ILPNET repositories on WWW: Inductive Logic Programming systems, datasets and bibliography

Nada Lavrač (1), Irene Weber (2), Darko Zupanič (1),
Dimitar Kazakov (3), Olga Štěpánková (3), Sašo Džeroski (1)

(1) Department of Intelligent Systems, J. Stefan Institute, Ljubljana, Slovenia
email: {nada.lavrac, darko.zupanic, saso.dzeroski}@ijs.si

(2) Department of Computer Science, University of Stuttgart, Germany
email: Irene.Weber@informatik.uni-stuttgart.de

(3) Faculty of Electrical Engineering, Czech Technical University, Prague, Czech Republic
email: {kazakov,step}@labe.felk.cvut.cz

Abstract

European research in Inductive Logic Programming (ILP) has been mainly conducted within two ESPRIT research project, ILP (1992–95) and ILP2 (1996–98), whereas the European Inductive Logic Programming Scientific Network ILPNET (1993–96) provided the infrastructure support for ILP research. The main results of ILPNET are outlined in this paper. Particular emphasis is given on the description of ILPNET repositories of ILP systems, datasets and bibliography, which have been made publicly available on WWW at http://www-ai.ijs.si/ilpnet.html.

1 Introduction

European research in Inductive Logic programming (ILP) has been mainly conducted within two ESPRIT research project, ILP (1992–95) and ILP2 (1996–98). The main topics of ILP research are the investigation of the theoretical framework for induction, the development of practical algorithms for inductive learning in first-order logic representation formalisms, and the practical applications of ILP algorithms in relational learning problems.

Standard ILP systems induce a general hypothesis H, represented as a set of clauses in a selected first-order hypothesis language, from training examples E ($E = P \cup N$, where P are positive and N are negative examples) and background knowledge theory T, such that H explains the examples E with respect to the theory T. Training examples are ground facts, and background knowledge is a set of clauses, giving intensional or extensional definitions of background knowledge predicates. In the classical ILP concept-learning task, H is said to explain E with regard to T if and only if $T \wedge H \models P$ (completeness) and $T \wedge H \wedge N \not\models \Box$ (consistency). Many systems apply a weaker notions of explanation; this is, for instance, due to the need of handling imperfect (noisy) data or due to a different formulation of the learning task. In the latter case, where the task is to find the properties that can be derived from T and E, for which the explanation based on non-monotonic reasoning is more appropriate.

In the ESPRIT III project no. 6020 Inductive Logic Programming (1992–95), research was grouped around the following research topics: theory of inductive learning, theory revision and multiple predicate learning, predicate invention, handling of imperfect data, declarative bias and applications of ILP. In the ESPRIT IV project no. 20237 Inductive Logic Programming 2 (1996–98), there is an important shift towards applications of ILP, but the following theorectically and practically important scientific problems remain to be further investigated: background knowledge (relevance, theory revision, predicate invention), complex hypotheses (multi-clause learning, deep clauses, recursion, structured hypotheses), built-in semantics (number handling, probabilities, constraints, built-in predicates), and sampling issues (large datasets, small datasets, statistical reliability).

The consortium of the ILP2 project consists of eight institutions: Katholieke Universiteit Leuven (coordinator), Oxford University, German National Research Center for Computer Science, J. Stefan Institute and Faculty of Computer Science and Informatics Ljubljana, University of Stockholm, Université Paris Sud, Tilburg University, and Università di Torino.

Whereas the ILP projects support research, the European Inductive Logic Programming Scientific Network ILPNET (1993–96) provided the infrastructure support for ILP research. The consortium of ILPNET is a superset of that of the ESPRIT projects. It also consists of: Faculty of Technical Sciences Maribor, Bulgarian Academy of Sciences, Czech Technical University Prague, Romanian Academy of Sciences, Austrian Research Institute for Artificial Intelligence, University of Porto, University of Dortmund, Technical University Graz, Hungarian Academy of Sciences, Daimler Benz, and Rudjer Bošković Institute Zagreb. It can be noted that many ILPNET partners are from Central and Eastern European countries. Being a scientific network (network no. CP93-44) funded under the umbrella of PECO/Copernicus, the aim of ILPNET was namely to stimulate the development, coordination, communication and exchange of results and personnel in ILP research and to disseminate the research results to a wider European research community. A particular aim was to build new communication and dissemination channels and to make them available to Central and Eastern European researchers interested in ILP.

The main results of the ILPNET scientific network, coordinated by the J. Stefan Institute, Ljubljana, are outlined in this paper. Information about ILPNET and its activities can be obtained on World-Wide Web (WWW) from the J. Stefan Institute ILPNET homepage at `http://www-ai.ijs.si/ilpnet.html`.

The following information is accessible on WWW: a brochure describing each ILPNET partner (much of this information was already published in the AI Communications [318]), issues of the ILP Newsletter, ILP-related PhD thesis abstracts, ILP and ILP-related books. In this paper, a particular emphasis is given on the description of ILPNET repositories of ILP systems, datasets and bibliography, which have been made publicly available on WWW.

In Section 2, the paper provides an overview of ILP systems gathered at the ILPNET system repository. As this paper is basically a report on ILPNET, its emphasis is on systems developed by ILPNET members. Descriptions of systems developed outside ILPNET are marked with an asterisk (*). For each system, a short description is given which summarizes its basic principles, the functionality and options it offers, and the input it expects. To facilitate the selection of a suitable system, the hardware and software prerequisites for running the system are described as well.

Section 3 describes a variety of ILP-related datasets that have been made available as a result of ILPNET activities. Most come from several broad application areas of molecular biology, finite element mesh design, natural language processing, the area of modelling, diagnosis and control, and chess. Sample datasets that illustrate the input/output performance of several ILP systems have been also made available, as well as several datasets that do not fall in the above application areas and are gathered under the heading 'miscellaneous'. Again, datasets not originating from the ILPNET consortium are marked with an asterisk (*).

Section 4 gives an exhaustive list of bibliographic items. There is no distinction between ILPNET and non-ILPNET contributions.

Despite its inevitable incompleteness, we believe that this overview contributes to the better understanding and accessibility of the results of ILP research to the interested readers.

Acknowledgement

We are grateful to the European Union for the grant supporting the ILPNET activities (network no. CP93-44) and for the support of ILP research (ESPRIT projects no. 6020 and no. 20237). Our thanks goes also to our local institutions and grants supporting this activity, as well as to the individual researchers who contributed to the results of ILPNET presented in this paper.

2 ILP systems

A common classification of ILP systems distinguishes between empirical ILP systems and interactive ILP systems [322, 105]. Empirical ILP systems are characterised as non-interactive batch learners inducing definitions for single predicates from scratch. Interactive systems, also called incremental systems, interactively learn multiple predicates, possibly starting with a preliminary incomplete or inconsistent theory. This classification is not to be taken too rigidly but rather as indicating two poles of the spectrum of exisiting ILP systems. To better fit the situation encountered at the ILP systems repository, we further divide the class of empirical systems according to the number of examples the systems expect to process. In the refined classification, the term *empirical ILP systems* then refers to single-predicate batch learning systems which are able to analyse large example sets requiring little or no user guidance.

Interactive ILP systems incrementally build complex domain theories consisting of multiple predicates where the user controls and initiates the subsequent steps of the model construction and refinement process. These systems usually offer a graphical user interface.

The third group of systems learns from small example sets in batch mode. One of the systems we assign to this group, namely FILP [41, 43] queries the user for missing examples, but as this interaction takes place in advance to induction, and as the induction proceeds autonomously once the example set is completed, the learning algorithm in essence performs batch-learning. The system MARKUS [244] is a non-interactive theory revisor learning single predicates, and consequently fits neither into the class of empirical ILP systems nor into the class of interactive ILP systems. Thus it seems natural to distinguish a third class of systems. As these systems require only small example sets and little user guidance, this class of systems may qualify as *programming assistants.*

Recently, the interest in ILP research has extended to include alternative task settings besides the classical ILP concept-learning task. The ILP systems repository includes two systems which are commonly assigned to the non-classical approach. They are described at the end of this section.

The emphasis of this report is on systems developed by ILPNET members. Descriptions of systems developed outside ILPNET are marked with an asterisk (*). These systems are described in less detail, expect for the system FOIL [452] which is one of the best-known and successful empirical ILP systems and has inspired much of further ILP research.

2.1 Empirical ILP systems

2.1.1 FOIL*

FOIL [452, 457] is a system for learning intensional concept definitions from relational tuples. The induced concept definitions are represented as function-free Horn clauses, optionally containing negated body literals. The background knowledge predicates are represented extensionally as sets of ground tuples. FOIL employs a heuristic search strategy which prunes vast parts of the hypothesis space.

System:	FOIL
Version:	6.4
Further specification:	empirical ILP system
Pointers:	`ftp://ftp.cs.su.oz.au/pub`
Code:	C source code, to be compiled using the enclosed `Makefile`
References:	[452], [457]
Other comments:	The release also includes a conversion program for transforming C4.5 input files into the FOIL format.

Table 1: FOIL: short description of the system.

As its general search strategy, FOIL adopts a covering approach. Induction of a single clause starts with a clause with an empty body which is specialised by repeatedly adding a body literal to the clause built so far. As candidate body literals, FOIL considers the literals which are constructed by variabilising the known predicates, that is, by distributing variables to the argument places of background knowledge predicates. Additionally, FOIL takes into account literals stating (un)equality of variables. Furthermore, literals may contain constants which the user has declared as theory (i.e. relevant) constants.

All literals conform to the type restrictions of the predicates. For further control of the language bias, FOIL provides parameters limiting the total number and maximum depth of variables in a single clause. In addition, FOIL incorporates mechanisms for excluding literals which might lead to endless loops in recursive hypothesis clauses. FOIL offers limited number handling capabilities and generates literals comparing numeric variables to each other or to thresholds it has derived.

Among the candidate literals, FOIL selects one literal to be added to the body of the hypothesis clause. The choice is determined by the information gain heuristic. The gain heuristic is an information-based measure estimating the utility of a literal in dividing positive from negative examples. FOIL stops adding literals to the hypothesis clause if the clause reaches the predefined minimum accuracy or if the encoding length of the clause exceeds the number of bits needed for explicitly encoding the positive examples it covers. This second stopping criterion prevents the induction of overly long and specific clauses in noisy domains. Induction of further hypothesis clauses stops if all positive examples are covered or if the set of induced hypothesis clauses violates the encoding length restriction. In a postprocessing stage, FOIL removes unneccessary literals from induced clauses as well as redundant clauses from the concept definition.

FOIL's greedy search strategy makes it very efficient, but also prone to exclude the intended concept definitions from the search space. Some refinements of the hill-climbing search alleviate its short-sightedness, such as including a certain class of literals with zero information gain into the hypothesis clause and a simple backtracking mechanism.

FOIL is a batch learning system which reads in all learning input from a single input file. For learning, positive as well as negative examples are required. A user may provide negative examples explicitely or, alternatively, instruct FOIL to construct negative examples automatically according to the Closed World Assumption (CWA). In the latter case, the set of positive examples must be complete up to a certain example complexity. For predicates with high arity, the CWA may generate a huge number of negative examples. FOIL offers a command line option allowing the user to specify the percentage of randomly-selected negative examples to be used for induction.

Examples and background knowledge for FOIL have to be formatted as tuples, that is, each ground instance of a predicate is represented as a sequence of argument values. For each predicate, the user provides a header defining its name and argument types. Optionally, the user may indicate the input/output mode of the predicates, thus further limiting the number of literals constructed by FOIL.

For convenient testing of the induced hypothesis, the user may provide test cases (i.e. classified examples) for the target predicates together with the learning input. FOIL then checks the

hypothesis on these cases and reports the results.

2.1.2 MFOIL

System:	mFOIL
Version:	
Further specification:	empirical ILP system
Pointers:	`http://www.gmd.de/ml-archive/ILP/public/software/mfoil`
Code:	Quintus Prolog source code
References:	[164], [322]
Other comments:	The implementation of mFOIL runs considerably slower than FOIL.

Table 2: MFOIL: short description of the system.

The system MFOIL [164, 322] is a descendant of FOIL which aims at improving its noise handling capacities which are of crucial importance when processing imperfect real-world datasets.

MFOIL integrates several noise-handling techniques from attribute-value learning approaches into FOIL. It offers two alternative accuracy-based search heuristics replacing FOIL's entropy-based information gain criterion, namely the Laplace-estimate and the more sophisticated m-estimate. The m-estimate takes into account the prior probabilities of examples, leading to a more reliable criterion for small example sets. The user-settable parameter m allows to control the influence of the prior probabilities. In MFOIL, FOIL's encoding-length based stopping criteria are replaced by criteria relying on statistical significance testing.

Further differences between FOIL and MFOIL concern the search strategy and the background knowledge. As FOIL, MFOIL adopts a covering strategy, but, unlike FOIL, it conducts beam search in order to overcome at least partially some of the disadvantages of FOIL's greedy hill-climbing search. On the other hand, some of FOIL's more advanced features, such as number handling, are not realised in MFOIL. Whereas FOIL is restricted to ground background knowledge, MFOIL is able to process intensionally defined background predicates as well. Furthermore, compared to FOIL, MFOIL allows the user to declare additional informations on the background predicates which reduce the number of possible body literals constructed during induction and thus help to gain efficiency.

2.1.3 GOLEM

System:	GOLEM
Version:	
Further specification:	empirical ILP system
Pointers:	`http://www.comlab/ox.ac.uk/oucl/groups/machlearn/golem.html` `http://www.gmd.de/ml-archive/ILP/public/software/golem`
Code:	C executable, running on Sun SparcStations
References:	[405]
Other comments:	A tar-file is available, containing the source files, a README file explaining the usage of GOLEM and some example datasets. GOLEM source can be obtained from both WWW sites.

Table 3: GOLEM: short description of the system.

As FOIL, GOLEM [405] is a 'classic' among empirical ILP systems. It has been applied successfully on real-world problems such as drug design [295] and finite element mesh design [160].

GOLEM copes efficiently with large datasets. It achieves this efficiency because it avoids searching a large hypothesis space for consistent hypotheses like, for instance, FOIL, but rather constructs a unique clause covering a set of positive examples relative to the available background knowledge. The principle is based on the relative least general generalisations (rlggs) introduced by Plotkin [440, 441]. GOLEM embeds the construction of rlggs in a covering approach. For the induction of a single clause, it randomly selects several pairs of positive examples and computes their rlggs. Among these rlggs, GOLEM chooses the one which covers the largest number of positive examples and is consistent with the negative examples. This clause is further generalised. GOLEM randomly selects a set of positive examples and constructs the rlggs of each of these examples and the clause obtained in the first construction step. Again, the rlgg with the greatest coverage is selected and generalised by the same process. The generalisation process is repeated as long as the coverage of the best clause stops increasing. GOLEM conducts a postprocessing step, which reduces induced clauses by removing irrelevant literals.

In the general case, the rlgg may contain infinitely many literals. Therefore, GOLEM imposes some restrictions on the background knowledge and hypothesis language which ensure that the length of rlggs grows at worst polynomially with the number of positive examples. The background knowledge of GOLEM is required to consist of ground facts. For the hypothesis language, the determinacy restriction applies, that is, for given values of the head variables of a clause, the values of the arguments of the body literals are determined uniquely. The complexity of GOLEM's hypothesis language is further controlled by two parameters, i and j, which limit the number and depth of body variables in a hypothesis clause.

GOLEM learns Horn clauses with functors. It may be run as a batch learner or in interactive mode where the induction can be controlled manually. GOLEM is able to learn from positive examples only. Negative examples are used for clause reduction in the postprocessing step, as well as input/output mode declarations for the predicates the user may optionally supply. For dealing with noisy data, GOLEM provides a system parameter enabling the user to define a maximum number of negative examples a hypothesis clause is allowed to cover.

2.1.4 LINUS

System:	LINUS
Version:	
Further specification:	empirical ILP system
Pointers:	`http://www.gmd.de/ml-archive/ILP/public/software/linus`
Code:	Quintus Prolog source code
References:	[312], [325], [325], [322]
Other comments:	The distribution includes an executable of CN2 running with SunOS 4.1.3.

Table 4: LINUS: short description of the system.

LINUS [325, 322] is an ILP learner which incorporates existing attribute-value learning systems. The idea is to transform a restricted class of ILP problems into propositional form and solve the transformed learning problem with an attribute-value learning algorithm. The propositional learning result is then re-transformed into the first-order language. On the one hand, this approach enhances the propositional learners with the use of background knowledge and the more expressive hypothesis language. On the other hand, it enables the application of successful propositional learners in a first-order framework. As various propositional learners can be integrated and accessed via LINUS, LINUS also qualifies as an ILP toolkit offering several learning algorithms with their specific strengths. The present distribution of LINUS provides interfaces to the attribute-value learners ASSISTANT, NEWGEM, and CN2. Other propositional learners may be added.

LINUS can be run in two modes. Running in CLASS mode, it corresponds to an enhanced attribute-value learner. In RELATION mode, LINUS behaves as an ILP system. Here, we focus on the RELATION mode only.

The basic principle of the transformation from first-order into propositional form is that all body literals which may possibly appear in a hypothesis clause (in the first-order formalism) are determined, thereby taking into account variable types. Each of these body literals corresponds to a boolean attribute in the propositional formalism. For each given example, its argument values are substituted for the variables of the body literal. Since all variables in the body literals are required to occur also as head variables in a hypothesis clause, the substitution yields a ground fact. If it is a true fact, the corresponding propositional attribute value of the example is true, and false otherwise. The learning results generated by the propositional learning algorithms are retransformed in the obvious way. The induced hypotheses are compressed in a postprocessing step.

In order to enable the transformation into propositional logic and vice versa, some restrictions on the hypothesis language and background knowledge are necessary. As in most systems, training examples are ground facts. These may contain structured, but nonrecursive terms. Negative examples can be stated explicitly or generated by LINUS according to the CWA. LINUS offers several options for controlling the generation of negative examples.

The hypothesis language of LINUS is restricted to constrained deductive hierarchical database clauses, that is, to typed program clauses with nonrecursive predicate definitions and nonrecursive types where the body variables are a subset of the head variables. Besides utility functions and predicates, hypothesis clauses consist of literals unifying two variables (X = Y) and of literals assigning a constant to a variable (X = a). Certain types of literals may appear in negated form in the body of a hypothesis clause.

Background knowledge has the form of deductive database clauses, that is, possibly recursive program clauses with typed variables. The variable type definitions which are required to be non-recursive have to be provided by the user. The background knowledge consists of two types of predicate definitions, namely utility functions and utility predicates. Utility functions are predicates which compute a unique output value for given input values. The user has to declare their input/output mode. When occuring in an induced clause, the output arguments are bound to constants. Utility predicates are boolean functions with input arguments only. For a given input, these predicates compute true or false.

An empirical comparison of FOIL, MFOIL and GOLEM can be found in [164]. [175] provides an empirical comparison of LINUS and FOIL.

2.1.5 PROGOL

System:	PROGOL
Version:	4.1
Further specification:	empirical ILP system
Pointers:	`ftp://ftp.comlab.ox.ac.uk/pub/Packages/ILP/PROGOL4.1/`
Code:	C source code
References:	[395], [396], [398]
Other comments:	Distributed together with a manual and examples. Freely available for academic research. Also available under license for commercial research. Version 4.2 [398] for learning from positive-only data is also available.

Table 5: PROGOL: short description of the system.

The system PROGOL [395, 396] provides the user with a standard Prolog interpreter augmented with inductive capacities. PROGOL can be run interactively or in batch mode. In

interactive mode, PROGOL behaves similar to a standard Prolog interpreter allowing the user to pose queries or assert new clauses. Additionally, the user can request the system to generalise the examples. In batch mode, PROGOL is called from the operating shell with the name of an input file containing examples and background knowledge as an argument.

PROGOL employs a covering approach like, e.g., FOIL. That is, it selects an example to be generalised and finds a consistent clause covering the example. All clauses made redundant by the found clause including all examples covered by the clause are removed from the theory. The example selection and generalisation cycle is repeated until all examples are covered. When constructing hypothesis clauses consistent with the examples, PROGOL conducts a general-to-specific search in the theta-subsumption lattice of a single clause hypothesis. In contrast to other general-to-specific searching systems, PROGOL computes the most specific clause covering the seed example and belonging to the hypothesis language. This most specific clause bounds the theta-subsumption lattice from below. On top, the lattice is bounded by the empty clause. The search strategy is an A^*-like algorithm guided by an approximate compression measure. Each invocation of the search returns a clause which is guaranteed to maximally compress the data, however, the set of all found hypotheses is not necessarily the most compressive set of clauses for the given example set. PROGOL can learn ranges and functions with numeric data (integer and floating point) by making use of the built-in predicates "is", $<$, $=<$, etc.

The hypothesis language of PROGOL is restricted by the means of mode declarations provided by the user. The mode declarations specify the atoms to be used as head literals or body literals in hypothesis clauses. For each atom, the mode declaration indicates the argument types, and whether an argument is to be instantiated with an input variable, an output variable, or a constant. Furthermore, the mode declaration bounds the number of alternative solutions for instantiating the atom. The types are defined in the background knowledge by unary predicates, or by Prolog built-in functions.

PROGOL's syntax for examples, background knowledge and hypotheses is Dec-10 Prolog with the usual augmentable set of prefix, postfix and infix operators. However, unlike in the Edinburgh DEC-10 Prolog syntax, a distinction is drawn in Progol between assertions, which are terminated in a "." and queries, which are terminated in a "?". Arbitrary Prolog programs are allowed as background knowledge. Besides the background theory provided by the user, standard primitive predicates are built into PROGOL and are available as background knowledge. Positive examples are represented as arbitrary definite clauses. Negative examples and integrity constraints are represented as headless Horn clauses. Using negation by failure (CWA), PROGOL is able to learn arbitrary integrity constraints. For instance, a clause $man(X) \vee woman(X) \leftarrow normal(X)$, represented as the Prolog integrity constraint `:- normal(X), not(man(X)), not(woman(X)).` can be learned using the four mode declarations `:- modeh(1,false)?`, `:- modeb(1,normal(+p))?`, `:- modeb(1,not(man(+p)))?`, and `:- modeb(1,not(woman(+p)))?`, from examples like `:- normal(leslie1).`

PROGOL provides a range of parameters for controlling the generalisation process. These parameters specify the maximum cardinality of hypothesis clauses, a depth bound for the theorem prover, the maximum layers of new variables, and an upper bound on the nodes to be explored when searching for a consistent clause. PROGOL allows to relax consistency by setting an upper bound on the number of negatives that can be covered by an acceptable clause.

