An Environment for Exploring and Validating Declarative Knowledge

Technical Memo

TM-95-03

Andreas Abecker, Harold Boley, Knut Hinkelmann, Holger Wache, and Franz Schmalhofer

**Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH**

**Technical
Memo**

TM-95-03

An Environment for Exploring and Validating Declarative Knowledge

**Andreas Abecker, Harold Boley, Knut Hinkelmann,
Holger Wache, and Franz Schmalhofer**

September 1995

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry of Education, Science, Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

An Environment for Exploring and Validating Declarative Knowledge

Andreas Abecker, Harold Boley, Knut Hinkelmann, Holger Wache, and Franz Schmalhofer

DFKI-TM-95-03

This work has been supported by a grant from The Federal Ministry of Education, Science, Research and Technology (FKZ ITWM-9405).

ISSN 0946-0071

An Environment for Exploring and Validating Declarative Knowledge

Andreas Abecker, Harold Boley*, Knut Hinkelmann
Holger Wache, and Franz Schmalhofer
German Research Center for Artificial Intelligence (DFKI) GmbH
P.O. Box 2080, D-67608 Kaiserslautern

September 22, 1995

Abstract

We discuss the extended LP environment of the project VEGA (knowledge validation and exploration by global analysis), a toolbox that supports the development and maintenance of declarative knowledge bases. The knowledge is represented in a declarative language that merges Horn logic with finite domains, sort hierarchies, functions, and integrity constraints. These KBs may contain problem-solving knowledge, ontologies, and cases. The VEGA environment includes tools that take into account these different kinds of knowledge to ensure the integrity of the knowledge base during its entire life-time. Besides such knowledge-*validation* components, VEGA provides for the interactive *exploration* of knowledge by inductive components, e.g. a generator of rules from facts. VEGA also allows to link foreign tools, and is thus an open architecture. The tools can be synergetically configured by visual programming. Knowledge validation can for example be performed on the output of knowledge exploration.

1 Introduction

In the past few years there have been a lot of extensions of logic programming aiming to make it more suitable for knowledge representation in artificial intelligence, e.g. abduction, induction, non-monontonic reasoning, and meta-reasoning [Kowalski, 1991]. These extensions concern both augmentations of the representation formalism and the development of new inference procedures.

Experiences with knowledge-based systems have shown that the maintenance of knowledge bases is of utmost importance. Since logic programs can be regarded as declarative knowledge bases and since the knowledge base is the core of a knowledge-based system, it is not sufficient that the programming (or knowledge-engineering) environment supports the development of a knowledge base. In addition it is necessary that future environments support the user to ensure the adequateness and integrity of a knowledge base during its entire life time.

Maintenance of software systems and knowledge-based systems consists of the removal of deficiencies, adaptation to changes in the application context and to requirements of users

*Please contact this author for further information: e-mail address: boley@dfki.uni-kl.de

concerning functionality and performance [Lehner, 1994]. For these we can distinguish two principal tasks that should be supported.

Validation: The objective of knowledge-base validation is to ensure that the knowledge base remains consistent after every modification. Since it can be assumed that the knowledge base is consistent prior to an update, validation has to be initiated only after a modification.

Exploration: By analyzing data and previous cases, commonalities and regularities may be detected that may lead to adaptations of the knowledge base. This new knowledge can be used to revise the knowledge base, which in turn may initiate the validation task.

In the VEGA project we have developed an engineering environment for developing and maintaining logic programs. This so-called *knowledge evolution system* (KES) is organized as an open toolbox including various algorithms for each of the maintenance tasks. We permit to apply each of the algorithms individually but also support the configuration of complex maintenance operations by a graphical interface.

The knowledge evolution system operates on knowledge bases written in a *declarative representation language* (DRL) that augments Horn logic by finite domains, sorts, functions, and integrity constraints.

A very important feature of knowledge-base maintenance is the organization of the knowledge base. Since knowledge-base maintenance is important in particular for large systems, we support the modularization of the knowledge base, where named modules can be configured into contexts. In our system we also distinguish three kinds of knowledge:

Ontologies: Sort hierarchies and integrity constraints representing domain or background knowledge.

Problem-solving knowledge: Horn clauses describing a particular application.

Cases: Ground facts and relational data from various applications of the system.

