Sebastián Uchitel. Generating obstacle conditions for requirements completeness. In *ICSE*, pages 705–715, 2012.

[Bar03] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[BR98] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artif. Intell.*, 101(1-2):285–297, 1998.

[BR14] Stefano Bragaglia and Oliver Ray. Nonmonotonic learning in large biological networks. In *24th Int. Conference on Inductive Logic Programming*, 2014.

[Bra99] Ivan Bratko. Refining Complete Hypotheses in ILP. In Saso Dzeroski and Peter A Flach, editors, *Inductive Logic Programming*, pages 44–55. Springer Berlin / Heidelberg, 1999.

[Bra01] Ivan Bratko. *Prolog (3rd ed.): programming for artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[Che94] Jianhua Chen. Inductive learning of stratified logic programs via non-monotonic inference. In *Florida Artificial Intelligence Research Symposium*, 1994.

[Cla77] Keith L. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322, 1977.

[Cor11] Domenico Corapi. *Nonmonotonic Inductive Logic Programming as Abductive Search*. PhD thesis, Imperial College London, 2011.

[CRL10] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming as abductive search. In Manuel Hermenegildo and Torsten Schaub, editors, *Technical Communications of the 26th International Conference on Logic Programming*, volume 2010, pages 54–63, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[CRL11] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In Stephen Muggleton, Alireza Tamaddoni-Nezhad, and Francesca A. Lisi, editors, *Inductive Logic Programming - 21st International Conference, ILP 2011, Windsor Great Park, UK, July 31 - August 3, 2011, Revised Selected Papers*, volume 7207 of *Lecture Notes in Computer Science*, pages 91–97. Springer, 2011.

[CRV+11] Domenico Corapi, Alessandra Russo, Marina De Vos, Julian A. Padget, and Ken Satoh. Normative design using inductive learning. *TPLP*, 11(4-5):783–799, 2011.

[DPZ09] Ana Luísa Duboc, Aline Paes, and Gerson Zaverucha. Using the bottom clause and mode declarations in fol theory revision from examples. *Machine learning*, 76(1):73–107, 2009.

[DvL96] Robert Darimont and Axel van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *FSE*, pages 179–190. ACM, 1996.

[EFLP03] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Computing preferred answer sets by meta-interpretation in answer set programming. *TPLP.*, 3(4):463–498, July 2003.

[Fla93] Peter A Flach. Predicate Invention in Inductive Data Engineering. In P Brazdil, editor, *Machine Learning ECML93 European Conference on Machine Learning Proceedings*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 83–94. Springer-Verlag, 1993.

[Fle97] Pierre Flener. Inductive logic program synthesis with DIALOGS. In Stephen Muggleton, editor, *Inductive Logic Programming*, volume 1314 of *Lecture Notes in Computer Science*, pages 175–198. Springer Berlin / Heidelberg, 1997.

[GKK+11] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.

[GKKS11] M. Gebser, R. Kaminski, A. König, and T. Schaub. Advances in *gringo* series 3. In J. Delgrande and W. Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, volume 6645 of *Lecture Notes in Artificial Intelligence*, pages 345–351. Springer-Verlag, 2011.

[GKNS07] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-driven answer set solving. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, page 386, 2007.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988.

[GRR10] Dov M. Gabbay, Odinaldo Rodrigues, and Alessandra Russo. *Revision, Acceptability and Context - Theoretical and Algorithmic Aspects*. Cognitive Technologies. Springer, 2010.

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

[IFKN10] Katsumi Inoue, Koichi Furukawa, Ikuo Kobayashi, and Hidetomo Nabeshima. Discovering Rules by Meta-level Abduction. In Luc De Raedt, editor, *Inductive Logic Programming*, volume 5989 of *Lecture Notes in Computer Science*, pages 49–64. Springer Berlin / Heidelberg, 2010.

[Ino04] Katsumi Inoue. Induction as consequence finding. *Machine Learning*, 55(2):109–135, 2004.

[IS00] Noboru Iwayama and Ken Satoh. Computing abduction by using TMS with top-down expectation. *J. Log. Program.*, 44(1-3):179–206, 2000.

[JB96] Alipio Jorge and Pavel Brazdil. Integrity constraints in ilp using a monte carlo approach. In *In Proc. 6th International Workshop on Inductive Logic Programming*, pages 137–151. Springer-Verlag, 1996.

[KAP14] Nikos Katzouris, Alexander Artikis, and George Paliouras. Incremental learning of event definitions with inductive logic programming. *CoRR*, abs/1402.5988, 2014.