PROGOL4.2 [398] is an upward compatible with version 4.1 but learns from positive-only data. The system is available from the author upon request. Papers [398] and [397], describing PROGOL4.2, can be obtained as Postscript by anonymous `ftp` from `ftp.comlab.ox.ac.uk` in files `pub/Packages/ILP/Papers/poslearn1.ps` and `pub/Packages/ILP/Papers/slp.ps`, respectively.

2.1.6 SPECTRE

SPECTRE (SPECialization by TRansformation and Elimination) [64] is an empirical ILP system that can handle large example sets very efficiently.

System:	SPECTRE
Version:	1.0
Further specification:	empirical ILP system
Pointers:	`http://www.dsv.su.se/~henke/SPECTRE/SPECTRE.html`
Code:	SICStus Prolog 3.1
References:	[64]
Other comments:	Provided both as a stand-alone application for SUN OS 4 and as a SICStus Prolog object file (requiring SICStus 3.1).

Table 6: SPECTRE: short description of the system.

Given sets of positive and negative examples and an overly general initial theory, that is, a theory which covers all positive and some of the negative examples, SPECTRE specialises an overly general initial theory in order to find a hypothesis which entails all positive examples but no negative examples.

SPECTRE employs a divide-and-conquer technique to specialize the overly general hypothesis until no negative examples are covered. Specialisation of the theory is performed by combining clause removal with the transformation rule unfolding. When SPECTRE finds a clause that covers a negative example and no positive examples, it removes the clause. When it finds a clause that covers both negative and positive examples, it unfolds the clause. Unfolding a clause requires the selection of a literal of the clause. The clause is resolved with all clauses in the theory with heads unifying with the selected literal. Then, the clause is replaced by the resulting resolvents. The choice of which literal to unfold upon is made such that the entropy of the resolvents is minimized. The resulting hypothesis consists of those clauses that cover positive examples only.

SPECTRE uses the overly general theory as a declarative bias that not only restricts what predicate symbols may occur in bodies of learned clauses, but also how these can be invoked. Specialised clauses defining the target predicate can only contain literals which occur in the initial target predicate definition or in clauses resolving with the initial or intermediate target predicate definitions. Therefore, the initial theory is crucial for the success of the induction.

In order to run SPECTRE, two input files are needed, namely a theory file containing the overly general theory in the form of a Prolog program in the Edinburgh syntax, and an example file with unit clauses in the Edinburgh syntax defining the positive and negative examples as arguments of the predicates `pos/1` and `neg/1`, respectively. SPECTRE assumes that the target predicate is non-recursive, i.e., the target predicate is not allowed to appear in bodies of clauses. Background predicates, however, may be recursive. SPECTRE provides a graphical interface. No system parameters need to be set.

2.1.7 MERLIN

System:	MERLIN
Version:	1.0
Further specification:	empirical ILP system
Pointers:	`http://www.dsv.su.se/~henke/MERLIN/MERLIN.html`
Code:	SICStus Prolog 3.1
References:	[63]
Other comments:	Provided both as a stand-alone application for SUN OS 4 and as a SICStus Prolog object file (requiring SICStus 3.1).

Table 7: MERLIN: short description of the system.

MERLIN (Model Extraction by Regular Language INference) [63] is a non-interactive, multiple

predicate learning system that has the ability to invent new predicates. Like SPECTRE, it uses an overly general hypothesis in the form of a logic program together with sets of positive and negative examples in order to find an inductive hypothesis which entails all positive examples but no negative examples.

The basic idea of the approach is to learn finite-state automata that represent allowed sequences of resolution steps. MERLIN first finds SLD-refutations for all examples using the overly general hypothesis, and then tries to find the minimal finite-state automaton that can generate all sequences of input clauses in the SLD-refutations of the positive examples and no sequences of input clauses in the SLD-refutations of the negative examples. After having learned the automaton, MERLIN produces a new theory that allows only those sequences of resolution steps which are allowed by both the initial theory and the learned automaton. This is done by calculating the intersection of the automaton and a grammar that corresponds to the overly general hypothesis. The intersection is used to derive the final hypothesis in the form of a logic program. During this process MERLIN may introduce new predicate symbols, i.e., it carries out a form of predicate invention.

As the induced hypothesis allows only such resolution sequences which are possible in the initial theory, the initial theory defines MERLIN's language bias. For instance, as the resolution sequences produced by the new theory involve only those clauses which are also part of resolution sequences allowed by the initial theory (besides clauses defining the newly introduced predicates), it is important that all relevant clauses occur in the resolution sequences produced by the initial theory.

MERLIN expects as input a theory file containing the initial theory in form of a Prolog program in Edinburgh syntax, and an example file with unit clauses that define the positive and negative examples as arguments of the predicates `pos/1` and `neg/1`, respectively. The arguments of the predicates pos/1 and neg/1 are instances of the same atom. MERLIN assumes that each positive example has at least one SLD-refutation such that there is no negative example with an SLD-refutation that has the same sequence of input clauses.

MERLIN provides a graphical interface. No parameters are to be set.

2.1.8 FOIDL*

System:	FOIDL*
Version:	1.0 Alpha
Further specification:	empirical ILP system
Pointers:	`ftp://ftp.cs.utexas.edu/pub/mooney/foidl/`
Code:	Quintus Prolog (version 3.1.3) source code and two LISP implementations
References:	[362], [363]
Other comments:	The Prolog version only creates non-recursive programs and always creates decision lists, so it is useful only for functional, non-recursive concepts. Two Lisp implementations are available, one version incorporating a Prolog interpreter, the other incorporating a Prolog compiler.

Table 8: FOIDL: short description of the system.

FOIDL [362, 363] is a descendant of FOIL differing from its predecessor in the following three ways. First, FOIDL is able to process intensionally defined background knowledge. Second, it substitutes the assumption of output completeness for explicit negative examples. The output completeness assumption requires that a mode declaration for the target predicate is given. It states that for every unique input pattern appearing in the training set, all correct output patterns occur in the examples in training the set. Together with the mode declaration, the positive examples then implicitly determine the negative examples. The third difference between FOIDL and FOIL

is that FOIDL supports the induction of decision lists. A decision list is an ordered set of clauses each ending with a cut. When answering a query, the decision list returns the answer of the first clause in the ordered set which succeeds in answering the query. FOIDL generates the clauses in the decision list in reverse order, that is, clauses learned first appear at the end of the decision list. As the covering algorithm tends to learn more general clauses covering many positive examples first the more general clauses are placed as default cases at the end of the decision list.

2.1.9 FOCL*

System:	FOCL*
Version:	
Further specification:	empirical ILP system
Pointers:	`http://www.ics.uci.edu/AI/ML/FOCL.html`
Code:	Common Lisp source code
References:	[436]
Other comments:	Available also as a Macintosh application, including a graphical interface to the machine learning program that shows the search space explored by FOCL, so it is a useful pedagogical tool for explaining inductive and explanation-based learning. It also provides facilities for creating and graphically editing knowledge-bases, tracing rules, and generating explanations, so the Mac version may be used as an expert system shell.

Table 9: FOCL: short description of the system.

The system FOCL [436] learns Horn clause programs from examples and, optionally, background knowledge. It integrates an explanation-based learning component with the inductive learning approach of FOIL. FOCL is able to use intensionally defined background knowledge and accepts as input a partial, possibly incorrect rule as an approximation of the target predicate. User-defined constraints which realise a declarative language bias allow to restrict the search space.

2.1.10 HYDRA*

System:	HYDRA*
Version:	
Further specification:	empirical ILP system
Pointers:	`http://www.ics.uci.edu/~mlearn/HYDRA.html`
Code:	Common List source code
References:	[21]
Other comments:	

Table 10: HYDRA: short description of the system.

The relational concept learner HYDRA [21] extends the machine learning program FOCL by adding likelihood ratios to the induced classification rules. HYDRA learns a concept description for each class. The concept descriptions compete to classify test examples using the likelihood ratios assigned to clauses of that concept description. This reduces the algorithm's susceptibility to noise.

2.1.11 FORTE*

System:	FORTE*
Version:	
Further specification:	empirical ILP system
Pointers:	ftp://ftp.cs.utexas.edu/pub/mooney/forte/
Code:	Quintus Prolog source code
References:	[460]
Other comments:	

Table 11: FORTE: short description of the system.

FORTE [460] (First Order Revision of Theories from Examples) is a system for automatically revising function-free first-order Horn clause theories. FORTE integrates a collection of specialisation and generalisation operators and conducts an iterative hill-climbing search through the space of revision operations. The system includes the operators delete-antecedent and delete-rule, adopted from propositional theory revision, FOIL-like operators for adding antecedents and new rules, two generalisation operators based on inverse resolution, and an antecedent-adding operator termed 'relational pathfinding'. Each iteration of the search identifies all possibilities for applying the operators. The revision operation resulting in a maximum increase in theory accuracy is performed. The revision process continues as long as revisions produce an improvement in accuracy or a reduction in theory size.

2.1.12 CHILLIN*

System:	CHILLIN*
Version:	1.0 Alpha
Further specification:	empirical ILP system
Pointers:	ftp://ftp.cs.utexas.edu/pub/mooney/chillin/
Code:	Quintus Prolog (version 3.1.3) source code
References:	[591]
Other comments:	In the actual implementation, it employs a modified search strategy in order to gain efficiency. It is able to handle induction problems with thousands of examples when running on a SparcStation 2.

Table 12: CHILLIN: short description of the system.

CHILLIN [591] is an ILP algorithm combining elements of top-down and bottom-up induction methods. CHILLIN's input consists of sets of ground facts representing positive and negative examples, and a set of background predicates expressed as definite clauses. Examples may contain functors. Basically, CHILLIN tries to construct a small, simple theory covering the positive, but not the negative examples by repeatedly compacting its current version of the program. Compactness is measured as the syntactic size of the theory.

The algorithm starts with a most specific theory, namely the set of all positive examples. Then it generalises the current theory, aiming to find a generalisation which allows to remove a maximum number of clauses from the theory while all positive examples remain provable.

Similar to GOLEM's approach, the generalisation algorithm finds a random sampling of pairs of clauses in the current program. These pairs are generalised by constructing their least-general-generalisations under theta-subsumption. If a generalisation covers negative examples, it is specialised by adding antecedents using a FOIL-like algorithm. If the specialisation with background predicates is not sufficient for preventing negative examples from being covered, CHILLIN tries to invent new predicates for further specialisation of the clause. At each step, CHILLIN considers a number of possible generalisations and implements the one that best compresses the theory.

CHILLIN is able to learn recursive predicates. It avoids generating theories leading to endless recursion by imposing syntactic restrictions on recursive predicates. However, CHILLIN may learn recursive predicates covering negative examples.

2.2 Programming assistants

2.2.1 FILP

System:	FILP
Version:	
Further specification:	programming assistant
Pointers:	`http://www.gmd.de/ml-archive/ILP/public/software/filp`
Code:	C Prolog source code
References:	[41], [43]
Other comments:	The C-Prolog interpreter is provided together with the system and runs on a SUN SPARCstation 1 as well as on a SUN4/200 under SunOS, but has problems running under Solaris.

Table 13: FILP: short description of the system.

FILP [41, 43] is an interactive system which learns functional logic programs. Functional means that for each sequence of input values for a predicate there is exactly one sequence of output values the predicate produces. This restriction applies to the induced predicates as well as to the predicates defined in the background knowledge. The functions are required to be total, i.e., for any input, an output must exist. The restriction to functional programs does not significantly affect the expressive power as any computable function can be represented by a functional logic program. The requirement that functions should be total is more restrictive. However, in practise some non-total functions can be learned as well (for instance, quicksort not using the append predicate quicksort(X,Acc,Y)) provided that appropriate examples and background knowledge are specified.

The restriction to functional programs is central to the approach. As an explicit restriction of the set of allowed hypotheses, it makes the learning task a lot easier, since it excludes a priori many clauses which otherwise must be generated and checked against the examples. Furthermore, it enables FILP to learn from positive examples only, since negative examples of the behaviour of induced predicates are implicitly given as the ones with the same input values but with different output values than the positive examples.

Unlike some other approaches, FILP does not require an example set which is complete up to a certain example complexity for the predicates to be learned, since, due to the functionality requirement, FILP can determine the examples needed for learning besides the given ones. For collecting the missing information, FILP queries the user. It presents examples with instantiated input values, and the user fills in the corresponding output values. This allows to start learning with a very limited number of initial examples, and more examples are added on request. This interactive way of example input is more convenient for the user than specifying the whole set of examples in advance, since FILP asks for all and only for the examples it really needs.

The background knowledge can be defined intensionally or extensionally. When using extensional background knowledge, FILP collects missing information on these background predicates as well. FILP is able to induce an intensional definition for such background predicates, thus realizing multiple predicate learning. FILP provably learns complete and consistent programs, that is, programs that cover all positive and no negative examples. Furthermore, if a complete and consistent program exists, FILP is guaranteed to find it. This is not the case for other approaches using extensionally defined background knowledge without example completion.

FILP works with flattened clauses, where functions are transformed into predicates. Besides the example set and background knowledge, the information FILP needs for learning includes a set of all the literals which may occur as body literals in the definition of the induced predicate. This literal set determines the hypothesis space for learning. As it affects the efficiency of learning and the number of literals FILP requests, this literal set should be specified carefully.

Furthermore, the user has to provide mode declarations for the induced predicate as well as for all background predicates. These mode declarations specify which arguments of a predicate are input arguments and which are output arguments. An aditional means for the user to influence induction is to declare 'forbidden clauses', that is clauses which must not be part of a solution even if satisfied by the positive examples.

FILP learns list manipulation programs such as member, quicksort or reverse, which are induced within a few seconds. For example, learning a functional variant of the member predicate member(Elem,List,yes/no) required four examples and about 12 seconds [43].

2.2.2 LILP

System:	LILP
Version:	
Further specification:	programming assistant
Pointers:	`http://www.gmd.de/ml-archive/ILP/public/software/lilp`
Code:	Poplog Prolog source code
References:	[343], [344]
Other comments:	Written for Poplog Prolog with Dec10 library. Running it with Quintus Prolog requires some minor adaptions such as adding `dynamic` statements and loading libraries.

Table 14: LILP: short description of the system.

The system LILP [343] encorporates an approach to concept learning from positive-only examples whose basic technique is borrowed from *lambda*-calculus. It relies on a generality ordering between Horn clauses called λ-subsumption, which is stronger than θ-subsumption and weaker than generalized subsumption. λ-subsumption allows to compare clauses in a local sense, i.e., with respect to a partial interpretation of the background knowledge. Consequently, the locality of λ-subsumption allows to search for clauses which are correct with respect to a small subset of the set of atoms they generally cover.

Induction of a single clause starts with a seed example. Variabilizing one argument of the seed example and keeping the other arguments fixed results in a logical expression which can be viewed as a function of the variable, say X. This function maps instances of X on *true* if the resulting instantiation of the expression matches an example, and on *false* otherwise. The set of *true* instances forms the λ *calculus model* of the expression. The algorithm searches a set of body literals which defines the set of true instances of the variable. This is done for each head argument in turn, and the found body literals are combined to a clause body. If more than one clause for a seed example is found, the system keeps the best one according to the proof complexity criterion. This technique is embedded into a covering algorithm. LILP postprocesses the clause set in order to remove redundant clauses. It does not alert the user if is not able to induce a clause set covering all positive examples.

Like FILP, LILP does not assume a Closed World in the sense that the set of positive examples up to a given example complexity must be complete. Rather, complete λ-models are required. LILP learns from positive examples, but is able to utilize negative examples if available. By providing negative examples, the user can force LILP to allow singleton head variables or to induce the more specific predicate definition when several predicate definitions of varied generality fit the given data.

When using LILP, the user has to specify the name and arity of predicates to be used as background knowledge as well as constants occuring in the examples and not be variabilised in the predicate definition, thus defining the system's language bias. The hypothesis language is further adjusted by two system parameters, one specifying the maximum number of body literals determining a single head argument, the other defining the maximum number of determinate literals in a clause. As FILP, LILP works with flattened clauses where functions are defined by predicates. Structured terms may occur, but are treated as constants. Due to implementation reasons, lists must not occur in examples. However, at a deeper level they are allowed.

The system is so fast as to induce, e.g., a definition for quicksort from extensionally defined background knowledge in less than a second.

2.2.3 MARKUS

System:	MARKUS
Version:	
Further specification:	programming assistant
Pointers:	`http://www.gmd.de/ml-archive/ILP/public/software/markus/`
Code:	Prolog source code
References:	[244], [245]
Other comments:	The system runs without modifications on Quintus Prolog (Vax, Sun), SICStus Prolog (Sun, HP) and Arity Prolog (IBM PC).

Table 15: MARKUS: short description of the system.

MARKUS [244, 245], a derivative of Shapiro's Model Inference System MIS [496], is a system for inducing Prolog programs from positive and negative examples. Like FILP and LILP, MARKUS employs a covering strategy. Unlike these systems, MARKUS induces clauses with functions and is able to start induction with a preliminary incomplement or inconsistent definition of the target predicate which is then refined.

For inducing single clauses, MARKUS searches a refinement graph. The search starts with the most general clause which is specialised by applying the refinement operators taken over from MIS. In contrast to MIS, MARKUS generates an optimal refinement graph, i.e., a refinement graph without duplicate nodes, thus improving efficiency. The refinement graph is searched by iterative deepening search. When MARKUS encounters a clause covering at least one yet uncovered positive and no negative example, the clause is added to the predicate definition. Redundant clauses are removed from the predicate definition.

As MARKUS realizes an exhaustive search, restricting the hypothesis space is crucial for learning. Therefore, MARKUS offers elaborate mechanisms for explicit and implicit declaration of the language bias. First of all, the language bias which is defined implicitly by the refinement operator can be adjusted to the learning task by choosing the appropriate refinement operator, as MARKUS offers a refinement operator for learning DCG clauses and a general refinement operator for inducing logic programs. The application of the general refinement operator is guided by input/output mode declarations and type definitions for the arguments of the target and background predicates specified by the user. A set of parameters determines the form of clauses generated by the refinement operator. E.g., these parameters allow to define the maximum number of body literals, and the maximum structure depth of head arguments. For further tuning of the search space, the user can specify various concrete syntactic restrictions within individual type and background predicate definitions in a uniform manner. E.g., these restrictions make it possible to define argument symmetry or to prevent particular combinations of literals.

MARKUS is a non-interactive learner, learning from positive and negative examples which offers various options for adjusting the system bias. It uses logic with functors and is able to

generate hypothesis clauses with negated body literals. The background knowledge is defined intensionally. The user has to provide mode declarations and type definitions for the arguments of target and background predicates.

With default parameter settings, MARKUS learns quicksort from 4 positive and 3 negative examples in 24.317 seconds on a Sun4/100.

2.3 Interactive ILP systems

2.3.1 MOBAL

System:	MOBAL
Version:	
Further specification:	interactive ILP system
Pointers:	`http://nathan.gmd.de/projects/ml/mobal/mobal.html`
Code:	executable, running on Sun Sparcs (sun arch 4) only
References:	[375]
Other comments:	Its interface is based on Tcl/Tk. The windowing system (Open Windows, Motif, etc.) is irrelevant as long as it is based on X11. The current version of the system is developped under SunOS 4.1.*, but running it under Solaris has shown to be unproblematic as well.

Table 16: MOBAL: short description of the system.

MOBAL [375] is a knowledge acquisition environment which assists the user in developing a model of an application domain in a first-order logical representation formalism. It implements the balanced cooperative modeling paradigm where knowledge acquisition is viewed as a cyclic and highly interactive modeling process. The system comes with a convenient graphical user interface providing means for manual input and inspection of a domain model, and access to a range of tools covering many of the substasks involved in knowledge acquisition, e.g., automated discovery of rules, knowledge revision, and theory restructuring. Additionally, MOBAL facilitates integration of external ILP tools, thus extending its own method pool as well as adding an interactive graphical interface to non-interactive ILP programs.

MOBAL's internal learning algorithm RDT solves the ILP task by inducing rules from positive and negative examples. The negative examples can be listed explicitly by the user or, optionally, can be defined implicitly via the Closed World Assumption. The logical dialect used by RDT is the function-free subset of Horn clause logic extended by negated literals. Negation is not treated as negation by failure as, for instance, in FOIL, but rather as proper negation, i.e., the negation of an atom is considered as true only when there exists a corresponding negated fact.

RDT requires extensionally defined background knowledge for learning. If intensional predicate definitions are entered into the system, MOBAL's built-in inference engine efficiently computes the corresponding extensional predicate definitions up to a prespecified depth limit. RDT's hypothesis space is spanned by a set of rule models specified by the user. A rule model is a rule where predicate variables replace actual domain predicates. RDT searches the hypothesis space defined by the given rule models by instantiating the predicate variables with compatible domain predicates.

Compatibility of predicates is defined with respect to a sort taxonomy and predicate topology holding in the domain model. The sort taxonomy divides the arguments of predicates into classes. The predicate topology reflects the inferential structure of the domain, i.e., it states which predicates are useful for defining other predicates. Ensuring that argument sorts and predicates in an instantiated rule model are compatible excludes useless rules from the hypothesis space. User-defined parameters control when to accept a rule as a hypothesis clause. Among others, these parameters take into account the number of positive and negative instances of the rule.

Rule models, sort taxonomy, and predicate topology enable very fine-grained adjustment of RDT's hypothesis space with explicit user control, however at the price that their specification can be quite demanding. Assistance in this task is offered by MOBAL's tools MAT (Model Acquisition Tool), PST (Predicate Structuring Tool), and SST (Sort Taxonomy Tool), which automatically derive rule models, a predicate topology, and a sort taxonomy from the current domain model. These provide insight into the structure of the domain model evolved so far and serve as a starting point for the specification of the structure of the intended domain model.

Further important components of MOBAL are a knowledge revision tool (KRT), a concept learning tool (CLT), and a theory restructuring tool (RRT). KRT assists in correcting inconsistencies which may arise while gradually developing a knowledge base. MOBAL's concept learning component CLT learns concept definitions from examples. Whereas rule induction algorithms such as RDT or FOIL usually produce rules stating sufficient conditions for a concept, CLT also searches for necessary conditions as well as for all other rules that use the concept. These tools interact in the following way. When the knowledge revision process indicates that more concepts are needed for the compact representation of the revised knowledge base, KRT calls CLT for the generation of such concepts. Thus, the system realizes predicate invention. For learning rules for the new concepts, CLT calls RDT.

MOBAL's theory restructuring tool (RRT) assesses the quality of the domain theory according to a set of formal and statistical criteria and helps to clean up the knowledge base.

In summary, MOBAL is a complex and sophisticated system, and fully exploiting its functionality requires some training. Usage and getting started are facilitated by a comprehensive userguide and an online tutorial.