The explicit distinction of these knowledge types can be taken into account for the evolution of the knowledge base. For instance, new rules for adapting the problem-solving knowledge can be induced from previous cases. Also, when there are various possibilities for revising a knowledge base it can be assumed that the ontologies are more stable and thus revisions of the problem-solving knowledge should be preferred.

In the main part of this paper we concentrate on the validation and exploration tasks. After proposing a methodology for knowledge-base maintenance in the following section, we give an overview of some specific features of our engineering environment which will be exemplified in Section 4.

2 Towards a Methodology for Knowledge Evolution

Recently, the idea is spreading that KB validation and exploration (i.e. machine learning) techniques should synergetically cooperate [Borrajo and de Antonio, 1993; Hoppe, 1991]. Most publications in this area describe a monolithic software system that implicitly encompasses such a cooperation in a "hard-wired" fashion in order to solve one specific KB

maintenance task (see e.g. [Craw and Sleeman, 1993; Lounis, 1993]). Our project aims at contributing to a KB evolution methodology where knowledge evolution means the integrated use of validation and exploration techniques for quality improvement during the entire life-cycle of a declarative KB.

Let us first consider two examples of such a validate-explore cooperation.

When incrementally building up, debugging and maintaining, or updating and augmenting a declarative KB, one always has to pass through similar sequences of activities. Principally, we can distinguish the case that a user interactively edits the KB and wants to insert some new knowledge or deletes some old and the case that a machine learning algorithm is employed in order to generate knowledge from cases. This distinction is reflected by the two example functionalities `update` and `explore` below.

(1.) The `update` functionality

`update`

1. add/delete some piece of knowledge (e.g. a fact or a rule)
2. analyze whether this leads to a conflict with existing knowledge
3. search candidate revision points for solving such conflicts
4. if candidate revision points were found, then choose a (e.g. minimal) revision and carry it out; else exit with `failure`

Here, we have both validation (item 2) and exploration (item 3) tasks, typically performed by deductive (2) in combination with abductive and inductive (3) inferences. A typical incarnation of this abstract schema is *theory revision* as described in [Adé *et al.*, 1993; Richards and Mooney, 1995].

(2.) The `explore` functionality

This functionality can also be found in *explorative data analysis* or *data mining*:

`explore`

1. analyze the case base and generate hypotheses about possible relationships
2. discard obviously inconsistent and redundant hypotheses
3. choose an interesting hypothesis to add to the problem-solving knowledge and call `update`; if `update` fails, then choose another hypothesis
4. goto `explore`

This *explore-validate cooperation* completely encompasses our first example. Furthermore, there are some basic functionalities useful for both tasks: e.g. the same validation algorithm can both find conflicts between new items and existing knowledge and detect hypotheses inconsistent w.r.t. the KB and the integrity constraints (ICs).

This leads to the idea of a knowledge-maintenance environment with a library of basic functionalities and the facility for building complex processes out of simpler ones. Besides technical aspects, such a toolbox project has a strong methodological component:

find the minimal commonly reusable pieces of functionality, describe the space of possible interactions/cooperations, and analyze the process of modeling complex tasks.

In the next sections, we sketch the basic elements of such a knowledge-evolution architecture and illustrate their use by a simple example.

3 Elements of a Knowledge-Evolution System

In the following we give an overview of principal functional units necessary for a knowledge evolution environment.

Functionality palette: a library of basic or composite functionalities. These functionalities can be arranged in a mixed decomposition-/specialization-hierarchy describing the sub-functionalities of complex processes and the alternative instantiations of functionality classes by available tools. Each functionality representation also contains information about input and output parameters and pre-/postconditions of the functionality.

Functionality configurator: allows the interactive specification of a sequence of maintenance activities by instantiating and linking together elements of the functionality palette and parts of the KB. This is not only possible in a top-down refinement manner of preproduced patterns as in Section 4 but also bottom-up for constructing new complex functionalities out of simpler ones. It must provide a scripting language (together with a set of corresponding graphical user-interface actions) for complex functionalities allowing e.g. loops and conditional expressions.

Ideally, in the functionality palette there are a lot of specific tools. So the user has the problem to choose and link together appropriate tools in the right way. To overcome this problem we suggest a **configuration assistant** which supports the user. This assistant is able to

- check consistency of user-specified functionality configurations
- provide the user with suggestions for configuration decisions
- instantiate non-executable configurations automatically[1]

These above elements give the slots of a methodological framework that have to be filled by research on different levels (cf. Section 5). But first we illustrate the intended use of our knowledge-evolution system.