[KBR09] Tim Kimber, Krysia Broda, and Alessandra Russo. Induction on Failure: Learning Connected Horn Theories. In *LPNMR*, pages 169–181, 2009.

[KKT92] Antonis C. Kakas, Robert A. Kowalski, and Francesca Toni. Abductive logic programming. *J. Log. Comput.*, 2(6):719–770, 1992.

[Kra95] Stefan Kramer. Predicate Invention: A Comprehensive View. *Intelligence*, 1995.

[KS86] Robert A. Kowalski and Marek Sergot. A logic-based calculus of events. *New generation computing*, 4(1):67–95, 1986.

[LPF+06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[LRB14] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 311–325. Springer, 2014.

[LZB08] Gregor Leban, Jure Zabkar, and Ivan Bratko. An Experiment in Robot Discovery with ILP. In *ILP 08 Proceedings of the 18th international conference on Inductive Logic Programming*, pages 77–90, 2008.

[MB88] Stephen Muggleton and Wray L Buntine. Machine Invention of First Order Predicates by Inverting Resolution. In *Machine Learning*, pages 339–352, 1988.

[MD94] Stephen Muggleton and Luc De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19-20(20):629–679, 1994.

[Mit97] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[ML13] Stephen Muggleton and Dianhuan Lin. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. IJCAI/AAAI, 2013.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[MS97] Eric McCreath and Arun Sharma. ILP with noise and fixed example size: A bayesian approach. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 1310–1315. Morgan Kaufmann, 1997.

[MSTn08] Stephen Muggleton, José Carlos Almeida Santos, and Alireza Tamaddoni-nezhad. TopLog : ILP using a logic program declarative bias. *ICLP*, 5366:687–692, 2008.

[Mug95] Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13(3):245–286, 1995.

[MV95] Lionel Martin and Christel Vrain. A three-valued framework for the induction of general logic programs. In *LECTURE NOTES IN COMPUTER SCIENCE*, page 2001. Springer-Verlag, 1995.

[Mye99] David G. Myers. *Exploring Psychology, 4th Edition*. Worth Publishers, New York, 1999.

[NdW97] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf, editors. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science*. Springer, 1997.

[OB10] Andrej Oblak and Ivan Bratko. Learning from noisy data using a non-covering ILP algorithm. In Paolo Frasconi and Francesca A. Lisi, editors, *Inductive Logic Programming - 20th International Conference, ILP 2010, Florence, Italy, June 27-30, 2010. Revised Papers*, volume 6489 of *Lecture Notes in Computer Science*, pages 190–197. Springer, 2010.

[QCj95] J. R. Quinlan and R. M. Cameron-jones. Induction of logic programs: Foil and related systems. *New Generation Computing*, 13:287–312, 1995.

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

[RBR03] Oliver Ray, Krysia Broda, and Alessandra Russo. Hybrid abductive inductive learning: A generalisation of progol. In Tamás Horváth, editor, *Inductive Logic Programming: 13th International Conference, ILP 2003, Szeged, Hungary, September 29-October 1, 2003, Proceedings*, volume 2835 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2003.

[RIB13] Tony Ribeiro, Katsumi Inoue, and Gauvain Bourgne. Combining answer set programs for adaptive and reactive reasoning. *TPLP*, 13(4-5-Online-Supplement), 2013.

[RM95] Bradley L. Richards and Raymond J. Mooney. Automated refinement of first-order horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[SI09] Chiaki Sakama and Katsumi Inoue. Brave induction: a logical framework for learning from incomplete information. *Machine Learning*, 67(1):3–35, 2009.

[Sri07] Ashwin Srinivasan. The aleph manual, 2007.

[Sta96] Irene Stahl. Predicate Invention in Inductive Logic Programming. In Luc De Raedt, editor, *Advances in Inductive Logic Programming*, pages 34–47. IOS Press, 1996.

[vEK76] Maarten H. van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976.

[WO91] Ruediger Wirth and Paul O'Rorke. Constraints on predicate invention. In Stephen H Muggleton, editor, *Proceedings of the Eighth International Workshop on Machine Learning*, pages 457–461. Morgan Kaufmann, 1991.

[Wro96] Stefan Wrobel. First order theory refinement. *Advances in inductive logic programming*, 32:14–33, 1996.

[ZMK94] John M. Zelle, Raymond J. Mooney, and Joshua B. Konvisser. Combining top-down and bottom-up techniques in inductive logic programming. In *in Proceedings of the Eleventh International Conference on Machine Learning ML-94, Morgan-Kaufmann*, pages 343–351. Morgan Kaufmann, 1994.