2.3.2 MILES

System:	MILES
Version:	
Further specification:	interactive ILP system
Pointers:	`http://www.gmd.de/ml-archive/ILP/public/software/miles`
Code:	Quintus Prolog (version 3.1.1) source code
References:	[525]
Other comments:	For using the system's X interface, version 3.1.1 is strictly required, later versions will fail.

Table 17: MILES: short description of the system.

MILES [525] is an ILP test environment designed to facilitate experiments with ILP methods and algorithms. It is not a ready-to-use ILP system in the sense that a user provides examples and background knowledge and the system returns a consistent theory. Rather, MILES contains an extensive collection of ILP operators used by common ILP algorithms and systems without embedding these operators into a fixed control mechanism and facilitates the integration of new operators. Thus, MILES is useful for investigating and comparing the effects of the available and newly defined operators as well as in teaching ILP methods. MILES facilitates rapid prototyping of specific ILP systems by adding a control mechanism which guides the application of operators taken from MILES's operator pool. A generic control procedure is provided.

MILES contains five types of operators, namely generalisation operators, specialisation operators, for generalising or specialising clauses or sets of clauses, reformulation operators, preprocessing operators, and evaluation operators.

The generalisation operators generalise single clauses or sets of clauses. MILES provides six variants of *least general generalisation* operators (e.g., the *rlgg* operator used by GOLEM), nine *inverse resolution* operators and five truncation operators. There are four specialisation operators

for single clauses (including three of the MIS operators also applied by MARKUS). Specialisation of sets of clauses is performed by a minimal base revision operator similar to the MOBAL's revision operator KRT. This operator and the fourth single clause specialisation operator specialise by inventing new predicates. The reformulation operators transform the knowledge base equivalently in order to facilitate the learning task. MILES provides a reduction operator for single clauses and the flattening/unflattening operators which transform a knowledge base into function-free logic and vice versa.

MILES' preprocessing operators serve for extracting implicit informations from the examples and for initializing hypothesis clauses. The first operator automatically determines argument types for all example predicates. The second operator determines a set of clause heads covering the positive examples based on the structure of the example arguments.

The evaluation operators defined in MILES assess the quality of the knowledge base according to different criteria. MILES contains predicates for checking whether the knowledge base is complete and consistent with respect to the examples and procedures for detecting culprit clauses, in case that it is not complete nor consistent. MILES provides an operator for evaluating the clauses in the knowledge base by determining the positive and negative examples they cover as well as the derivations of examples in which the clauses participate. These informations allow to compute commonly employed measures as, for instance, the information gain used by FOIL. Furthermore, MILES contains two operators for computing the compression of the knowledge base. These operators can be instantiated with six different encoding schemes.

The generic control procedure available in MILES takes twelve parameters. These have to be instantiated with predicates defining the initialization of the hypothesis, a stopping criterion for the induction process, a quality criterion for accepting hypothesis clauses, the selection of the appropriate refinement operator etc. Example instantiations of the generic control realizing ILP algorithms for special types of logic programs (regular unary logic programs, definite clause grammars, constrained programs), and a FOIL-like algorithm are included.

Comparing MILES to MOBAL we find out that, although the systems contain similar components (knowledge base access and maintenance procedures, an inference mechanism, knowledge induction and revision operators, mechanisms for deriving argument types, theory evaluation criteria) they serve quite different purposes. Whereas MOBAL provides an ILP toolbox to be used for applications, MILES is a toolbox for investigation of and experimentation with ILP methods. Since MILES serves for rapid prototyping of ILP algorithms, it may also be viewed as an ILP construction kit. As MILES is a very flexible system providing a large range of operators with various parameters and options, a user either requires good knowledge of ILP or the willingness to acquire it. MILES offers a graphical X interface which, however, does not include access to the generic control. Usage and function of the operators are described in detailed comments in the source code of MILES.

2.3.3 CLINT

System:	CLINT
Version:	
Further specification:	interactive ILP system
Pointers:	`http://www.gmd.de/ml-archive/ILP/public/software/clint`
Code:	executable for Apple Macintosh
References:	[105], [129]
Other comments:	It runs on Apple Macintosh computers under system 7 with at least two megabytes of free memory. It offers a convenient window-based interface designed according to Apple's standards.

Table 18: CLINT: short description of the system.

CLINT [105, 129] is an interactive theory revisor which allows the user to incrementally build and revise a logical knowledge base. It is often described as opportunistic, i.e., learning self-initiatively whenever possible, and user-friendly, i.e., easy to use without knowledge of its internal mechanisms.

CLINT is interactive in the sense that it queries the user in order to collect missing information on the predicates to be learned. The queries posed by CLINT are moderately complex, being either membership questions (the user is asked whether a given ground fact is true or false in the intended knowledge base), or existential questions (the user is asked to enter a ground substitution for a given non-ground fact such that the instantiated fact is true in the intended knowledge base). This contrasts to interactiveness in MOBAL's sense, where interactive means that the user and the system's components cooperate and, in particular, that the system relies on the user to determine which step should be taken next.

CLINT's prominent features include the integration of an integrity theory defining constraints satisfied by the knowledge base, an abductive component, postponing of examples, parameterized language series for declaring and shifting the system's language bias, and a set of special multi-valued logical frameworks to be chosen by the user.

CLINT learns from positive and negative examples and from integrity constraints. Its hypothesis language is a subset of range-restricted functor-free Horn logic. Building and revising complete knowledge bases, CLINT is able to learn multiple predicates, so that the examples may concern several predicates.

CLINT's basic algorithm is a loop where the user repeatedly enters a new example or integrity constraint and CLINT revises the current theory in order to make it consistent with the new input. First, we briefly sketch the way CLINT processes examples. If the user enters examples consistent with the current theory, it suffices to simply remember these examples, otherwise the theory has to be revised. If a newly entered negative example is covered by the theory, CLINT searches the clauses used for proving the negative examples for an incorrect clause and retracts it. The incorrect clause is identified with the help of user queries. Positive examples which become uncovered by removing the incorrect clause are input into CLINT's basic loop and thus trigger further theory revision. If CLINT encounters an uncovered positive example, it calls its abductive or inductive procedure. The abductive procedure completes the knowledge base by learning new facts. It constructs incomplete proofs of the uncovered positive example which are to be completed by entering ground facts to the theory (without making the theory inconsistent).

If no such facts can be found, the inductive procedure for learning new rules is triggered. CLINT's induction algorithm consists of two steps. In the first step, the so-called justifications for positive example are generated. A justification is a most specific Horn clause covering the positive example and not covering any negative examples. In the second step, the justifications are generalised by dropping literals. CLINT relies on negative examples and user queries to determine which literals can be dropped safely.

The justifications which can be constructed for a positive example depend on the hypothesis language the system employs. In CLINT, parameterized languages are used for declaring the language bias. The language parameters specify the number and depth of variables in hypothesis clauses. The languages are ordered into sequences with increasing generality. If the system detects that the current language is not sufficient for learning a consistent predicate definition, it automatically shifts its bias to the next general language in the series. CLINT offers four predefined series, but the user may customize additional series as well.

Postponing of examples becomes necessary when none of the available languages contains a consistent justification for a positive example. In this case, the knowledge base misses relevant information. CLINT then automatically postpones the uncovered positive example until more information is available.

CLINT can learn from integrity constraints as well. An integrity constraint is a first-order clause describing properties of the knowledge base to be built. CLINT treats integrity constraints as generalised examples. Roughly speaking, this is done as follows. If CLINT detects that an

integrity constraint is violated by the current knowledge base, the violated literal in the constraint is located via user queries. A violated body literal (i.e., a body literal that is wrongly true when instantiated by the violated substitution of the constraint) corresponds to a negative example, a violated head literal (i.e., a literal that is wrongly false) corresponds to a positive example. Both cases are passed into CLINT's main loop and processed accordingly.

Among a range of other user-adjustable parameters, CLINT enables the user to select a suitable logical framework. Besides the default setting, where negation is treated as negation by failure, there are three variants of multi-valued logic, providing the truth-values inconsistent, unknown or both of them in addition to the truth-values true and false.

2.4 Alternative ILP tasks

2.4.1 Inductive data engineering: INDEX

System:	INDEX
Version:	
Further specification:	alternative ILP system
Pointers:	`ftp://ftp.gmd.de/MachineLearning/ILP/public/software/index/`
Code:	Prolog source
References:	[210]
Other comments:	It is an experimental system, not able to process large datasets.

Table 19: INDEX: short description of the system.

INDEX [210] is a system for inductive data engineering. Inductive data engineering denotes the interactive process of restructuring a knowledge base by means of induction.

The underlying idea is to analyse a given database in order to detect hidden regularities which can be used for restructuring the database, yielding a more compact and meaningful representation. This approach does not conform to the classical ILP setting where the aim is to induce a hypothesis which, when combined with the background theory, explains the given examples, thus completing the theory. Rather, the approach uncovers information which is, though hidden, already present in the data. As Flach states, the process is nonetheless inductive, since it derives general rules from specific data [210]. The general setting, of which INDEX realizes a variant, is called the nonmonotonic setting of ILP [403] or the confirmatory setting of ILP [214].

INDEX expects as input an extensional relation, that is, a set of ground facts defining the relation. This relation is searched for funtional and multi-valued dependencies among the attributes, i.e., arguments of the relation. Let X and Y denote sets of attributes. A functional dependency $X \rightarrow Y$ states that whenever two tuples in the relation have identical values of attributes X, they also have identical values of attributes Y. A multi-valued dependency $X \rightarrow\rightarrow Y$ generalises a functional dependency to sets of values of the dependent attributes. This means that the values of the dependent attributes Y are not determined uniquely by the values of the attributes X, but rather, each of a fixed set of attribute value combinations of Y occurs in the relation for given values of X. INDEX searches the relation for functional and multivalued dependencies in a MIS-like manner [496], thereby exploiting the generality ordering of dependencies.

In the second step, INDEX restructures the relation, based on the attribute dependencies it has found to be holding in or violated by the relation. In either case, restructuring means that the relation is split into smaller relations allowing to reconstruct the original relation. Since the restructuring of the database involves the construction of new relations, INDEX performs predicate invention.

If an attribute dependency $X \rightarrow Y$ or $X \rightarrow\rightarrow Y$ is found to hold in the relation, it induces a so-called horizontal decomposition. The dependent attributes Y are removed from the original

relation, and stored in a separate relation together with the determining attributes X. This yields two new, smaller relations from which the original relation is reconstructed by a JOIN operation on the common attributes X. This type of split of a relation is called a horizontal decomposition since it is to reverted by the horizontal JOIN operation [210]. If INDEX discovers a satisfied attribute dependency, it automatically performs the horizontal decomposition and constructs the clause which expresses the join reverting it. INDEX queries the user to enter meaningful names for the new relations resulting from the decomposition.

Attribute dependencies which are violated in the relation induce vertical decompositions of a relation. The relation is divided into smaller relations such that the attribute dependency holds in each of the partial relations. This type of decomposition operation is reverted by UNION operations. In general, an attribute dependency induces many alternative decompositions of a relation. The selection of meaningful decompositions is guided by heuristics, but still requires user interaction.

2.4.2 Clausal discovery: CLAUDIEN

System:	CLAUDIEN
Version:	3.0
Further specification:	alternative ILP system
Pointers:	`http://www.cs.kuleuven.ac.be/cwis/research/ai/Research/claudien-E.shtml` `ftp://ftp.cs.kuleuven.ac.be/pub/logic-prgm/ilp/dlab`
Code:	BIM Prolog
References:	[131], [135]
Other comments:	Available for academic purposes as a stand-alone system running without a Bim Prolog system. If a Bim Prolog compiler is available, CLAUDIEN is able to make use of it, thereby significantly improving its execution times. The DLab mechanism is also available as a Prolog library for the use with concept learning and knowledge discovery approaches other than CLAUDIEN.

Table 20: CLAUDIEN: short description of the system.

The interactive system CLAUDIEN [131, 135] performs the task of clausal discovery, that is, it searches a given database for hidden regularities. Both the database and the regularities are represented as first-order clausal theories. As the system INDEX, CLAUDIEN belongs to nonmonotonic setting of ILP. Whereas INDEX utilises detected regularities for restructuring the database, CLAUDIEN regards the discovered regularities as an aim of themselves. As CLAUDIEN provides a powerful mechanism for specifying the type of regularities to be detected, CLAUDIEN can be applied for detecting various kinds of regularities in databases, such as integrity constraints in databases, functional dependencies and determinations, or properties of sequences.

The basic principle of CLAUDIEN's discovery algorithm is to subseqently generate the clauses contained in the hypothesis language and check them against the database. The clauses which are found to represent an actual regularity of the data are added to the hypothesis. The algorithm searches the hypothesis space from general to specific, thereby exploiting the subsumption relations among of clauses for pruning the search space. While running, CLAUDIEN successively enlarges the set of discovered regularities. The longer the algorithm runs, the more regularities may be found. As the search outputs a valid hypothesis whenever it is interrrupted, CLAUDIEN can be regarded as an anytime algorithm.

CLAUDIEN's background theory and examples (termed 'observations') are represented as conjunctions of first order Horn clauses. The hypothesis may consist of arbitrary clauses. CLAUDIEN incorporates a mechanism for the syntactical declaration of the hypothesis language called DLAB. This mechanism allows the user to specify general clause templates for hypothesis clauses. Each template defines sets of clauses. DLAB derives refinement operators from the clause templates

which map the expansion of the template into clause sets on sequences of specialisation operations under theta-subsumptions. This enables enables pruning of the search.

CLAUDIEN provides a range of control parameters. Some of these allow further control of the hypothesis language. Another group concerns semantical aspects of the hypothesis as, e.g., the minimum accuracy and coverage of discovered clauses. Additionally, the user can choose among four search strategies. Optionally, the system can be requested to produce non-redundant hypotheses, that is, hypotheses not containing clauses which are logically entailed by the background knowledge or other discovered regularities. Furthermore, CLAUDIEN provides mechanisms for the convenient management of different configurations of discovery experiments.

3 ILP datasets

A variety of ILP-related datasets have been made available as a result of ILPNET activities. Most come from several broad application areas, which are: molecular biology, finite element mesh design, natural language processing, the area of modelling, diagnosis and control, and chess. Speaking very roughly, the areas are listed in descending order according to how close to real life the applications (from which the datasets originated) are. Sample datasets that illustrate the input/output performance of several ILP systems have been also made available, as well as several datasets that do not fall in the above application areas and are put together under the heading 'miscellaneous'. Datasets not originating from the ILPNET consortium are marked with an asterisk (*).

3.1 Molecular biology datasets

Two main ILP applications have emerged in the area of molecular biology: predicting protein secondary structure and learning rules for predicting structure-activity relationships (SARs). Several applications of the latter type exist in the area of drug design, notably for pyrimidines and triazines, drugs for Alzheimer's disease, and suramin analogues. Predicting mutagenesis is also an instance of the problem of predicting structure-activity relationships. While this problem is mostly addressed in its quantitative form (predicting QSARs), ILP systems address the problem of comparing the activities of pairs of drugs or distinguishing active from inactive compounds.

3.1.1 Learning rules for predicting protein secondary structure

Application domain:	Learning Rules for Predicting Protein Secondary Structure
Further specification:	datasets, html and LaTeX documentation
Pointers:	http://www.comlab.ox.ac.uk/oucl/groups/machlearn/proteins.html
Dataset size:	46 KB (tar, gzip)
Data format:	GOLEM
References:	[406]

Table 21: Predicting protein secondary structure: dataset short description.

Predicting the secondary structure (three-dimensional shape) of proteins from their amino acid sequence (primary structure) is widely believed to be one of the hardest unsolved problems in molecular biology. The amino acids can be arranged in different patterns (spirals, turns, flat sections etc.) which are of considerable interest to pharmaceutical companies since a protein's shape generally determines its function as an enzyme. The dataset of 16 proteins (consisting of 20 aminoacids), studied in [406], has been created with the goal to learn rules to identify whether a position in a protein is in an alpha-helix. The positive examples state which positions of the

chosen proteins are in an alpha-helix and the negative examples identify the positions which are not in an alpha-helix.

The constants of the considered language denote all the 20 existing amino acids and the values of some physical or chemical properties, such as sizes, hydrophobicities, and polarities (e.g., `polar0` and `polar1`). The background knowledge is expressed using the following predicates:

- `position(A,B,C)` meaning "residue of protein A at position B is C".
- `octf(A,B,C,D,E,F,G,H,I)` provides information that allows to index groups of nine adjacent positions in a protein (positions A–I occur in sequence).
- `alpha_triplet(A,B,C)`, `alpha_pair(A,B)`, index groups of three or two adjacent positions in a protein, respectively.
- `alpha_pair4(A,B)` holds if a pair of positions A, B is separated by 4 positions in a protein.

Additional unary predicates characterize some physical and chemical properties of the individual residues (hydrophobicity, hydrophilicity, charge, size, polarity, whether a residue is aliphatic or aromatic, whether it is a hydrogen donor or acceptor etc.). Ordering relations between some constants, such as `less_than(polar0,polar1)`, are also provided.

3.1.2 Learning structure-activity rules for pyrimidines and triazines

Application domain:	Structure-Activity Rules for Inhibition of E. Coli Dihydrofolate Reductase
Further specification:	two datasets (pyrimidines and triazines) html and LaTeX documentation
Pointers:	http://www.gmd.de/ml-archive/ILP/public/data/drug http://wwwcomlab.ox.ac.uk/oucl/groups/machlearn/e_coli.html
Dataset size:	3.5 MB
Data format:	GOLEM
References:	[294], [297]

Table 22: Learning SARs for pyrimidines and triazines: dataset short description.

Structure activity relationships (SAR) describe empirically derived relationships between the chemical structure and the activity of considered drugs. In a typical SAR problem a set of chemicals of known stucture and activity are given, and the goal is to construct a predictive theory relating the structure of a compound to its activity. Both available datasets, described in [294] and [297] concern the classical drug design problem of inhibition of E. Coli Dihydrofolate Reductase by pyrimidines and triazines.

Pyrimidine compounds are antibiotics based on a common template to which chemical groups can be added at 3 possible substitution positions. A chemical group is an atom or a set of structurally connected atoms (that can be substituted together as a unit) characterized by well defined chemical properties. Some of these properties are encoded within the background knowledge as facts, e.g., the fact `polar(br,polar3)` states that bromine atoms have polarity 3. Pyrimidine compounds are identified by the substituents at the 3 substitution positions. Examples are pairs of drugs, positive when the activity of the first is known to be higher than that of the second and negative otherwise.

Triazines act as anti-cancer agents by preferentially inhibiting reproducing cells. Like pyrimidines, they have a common template, but this one is much more complicated with 5 possible substitution positions. The research goal is to find rules for predicting the activity of different compounds. This task and its structure are closely related to the pyrimidine problem.

3.1.3 Learning structure-activity rules in drugs for Alzheimer's disease

Application domain:	Learning Structure-Activity Rules in Drugs for Alzheimer's disease
Further specification:	datasets, html and LaTeX documentation
Pointers:	http://www.comlab.ox.ac.uk/oucl/groups/machlearn/drugs.html
Dataset size:	37 KB
Data format:	Prolog
References:	[297]

Table 23: Learning SARs in drugs for Alzheimer's disease: dataset short description.

Low toxicity is a *conditio sine qua non* when searching for a new drug treatment of any disease. The following specific properties are considered as essential in the case of Alzheimer's disease:

- high acetocholinease inhibition,
- good reversal of scopolamine induced deficiency and
- inhibition of amine re-uptake.

Recently the drug Tacrine has drawn considerable attention as it exhibits the requested specific effects. Unfortunately, large scale clinical tests show its high toxicity. Variants of Tacrine have been studied which are created by substituting new components for its R and X subgroups.

From the Alzheimer's disease dataset, the problem is to find rules concerning the upper mentioned properties from the positive and negative examples providing pairwise comparisons of various drugs. To describe the required background knowledge a representation is used that is similar to the one used for predicting protein secondary structure. Moreover, additional information is included:

- Some physical and chemical properties of chemicals that can be substituted for `R` or `X`. These are described by unary predicates in the background knowledge, such as hydrophobicity, hydrophilicity, charge, size, polarity, etc.
- The predicate `x_subst(Drug,Position,Subs)` states that the `X` substitution for `Drug` in position `Position` is `Subs`.
- The `R` substitution is split into parts corresponding to bonds between different chemical structures.
- Sometimes the `R` substitution involves the presence of one or more ring structures. These are benzyl derivatives that arise from substitutions within the basic benzene ring structure. The number of such substitutions in the basic benzene structure, the position of the substitutions and the actual substituent are provided.

The obtained results have been published in [297].

3.1.4 Learning structure-activity rules in suramin analogues

Suramin analogues act as anti-cancer agents by interfering with the formation of blood vessels for the tumor. Using the atomic structure and bond relationships present in suramin compounds, the task for an ILP system is to discover structural features that can be used to improve such compounds. Formally, this goal is closely related to the problem studied in connection with the Alzheimer's disease. The structure of the data and the language applied are closely related to the Alzheimer's case.

Application domain:	Learning Structure-Activity Rules for Suramin Analogues
Further specification:	datasets, html and LaTeX documentation
Pointers:	http://www.comlab.ox.ac.uk/oucl/groups/machlearn/drugs.html
Dataset size:	23KB (7 positive and 4 negative examples)
Data format:	PROGOL
References:	

Table 24: Learning SARs for suramin analogues: dataset short description.

3.1.5 Learning rules for predicting mutagenesis

Application domain:	Learning Rules for Predicting Mutagenesis
Further specification:	datasets, html and LaTeX documentation
Pointers:	http://www.comlab.ox.ac.uk/oucl/groups/machlearn/mutagenesis.html
Dataset size:	131 KB compressed file with positive and negative examples for the subsets of 188 and 42 compounds
Data format:	PROGOL
References:	[296], [511], [512], [513]

Table 25: Learning rules for predicting mutagenesis: dataset short description.

The prediction of mutagenesis is important as it is relevant to the understanding and prediction of carcinogenesis. Not all compounds can be empirically tested for mutagenesis, e.g. antibiotics. The considered dataset has been collected with the intention to search for a method for predicting the mutagenicity of aromatic and heteroaromatic nitro compounds. It comprises 230 compounds.

Mutagenicity is measured by the Ames test using S. typhimurium TA98. Of the 230 compounds, 138 have positive levels of log mutagenicity and are labeled as active, i.e., are treated as positive examples. The remaining 92 compounds are treated as negative examples.