4 A Sample Functionality Configuration

We sketch the modeling of a simple knowledge evolution functionality. Although simplified in some technical details, the example gives an impression of how to use our knowledge-evolution system.

[1]Note that our term of functionality *configuration* comes from a strong analogy to the structure-oriented configuration of technical products, where we have also decomposition and specialization decisions, parametrizing and optimality criteria. Automatic functionality configuration is a rather similar task.

Suppose, the user wants to configure an instance of the explore-validate-activity. He selects the appropriate item in an interactive graphical user interface and asks for the definition of this activity. The system presents a graphical decomposition into sub-activities as shown in Fig. 1.

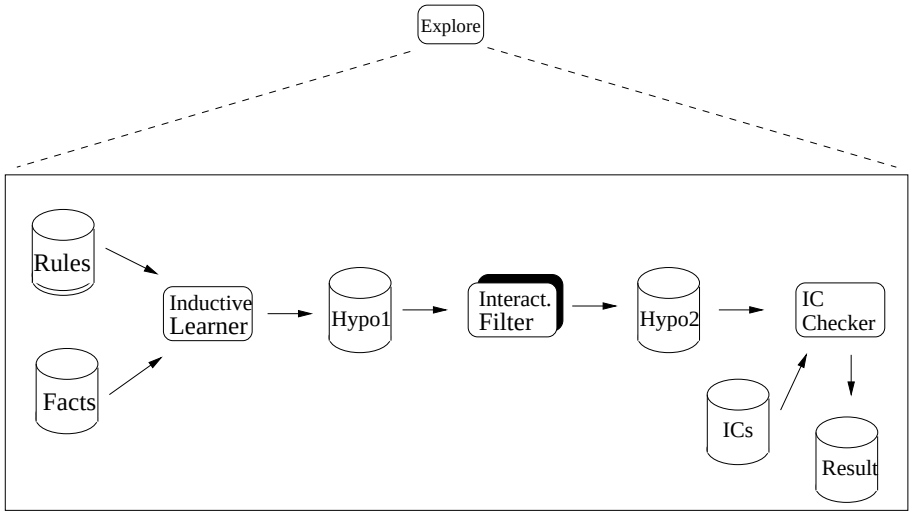

Figure 1: Decomposition of the explore-validate functionality

The task decomposition shows the functionality to be composed out of three functional units: (i) an inductive learning system, (ii) an interactive hypothesis filter, and (iii) an integrity constraint (IC) checker.

The inductive learning system takes some learning input (cases) and a background theory (ontology and problem-solving knowledge) and produces a set of hypotheses that should comprise the input facts. Typically, there are a lot of unnecessary hypotheses, such that there is a need for a filtering process. The filtered hypotheses have to be checked against the integrity constraints: because state-of-the-art inductive learning systems usually do not take ICs into account, it is possible that the inductive component produces some output incompatible with the IC theory, e.g. due to over-generalization.

Except the interactive filter, which is already an executable program (as indicated by the shadowed icon), the two other functional units have to be further specified, or parametrized.

In order to do this, the user expands the inductive-learning activity. In this case, the system does not decompose the activity but presents instances of this functionality class that are provided by the toolbox (see Fig. 2).

Here is a brief description of the two algorithms currently available in the KES toolbox:

RuleGen+ is an extension of the state-of-the-art inductive-logic-programming system CLAUDIEN [Raedt and Bruynooghe, 1993]. The system takes as input a Horn clause theory and a set of ground facts as examples. The system follows the *weak setting of ILP* in that it induces only Horn clauses that are logically entailed by the input theory and the example set. A possible application for this kind of learning can be the compression of many examples to a small set of rules. RuleGen+ extends CLAUDIEN among other things

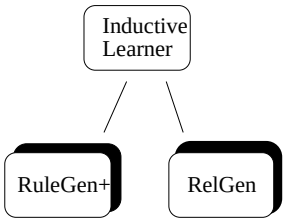

Figure 2: Alternative specializations of the inductive-learner class

by employing a sort hierarchy to improve efficiency.