The basic background knowledge contains the generic description of the compounds consisting of the constituent atoms and their bond connectivities. Each compound is represented by a sets of facts of the form: `atm( 127, 127_1, c, 22, 0.191 )` and `bond(127, 127_1, 127_6, 7 )`. These two predicates give a completely generic method of describing molecular structure in drug design. They also allow a straightforward definition of generic chemistry knowledge that defines higher level chemical concepts (for example, ring structures). Some definitions of this kind are also provided with the dataset.

Four additional attributes are provided for each compound:

- its hydrophobicity,
- the energy level of the lowest unoccupied molecular orbit,
- two boolean valued attributes identifying components with 'three or more benzol rings' and compounds termed accenthryls.

The considered nitro compounds are more heterogeneous structurally than any of those in the other ILP datasets concerning chemical structure activity. Results of relevance to the ML community are available in [511, 512, 513], relevant chemical results can be found in [296].

3.2 Finite element mesh design

The datasets on finite element mesh design come from the area of mechanical engineering where finite element meshes are used to analyse the behaviour of structures under different kinds of stress. The problem addressed here is to determine an appropriate resolution of a finite element mesh for a given structure so that the corresponding computations are both accurate and fast. A complete and a partial dataset exist, the latter containing data on five of the ten structures described in the former.

3.2.1 Finite element mesh design (complete dataset)

Application domain:	Finite Element Mesh Design (complete dataset)
Further specification:	dataset
Pointers:	http://www.gmd.de/ml-archive/general/data/mesh_design
Dataset size:	642 positive examples + 3804 background facts
Data format:	Prolog facts
References:	[155], [160], [173], [158], [156]

Table 26: Finite element mesh design: short description of the complete dataset.

The resolution of a finite element (FE) mesh is determined by the number of elements on each of its edges. It depends on the geometry of the body studied and on the boundary conditions. Given are descriptions of ten structures for which experts have determined an appropriate mesh resolution. The positive examples are of the form `mesh(Edge,NumberOfElements)`. The task is to learn rules that determine an appropriate resolution of a FE mesh (i.e., an appropriate resolution for each given edge) from the geometry of the body, the types of edges, boundary conditions and loadings.

The background knowledge can be divided into two parts: attribute description of the edges and geometric relations between the edges. The first part consists of unary predicates that have edges as arguments. These predicates can be grouped in three subgroups:

- predicates that describe the type of the edge (`long`, `usual`, `short`, `circuit`, `half_circuit`, `quarter_circuit`, `short_for_hole`, `long_for_hole`, `circuit_hole`, `half_circuit_hole`, `quarter_circuit_hole`, `not_important`),
- predicates that describe the supports of the edge (`free`, `one_side_fixed`, `two_side_fixed`, `fixed`),
- predicates that describe the loads (`not_loaded`, `one_side_loaded`, `two_side_loaded`, `cont_loaded`).

The second part contains the binary relations `neighbour` and `opposite`, that describe the geometrical relations between edges. These two relations are nondeterminate. A determinate version also exists, where each of the two relations is replaced with three relations, e.g., `neighbour_xy`, `neighbour_yz`, and `neighbour_zx`.

The problem domain and early results (obtained on the partial dataset) are described in [155, 160, 158, 173]. The complete dataset and experiments on this dataset with CLAUDIEN are decribed in [156].

3.2.2 Finite element mesh design (partial dataset)

This dataset is a preliminary version of the more complete dataset described above. It is based on the descriptions of five structures for which experts have determined an appropriate mesh

Application domain:	Finite Element Mesh Design (partial dataset)
Further specification:	dataset
Pointers:	http://www.comlab.ox.ac.uk/oucl/groups/machlearn/mesh.html
Dataset size:	57 KB (tar, gzip)
Data format:	GOLEM
References:	[160]

Table 27: Finite element mesh design: short description of the partial dataset.

resolution. Only the determinate version of the geometrical relations is provided, as this particular version of the dataset has been used for experiments with GOLEM, as reported in [160].

3.3 Natural language processing datasets*

This section includes the descriptions of three datasets that originate from applications in the area of natural language processing: natural language parsing, mapping natural language queries to database queries, and the formation of the past tense form of English verbs. A dataset originating from the related application domain of document understanding is also described.

3.3.1 Natural language parsing (M&K data)*

Application domain:	Natural Language Parsing (M&K Data)*
Further specification:	dataset and papers
Pointers:	ftp://ftp.cs.utexas.edu/pub/mooney/nl-ilp-data ftp://ftp.cs.utexas.edu/pub/mooney/papers/chill*
Dataset size:	1450 facts
Data format:	Prolog
References:	[588], [589]

Table 28: Natural language parsing M&K data: dataset short description.

The problem addressed here is the construction of semantic grammars. This is a difficult and interesting problem which has been treated by machine learning techniques recently [588], viewing the semantic-grammar acquisition problem as a problem of learning search-control heuristics. Appropriate control rules are learned using the first-order induction algorithm CHILL that automatically invents useful syntactic and semantic categories. The logic programming formalism is used to represent these control rules. The learning task can be also viewed as an n-way categorization problem for complex tuples. Empirical results show that the learned parsers generalize well to novel sentences and outperform previous approaches based on connectionnist techniques [589].

The data come in two ILP formats:

1. Examples of the predicate `parse`, which contain pairs of sentences with their case-role analysis, such as:

   ```
   parse([the,man,ate],[ate,agt:[man,det:the]]).
   ```

2. Control examples for parsing the input sentences using a shift/reduce parser. Transitions of the parser are described by 4-tuples `(Stack,Input,NewStack,NewInput)`. In this data set, lists of parser states where the first two atributes `Stack, Input` are instantiated, are matched to the corresponding semantic attachment of the stack items. Consider the following examples:

```
op([S1,S2|SRest],Inp,[SNew|SRest],Inp):- attach(S1,prep,S2,SNew).
op([[rock,det:the],with,[hit,pat:[rock,det:the],agt:[girl,det:the]]],[],
   NewStack,NewInput).
op([[hammer,det:the],with,[hit,pat:[rock,det:the],agt:[girl,det:the]]],[],
   NewStack,NewInput).
```

In the two instantiated clauses of `op/4` shown above, the first two items of the stack will be attached to each other (according to the general clause at the top) and will produce a prepositional phrase.

More details on this approach can be found in the papers on CHILL, which are accessible via ftp (see table). M&K stands for McCelland and Kawamoto, since this artificially generated data originates from their work.

3.3.2 Mapping natural language queries to database queries (Geoqueries)*

Application domain:	Mapping natural language queries to database queries (Geoqueries)*
Further specification:	dataset and papers
Pointers:	ftp.cs.utexas.edu/pub/mooney/nl-ilp-data ftp://ftp.cs.utexas.edu/pub/mooney/papers/chill-dissertation-95.ps ftp://ftp.cs.utexas.edu/pub/mooney/papers/chill-bkchapter-95.ps
Dataset size:	250 facts
Data format:	Prolog
References:	[590]

Table 29: Geoqueries: dataset short description.

These are data for natural-language learning experiments (Geoquery) in a form suitable for ILP systems. The set of examples relates English queries about a simple U.S. geography database to executable Prolog queries and has the following form:

```
parse([how,many,people,live,in,hawaii,?],
    answer(B,(population(A,B),const(A,stateid(hawaii))))).
```

ILP is applied to induce rules defining an appropriate natural-language database front-end [590]. Details are in the dissertation of Zelle which is available on-line via ftp along with a summary in a book chapter (see table). This data set is also related to the task described in the previous subsection.

3.3.3 English past tense*

Application domain:	English past tense*
Further specification:	datasets and papers
Pointers:	ftp.cs.utexas.edu/pub/mooney/nl-ilp-data ftp://ftp.cs.utexas.edu/pub/mooney/papers/foidl-jair-95.ps ftp://ftp.cs.utexas.edu/pub/mooney/papers/foidl-bkchapter-95.ps
Dataset size:	1392 facts
Data format:	Prolog
References:	[362], [335]

Table 30: English past tense: dataset short description.

This dataset provides a Prolog form of the data studied in [335], originally assembled by Brian McWhinney at CMU. The dataset contains the past-tense of a set of English verbs. The data are divided into three files. The file `alphabetic-past-data` contains the normal English spelling forms for the verbs and their past tense forms. An example fact from this file is:

```
past(['A','I','D'],['A','I','D','E','D']).
```

The file `phonetic-past-data` contains the phonetic forms for all of the verbs in the data set and `regular-phonetic-past-data` contains the phonetic forms for the regular verbs only. Finally, the file `past-background-defn` consists of the intensional definition for the predicate `split/3`, the background predicate used in experiments on this dataset with FOIDL and IFOIL [362].

3.3.4 Document understanding*

Application domain:	Document understanding*
Further specification:	datasets
Pointers:	http://www.gmd.de/ml-archive/general/data/doc-understanding
Dataset size:	112 KB (compressed files)
Data format:	FOCL format, FOIL format
References:	[339], [194], [195], [494], [196]

Table 31: Document understanding: dataset short description.

The problem considered here is to classify some parts of a business letter using information about the layout of a one page document. There are five concepts to be learned. These concepts are expressed as predicates, namely `sender`, `receiver`, `logotype`, `reference number` and `date`. The representation used characterizes some properties of the text-blocks (their width and height, their position in a page etc.), as well as the relative positioning of two blocks (e.g., `aligned-only-upper-row(X,Y)`).

The problem is complicated by the presence of dependencies among the five target concepts. The problem can be cast as a multiple predicate learning problem. Experimental results prove that learning contextual rules, that is rules in which concept dependencies are explicitly considered, leads to good results.

In the experimentation, 30 single page documents were considered. Six trials were performed by randomly selecting 20 documents for the training set and 10 for the test set. The training and testing set for each of the six trials are given in FOCL format, along with a file that contains information on all documents in FOIL format.

Initially, results on these datasets were published in a technical report [339], summaries of the results appear in [194, 195, 494]. The integral document processing system is treated in detail in [196].

3.4 ILP datasets for diagnosis, control, and modelling

The datasets described in this section originate from ILP applications related to dynamical systems, their diagnosis, control, and modelling. These are: learning diagnostic rules from a qualitative model of the power supply subsystem of a satellite, learning rules for transitions between flight stages in a flight simulator, learning relational concepts from sensor data gathered by a mobile robot, and learning qualitative models of dynamical systems from example behaviours.

Application domain:	Learning diagnostic rules for a satellite power supply subsystem
Further specification:	dataset
Pointers:	http://www.comlab.ox.ac.uk/oucl/groups/machlearn/satellite.html
Dataset size:	78 KB (tar, gzip)
Data format:	GOLEM
References:	[197]

Table 32: Satellite power supply subsystem data: dataset short description.

3.4.1 Learning diagnostic rules for a satellite power supply

One of the first practical applications of ILP was within the aerospace industry, namely the diagnosis of power-supply failures in a communications satellite [197]. The satellite recharges its batteries using solar energy. The charging system can be described through a qualitative model consisting of 40 components and 29 sensors. As the satellite orbits the Earth, its position relative to the Sun changes, driving its power-supply subsystem through four distinct stages: battery charging, solstice, eclipse, and battery reconditioning. Qualitative simulation makes it possible to predict the behaviour of the power supply in each of these stages. By provoking simulated faults in the components, the simulation can generate examples of relations between a fault and the supply's behaviour. These examples form the present dataset, which served as an input to the ILP program GOLEM.

GOLEM induced a set of rules for diagnosing power supply failures. In generating the examples, faults were provoked in all possible components, thus guaranteeing that the rules are complete and correct for all single faults. Because the power-supply's behaviour changes with time, the formalism used to describe the examples is based on temporal logic which proves to be suitable for ILP learning.

3.4.2 Learning flight stage transition rules

Application domain:	Learning flight stage transition rules
Further specification:	dataset
Pointers:	http://www.gmd.de/ml-archive/general/data/stagedata/ Welcome.html
Dataset size:	160KB (tar, gzip)
Data format:	PROGOL
References:	[83], [350]

Table 33: Learning flight stage transition rules: dataset short description.

Data from behavioural traces of several human pilots flying an F-16 flight simulator have been used to reverse engineer human flight skill [350]. In that experiment, the pilots followed a flight plan, i.e., a sequence of flight stages: take-off, climb, leveling, straight and level flight, left turn, align with runway, descent, landing. For each stage and for each aircraft control a decision tree was induced. When using the induced controller as an auto-pilot the values for the aircraft controls are obtained by interpreting the corresponding decision tree for that control and for that flight stage. The transitions from stage to stage were done by means of hand-coded rules.

The task addressed here is to **learn** such stage transition rules from examples and background knowledge. Eight different concepts have to be learned, corresponding to the preconditions for take-off and the seven stage transitions. The ILP system INDLOG [83] was applied to these data.

3.4.3 Learning relational concepts from sensor data of a mobile robot

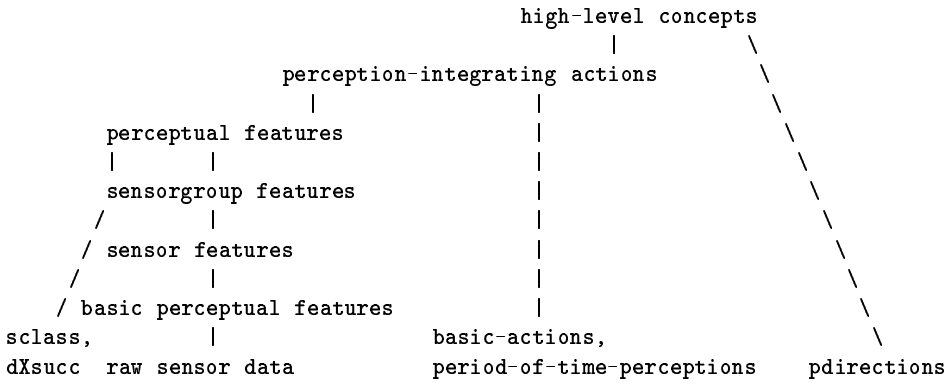
Application domain:	Learning relational concepts from sensor data of a mobile robot
Further specification:	set of sensor datasets
Pointers:	http://www-ai.informatik.uni-dortmund.de/ blearn/data-sets.html
Dataset size:	1.5MB (tar)
Data format:	first order logic
References:	[300], [299]

Table 34: Robot sensor datasets: short description.

This set of datasets was gathered during experiments with a real mobile robot described in detail [300, 299]. Data have been collected with the goal to learn abstract operational concepts, e.g., 'the robot moves along a wall', from sequences of sonar sensor measurements and robot actions. Each dataset is represented in first order logic, where the following restrictions are applied: facts can be linked using the argument "Time", and there are never two different facts concerning the same sensor and the same point in time.

Positive and negative examples are given through the sensor feature predicates structured according to a general pattern `sf(Tr,S_id,Start,End,Rel_or)`. Its intended interpretation is "In a trace `Tr`, a specific sensor `S_id` percieved an object corresponding to the predicate's name `sf`. This happened during the time interval between `Start` and `End` while moving in a relative orientation `Rel_or` along the object."

Each data set corresponds to learning disjoint concepts at one level. The levels are organized in a hierarchy as shown below:



Each node in the hierarchy denotes a set of predicates. The links are directed from the bottom to the top. They link sets of predicates in lower nodes to a set of predicates in a higher node if the predicates of the lower level are necessary to learn the concepts of the higher level. Hence, a sequence of learning passes can learn high-level concepts from raw sensor data.

3.4.4 Learning qualitative models from example behaviors

The task here is to learn a QSIM type qualitative model of two connected containerns - a dynamic system known as the U-tube. The target predicate `legalstate/4` has as arguments the qualitative values of the four system variables (water levels and their rates of change for each of the two containers). The positive facts corresponding to this predicate describe the states that appear

Application domain:	Learning qualitative models from example behaviors
Further specification:	dataset
Pointers:	http://www.gmd.de/ml-archive/ILP/public/data/utube/
Dataset size:	4 positive and 543 negative examples
Data format:	Prolog (MFOIL format)
References:	[322]

Table 35: Learning qualitative models from example behaviors: dataset short description.

in a qualitative behavior of the U-tube (4 facts). The negative facts (543) represent randomly selected states that are impossible in such a system. The background knowledge defines the qualitative constraints of the QSIM formalism: `add/3`, `multiply/3`, `deriv/2`, `m_plus/2`, `m_minus/2`, and `minus/2`. A more detailed description of the data and the results of experiments with MFOIL can be found in [322].

3.5 Chess datasets

Two datasets from the chess endgame White King and Rook vs. Black King (KRK) are described here. The first concerns the problem of learning the concept of an illegal white-to-move position. The second concerns the problem of learning the optimal depth to win in the same chess endgame.

3.5.1 Learning rules for illegal KRK positions

The data provided here concern the chess endgame KRK with three pieces left on the chess board: White King, White Rook and Black King. The problem of learning rules for recognising illegal positions when it is white's turn to move (WTM) was first proposed by [400]. This has since become a widely accepted test-bed for ILP systems.

The following information is used to learn the concept of illegal position. The examples are represented by the predicate `illegal/6` specifying the column and row coordinates of White King, White Rook and Black King, respectively; the data on ordering and adjacency of rows and columns are provided as background knowledge.

Application domain:	Learning rules for illegal positions in the King and Rook vs. King chess endgame
Further specification:	set of datasets
Pointers:	http://www.gmd.de/ml-archive/ILP/public/data/KRK/
Dataset size:	A test set of 5000 examples and 5 sets of 100 examples each. Variants of the latter with three different types and six different levels of noise.
Data format:	Prolog
References:	[162], [164], [322], [400]

Table 36: Learning illegal KRK positions: dataset short description.

The data in this data set originates from [400], where 5 sets of 100 examples and 5 sets of 1000 examples were used. The latter are now in one single file of 5000 examples named `test.pl`. To test the performance on ILP systems in the presence of a controlled amount of noise, three different types of noise were added to the smaller sets of examples at different noise levels: noise in arguments (na), noise in class (nc) and noise in both arguments and class (nb). The directory contains corresponding variants of the original data annotated by the originating set (1–5), the type (na, nb, nc) and the noise level (00–80). Extensive experiments on the noisy datasets were performed

with MFOIL, an ILP system based on FOIL, which includes techniques for handling imperfect (noisy) data. A detailed description of the datasets here (e.g., how noise was introduced) as well as the experiments on these datasets with FOIL [4] and MFOIL can be found in [162, 164, 322].

3.5.2 Learning rules for optimal depth to win in KRK

Application domain:	Learning rules for optimal depth to win in the King and Rook vs. King chess endgame
Further specification:	two datasets
Pointers:	http://www.comlab.ox.ac.uk/oucl/groups/machlearn/chess.html
Dataset size:	184 KB (tar, compress)
Data format:	GOLEM
References:	[32], [400]

Table 37: Learning rules for optimal depth to win in KRK: dataset short description.

A different dataset on learning illegal KRK positions is included in this distribution, used in experiments with GOLEM. A training and a testing set of 10000 posititions each are given, as well as the background knowledge.

Another, more difficult problem related to the same chess endgame is to do with predicting optimal depth of win (that is, the number of moves to checkmate). In the data here, the exhaustive database for the KRK domain acts as a source of example positions, each of which has associated with it optimal depth of win information. Here this is represented by the predicate `krk/7`. The arguments for this predicate stand for depth of win, and the coordinates of the three pieces on the chess board.

The typical ILP task is to learn the sub-concept "black-to-move KRK position won optimally for white in N moves". Preliminary results of classifiers for "won in 0 moves" up to "won in 5 moves" can be found in [32]. The background knowledge consists of ground instances of the predicates "symmetric difference" and "strictly less than".

3.6 Sample datasets with ILP systems

This section reports on datasets that are distributed together with ILP systems with the aim to demonstrate the functionalities of these systems. These are typically small datasets, often taken from the literature or contrived by the authors of the systems themselves. Datasets that come with LINUS [325], FORTE [460], SKILIT [272], HAIKU [417] and WIM [449] are described.

3.6.1 LINUS

Application domain:	LINUS
Further specification:	system, datasets and paper
Pointers:	http://www.gmd.de/ml-archive/ILP/public/software/linus/ (datasets and LINUS sources) http://www.gmd.de/ml-archive/ILP/public/papers/nonrec-learn.ps (paper)
Dataset size:	small datasets
Data format:	LINUS
References:	[152], [325], [312], [349], [452]

Table 38: Short description of LINUS datasets.

This distribution includes several small datasets adapted from the ML literature in a format appropriate for LINUS [325, 312]. The problems include:

- learning family relationships [349] (file `dhdbmoth.pl`)
- learning the concept of an arch (file `dhdbarch.pl`)
- learning where trains are heading [349] (file `dhdbeast.pl`)
- learning rules governing card sequences in the game Eleusis [152] (files `dhdbele[123m].pl`)

All the mentioned references point to the original sources of the data. FOIL [452] was applied to all of these domains and the datasets were prepared starting from this paper, where descriptions of the domains can be found. The results obtained from LINUS are described in detail in a Technical Report [312] and summarized in the paper [325].

3.6.2 FORTE (First Order Revision of Theories from Examples)*

Application domain:	FORTE (First Order Revision of Theories from Examples)*
Further specification:	system and datasets
Pointers:	http://www.cs.utexas.edu/users/ml/forte.html
Dataset size:	small datasets
Data format:	FORTE format
References:	[460]

Table 39: Short description of FORTE datasets.

The system FORTE (First Order Revision of Theories from Examples) has been described in [460]. It is a machine learning system for modifying a first-order Horn-clause domain theory to fit a set of training examples. FORTE uses a hill-climbing approach to revise theories, it identifies possible errors in an input theory and calls on a library of operators to develop possible revisions. These operators are constructed from methods such as propositional theory refinement, first-order induction, and inverse resolution.

To illustrate the behaviour of FORTE several sample datasets are provided, such as the datasets for learning family relationships, learning illegal positions in the KRK chess endgame and learning a program to insert an element in a list after a given element.

3.6.3 SKILit (Recursive Theories)

Application domain:	SKILit (Recursive Theories)
Further specification:	data + background knowledge
Pointers:	http://www.up.pt/~amjorge/skil/index.html
Dataset size:	small datasets
Data format:	Prolog
References:	[272], [77], [271]

Table 40: Short description of SKILit datasets.

The system SKILit described in [271] is designed to induce recursive theories from small sets of positive examples using iterative bootstrapping. The system does not start from scratch but searches for a program matching the input programming schema of the intended program. Moreover, SKILit offers means to specify both

- some characteristics of the target predicate, e.g., its input/output mode and type declarations
- and integrity constraints.

Auxiliary predicates are provided in the background theory (environment files). The present dataset includes training and testing data used for verification of the functionality of SKILIT when inducing simple arithmetic and list processing concepts (e.g., sorting).

3.6.4 HAIKU

Application domain:	HAIKU
Further specification:	system and datasets
Pointers:	Fabien.Torre@lri.fr
Dataset size:	small datasets
Data format:	Prolog
References:	[417]

Table 41: Short description of HAIKU datasets.

This dataset contains examples as well as background knowledge for three well-known concepts: 'arch', 'cup', and 'tic-tac-toe'. The data have been used by the HAIKU system.

3.6.5 WIM

Application domain:	WIM
Further specification:	system and datasets
Pointers:	popel@fi.muni.cz
Dataset size:	small datasets
Data format:	Prolog
References:	[449], [448], [450]

Table 42: Short description of WIM datasets.

The data set contains the minimal example sets for learning basic list processing predicates (`member/2`, `append/3`, `reverse/2`, `split/3`, `delete/3`, `sublist/2` etc.), for the set operation `union/3` as well as for predicates that use Peano's arithmetics (`plus/3`, `lessOrEqual/2`, `listLength/2`, `extractNth/3`) by the system WIM [449, 450].

For each target predicate, the example set contains the worst possible examples such that WIM can learn the target predicate. For the current version of WIM 'the worst possible examples' have to meet the following three requirements:

- an instance of each base clause has to appear in the example set (e.g., for the predicate `member/2` this may be `member(a,[a,b])`).
- positive examples are not on the same resolution chain (e.g., `member(d,[c,d,e])` is enough together with `member(a,[a,b])`) and
- at most one negative example is used (this negative example is generated by WIM itself).

Experiments with WIM to rebuild database schemata were described in [448]. In deductive object-oriented databases both classes and attributes may be defined by rules. Example sets for

learning subclasses (`japaneseCar/1` as a subclass of `car/1`, `isMother/1` as a subclass of `person/1`, `electricalVehicle/1` as a subclass of `car/1` and `publicTransportVehicle/1`), superclasses (`factory/1`, `person/1`) as well as example sets for learning classes of new objects (`family/2`) and for learning a new attribute (`personManagedBy/2`) are included.

3.7 Miscellaneous datasets

Datasets on spatial layouts of Japanese houses, trains that head east or west, and topographic descriptions of geographic objects.

3.7.1 Spatial layout of Japanese houses*

Application domain:	Spatial layout of Japanese houses*
Further specification:	dataset
Pointers:	ftp mizo01.ia.noda.sut.ac.jp/incoming/Floor_data.tar
Dataset size:	174 positive and 214 negative examples 1500 clauses of background knowledge
Data format:	Prolog
References:	[354]

Table 43: Spatial layout of Japanese houses: dataset short description.

The examples provide symbolic and numerical information about spatial layout of Japanese houses, e.g., the position and size of different rooms, their spatial relations or the total size of the house. There are two types of background knowledge — one of them allows to transform the numerical information into a qualitative form. The goal is to induce rules in the form of constrained clauses containing both symbolic and numerical information, e.g., "If the space `X` is in the interval `[Value1,Value2]`, it is concluded with the likelihood `P%` that the living room is adjacent to the dining room." This dataset has been generated to test the GKS system [354], an ILP system for induction of constraint logic programs.

3.7.2 East-West challenge*

Application domain:	East-West Challenge*
Further specification:	datasets and results
Pointers:	http://www.gmd.de/ml-archive/ILP/public/data/east_west/
Dataset size:	20/24/100 trains
Data format:	Prolog
References:	[351], [549]

Table 44: Short description of the East-West challenge data.

The problem of learning where trains are heading has been proposed by Michalski and Larson in 1977: the task is to discover simple rules that distinguish Eastbound trains from those heading in the opposite direction. An example of such a rule is it "If a train has a long closed car with more than one load, then it is heading west." This problem has numerous variants depending on the initial description of the considered examples. Several variants have been presented by [351] as a challenge to the international computing community in the form of a competition with deadline early in 1995.

This entry has 3 parts:

- The Challenge files describe the competition and contain all the material that was made available to the competitors.
- The Results provide results of all the competitions, some comments on them and the winning program.
- RL-ICET files describe an ILP solution by Peter Turney (full paper and the data preprocessor).

3.7.3 Geographic data

Application domain:	Geographic data
Further specification:	dataset
Pointers:	popel@fi.muni.cz
Dataset size:	100 positive examples
Data format:	Prolog
References:	unpublished data, contact popel@fi.muni.cz

Table 45: Short description of geographic data.

The dataset contains a topographic description of rivers, roads, railways as well as woods/forests and buildings (one fact per object). Each object is decribed by its geometry in 2-dimensional space, some additional characteristics are used when neccessary, e.g., for the roads and railways. The example set is based on real-world topographic data, which have been modified to keep their confidentiality. The applied modification maintains the essential geographical features of the original data with clear geometrical interpretation (e.g., line intersections, the size of the considered area etc.).

The provided dataset is intended for the induction of rules for identification of concepts like bridge, forest in contrast to wood, railway station in contrast to a house nearby the railway etc.

4 ILP bibliography

This section gives a list of collected bibliographic items. Despite its inevitable incompleteness, we believe that the collection contributes to the better accessibility of the ILP literature to interested readers. Please inform us of discovered mistakes and/or omissions.

[1] H. Adé. Inverse resolutie en automatisch leren. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1990. (In Dutch).

[2] H. Adé and H. Boström. JIGSAW: Puzzling together RUTH and SPECTRE. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 263–266. Springer-Verlag, 1995.

[3] H. Adé and M. Bruynooghe. A comparative study of declarative and dynamically adjustable language bias in concept learning. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[4] H. Adé and M. Bruynooghe. A comparative study of declarative and dynamically adjustable language bias in concept learning. In *Proceedings of the ML-92 Workshop on Biases in Inductive Learning*, 1992.

[5] H. Adé, L. De Raedt, and M. Bruynooghe. Inverse resolutie in een geintegreerd inductief - deductief leersysteem: SWAN. In *Proceedings of the 4th Dutch AI Conference*, pages 299–308, 1991. (In Dutch).

[6] H. Adé, L. De Raedt, and M. Bruynooghe. Inverse resolution in an integrated inductive - deductive learning system. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 456–457. John Wiley, 1992.

[7] H. Adé, L. De Raedt, and M. Bruynooghe. Révision de théorie: Approche logique. In *Actes des Journées Francophones sur l'Apprentissage*, 1993.

[8] H. Adé, L. De Raedt, and M. Bruynooghe. Theory revision. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 179–192. J. Stefan Institute, 1993.

[9] H. Adé, L. De Raedt, and M. Bruynooghe. Declarative Bias for Specific-to-General ILP Systems. *Machine Learning*, 20(1/2):119–154, 1995.

[10] H. Adé and M. Denecker. AILP: Abductive inductive logic programming. In C.S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1201–1207. Morgan Kaufmann, 1995.

[11] H. Adé, B. Malfait, and L. De Raedt. RUTH: An ILP approach to theory revision. In *Proceedings of the 4th Belgian-Dutch Conference on Machine Learning*, pages 64–74, 1994.

[12] H. Adé, B. Malfait, and L. De Raedt. RUTH: An ILP theory revision system. In *Proceedings of the 8th International Symposium on Methodologies for Intelligent Systems*, volume 869 of *Lecture Notes in Artificial Intelligence*, pages 336–345. Springer-Verlag, 1994.

[13] D.W. Aha. Relating relational learning algorithms. In S. Muggleton, editor, *Inductive Logic Programming*, pages 233–260. Academic Press, 1992.

[14] D.W. Aha, S. Lapointe, C.X. Ling, and S. Matwin. Inverting implication with small training sets. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 31–48. Springer-Verlag, 1994.

[15] D.W. Aha, S. Lapointe, C.X. Ling, and S. Matwin. Learning recursive relations with randomly selected small training sets. In W.W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 12–18. Morgan Kaufmann, 1994.

[16] D.W. Aha, C.X. Ling, S. Matwin, and S. Lapointe. Learning singly-recursive relations from small datasets. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 47–58. Morgan Kaufmann, 1993.

[17] K. Akama. A theory of predicate invention. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[18] Z. Alexin, T. Gyimóthy, and H. Boström. Integrating algorithmic debugging and unfolding transformation in an interactive learner. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 437–452. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[19] Z. Alexin, T. Gyimóthy, and H. Boström. Integrating algorithmic debugging and unfolding transformation in an interactive learner. In W. Wahlster, editor, *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 403–407. John Wiley, 1996.

[20] K.M. Ali. *Learning Probabilistic Relational Concept Descriptions.* PhD thesis, Department of Information and Computer Science, University of California, Irvine, 1995.

[21] K.M. Ali and M.J. Pazzani. Hydra: A noise-tolerant relational concept learning algorithm. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1064–1071. Morgan Kaufmann, 1993.

[22] A. Amin, C.P. Chen, C. Sammut, and K.C. Sum. Learning to recognise hand-printed Chinese characters using ILP. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 263–272. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[23] J. Arima. Automatic logic programming under highly redundant background knowledge. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 355–372. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[24] B. Bain, C. Sammut, A. Sharma, and J. Shepherd. ReDuce: Automatic structuring and compression in relational databases. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing*, pages 41–52, 1996.

[25] M. Bain. Experiments in non-monotonic first-order induction. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 195–206, 1991.

[26] M. Bain. Experiments in non-monotonic leaning. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 380–384. Morgan Kaufmann, 1991.

[27] M. Bain. Experiments in non-monotonic first-order induction. In S. Muggleton, editor, *Inductive Logic Programming*, pages 423–436. Academic Press, 1992.

[28] M. Bain. Learning optimal KRK strategies. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[29] M. Bain and S. Muggleton. Non-monotonic learning. In J.E. Hayes-Michie and E. Tyugu, editors, *Machine Intelligence*, volume 12. Oxford University Press, 1991.

[30] M. Bain and S. Muggleton. Non-monotonic learning. In S. Muggleton, editor, *Inductive Logic Programming*, pages 145–161. Academic Press, 1992.

[31] M. Bain and S. Muggleton. Learning optimal chess strategies. In S. Muggleton, D. Michie, and K. Furukawa, editors, *Machine Intelligence*, volume 13. Oxford University Press, 1993.

[32] M. Bain and A. Srinivasan. Incremental inductive logic programming from large scale unstructured data. In *Machine Intelligence*, volume 14. Oxford University Press, 1994.

[33] R.B. Banerji. Learning theoretical terms. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 3–22, 1991.

[34] R.B. Banerji. Learning theoretical terms. In S. Muggleton, editor, *Inductive Logic Programming*, pages 93–112. Academic Press, 1992.

[35] S. Bell. Discovery and maintenance of functional dependencies by independencies. In *Proceedings of the Workshop on Knowledge Discovery in Databases*, pages 27–32. AAAI Press, 1995.

[36] S. Bell. Deciding distinctness of query result by discovered constraints. In *Practical Application of Constraint Technology*, pages 399–416. Practical Application Company, 1996.

[37] S. Bell and S. Weber. On the close logical relationship between FOIL and the frameworks of Helft and Plotkin. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 1–10. J. Stefan Institute, 1993.

[38] F. Bergadano. Test case generation by means of learning techniques. In *Proceedings of the ACM-SIGSOFT Conference*, 1993.

[39] F. Bergadano. Towards an inductive logic programming language. Technical Report ESPRIT project no. 6020 ILP Deliverable TO1, Computer Science Department, University of Torino, 1993.

[40] F. Bergadano, S. Brussoti, D. Gunetti, and U. Trinchero. Inductive test case generation. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 11–24. J. Stefan Institute, 1993.

[41] F. Bergadano and D. Gunetti. Functional inductive logic programming with queries to the user. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 323–328. Springer-Verlag, 1993.

[42] F. Bergadano and D. Gunetti. Inductive synthesis of logic programs and inductive logic programming. In Y. Deville, editor, *Proceedings of the 3rd International Workshop on Logic Program Synthesis and Transformation*, pages 45–56. Workshops in Computing, Springer-Verlag, 1993.

[43] F. Bergadano and D. Gunetti. An interactive system to learn functional logic programs. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1044–1049. Morgan Kaufmann, 1993.

[44] F. Bergadano and D. Gunetti. Learning relations: Basing top-down methods on inverse resolution. In P. Torasso, editor, *Proceedings of the 3rd Congress of the Italian Association for Artificial Intelligence*, volume 728 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1993.

[45] F. Bergadano and D. Gunetti. Learning clauses by tracing derivations. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 11–30. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[46] F. Bergadano and D. Gunetti. Learning relations and logic programs. *The Knowledge Engineering Review*, 9(1), 1994.

[47] F. Bergadano and D. Gunetti. *Inductive Logic Programming: From Machine Learning to Software Engineering.* The MIT Press, 1995.

[48] F. Bergadano, D. Gunetti, M. Nicosia, and G Ruffo. Learning logic programs with negation as failure. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 33–52. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[49] F. Bergadano, D. Gunetti, M. Nicosia, and G. Ruffo. Learning logic programs with negation as failure. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 107–123. IOS Press, 1996.

[50] F. Bergadano, D. Gunetti, and U. Trinchero. The difficulties of learning logic programs with cut. *Journal of Artificial Intelligence Research*, 1:91–107, 1993.

[51] G. Bisson. Conceptual clustering in a first-order logic representation. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 458–462. John Wiley, 1992.

[52] G. Bisson. Learning in FOL with a similarity measure. In *Proceedings of the 10th National Conference on Artificial Intelligence*. Morgan Kaufmann, 1992.

[53] E. Bleken. Machine learning in an adventure game. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1992.

[54] H. Blockeel. Natural language processing and inductive logic programming. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1994.

[55] H. Blockeel and L. De Raedt. Inductive database design. In *Proceedings of the 10th International Symposium on Methodologies for Intelligent Systems*, volume 1079 of *Lecture Notes in Artificial Intelligence*, pages 376–385. Springer-Verlag, 1996.

[56] H. Blockeel and L. De Raedt. Relational knowledge discovery in databases. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing*, pages 111–124, 1996.

[57] H. Blockeel and L. De Raedt. Relational knowledge discovery in databases. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 1–13. Stockholm University, Royal Institute of Technology, 1996.

[58] D. Bonne and A. Polley. An application of inductive logic programming in banking. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1994. (In Dutch).

[59] H. Boström. *Explanation-Based Transformation of Logic Programs.* PhD thesis, Department of Computer and Systme Sciences, Stockholm University and Royal Institute of Technology, 1993.

[60] H. Boström. Improving example-guided unfolding. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 124–135. Springer-Verlag, 1993.

[61] H. Boström. Covering vs. divide-and-conquer for top-down induction of logic programs. In C.S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1194–1200. Morgan Kaufmann, 1995.

[62] H. Boström. Specialization of recursive predicates. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 92–106. Springer-Verlag, 1995.

[63] H. Boström. Theory-guided induction of logic programs by inference of regular languages. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*, pages 46–53. Morgan Kaufmann, 1996.

[64] H. Boström and P. Idestam-Almquist. Specialization of logic programs by pruning SLD-trees. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 31–48. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[65] I. Bratko. Innovative design as learning from examples. In *Proceedings International Conference Design to Manufacture in Modern Industry*, pages 355–362, 1993.

[66] I. Bratko. Machine learning and qualitative reasoning. *Machine Learning*, 14(3):305–312, 1994.

[67] I. Bratko and S. Džeroski. Engineering applications of ILP. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):313–333, 1995.

[68] I. Bratko and M. Grobelnik. Inductive learning applied to program construction and verification. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 279–292. J. Stefan Institute, 1993.

[69] I. Bratko and M. Grobelnik. Inductive learning applied to program construction and verification. In J. Cuena, editor, *Pattern-Directed Inference Systems*, pages 169–182. North-Holland, 1993.

[70] I. Bratko and R.D. King. Applications of inductive logic programming. *SIGART Bulletin*, 5(1):43–56, 1994.

[71] I. Bratko and S. Muggleton. Applications of inductive logic programming. *Communications of the ACM*, 1995.

[72] I. Bratko, S. Muggleton, and A. Varšek. Learning qualitative models of dynamic systems. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 207–224, 1991.

[73] I. Bratko, S. Muggleton, and A. Varšek. Learning qualitative models of dynamic systems. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 385–388. Morgan Kaufmann, 1991.

[74] I. Bratko, S. Muggleton, and A. Varšek. Learning qualitative models of dynamic systems. In S. Muggleton, editor, *Inductive Logic Programming*, pages 437–452. Academic Press, 1992.

[75] P. Brazdil and A. Jorge. Modular approach to ILP: Learning from interaction between modules. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[76] P. Brazdil and A. Jorge. Exploiting algorithm sketches in ILP. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 193–204. J. Stefan Institute, 1993.

[77] P. Brazdil and A. Jorge. Learning by refining algorithm sketches. In A.G. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 443–447. John Wiley, 1994.

[78] P. Brazdil and S. Muggleton. Learning to relate terms in a multiple agent environment. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 424–439. Springer-Verlag, 1991.

[79] P. Brockhausen and K. Morik. Direct access of an ILP algorithm to a database management system. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing*, pages 95–110, 1996.

[80] C.A. Brunk and M.J. Pazzani. An investigation of noise-tolerant relational concept learning algorithms. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 389–393. Morgan Kaufmann, 1991.

[81] W. Buntine. Induction of Horn-clauses: Methods and the plausible generalization algorithm. *International Journal of Man-Machine Studies*, 26:499–520, 1987.

[82] W. Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36:375–399, 1988.

[83] R. Camacho. Learning stage transition rules with Indlog. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 273–290. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[84] M.R. Cameron-Jones and J.R. Quinlan. Efficient top-down induction of logic programs. *SIGART Bulletin*, 5(1):33–42, 1994.

[85] R.M. Cameron-Jones and J.R. Quinlan. Avoiding pitfalls when learning recursive theories. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1050–1057. Morgan Kaufmann, 1993.

[86] M. Champesme, P. Brézellec, and H. Soldano. Empirically conservative search space reductions. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 387–402. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[87] G. Chaouat. Towards an autonomous learning agent in an adventure game. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1991.

[88] V. Coget. Learning evaluation functions in an adventure game. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1993.

[89] W.W. Cohen. Compiling prior knowledge into an explicit bias. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 102–110. Morgan Kaufmann, 1992.

[90] W.W. Cohen. Learnability of restricted logic programs. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 41–72. J. Stefan Institute, 1993.

[91] W.W. Cohen. PAC-learning a restricted class of recursive logic programs. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 73–86. J. Stefan Institute, 1993.

[92] W.W. Cohen. Rapid prototyping of ILP systems using explicit bias. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 24–35. Morgan Kaufmann, 1993.

[93] W.W. Cohen. Rule learning systems. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 988–994. Morgan Kaufmann, 1993.

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

[95] W.W. Cohen. Recovering software specifications with ILP. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 142–148. Morgan Kaufmann, 1994.

[96] W.W. Cohen. Learning to classify English text with ILP methods. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 3–24. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[97] W.W. Cohen. Learning to classify English text with ILP methods. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 124–143. IOS Press, 1996.

[98] W.W. Cohen and C.D. Page. Polynomial learnability and inductive logic programming: Methods and results. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):369–410, 1995.

[99] D. Conklin and I.H. Witten. Complexity-based induction. *Machine Learning*, 16(3):203–226, 1994.

[100] A. Cornuejols. Getting order independence in incremental learning. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 196–212. Springer-Verlag, 1993.

[101] J. Cussens. Estimating rule accuracies from training data. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[102] P. Datta and D. Kibler. Concept sharing: A means to improve multi-concept learning. In P. Utgoff, editor, *Proceedings of the 10th International Conference on Machine Learning*, pages 89–96. Morgan Kaufmann, 1993.

[103] L. De Raedt. *Interactive Concept-Learning.* PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1991.

[104] L. De Raedt. Inductive logic programming: An introduction. In *Proceedings of the 14th International Conference on Information Technology Interfaces*, pages 17–22, 1992. (Invited paper).

[105] L. De Raedt. *Interactive Theory Revision: An Inductive Logic Programming Approach.* Academic Press, 1992.

[106] L. De Raedt. A brief introduction to inductive logic programming. In D. Miller, editor, *Proceedings of the 7th International Symposium on Logic Programming*, pages 45–51. The MIT Press, 1993.

[107] L. De Raedt. Inductive logic programming and scientific discovery. *Academia Analecta*, 57(2):101–128, 1995.

[108] L. De Raedt, editor. *Advances in Inductive Logic Programming.* IOS Press, 1996.

[109] L. De Raedt. Induction in logic. In *Proceedings of the 3nd International Workshop on Multistrategy Learning*, pages 29–38. AAAI Press, 1996.

[110] L. De Raedt. The inductive logic programming project. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 3–13. IOS Press, 1996.

[111] L. De Raedt. PAC-learning logic programs under the closed world assumption. In *Proceedings of the 10th International Symposium on Methodologies for Intelligent Systems*, volume 1079 of *Lecture Notes in Artificial Intelligence*, pages 531–540. Springer-Verlag, 1996.

[112] L. De Raedt, E. Bleken, V. Coget, C. Ghil, B. Swennen, and M. Bruynooghe. Learning to survive. In *Proceedings of the 2nd International Workshop on Multistrategy Learning*, pages 92–106, 1993.

[113] L. De Raedt and M. Bruynooghe. On interactive concept-learning and assimilation. In D. Sleeman, editor, *Proceedings of the 3rd European Working Session on Learning.* Pitman, 1988.

[114] L. De Raedt and M. Bruynooghe. Constructive induction by analogy. In *Proceedings of the 6th International Workshop on Machine Learning*, pages 476–477. Morgan Kaufmann, 1989.

[115] L. De Raedt and M. Bruynooghe. Constructive induction by analogy: A method to learn how to learn? In K. Morik, editor, *Proceedings of the 4th European Working Session on Learning*, pages 189–200. Pitman, 1989.

[116] L. De Raedt and M. Bruynooghe. On explanation and bias in inductive concept-learning. In *Proceedings of the 3rd European Knowledge Acquisition for Knowledge Based Systems Workshop*, pages 338–353, 1989.

[117] L. De Raedt and M. Bruynooghe. Towards friendly concept-learners. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 849–856. Morgan Kaufmann, 1989.

[118] L. De Raedt and M. Bruynooghe. Indirect relevance and bias in inductive concept-learning. *Knowledge Acquisition*, 2:365–390, 1990.

[119] L. De Raedt and M. Bruynooghe. Interactief leren van concepten: Een overzicht. In *Proceedings of the 3rd Dutch AI Conference*, 1990. (In Dutch).

[120] L. De Raedt and M. Bruynooghe. On negation and three-valued logic in interactive concept-learning. In L.C. Aiello, editor, *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 207–212. Pitman, 1990.