RelGen is a prototypical bottom-up induction system over database relations. It discovers join-like relationships between tables that are deterministically or probabilistically entailed by the KB. These relationships can be translated into suggestions for program Horn clauses and ICs, which is a novel feature compared to the standard ILP approach.

Suppose the user specifies RuleGen+ as inductive component and asks the system for a completion of the functionality configuration. Now the system selects one of the available IC checkers:

SLIC-Resolution extends SLDNF-Resolution for checking integrity constraints. The inference process is similar to the methods proposed by [Decker and Celma, 1994; Sadri and Kowalski, 1988]. Its efficiency is improved by a combined bottom-up/top-down algorithm considering only facts and rules that can contribute to an inconsistency.

The consequence-finding transformation as described in [Hinkelmann, 1994] is a rewriting approach for integrity checking, which restricts the derivations of a logic program to exactly those facts that depend on an explicitly given update information. The consequence-finding transformation extends Generalized Magic Sets rewriting by an *up* propagation in addition to the usual *down* propagation. The approach has been applied for integrity checking of abductively inferred hypotheses [Hinkelmann, 1995].

Suppose the system chooses *SLIC-Resolution* as the default IC checker and employs its knowledge about I/O characteristics of the combined algorithms in order to test the compatibility of the configuration. After linking the input and output parameters with appropriate modules in the knowledge-base management part of the system (see below), we have an executable configuration: Fig. 3.

5 Research Issues Emerging From the Framework

The presentation of our environment already suggests the dimensions of the research questions we have to face before being able to put the system into routine action for KB maintenance. For some of these questions there exist first results, other ones are under work. We discuss two levels of research:

Evolution-tool level

The main topic in developing tools for knowledge evolution is lifting them from relatively simple representation mechanisms to more complex ones. Generally, this is still an open

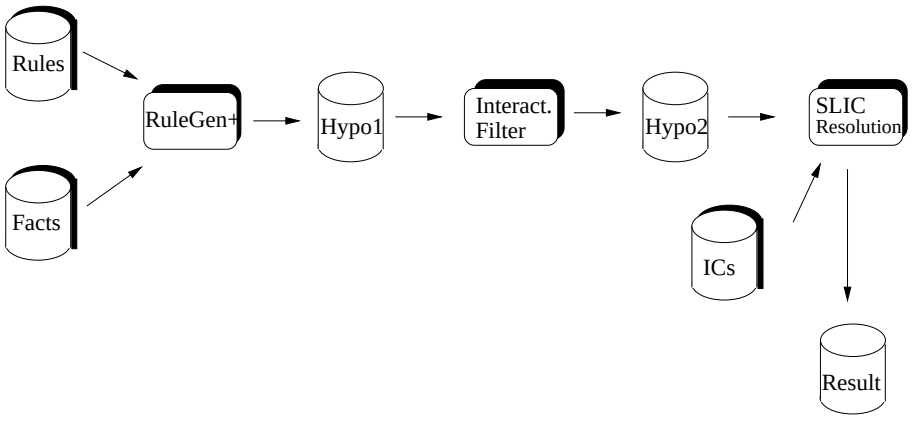

Figure 3: An executable functionality configuration

research question also tackled in the communities of ILP and theory revision.

Before the above-mentioned implementations (RuleGen+, RelGen, SLIC-Resolution), we had developed a partial anti-unification algorithm [Fischer, 1994] that can handle finite domains and sorts.

An andvantage of our flexible system architecture is that we can circumvent some of the problems arising when introducing complex representation formalisms. For example, the functionality configuration in Section 4 shows how a sequence of functionalities reduces the problem that state-of-the-art ILP tools are not able to handle ICs as background knowledge.

Integration level

This topic provides us with many interesting problems. Most of them have not yet been tackled because the implementation of the toolbox is the first step, which can be followed by gaining experience with evolution activities.

One question is how to acquire the applicability and optimality knowledge necessary for decision support and automatic functionality configuration. Even for the narrower field of pure machine learning there have been only few approaches to gather such knowledge (e.g. within the MLT machine learning toolbox project [Sleeman, 1994]).