[121] L. De Raedt and M. Bruynooghe. Using interactive concept-learning for knowledge base validation, verification and testing. In *Proceedings of the ECAI-90 Workshop on Validation, Verification and Testing of Knowledge-Based Systems*, 1990.

[122] L. De Raedt and M. Bruynooghe. CLINT: A multistrategy interactive concept-learner and theory revision system. In *Proceedings of the 1st International Workshop on Multistrategy Learning*, pages 175–191, 1991.

[123] L. De Raedt and M. Bruynooghe. Belief updating from integrity constraints and queries. *Artificial Intelligence*, 53:291–307, 1992.

[124] L. De Raedt and M. Bruynooghe. A clausal discovery engine. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992. (Extended abstract).

[125] L. De Raedt and M. Bruynooghe. A clausal discovery engine. In *Proceedings of the 6th International School for the Synthesis of Expert Knowledge*, 1992. (Extended abstract).

[126] L. De Raedt and M. Bruynooghe. Discovering clausal theories in databases. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[127] L. De Raedt and M. Bruynooghe. Discovering integrity constraints in deductive databases. In *Proceedings of the 1st Esprit Compulog Network Workshop on Logic Programming and Artficial Intelligence*, 1992. (Extended abstract).

[128] L. De Raedt and M. Bruynooghe. Interactive concept learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150, 1992.

[129] L. De Raedt and M. Bruynooghe. An overview of the interactive concept-learner and theory revisor CLINT. In S. Muggleton, editor, *Inductive Logic Programming*, pages 163–192. Academic Press, 1992.

[130] L. De Raedt and M. Bruynooghe. A unifying framework for concept-learning algorithms. *The Knowledge Engineering Review*, 7(3):251–269, 1992.

[131] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann, 1993.

[132] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 25–40. J. Stefan Institute, 1993.

[133] L. De Raedt and M. Bruynooghe. Interactive theory revision. In R.S. Michalski and G. Tecuci, editors, *Machine Learning: A Multistrategy Approach*, volume 4. Morgan Kaufmann, 1994.

[134] L. De Raedt, M. Bruynooghe, and B. Martens. Integrity constraints and interactive concept-learning. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 394–398. Morgan Kaufmann, 1991.

[135] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 1996. (To appear).

[136] L. De Raedt and S. Džeroski. First-order jk-clausal theories are PAC-learnable. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 49–50. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994. (Abstract).

[137] L. De Raedt and S. Džeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.

[138] L. De Raedt, J. Feyaerts, and M. Bruynooghe. Acquiring object-knowledge for learning systems. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 245–264. Springer-Verlag, 1991.

[139] L. De Raedt, J. Feyaerts, and M. Bruynooghe. Acquiring object-knowledge. *Journal of Experimental and Theoretical Artificial Intelligence*, 4:213–232, 1992.

[140] L. De Raedt, B. Krekels, M. Bruynooghe, and D. Van Meir. Using Shapiro's model inference system for concept-learning. In *Proceedings of the 4th International Conference on Artificial Intelligence and Control Systems of Robots*, pages 191–196, 1987.

[141] L. De Raedt and N. Lavrač. The many faces of inductive logic programming. In J. Komorowski and Z.W. Raś, editors, *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems*, volume 689 of *Lecture Notes in Artificial Intelligence*, pages 435–449. Springer-Verlag, 1993. (Invited paper).

[142] L. De Raedt and N. Lavrač. Multiple predicate learning in two inductive logic programming settings. *Journal on Pure and Applied Logic*, 4(2):227–254, 1996.

[143] L. De Raedt, N. Lavrač, and S. Džeroski. Multiple predicate learning. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 221–240. J. Stefan Institute, 1993.

[144] L. De Raedt, N. Lavrač, and S. Džeroski. Multiple predicate learning. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1037–1043. Morgan Kaufmann, 1993.

[145] L. De Raedt, G. Sablon, and M. Bruynooghe. Using interactive concept-learning for knowledge base validation and verification. In M. Ayel and J.P. Laurent, editors, *Validation, Verification and Testing of Knowledge Based Systems*, pages 177–190. John Wiley, 1991.

[146] L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proceedings of the 5th Conference on Algorithmic Learning Theory*, volume 997 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1995.

[147] L. Dehaspe, H. Blockeel, and L. De Raedt. Induction, logic and natural language processing. In *Proceedings of the joint ELSNET/COMPULOG-NET/EAGLES Workshop on Computational Logic for Natural Language Processing*, 1995.

[148] L. Dehaspe and L. De Raedt. Parallel inductive logic programming. In *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, 1995.

[149] L. Dehaspe and L. De Raedt. DLAB: A declarative language bias formalism. In *Proceedings of the 10th International Symposium on Methodologies for Intelligent Systems*, volume 1079 of *Lecture Notes in Artificial Intelligence*, pages 613–622. Springer-Verlag, 1996.

[150] L. Dehaspe, L. De Raedt, and W. Van Laer. CLAUDIEN: User's manual. Technical report, Department of Computer Science, Katholieke Universiteit Leuven, 1994.

[151] L. Dehaspe, W. Van Laer, and L. De Raedt. Applications of a logical discovery engine. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 291–304. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[152] T.G. Dietterich and R.S. Michalski. Learning to predict sequences. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine learning: An artificial intelligence approach*, volume 2. Morgan Kaufmann, 1986.

[153] Y. Dimopoulos and A. Kakas. Learning non-monotonic logic programs: Learning exceptions. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 122–137. Springer-Verlag, 1995.

[154] Y. Dimopoulos and A. Kakas. Abduction and inductive learning. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 144–171. IOS Press, 1996.

[155] B. Dolšak. Constructing finite element meshes using artificial intelligence methods. Master's thesis, Faculty of Technical Sciences, University of Maribor, Slovenia, 1991. (In Slovenian).

[156] B. Dolšak. *A contribution to Intelligent Mesh Design for FEM Analyses.* PhD thesis, Faculty of Technical Sciences, University of Maribor, Slovenia, 1996. (In Slovenian).

[157] B. Dolšak, I. Bratko, and A. Jezernik. Finite element mesh design: An engineering domain for ILP application. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 305–320. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[158] B. Dolšak, A. Jezernik, and I. Bratko. A knowledge base for finite element mesh design. In *Proceedings of the 6th International School for the Synthesis of Expert Knowledge*, 1992.

[159] B. Dolšak and S. Muggleton. The application of ILP to finite element mesh design. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 225–242, 1991.

[160] B. Dolšak and S. Muggleton. The application of inductive logic programming to finite-element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*, pages 453–472. Academic Press, 1992.

[161] M.J. Dovey. Analysis of Rachmaninoff's piano performances using inductive logic programming. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 279–282. Springer-Verlag, 1995.

[162] S. Džeroski. Handling noise in inductive logic programming. Master's thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, 1991.

[163] S. Džeroski. Learning qualitative models with inductive logic programming. *Informatica*, 16(4):30–41, 1992.

[164] S. Džeroski. Handling imperfect data in inductive logic programming. In *Proceedings of the 4th Scandinavian Conference on Artificial Intelligence*, pages 111–125. IOS Press, 1993.

[165] S. Džeroski. Inductive logic programming and knowledge discovery in databases. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 118–152. The MIT Press, 1995.

[166] S. Džeroski. Learning first-order clausal theories in the presence of noise. In A. Aamodt and J. Komorowski, editors, *Proceedings of the 5th Scandinavian Conference on Artificial Intelligence*, pages 51–60. IOS, Amsterdam, 1995.

[167] S. Džeroski. *Numerical constraints and learnability in inductive logic programming.* PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia, 1995.

[168] S. Džeroski and I. Bratko. Handling noise in inductive logic programming. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[169] S. Džeroski and I. Bratko. Using the m-estimate in inductive logic programming. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[170] S. Džeroski and I. Bratko. Applications of inductive logic programming. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 65–81. IOS Press, 1996.

[171] S. Džeroski, L. Dehaspe, B. Ruck, and W. Walley. Classification of river water quality data using machine learning. In *Proceedings of the 5th International Conference on the Development and Application of Computer Techniques to Environmental Studies*, 1995.

[172] S. Džeroski and B. Dolšak. A comparison of relation learning algorithms on the problem of finite element mesh design. In *Proceedings of the 26th Yugoslav Conference of the Society for ETAN*, 1991. In Slovenian.

[173] S. Džeroski and B. Dolšak. Comparison of ILP systems on the problem of finite element mesh design. In *Proceedings of the 6th International School for the Synthesis of Expert Knowledge*, 1992.

[174] S. Džeroski and N. Lavrač. Learning relations from imperfect data. Technical Report IJS-DP-6163, Jožef Stefan Institute, 1991.

[175] S. Džeroski and N. Lavrač. Learning relations from noisy examples: An empirical comparison of LINUS and FOIL. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 399–402. Morgan Kaufmann, 1991.

[176] S. Džeroski and N. Lavrač. Refinement graphs for FOIL and LINUS. In S. Muggleton, editor, *Inductive Logic Programming*, pages 319–334. Academic Press, 1992.

[177] S. Džeroski and N. Lavrač. Inductive learning in deductive databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):939–949, 1993.

[178] S. Džeroski, S. Muggleton, and S. Russel. Learnability of constrained logic programs. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 342–347. Springer-Verlag, 1993.

[179] S. Džeroski, S. Muggleton, and S. Russell. PAC-learnability of constrained nonrecursive logic programs. In *Proceedings of the 3rd International Workshop on Computational Learning Theory and Natural Learning Systems*, 1992.

[180] S. Džeroski, S. Muggleton, and S. Russell. PAC-learnability of determinate logic programs. In *Proceedings of the 5th ACM Workshop on Computational Learning Theory*, pages 128–135. ACM Press, 1992.

[181] S. Džeroski, S. Schulze-Kremer, K.R. Heidtke, K. Siems, and D. Wettschereck. Applying ILP to diterpene structure elucidation from ^{13}C NMR spectra. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 14–27. Stockholm University, Royal Institute of Technology, 1996.

[182] S. Džeroski, S. Schulze-Kremer, R.K. Heidtke, K. Siems, and D. Wettschereck. Applying ILP to diterpene structure elucidation from ^{13}C NMR spectra. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing*, pages 12–24, 1996.

[183] S. Džeroski, L. Todoroski, and T. Urbančič. Handling real numbers in inductive logic programming: A step towards better behavioural clones. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 283–286. Springer-Verlag, 1995.

[184] S. Džeroski and L. Todorovski. Discovering dynamics. In *Proceedings of the AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 125–137. AAAI Press, 1993.

[185] S. Džeroski and L. Todorovski. Discovering dynamics: From inductive logic programming to machine discovery. In *Proceedings of the 10th International Conference on Machine Learning*, pages 97–103. Morgan Kaufmann, 1993.

[186] S. Džeroski and L. Todorovski. Discovery dynamics: From inductive logic programming to machine discovery. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 1–13. Morgan Kaufmann, 1993.

[187] S. Džeroski and L. Todorovski. Handling real numbers in inductive logic programming. In *Proceedings of the 3rd Electrotechnical and Computer Science Conference*, volume B, pages 143–146. Slovenian Section IEEE, Ljubljana, 1994.

[188] S. Džeroski and L. Todorovski. Discovering dynamics: From inductive logic programming to machine discovery. *Journal of Intelligent Information Systems*, 4:89–108, 1995.

[189] W. Emde. Inductive learning of characteristic concept descriptions. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 51–70. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[190] W. Emde. Inductive learning of characteristic concept descriptions from small sets to classified examples. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 103–121. Springer-Verlag, 1994.

[191] W. Emde and D. Wettschereck. Relational instance-based learning. In *Fachgruppentreffen der Fachgruppe Machinelles Lernen der GI, FGML'95*. University of Dortmund, 1995.

[192] W. Emde and D. Wettschereck. Relational instance-based learning. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann, 1996.

[193] F. Esposito, A. Laterza, D. Malerba, and G. Semeraro. Refinement of Datalog programs. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing*, pages 73–94, 1996.

[194] F. Esposito, D. Malerba, and G. Semeraro. Automated acquisition of rules for document understanding. In *Proceedings of the 2nd International Conference on Document Analysis and Recognition*, pages 650–654, 1993.

[195] F. Esposito, D. Malerba, and G. Semeraro. Learning contextual rules in first-order logic. In *Proceedings of the 4th Italian Workshop on Machine Learning*, pages 111–127, 1993.

[196] F. Esposito, D. Malerba, and G. Semeraro. Multistrategy learning for document recognition. *Applied Artificial Intelligence*, 8:33–84, 1994.

[197] C. Feng. Inducing temporal fault diagnostic rules from a qualitative model. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 243–258, 1991.

[198] C. Feng. Inducing temporal fault diagnostic rules from a qualitative model. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 403–406. Morgan Kaufmann, 1991.

[199] C. Feng. Inducing temporal fault diagnostic rules from a qualitative model. In S. Muggleton, editor, *Inductive Logic Programming*, pages 473–494. Academic Press, 1992.

[200] C. Feng and S. Muggleton. Towards inductive generalization in higher order logic. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[201] C. Feng and S. Muggleton. Towards inductive generalization in higher order logic. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 154–162. Morgan Kaufmann, 1992.

[202] D. Fensel, M. Zickwolff, and M. Wiese. Are substitutions the better examples? Learning complete sets of clauses with Frog. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 453–474. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[203] P.A. Flach. Inductive characterisation of database relations. In Z.W. Raś, M. Zemankova, and M.L. Emrich, editors, *Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems*, volume 5, pages 371–378. North-Holland, 1990.

[204] P.A. Flach. Towards a logical theory of inductive learning. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 23–32, 1991.

[205] P.A. Flach. Towards a theory of inductive logic programming. In Z.W. Raś and M. Zemankowa, editors, *Proceedings of the 5th International Symposium on Methodologies for Intelligent Systems*, volume 542 of *Lecture Notes in Artificial Intelligence*, pages 510–519. Springer-Verlag, 1991.

[206] P.A. Flach. An analysis of various forms of 'jumping to conclusions'. In K. Jantke, editor, *Proceedings 3rd International Workshop on Analogical and Inductive Inference*, volume 642 of *Lecture Notes in Artificial Intelligence*, pages 170–186. Springer-Verlag, 1992.

[207] P.A. Flach. A framework for inductive logic programming. In S. Muggleton, editor, *Inductive Logic Programming*, pages 193–212. Academic Press, 1992.

[208] P.A. Flach. Generality revisited. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[209] P.A. Flach. Logical approaches to machine learning - An overview. *THINK*, 1(2):25–36, 1992.

[210] P.A. Flach. Predicate invention in inductive data engineering. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 83–94. Springer-Verlag, 1993.

[211] P.A. Flach. Inductive logic programming and philosophy of science. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 71–84. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[212] P.A. Flach. A model of inductive reasoning. In M. Masuch and L. Polos, editors, *Knowledge Representation and Reasoning under Uncertainty*, volume 808 of *Lecture Notes in Artificial Intelligence*, pages 41–56. Springer-Verlag, 1994.

[213] P.A. Flach. *Simply Logical: Intelligent Reasoning by Example*. John Wiley, 1994.

[214] P.A. Flach. *Conjectures: An Inquiry Concerning the Logic of Induction*. PhD thesis, Katholieke Universiteit Brabant, 1995.

[215] P.A. Flach. Abduction and induction: inferential and syllogistic perspectives. In P.A. Flach and A. Kakas, editors, *Proceedings of the ECAI-96 Workshop on Abductive and Inductive Reasoning*, pages 31–35, 1996.

[216] P.A. Flach. Rationality postulates for induction. In Y. Shoham, editor, *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 267–281. Morgan Kaufmann, 1996.

[217] P. Flener. *Logic Program Synthesis from Incomplete Information*, volume 295 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, 1994.

[218] P. Flener. Inductive logic program synthesis with Dialogs. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 28–51. Stockholm University, Royal Institute of Technology, 1996.

[219] M. Franova, Y. Kodratoff, and M. Gross. Constructive matching methodology: Formally creative or intelligent inductive theorem proving? In J. Komorowski and Z.W. Raś, editors, *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems*, volume 689 of *Lecture Notes in Artificial Intelligence*, pages 476–485. Springer-Verlag, 1993.

[220] M. Frazier and C.D. Page. Learnability in inductive logic programming: Some basic results and technique. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 93–98. Morgan Kaufmann, 1993.

[221] M. Frazier and C.D. Page. Learnability of recursive, non-determinate theories: Some basic results and techniques. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 103–126. J. Stefan Institute, 1993.

[222] A.M. Frisch and C.D. Page. Building theories into instantiation. In C.S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1210–1216. Morgan Kaufmann, 1995.

[223] H. Fujita, N. Yagi, T. Ozaki, and K. Furukawa. A new design and implementation of Progol by bottom-up computation. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 52–58. Stockholm University, Royal Institute of Technology, 1996.

[224] J. Fürnkranz. Avoiding noise fitting in a FOIL-like learning algorithm. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 14–23. Morgan Kaufmann, 1993.

[225] J. Fürnkranz. A comparison of pruning methods for relational concept learning. In *Proceedings of the AAAI-94 Workshop on Knowledge Discovery in Databases*, 1994.

[226] J. Fürnkranz. *Efficient Pruning Methods for Relational Learning.* PhD thesis, Technical University of Vienna, 1994.

[227] J. Fürnkranz. Fossil: A robust relational learner. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 122–137. Springer-Verlag, 1994.

[228] J. Fürnkranz. Pruning methods for rule learning algorithms. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 321–336. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[229] J. Fürnkranz. Top-down pruning in relational learning. In A.G. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 453–457. John Wiley, 1994.

[230] J. Fürnkranz. A tight integration of pruning and learning. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 291–294. Springer-Verlag, 1995.

[231] J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In W.W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 70–77. Morgan Kaufmann, 1994.

[232] D. Gabbay, D. Gilles, A. Hunter, S. Muggleton, Y. Ng, and B. Richards. The rule-based systems project. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 95–106, 1991.

[233] D. Gabbay, D. Gillies, A. Hunter, S. Muggleton, Y. Ng, and B. Richards. The rule-based systems project: Using confirmation theory and non-monotonic logics for incremental learning. In S. Muggleton, editor, *Inductive Logic Programming*, pages 213–230. Academic Press, 1992.

[234] D. Gamberger and N. Lavrač. Noise detection and elimination applied to noise handling in a KRK chess endgame. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 59–75. Stockholm University, Royal Institute of Technology, 1996.

[235] P. Geibel and F. Wysotzki. Learning context dependent concepts. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 323–337. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[236] P. Geibl and F. Wysotzki. Learning relational concepts with decision trees. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*, pages 166–174. Morgan Kaufmann, 1996.

[237] P. Geibl and F. Wysotzki. Relational learning with decision trees. In W. Wahlster, editor, *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 428–432. John Wiley, 1996.

[238] R. Gennaro. PAC-learning Prolog clauses with or without errors. Technical Report 500, Laboratory for Computer Science, MIT, 1995.

[239] D. Gillies. Confirmation theory and machine learning. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[240] A. Giordana and F. Neri. Search-intensive concept induction. *Evolutionary Computation Journal*, 3(4):375–416, 1996.

[241] A. Giordana, L. Saitta, and D. Roverso. Abstracting concepts with inverse resolution. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 142–146. Morgan Kaufmann, 1991.

[242] A. Giordana and C. Sale. Learning structured concepts using genetic algorithms. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 169–178. Morgan Kaufmann, 1992.

[243] M-E. Goncalves. Handling quantifiers in ILP. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 76–96. Stockholm University, Royal Institute of Technology, 1996.

[244] M. Grobelnik. MARKUS: An optimized model inference system. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[245] M. Grobelnik. Induction of Prolog programs with MARKUS. In Y. Deville, editor, *Proceedings of the 3rd International Workshop on Logic Program Synthesis and Transformation.* Workshops in Computing, Springer-Verlag, 1993.

[246] D. Gunetti. Using abduction to learn Horn theories. In G. della Riccia, R. Kruse, and R. Viertl, editors, *Proceedings of the Workshop on Mathematical and Statistical Methods in Artificial Intelligence*, volume 363 of *CISM Courses and Lectures*, pages 131–146. Springer-Verlag, 1995.

[247] D. Gunetti and U. Trinchero. Intensional learning of logic programs. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 359–362. Springer-Verlag, 1994.

[248] T. Gyimóthy and J. Paakki. Static slicing of logic programs. In M. Ducasse, editor, *Proceedings of AADEBUG'95*, pages 87–105. IRISA/INSA France, 1995.

[249] A. Hamfelt and J.F. Nilsson. Inductive metalogic programming. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 85–96. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[250] M. Harao. Abstraction based analogical reasoning for natural deduction proof. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[251] N. Helft. Induction as nonmonotonic inference. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 149–156. Morgan Kaufmann, 1989.

[252] E. Hirowatari and S. Arikawa. Explanation-based generalization by analogical reasoning. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[253] T. Horváth, R. Sloan, and G. Turán. Learning logic programs with random classification noise. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 97–118. Stockholm University, Royal Institute of Technology, 1996.

[254] T. Horváth and G. Turán. Learning logic programs with structured background knowledge. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 53–76. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[255] T. Horváth and G. Turán. Learning logic programs with structured background knowledge. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 172–191. IOS Press, 1996.

[256] D. Hume. Learning procedures by environment-driven constructive induction. In B. Porter and R.J. Mooney, editors, *Proceedings of the 7th International Conference on Machine Learning*, pages 113–121. Morgan Kaufmann, 1990.

[257] D. Hume. Using inverse resolution to learn relations from experiments. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 412–416. Morgan Kaufmann, 1991.

[258] D. Hume and C. Sammut. Applying inductive logic programming in reactive environments. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 279–290, 1991.

[259] D. Hume and C. Sammut. Using inverse resolution to learn relations from experiments. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 412–416. Morgan Kaufmann, 1991.

[260] D. Hume and C. Sammut. Applying inductive logic programming in reactive environments. In S. Muggleton, editor, *Inductive Logic Programming*, pages 539–549. Academic Press, 1992.

[261] P. Idestam-Almquist. Learning missing clauses by inverse resolution. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1992.

[262] P. Idestam-Almquist. *Generalization of clauses.* PhD thesis, Stockholm University, Department of Computer and Systems Sciences, 1993.

[263] P. Idestam-Almquist. Generalization under implication by recursive anti-unification. In P. Utgoff, editor, *Proceedings of the 10th International Conference on Machine Learning*, pages 151–158. Morgan Kaufmann, 1993.

[264] P. Idestam-Almquist. Generalization under implication by using or-introduction. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 56–64. Springer-Verlag, 1993.

[265] P. Idestam-Almquist. Recursive anti-unification. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 241–254. J. Stefan Institute, 1993.

[266] P. Idestam-Almquist. Efficient induction of recursive definitions by structural analysis of saturations. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 77–94. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[267] P. Idestam-Almquist. Efficient induction of recursive definitions by structural analysis of saturations. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 192–205. IOS Press, 1996.

[268] N. Inuzuka, M. Kamo, N. Ishii, H. Seki, and H. Itoh. Top-down induction of logic programs from incomplete samples. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 119–136. Stockholm University, Royal Institute of Technology, 1996.

[269] C.G. Jansson, H. Boström, and P. Idestam-Almquist. Theory revision in a logic programming framework. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[270] K. Jantke. Fundamental properties of inductive reasoning. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[271] A. Jorge and P. Brazdil. Architecture for iterative learning of recursive definitions. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 95–108. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[272] A. Jorge and P. Brazdil. Learning recursion with iterative bootstrap induction. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 299–302. Springer-Verlag, 1995.

[273] A. Jorge and P. Brazdil. Architecture for iterative learning of recursive definitions. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 206–218. IOS Press, 1996.

[274] A. Jorge and P. Brazdil. Integrity constraints in ILP using a Monte Carlo approach. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 137–151. Stockholm University, Royal Institute of Technology, 1996.

[275] B. Jung. On inverting generality relations. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 87–102. J. Stefan Institute, 1993.

[276] A. Kakas. Abduction and inductive learning. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 25–28. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[277] A. Karalič. Relational regression: First steps. Technical Report IJS-DP-7001, Jožef Stefan Institute, Ljubljana, 1994.

[278] A. Karalič. *First order regression.* PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia, 1995.

[279] A. Karalič. First-order regression: Applications in real-world domains. In *Proceedings of the 2nd International Workshop on Artificial Intelligence Techniques*, 1995.

[280] T. Kawamura and K. Furukawa. Towards inductive generalization in constraint logic programs. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 93–104. Morgan Kaufmann, 1993.

[281] J-U. Kietz. A comparative study of structural most specific generalizations used in machine learning. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[282] J-U. Kietz. A comparative study of structural most specific generalizations used in machine learning. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 149–164. J. Stefan Institute, 1993.

[283] J-U. Kietz. Some lower bounds for the computational complexity of inductive logic programming. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 115–123. Springer-Verlag, 1993.

[284] J-U. Kietz and S. Džeroski. Inductive logic programming and learnability. *SIGART Bulletin*, 5(1):22–32, 1994.

[285] J-U. Kietz and M. Lübbe. An efficient subsumption algorithm for inductive logic programming. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 97–106. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[286] J-U. Kietz and M. Lübbe. An efficient subsumption algorithm for inductive logic programming. In W.W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 130–138. Morgan Kaufmann, 1994.

[287] J-U. Kietz and K. Morik. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14(2):193–218, 1994.

[288] J-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 107–126, 1991.

[289] J-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*, pages 335–359. Academic Press, 1992.

[290] B. Kijsirikul, M. Numao, and M. Shimura. Efficient learning of logic programs with non-determinate, non-discriminating literals. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 33–40, 1991.

[291] B. Kijsirikul, M. Numao, and M. Shimura. Efficient learning of logic programs with non-determinante, non-discriminating literals. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 417–421. Morgan Kaufmann, 1991.

[292] B. Kijsirikul, M. Numao, and M. Shimura. Discrimination-based constructive induction of logic programs. In *Proceedings of the 10th National Conference on Artificial Intelligence*. Morgan Kaufmann, 1992.

[293] B. Kijsirikul, M. Numao, and M. Shimura. Efficient learning of logic programs with non-determinate, non-discriminating literals. In S. Muggleton, editor, *Inductive Logic Programming*, pages 361–372. Academic Press, 1992.

[294] R.D. King, S. Muggleton, R. Lewis, and M.J.E Sternberg. Drug design by machine learning. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[295] R.D. King, S. Muggleton, R. Lewis, and M.J.E Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences of the USA*, 89(23):11322–11326, 1992.

[296] R.D. King, S. Muggleton, A. Srinivasan, and M.J.E. Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93:438–442, 1996.

[297] R.D. King and A. Srinivasan. Relating chemical activity to structure: An examination of ILP successes. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):411–434, 1995.

[298] M. Kirschenbaum and L.S. Sterling. Refinement strategies for inductive learning of simple Prolog programs. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1991.

[299] V. Klingspor. GRDT: Enhancing model-based learning for its application in robot navigation. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 107–122. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[300] V. Klingspor, K. Morik, and A. Rieger. Learning concepts from sensor data of a mobile robot. *Machine Learning*, 23:305–332, 1996.

[301] G. Koch. A personal approach to linguistics oriented inductive logic programming. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[302] G. Kókai, Z. Alexin, and T. Gyimóthy. Classifying ECG waveforms in Prolog. In P. Reintjes, editor, *Proceedings of the 4th International Conference on the Practical Application of Prolog*, pages 173–199. Practical Application Company Limited, 1996.

[303] G. Kókai, Z. Alexin, and T. Gyimóthy. Learning biomedical patterns. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 152–170. Stockholm University, Royal Institute of Technology, 1996.

[304] M. Kovačič. MDL heuristic in ILP revised. In *ICML94 Workshop on Applications of Descriptional Complexity to Inductive, Statistical, and Visual Inference*, 1994.

[305] M. Kovačič. MILP - A stochastic approach to inductive logic programming. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 123–138. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[306] M. Kovačič. *Stochastic Inductive Logic Programming*. PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia, 1994.

[307] M. Kovačič, N. Lavrač, M. Grobelnik, D. Zupanič, and D. Mladenić. Stochastic search in inductive logic programming. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 444–445. John Wiley, 1992.

[308] P. Langley. Induction of recursive Bayesian classifiers. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 153–164. Springer-Verlag, 1993.

[309] S. Lapointe, C.X. Ling, and S. Matwin. Constructive inductive logic programming. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 255–264. J. Stefan Institute, 1993.

[310] S. Lapointe, C.X. Ling, and S. Matwin. Constructive inductive logic programming. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1030–1036. Morgan Kaufmann, 1993.

[311] S. Lapointe and S. Matwin. Sub-unification: A tool for efficient induction of recursive programs. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 273–281. Morgan Kaufmann, 1992.

[312] N. Lavrač, S. Džeroski, and M. Grobelnik. Experiments in learning nonrecursive definitions of relations with LINUS. Technical Report IJS-DP-5863, J. Stefan Institute, Ljubljana, Slovenia, 1990.

[313] N. Lavrač. *Principles of knowledge acquisition in expert systems.* PhD thesis, Faculty of Technical Sciences, University of Maribor, 1990.

[314] N. Lavrač. Inductive concept learning using background knowledge. In T. Pequeno and F. Carvalho, editors, *Proceedings of the 11th Brasilian Symposium on Artificial Intelligence*, pages 1–16, 1994. (Invited paper).

[315] N. Lavrač. Inductive logic programming. In N.E. Fuchs and G. Gottlob, editors, *Proceedings of the 10th Logic Programming Workshop*, Technical Report, IFI-94.10. Zürich University, 1994. (Invited paper).

[316] N. Lavrač, B. Cestnik, and S. Džeroski. Search heuristics in empirical inductive logic programming. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[317] N. Lavrač, B. Cestnik, and S. Džeroski. Use of heuristics in empirical inductive logic programming. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[318] N. Lavrač and L. De Raedt. Inductive logic programming: A survey of European research. *AI Communications*, 8(1):3–19, 1995.

[319] N. Lavrač and S. Džeroski. Inductive learning of relational descriptions from noisy examples. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 259–278, 1991.

[320] N. Lavrač and S. Džeroski. Background knowledge and declarative bias in inductive concept learning. In K. Jantke, editor, *Proceedings 3rd International Workshop on Analogical and Inductive Inference*, pages 51–71. Springer-Verlag, 1992. (Invited paper).

[321] N. Lavrač and S. Džeroski. Inductive learning of relations from noisy examples. In S. Muggleton, editor, *Inductive Logic Programming*, pages 495–516. Academic Press, 1992.

[322] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[323] N. Lavrač and S. Džeroski. Weakening the language bias in LINUS. *Journal of Experimental and Theoretical Artificial Intelligence*, 6(1):95–119, 1994.

[324] N. Lavrač, S. Džeroski, and I. Bratko. Handling imperfect data in inductive logic programming. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 48–64. IOS Press, 1996.

[325] N. Lavrač, S. Džeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 265–281. Springer-Verlag, 1991.

[326] N. Lavrač, S. Džeroski, V. Pirnat, and V. Križman. Learning rules for early diagnosis of rheumatic diseases. In *Proceedings of the 3rd Scandinavian Conference on Artificial Intelligence*, pages 138–149. IOS Press, 1991.

[327] N. Lavrač, S. Džeroski, V. Pirnat, and V. Križman. The utility of background knowledge in learning medical diagnostic rules. *Applied Artificial Intelligence*, 7:273–293, 1993.

[328] N. Lavrač, D. Gamberger, and S. Džeroski. An approach to dimensionality reduction in learning from deductive databases. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 337–354. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[329] G. Le Blanc. BMWk revisited - Generalization and formalization of an algorithm for detecting recursive relations in term sequences. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 183–197. Springer-Verlag, 1994.

[330] C.X. Ling. Inductive learning from good examples. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1991.

[331] C.X. Ling. Inventing necessary theoretical terms. Technical Report Nr. 302, Dept. of Computer Science, University of Western Ontario, 1991.

[332] C.X. Ling. Logic program synthesis from good examples. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 41–58, 1991.

[333] C.X. Ling. Non-monotonic specialization. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 59–68, 1991.

[334] C.X. Ling. Logic program synthesis from good examples. In S. Muggleton, editor, *Inductive Logic Programming*, pages 113–130. Academic Press, 1992.

[335] C.X. Ling. Learning the past tense of English verbs: The symbolic pattern associator vs. connectionist models. *Journal of Artificial Intelligence Research*, 1:209–229, 1994.

[336] C.X. Ling and M.A. Narayan. Comparison of methods based on inverse resolution. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 168–172. Morgan Kaufmann, 1991.

[337] C.X. Ling and M.A. Narayan. A critical comparison of various methods based on inverse resolution. In S. Muggleton, editor, *Inductive Logic Programming*, pages 131–144. Academic Press, 1992.

[338] D. Lorenzo. Application of clausal discovery to temporal databases. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing*, pages 25–40, 1996.

[339] D. Malerba. Document understanding: A machine learning approach. Technical report, Esprit Project 5203 INTERRID, 1993.

[340] B. Malfait. Een incrementele, niet-interactieve aanpak van het Theory Revision-probleem binnen ILP. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1994. (In Dutch).

[341] Z. Markov. Inductive inference in networks of relations. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 265–278. J. Stefan Institute, 1993.

[342] Z. Markov. Relational learning by heuristic evaluation of ground data. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 337–350. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[343] Z. Markov. A functional approach to ILP. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 267–280. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[344] Z. Markov. λ-subsumption and its application to learning from positive-only examples. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 171–190. Stockholm University, Royal Institute of Technology, 1996.

[345] L. Martin and C. Vrain. MULT_ICN: An empirical multiple predicate learner. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 129–144. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[346] L. Martin and C. Vrain. A three-valued framework for the induction of general program. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 109–128. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[347] L. Martin and C. Vrain. A three-valued framework for the induction of general logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 219–235. IOS Press, 1996.

[348] M. Meyer and P. Hanschke. An alternative to q-subsuption based on terminological reasoning. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[349] R.S. Michalski. Pattern recognition as rule-guided inductive inference. In *Proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 349–361, 1980.

[350] D. Michie and R. Camacho. Building symbolic representations of intuitive real-time skills from performance data. In K. Furukawa, D. Michie, and S. Muggleton, editors, *Intelligence 13*, pages 385–418. Oxford University Press, 1994.

[351] D. Michie, S. Muggleton, D. Page, and A. Srinivasan. To the international computing community: A new East-West challenge. Technical report, Oxford University Computing laboratory, Oxford,UK, 1994. URL: ftp://ftp.comlab.ox.ac.uk/pub/Packages/ILP/trains.tar.Z.

[352] F. Mizoguchi and H. Ohwada. Constraint-directed generalization for learning spatial relations. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[353] F. Mizoguchi and H. Ohwada. Constrained relative least general generalization for inducing constraint logic programs. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):335–368, 1995.

[354] F. Mizoguchi and H. Ohwada. An inductive logic programming approach to constraint acquisition for constraint-based problem solving. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 297–322. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[355] F. Mizoguchi, H. Ohwada, M. Daidoji, and S. Shirato. Learning rules that classify ocular fundus images for glaucoma diagnosis. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 191–204. Stockholm University, Royal Institute of Technology, 1996.

[356] D. Mladenić, I. Bratko, R.J. Paul, and M. Grobelnik. Using machine learning techniques to interprete results from discrete event simulation. In *Proceedings of the 15th International Conference on Information Technology Interfaces*, pages 401–406, 1993.

[357] D. Mladenić, I. Bratko, R.J. Paul, and M. Grobelnik. Using machine learning techniques to interpret results from discrete event simulation. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 399–402. Springer-Verlag, 1994.

[358] C.R. Mofizur and M. Numao. Logic program synthesis as a controlled search through appropriate hypothesis sub-space. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 373–386. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[359] C.R. Mofizur and M. Numao. Top-down induction of recursive programs from small number of sparse examples. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 161–180. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[360] C.R. Mofizur and M. Numao. Top-down induction of recursive programs from small number of sparse examples. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 236–253. IOS Press, 1996.

[361] R.J. Mooney. Inductive logic programming for natural language processing. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 205–224. Stockholm University, Royal Institute of Technology, 1996.

[362] R.J. Mooney and M.E. Califf. Induction of first–order decision lists: Results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research*, 3:1–24, 1995.

[363] R.J. Mooney and M.E. Califf. Induction of first-order decision lists: Results on learning the past tense of English verbs. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 145–146. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[364] R.J. Mooney and B. Richards. Automated debugging of logic programs via theory revision. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[365] R.J. Mooney and J. M. Zelle. Integrating ILP and EBL. *SIGART Bulletin*, 5(1):12–21, 1994.

[366] E. Morales. Learning chess patterns. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 291–307, 1991.

[367] E. Morales. Learning features by experimentation in chess. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 494–511. Springer-Verlag, 1991.

[368] E. Morales. *First-order induction of patterns in chess.* PhD thesis, Department of Computer Science, University of Strathclyde, 1992.

[369] E. Morales. Learning chess patterns. In S. Muggleton, editor, *Inductive Logic Programming*, pages 517–538. Academic Press, 1992.

[370] E. Morales. Learning patterns for playing KRK. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[371] E. Morales. Testing the applicability of ILP systems. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[372] K. Morik. Integrating manual and automatic knowledge aquisition - BLIP. In K.L. McGraw and C.R. Westphal, editors, *Readings in Knowledge Acquisition – Current Practices and Trends*, pages 213–232. Ellis Horwood, 1990.

[373] K. Morik. The art of ILP aplications. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 3–4. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[374] K. Morik and P. Brockhausen. A multistrategy approach to relational knowledge discovery in databases. In *Proceedings of the 3nd International Workshop on Multistrategy Learning*, pages 17–28. AAAI Press, 1996.

[375] K. Morik, S. Wrobel, J-U. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications.* Academic Press, 1993.

[376] I. Mozetič and N. Lavrač. Incremental learning from examples in a logic-based formalism. In P. Brazdil, editor, *Proceedings of the Workshop on Machine Learning, Meta-Reasoning and Logics*, pages 109–127, 1988.

[377] S. Muggleton. A strategy for constructing new predicates in first order logic. In D. Sleeman, editor, *Proceedings of the 3rd European Working Session on Learning*, pages 123–130. Pitman, 1988.

[378] S. Muggleton. Inductive logic programming. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 43–62. Ohmsma, Tokyo, Japan, 1990.

[379] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–317, 1991.

[380] S. Muggleton. Inverting the resolution principle. In J.E. Hayes-Michie and E. Tyugu, editors, *Machine Intelligence*, volume 12, pages 93–104. Oxford University Press, 1991.

[381] S. Muggleton, editor. *Proceedings of the 1st International Workshop on Inductive Logic Programming*. University of Porto, 1991.

[382] S. Muggleton. Inductive logic programming. In S. Muggleton, editor, *Inductive Logic Programming*, pages 3–28. Academic Press, 1992.

[383] S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.

[384] S. Muggleton. Inverting implication. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, pages 19–39, 1992.

[385] S. Muggleton. Inverting implication. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[386] S. Muggleton, editor. *Proceedings of the 2nd International Workshop on Inductive Logic Programming*. ICOT, Tokyo, 1992.

[387] S. Muggleton. A theoretical framework for predicate invention. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, page 18, 1992. Abstract only.

[388] S. Muggleton. Inductive logic programming: Derivations, successes and shortcomings. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 21–38. Springer-Verlag, 1993.

[389] S. Muggleton. Logic and learning: Turing's legacy. In S. Muggleton, D. Michie, and K. Furukawa, editors, *Machine Intelligence*, volume 13, pages 37–56. Oxford University Press, 1993.

[390] S. Muggleton. Optimal layered learning: A PAC approach to incremental sampling. In K. Jantke, S. Kobayashi, E. Tomita, and T. Yokomori, editors, *Proceedings of the 4th Conference on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence, pages 37–44. Springer-Verlag, 1993.

[391] S. Muggleton, editor. *Proceedings of the 3rd International Workshop on Inductive Logic Programming*. Jožef Stefan Institute, 1993.

[392] S. Muggleton. Bayesian inductive logic programming. In W.W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 371–379. Morgan Kaufmann, 1994.

[393] S. Muggleton. Inductive logic programming. *SIGART Bulletin*, 5(1):5–11, 1994.

[394] S. Muggleton. Predicate invention and utility. *Journal of Experimental and Theoretical Artificial Intelligence*, 6(1):121–130, 1994.

[395] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[396] S. Muggleton. Inverting entailment and Progol. In *Machine Intelligence*, volume 14, pages 133–188. Oxford University Press, 1995.

[397] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, page 29. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[398] S. Muggleton. Learning from positive data. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 225–244. Stockholm University, Royal Institute of Technology, 1996.

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

[400] S. Muggleton, M. Bain, J. Hayes-Michie, and D. Michie. An experimental comparison of human and machine learning formalisms. In *Proceedings of the 6th International Workshop on Machine Learning*, pages 113–118. Morgan Kaufmann, 1989.

[401] S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Workshop on Machine Learning*, pages 339–351. Morgan Kaufmann, 1988.

[402] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In S. Muggleton, editor, *Inductive Logic Programming*, pages 261–280. Academic Press, 1992.

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

[404] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan, 1990.

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

[406] S. Muggleton, R.D. King, and M.J.E. Sternberg. Protein secondary structure prediction using logic. *Protein Engineering*, 7:647–657, 1992.

[407] S. Muggleton, R.D. King, and M.J.E Sternberg. Protein secondary structure prediction using logic. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, pages 228–259, 1992.

[408] S. Muggleton and C.D. Page. A learnability model for universal representations. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 139–160. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[409] S. Muggleton and C.D. Page. Self-saturation of definite clauses. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 161–174. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[410] S. Muggleton, C.D. Page, and A. Srinivasan. An initial experiment into stereochemistry-based drug design using ILP. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 245–261. Stockholm University, Royal Institute of Technology, 1996.

[411] S. Muggleton and D. Page. Beyond first-order learning: Inductive learning with higher-order logic. Technical Report PRG-TR-13-94, Oxford University, Oxford, 1994.

[412] S. Muggleton, A. Srinivasan, and M. Bain. MDL codes for non-monotonic learning. Technical Report TIRM-91-049, Turing Institute, 1991.

[413] S. Muggleton, A. Srinivasan, and M. Bain. Compression, significance and accuracy. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 338–347. Morgan Kaufmann, 1992.

[414] M. Nagiya. An iterative and bottom-up procedure for proving-by-example. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 336–341. Springer-Verlag, 1993.

[415] C. Nédellec. How to specialize by theory refinement. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 474–478. John Wiley, 1992.

[416] C. Nédellec and C. Rouveirol. Hypothesis selection biases for incremental learning. In *Proceedings of the AAAI Spring Symposium Series workshop on Issues for Incremental Learning*, 1993.

[417] C. Nédellec and C. Rouveirol. Specification of the HAIKU system. Technical Report 928, Université Paris-Sud, 1994.

[418] C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, and B. Tausend. Declarative bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 82–103. IOS Press, 1996.

[419] F. Neri and A. Giordana. A distributed genetic algorithm for concept learning. In *Proceedings of the International Conference on Genetic Algorithms*, pages 436–443. Morgan Kaufmann, 1995.

[420] F. Neri and L. Saitta. A formal analysis of selection schemes. In *Proceedings of the International Conference on Genetic Algorithms*, pages 32–39. Morgan Kaufmann, 1995.

[421] F. Neri and L. Saitta. An analysis of the universal suffrage selection operator. *Evolutionary Computation Journal*, 4(1):89–109, 1996.

[422] T. Niblett. A study of generalisation in logic programs. In D. Sleeman, editor, *Proceedings of the 3rd European Working Session on Learning*, pages 131–138. Pitman, 1988.

[423] T. Niblett. A note on refinement operators. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 329–335. Springer-Verlag, 1993.

[424] S-H. Nienhuys-Cheng and R. de Wolf. The subsumption theorem in inductive logic programming: Facts and fallacies. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 147–160. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[425] S-H. Nienhuys-Cheng and R. de Wolf. A complete method for program specialization based on unfolding. In W. Wahlster, editor, *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 438–442. John Wiley, 1996.

[426] S-H. Nienhuys-Cheng and R. de Wolf. Least generalisations under implication. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 262–275. Stockholm University, Royal Institute of Technology, 1996.

[427] S-H. Nienhuys-Cheng and R. de Wolf. The subsumption theorem in inductive logic programming: Facts and fallacies. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 265–276. IOS Press, 1996.

[428] S-H. Nienhuys-Cheng and P.A. Flach. Consistent term mappings, term partitions and inverse resolution. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 361–374. Springer-Verlag, 1991.

[429] S-H. Nienhuys-Cheng and M. Polman. Sample PAC-learnability in model inference. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 217–230. Springer-Verlag, 1994.

[430] J. Paakki, T. Gyimóthy, and T. Horváth. Effective algorithmic debugging for inductive logic programming. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 175–194. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[431] C.D. Page and A.M. Frisch. Genaralizing sorted and constrained atoms. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 69–82, 1991.

[432] C.D. Page and A.M. Frisch. Learning constrained atoms. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 427–431. Morgan Kaufmann, 1991.

[433] C.D. Page and A.M. Frisch. Generalization and learnability: A study of constrained atoms. In S. Muggleton, editor, *Inductive Logic Programming*, pages 29–62. Academic Press, 1992.

[434] M.J. Pazzani, C.A. Brunk, and G. Silverstein. A knowledge-intensive approach to learning relational concepts. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 432–436. Morgan Kaufmann, 1991.

[435] M.J. Pazzani, C.A. Brunk, and G. Silverstein. An information-based approach to integrating empirical and explanation-based learning. In S. Muggleton, editor, *Inductive Logic Programming*, pages 373–394. Academic Press, 1992.

[436] M.J. Pazzani and D. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9(1):57–94, 1992.

[437] B. Pfahringer. Controlling constructive induction in CIPF: An MDL approach. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 242–256. Springer-Verlag, 1994.

[438] B. Pfahringer. A new MDL measure for robust rule induction. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 331–334. Springer-Verlag, 1995.

[439] G.D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

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

[441] G.D. Plotkin. A further note on inductive generalization. In *Machine Intelligence*, volume 6, pages 101–124. Edinburgh University Press, 1971.

[442] U. Pompe. SFOIL: Stochastic approach to inductive logic programming. In *Proceedings of the 2nd Slovenian Conference on Electrotechnical and Computer*, 1993.

[443] U. Pompe. Stochastic inductive logic programming. BSc Thesis, 1993. (In Slovenian).

[444] U. Pompe. Efficient proof encoding. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 276–291. Stockholm University, Royal Institute of Technology, 1996.

[445] U. Pompe and I. Kononenko. Linear space induction in first order logic with RELIEFF. In R. Kruse, R. Viertl, and G. Della Ricci, editors, *Mathematical and Statistical Methods in Artificial Intelligence, CISM Course and Lecture Notes 363*, pages 185–220. Springer-Verlag, 1995.

[446] U. Pompe and I. Kononenko. Naive Bayesian classifier within ILP-R. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 417–436. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[447] U. Pompe, I. Kononenko, and T. Makše. An application of ILP in a musical database: Learning to compose the two-voice counterpoint. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing*, pages 1–11, 1996.

[448] L. Popelinský. Object-oriented data modelling and rules: ILP meets databases. In *Proceedings of the ECML'95 Workshop Knowledge Level Modelling*, 1995.

[449] L. Popelinský, P. Flener, and O. Štěpánková. ILP and automatic programming: Towards three approaches. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 351–364. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[450] L. Popelinský and O. Štěpánková. WiM: A study on top-down ILP program. In *Proceedings of the AIT'95 Workshop*, 1995.

[451] J.R. Quinlan. Learning relations: Comparison of a symbolic and a connectionist approach. Technical report, Basser Department of Computer Science, University of Sydney, 1989.

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

[453] J.R. Quinlan. Determinate literals in inductive logic programming. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 442–446. Morgan Kaufmann, 1991.

[454] J.R. Quinlan. Determinate literals in inductive logic programming. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1991.

[455] J.R. Quinlan. Knowledge acquisition from structured data - Using determinate literals to assist search. *IEEE Expert*, 6(6):32–37, 1991.

[456] J.R. Quinlan. The minimum description lenght principle and categorical theories. In W.W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 233–241. Morgan Kaufmann, 1994.

[457] J.R. Quinlan and R.M. Cameron-Jones. FOIL: A midterm report. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 3–20. Springer-Verlag, 1993.

[458] J.R. Quinlan and R.M. Cameron-Jones. Introduction of logic programs: FOIL and related systems. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):287–312, 1995.

[459] B.L. Richards and R.J. Mooney. First-order theory revision. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 447–451. Morgan Kaufmann, 1991.

[460] B.L. Richards and R.J. Mooney. Refinement of first-order Horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.

[461] A. Rieger. Restructuring chain Datalog programs. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 292–311. Stockholm University, Royal Institute of Technology, 1996.

[462] C. Rouveirol. Saturation: Postponing choices when inverting resolution. In L.C. Aiello, editor, *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 557–562. Pitman, 1990.

[463] C. Rouveirol. Completeness for inductive procedures. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 452–456. Morgan Kaufmann, 1991.

[464] C. Rouveirol. ITOU: Induction of first order theories. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 127–158, 1991.

[465] C. Rouveirol. Semantic model for induction of first order theories. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1991.

[466] C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In S. Muggleton, editor, *Inductive Logic Programming*, pages 63–92. Academic Press, 1992.

[467] C. Rouveirol. Flattening and saturation: Two representation changes for generalization. *Machine Learning*, 14(2):219–232, 1994.

[468] C. Rouveirol, H. Adé, and L. De Raedt. Bottom-up generalization in ILP. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 59–70. Morgan Kaufmann, 1993.

[469] C. Rouveirol, H. Adé, and L. De Raedt. Bottom up generalization in ILP. Technical Report KUL-CW-171, Department of Computer Science, Katholieke Universiteit Leuven, 1993.

[470] C. Rouveirol and P. Albert. Knowledge level model of a configurable learning system. In *Proceedings of the 8th European Knowledge Acquisition Workshop*. Springer-Verlag, 1994.

[471] C. Rouveirol and L. De Raedt. The use of background knowledge for generalization in ILP. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[472] C. Rouveirol and J-F. Puget. A simple solution for inverting resolution. In K. Morik, editor, *Proceedings of the 4th European Working Session on Learning*, pages 201–210. Pitman, 1989.

[473] C. Rouveirol and J-F. Puget. Beyond inversion of resolution. In B. Porter and R.J. Mooney, editors, *Proceedings of the 7th International Conference on Machine Learning*, pages 122–131. Morgan Kaufmann, 1990.

[474] C. Rouveirol and M. Sebag. Constraint inductive logic programming. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 181–198. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[475] G. Sablon. *Iterative versionspaces with an application in inductive logic programming.* PhD thesis, Katholieke Universiteit Leuven, Belgium, 1995.

[476] G. Sablon, H. Adé, L. De Raedt, and M. Bruynooghe. Some thoughts on inverse resolution. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 83–91, 1991.

[477] G. Sablon, H. Adé, L. De Raedt, and M. Bruynooghe. Some thoughts on inverse resolution. In S. Muggleton, editor, *Inductive Logic Programming*, pages 409–422. Academic Press, 1992.

[478] G. Sablon and M. Bruynooghe. Using the event calculus to integrate planning and learning in an intelligent autonomous agent. In E. Plaza, editor, *Proceedings of the ECML-93 Workshop on Integrated Learning Architectures*, 1993.

[479] G. Sablon and M. Bruynooghe. Using the event calculus to integrate planning and learning in an intelligent autonomous agent. In C. Bäckström and E. Sandewall, editors, *Current Trends in AI Planning*, pages 254–265. IOS Press, 1994.

[480] G. Sablon, L. De Raedt, and M. Bruynooghe. Iterative versionspaces. *Artificial Intelligence*, 69, 1994.

[481] K. Sadohara and M. Haraguchi. Analogical logic program synthesis from examples. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 232–244. Springer-Verlag, 1995.

[482] L. Saitta. ILP: An alternative view. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 5–6. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[483] C. Sammut. Automatically constructing control systems by observing human behaviour. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[484] C. Sammut. The origins of inductive logic programming: A prehistoric tale. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 127–148. J. Stefan Institute, 1993.

[485] C. Sammut. Using background knowledge to build multistrategy learners. In *Proceedings of the 3nd International Workshop on Multistrategy Learning*, pages 11–16. AAAI Press, 1996.

[486] C. Sammut and R. Banerji. Learning concepts by asking questions. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume 2*, pages 167–191. Morgan Kaufmann, 1986.

[487] I. Savnik and P.A. Flach. Bottom-up induction of functional dependencies from relations. In G. Piatetsky-Shapiro, editor, *Proceedings of the AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 174–185, 1993.

[488] T. Scheffer, R. Herbrich, and F. Wysotzki. Efficient θ-subsumption based on graph algorithms. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 312–329. Stockholm University, Royal Institute of Technology, 1996.

[489] M. Sebag. A constraint-based induction algorithm in FOL. In W.W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 275–283. Morgan Kaufmann, 1994.

[490] M. Sebag. Using constraints to building version spaces. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 257–271. Springer-Verlag, 1994.

[491] M. Sebag and C. Rouveirol. Induction of maximally general clauses consistent with integrity constraints. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 195–216. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[492] M. Sebag and C. Rouveirol. Constraint inductive logic programming. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 277–294. IOS Press, 1996.

[493] M. Sebag and C. Rouveirol. Polynomial approximate learning in (constraint) logic programming. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 330–351. Stockholm University, Royal Institute of Technology, 1996.

[494] G. Semeraro, F. Esposito, and D. Malerba. Learning contextual rules for document understanding. In *Proceedings of the 10th IEEE Conference on Artificial Intelligence for Applications*, pages 108–115, 1994.

[495] E.Y. Shapiro. An algorithm that infers theories from facts. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 446–452. Morgan Kaufmann, 1981.

[496] E.Y. Shapiro. *Algorithmic Program Debugging*. The MIT Press, 1983.

[497] E.Y. Shapiro. Inductive inference of theories from facts. In J.L. Lassez and G.D. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 199–255. The MIT Press, 1991.

[498] G. Silverstein and M.J. Pazzani. Relational cliches: Constraining constructive induction during relational learning. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 203–207. Morgan Kaufmann, 1991.

[499] G. Silverstein and M.J. Pazzani. Learning relational cliches. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 71–82. Morgan Kaufmann, 1993.

[500] E. Siou. A bidirectional search for clauses. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 365–376. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[501] E. Sommer. Learning relations without closing the world. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 419–422. Springer-Verlag, 1994.

[502] E. Sommer. Rulebase stratifications: An approach to theory restructuring. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 377–390. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[503] E. Sommer. An approach to quantifying the quality of induced theories. In *Proceedings of the IJCAI Workshop on Machine Learning and Comprehensibility*. Morgan Kaufmann, 1995.

[504] E. Sommer. FENDER: An approach to theory restructuring. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 356–359. Springer-Verlag, 1995.

[505] E. Sommer. Theory restructuring: Coarse-grained integration of strategies for induction & maintanance of knowledge bases. In *Proceedings of the 3nd International Workshop on Multistrategy Learning*, pages 177–190. AAAI Press, 1996.

[506] A. Srinarasan, S. Muggleton, and M. Bain. Distinguishing exceptions from noise in non monotonic learning. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[507] A. Srinivasan and R.D. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 352–367. Stockholm University, Royal Institute of Technology, 1996.

[508] A. Srinivasan, S. Muggleton, and M. Bain. Distinguishing exceptions from noise in non-monotonic learning. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[509] A. Srinivasan, S. Muggleton, and M. Bain. The justification of logical theories. In S. Muggleton, D. Michie, and K.Furukawa, editors, *Machine Intelligence*, volume 13, pages 87–121. Oxford University Press, 1994.

[510] A. Srinivasan, S. Muggleton, and R.D. King. Comparing the use of background knowledge by inductive logic programming systems. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 199–230. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[511] A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[512] A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. The effect of background knowledge in inductive logic programming: A case study. Technical report, PRG-TR-9-95 Oxford University Computing Laboratory, 1995.

[513] A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. Theories for mutagenecity: A study of first-order and feature based induciton. Technical report, PRG-TR-8-95 Oxford University Computing Laboratory, 1995.

[514] A. Srinivasan, S. Muggleton, M.J.E. Sternberg, and R.D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1,2), 1996.

[515] I. Stahl. Predicate invention in ILP - An overview. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 313–322. Springer-Verlag, 1993.

[516] I. Stahl. On the utility of predicate invention in inductive logic programming. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 272–286. Springer-Verlag, 1994.

[517] I. Stahl. Properties of inductive logic programming in function-free Horn logic. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 423–426. Springer-Verlag, 1994.

[518] I. Stahl. The appropriateness of predicate invention as bias shift operation in ILP. *Machine Learning*, 20(1/2):95–118, 1995.

[519] I. Stahl. Compression measures in ILP. Technical report, Fakultät Informatik, Universität Stuttgart, 1995.

[520] I. Stahl. Compression measures in ILP. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 280–296. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[521] I. Stahl. The efficiency of bias shift operations in ILP. Technical report, Fakultät Informatik, Universität Stuttgart, 1995.

[522] I. Stahl. The efficiency of bias shift operations in ILP. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 231–246. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[523] I. Stahl. Compression measures in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 295–307. IOS Press, 1996.

[524] I. Stahl. Predicate invention in inductive logic programming. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 34–47. IOS Press, 1996.

[525] I. Stahl and B. Tausend. MILES – A modular inductive logic programming experimentation system. Technical report, Fakultät Informatik, Universität Stuttgart, 1994.

[526] I. Stahl, B. Tausend, and R. Wirth. General-to-specific learning of Horn clauses from positive examples. In *Proceedings of the CompEuro92*, 1992.

[527] I. Stahl, B. Tausend, and R. Wirth. Induction of disjunctive concepts via partitioning of the example set. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[528] I. Stahl, B. Tausend, and R. Wirth. Two methods for improving inductive logic programming systems. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 41–55. Springer-Verlag, 1993.

[529] I. Stahl and I. Weber. The arguments of newly invented predicates in ILP. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 233–246. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[530] M.J.E. Sternberg, J.D. Hurst, R.A. Lewis, R.D. King, A. Srinivasan, and S. Muggleton. Application of machine learning to protein structure prediction and drug design. In *Proceedings of the Colloquium on Molecular Bioinformatics*, 1994.

[531] B. Swennen. Learning by discovery in an adventure game. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1991.

[532] S. Tangkitvanich, M. Numao, and M. Shimura. Correcting multiple faults in the concept and subconcepts by learning and abduction. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

[533] S. Tankitvanitch and M. Shimura. Refining a relational theory with multiple faults in the concept and subconcepts. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 436–444. Morgan Kaufmann, 1992.

[534] B. Tausend. Lernen von Hornklauseln mit Programmierschemata. In K. Reiss, M. Reiss, and H. Spandl, editors, *Maschinelles Lernen - Modellierung von Lernen mit Maschinen*. Springer, 1992. (In German).

[535] B. Tausend. Using and adapting schemes for the induction of Horn clauses. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[536] B. Tausend. A unifying representation for language restrictions. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 205–220. J. Stefan Institute, 1993.

[537] B. Tausend. *Beschränkungen der Hypothesensprache und ihre Repräsentation in der Induktiven Logischen Programmierung*. PhD thesis, Fakultät Informatik, Universität Stuttgart, 1994. (In German).

[538] B. Tausend. Biases and their effects in inductive logic programming. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 431–434. Springer-Verlag, 1994.

[539] B. Tausend. Representing biases for inductive logic programming. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 427–430. Springer-Verlag, 1994.

[540] B. Tausend. A guided tour through hypothesis spaces in ILP. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 245–259. Springer-Verlag, 1995.

[541] B. Tausend and S. Bell. Analogical reasoning for logic programming. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 159–166, 1991.

[542] B. Tausend and S. Bell. Analogical reasoning for logic programming. In S. Muggleton, editor, *Inductive Logic Programming*, pages 397–408. Academic Press, 1992.

[543] K. Taylor. Inverse resolution of normal clauses. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 165–178. J. Stefan Institute, 1993.

[544] G. Tecuci. Building knowledge bases through multistrategy learning and knowledge acquisition. In Y. Kodratoff and R.S. Michalski, editors, *Machine Learning and Knowledge Acquisition: Integrated Approaches*, pages 13–50. Academic Press, 1995.

[545] G. Tecuci, S. Calinoiu, and D. Marcu. Using multistrategy learning as a framework for building knowledge-based systems. In C. Unger and I.A. Letia, editors, *Proceedings of the International Conference on Intelligent Communication ICCU95*. Technical University of Cluj-Napoca, 1995.

[546] G. Tecuci and D. Duff. A framework for knowledge base refinement through multistrategy learning and knowledge acquisition. *Knowledge Acquisition*, 6(2):137–161, 1994.

[547] L. Torgo. Controlled redundancy in incremental rule learning. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 185–195. Springer-Verlag, 1993.

[548] G. Turán. Lower bounds for PAC-learning with queries. In *Proceedings of the 6th ACM Workshop on Computational Learning Theory*, 1993.

[549] P. Turney. Low size-complexity inductive logic programming: The East-West challenge considered as a problem in cost-sensitive classification. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 247–263. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[550] P. Turney. Low size-complexity inductive logic programming: The East-West challenge considered as a problem in cost-sensitive classification. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 308–321. IOS Press, 1996.

[551] E. Van Baelen and L. De Raedt. Analysis and prediction of piano performances using inductive logic programming. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 368–378. Stockholm University, Royal Institute of Technology, 1996.

[552] P.R.J. van der Laag. *An analysis of refinement operators in inductive logic programming.* PhD thesis, Erasmus Universiteit, Rotterdam, the Netherlands, 1995.

[553] P.R.J. van der Laag and S-H. Nienhuys-Cheng. Subsumption and refinement in model inference. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 95–114. Springer-Verlag, 1993.

[554] P.R.J. van der Laag and S-H. Nienhuys-Cheng. Existence and nonexistence of complete refinement operators. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 307–322. Springer-Verlag, 1994.

[555] P.R.J. van der Laag and S-H. Nienhuys-Cheng. A note on ideal refinement operators in ILP. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 247–262. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[556] W. Van Holder, L. De Raedt, H. Adé, G. Sablon, M. Bruynooghe, and Y.D. Willems. MacCLINT: An interactive machine learning system. In *Proceedings of the European Apple University Consortium Conference*, 1992.

[557] W. Van Laer and L. De Raedt. Discovering quantitative laws in inductive logic programming. In *Proceedings of the Familiarization Workshop of the ESPRIT Network of Excellence on Machine Learning*, pages 8–11, 1993. (Extended abstract).

[558] W. Van Laer, L. Dehaspe, and L. De Raedt. Applications of a logical discovery engine. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pages 263–274, 1994.

[559] W. Van Laer, S. Džeroski, and L. De Raedt. Multi-class problems and discretization in ICL. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing*, pages 53–60, 1996.

[560] A. Varšek. *Inductive Logic Programming with Genetic Algorithms.* PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia, 1993. (In Slovenian).

[561] A. Varšek. Genetic inductive logic programming. Technical report, ESPRIT III project 6020 deliverable D.LAI.3.6., 1995.

[562] P. Vitanyi. Inductive reasoning. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 7–10. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[563] C. Vrain and L. Martin. Inductive learning of normal clauses. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 435–438. Springer-Verlag, 1994.

[564] L. Watanabe and L. Rendell. Feature construction in structural decision trees. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 218–222. Morgan Kaufmann, 1991.

[565] L. Watanabe and L. Rendell. Learning structural decision trees from examples. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1991.

[566] I. Weber and B. Tausend. A three-tiered confidence model for revising logical theories. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 391–402. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[567] I. Weber, B. Tausend, and I. Stahl. Language series revisited: The complexity of hypothesis spaces in ILP. In N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 360–363. Springer-Verlag, 1995.

[568] M. Wiese. A bi-directional ILP algorithm. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing*, pages 61–72, 1996.

[569] R. Wirth. Learning by failure to prove. In D. Sleeman, editor, *Proceedings of the 3rd European Working Session on Learning*, pages 239–250. Pitman, 1988.

[570] R. Wirth. Completing logic programs by inverse resolution. In K. Morik, editor, *Proceedings of the 4th European Working Session on Learning*. Pitman, 1989.

[571] R. Wirth. *Lernverfahren zur Vervollstaendigung von Hornklauselmengen durch inverse Resolution*. PhD thesis, University of Stuttgart, 1989. (In German).

[572] R. Wirth and P. O'Rorke. Constraints on predicate invention. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 457–461. Morgan Kaufmann, 1991.

[573] R. Wirth and P. O'Rorke. Inductive completion of SLD proofs. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 167–176, 1991.

[574] R. Wirth and P. O'Rorke. Constraints for predicate invention. In S. Muggleton, editor, *Inductive Logic Programming*, pages 299–318. Academic Press, 1992.

[575] J. Wogulis. Revising relational domain theories. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 462–466. Morgan Kaufmann, 1991.

[576] J. Wogulis. Handling negation in first-order theory revision. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 36–46. Morgan Kaufmann, 1993.

[577] J. Wolff. On the integration of learning, logical deduction and probabilistic inference. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 177–192, 1991.

[578] S. Wrobel. Automatic representation adjustment in an observational discovery system. In D. Sleeman, editor, *Proceedings of the 3rd European Working Session on Learning*, pages 253–262. Pitman, 1988.

[579] S. Wrobel. Demand driven concept-formation. In K. Morik, editor, *Knowledge Representation and Organization in Machine Learning*, volume 347 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1989.

[580] S. Wrobel. On the proper definition of minimality in specialization and theory revision. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 65–82. Springer-Verlag, 1993.

[581] S. Wrobel. *Concept Formation and Knowledge Revision.* Kluwer Academic Publishers, 1994.

[582] S. Wrobel. Concept formation during interactive theory revision. *Machine Learning*, 14(2):169–192, 1994.

[583] S. Wrobel. From balanced cooperative modeling to embedded adaptivity – Using inductive logic programming techniques for knowledge acquisition. In Y. Kodratoff and G. Tecuci, editors, *Machine Learning and Knowledge Acquisition: Integrated Approaches*, pages 115–143. Academic Press, 1995.

[584] S. Wrobel. First order theory refinement. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 14–33. IOS Press, 1996.

[585] S. Wrobel. Inductive logic programming. In G. Brewka, editor, *Advances in Knowledge Representation and Reasoning.* CSLI Publishers, 1996. (To appear).

[586] J.M. Zelle and R.J. Mooney. Combining FOIL and EBG to speed-up logic programming. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1106–1113. Morgan Kaufmann, 1993.

[587] J.M. Zelle and R.J. Mooney. ILP techniques for learning semantic grammars. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 83–92. Morgan Kaufmann, 1993.

[588] J.M. Zelle and R.J. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 817–822. AAAI Press/MIT Press, 1993.

[589] J.M. Zelle and R.J. Mooney. Inducing deterministic Prolog parsers from treebanks: A machine learning approach. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 748–753. AAAI Press/MIT Press, 1994.

[590] J.M. Zelle and R.J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 1050–1055. AAAI Press/MIT Press, 1996.

[591] J.M. Zelle, R.J. Mooney, and J.B. Konvisser. Combining top-down and bottom-up techniques in inductive logic programming. In W.W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 343–351. Morgan Kaufmann, 1994.

[592] J.M. Zelle, C.A. Thompson, M.E. Califf, and R.J. Mooney. Inducing logic programs without explicit negative examples. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 403–416. Department of Computer Science, Katholieke Universiteit Leuven, 1995.