Another issue is the cooperation of several kinds of knowledge (considering both syntactic and semantic structure). The most simple (and in most environments, the only) example is the use of background knowledge and learning input for inductive learners (cf. e.g. [Pazzani and Kibler, 1992]). Another idea is that problems during the analysis of one kind of knowledge trigger the restructuring of another kind of knowledge. To our knowledge, the only systems that allow such a mechanism are MOBAL [Wrobel, 1994] and the system described by Lounis in [Lounis, 1993], where the introduction of new sorts allows to improve the rule representation. Some simple application-driven forms of cooperation are described in [Hinkelmann and Kühn, 1995]: e.g. abstracting problem-solving rules from case descriptions or checking cases vs. rules.

The last methodological point (and the most typical for a toolbox project) is the task of finding minimal reusable pieces of functionality. A very useful concession to real-world

conditions is that our environment allows to incorporate available tools of arbitrary granularity. Furthermore, the open architecture allows to easily apply specialized tools for specific representation formalisms without the need of transforming them into our representation languages.

6 Summary and Related Work

We extended standard LP environment concepts to the maintenance of logic programs or knowledge bases during their entire life cycle. The two principal cooperating maintenance tasks, validation and exploration, are identified. Because there exists a large variety of applicable techniques, a toolbox of functionalities for validation and exploration was developed. Complex functionalities can be combined from other, simpler ones. An open problem is to identify a minimal set of basic functionalities from which complex ones can be built. Another interesting point is to support (or guide) a user in choosing and linking together the functionalities available in the toolbox.

We have designed and partially implemented an integration of ideas coming from three different areas of research:

1. A general framework for the cooperation of exploration and validation in KB maintenance has been proposed, whereas most existing approaches sketch specific examples of possible explore-validate cooperations (cf. [Craw and Sleeman, 1993; Lounis, 1993]).

2. The methodological aspect of toolbox projects (like MLT in machine learning) was generalized to validation and exploration activities.

3. The visual programming approach of some data mining tools (like Clementine [Khabaza and Shearer, 1995] or CAKETool [Steuernagel, 1995]) was employed to provide a user-friendly ergonomic experimentation environment for knowledge evolution.

State of the implementation:

- We have implemented a knowledge-base management system for our augmented Horn logic. In order to abstract from the syntax of knowledge items, this *knowledge-base server* (KBServ) provides an ask/tell interface between the knowledge base and the knowledge-evolution system. KBServ supports different kinds of knowledge and a flexible KB modularization facility [Herfert, 1995] holding a forest of named modules interactively configurable into knowledge contexts.

- Where possible, KBServ offers standard LP environment components extended for the augmented Horn logic as well as specific maintenance tools for new representation facilities. For example, the sort hierarchy is supported by a browser, checkers for, e.g., acyclicity, and an individuals-to-sort abstractor.

- Several tools (Partial anti-unification, RuleGen+, RelGen, SLIC-Resolution, CF Transformation) have been implemented and cooperate for simple KB maintenance tasks. Extensions and new cooperation types are under development. Technically spoken, a main goal is to complement deductive queries by abductive ones and by inductive algorithms.

- The KES toolbox with a small functionality palette and a simple sequential scripting language (including a graphical user interface for interactive functionality configuration) has been implemented. The open system architecture and - at the implementation level - an especially designed UNIX process-communication support permit the combination of VEGA's tools with foreign ones independent from their implementation language (TCL/TK, C, LISP, PROLOG, ...). The configuration assistant is not yet implemented.

- Although the VEGA environment is not yet fully implemented, several prototypical engineering applications have already been built employing ideas of the knowledge evolution scenario, such as for materials selection [Boley *et al.*, 1994], for product/production planning [Hinkelmann *et al.*, 1994], and for conserving crankshaft design rules [Kühn *et al.*, 1994].

References

[Adé *et al.*, 1993] H. Adé, L. de Raedt, and M. Bruynooghe. Theory revision. In Stephen Muggleton, editor, *ILP '93, Proc. of the 3rd Int. Workshop on Inductive Logic Programming, Bled, Slovenia*, Stefan Institute Technical Report, Ljubljana, Slovenia, pages 179–182, 1993.

[Boley *et al.*, 1994] H. Boley, U. Buhrmann, and Ch. Kremer. Towards a sharable knowledge base on recyclable plastics. In J.K. McDowell and K.J. Meltsner, editors, *Knowledge-Based Applications in Materials Science and Engineering*, pages 29–42. TMS, 1994.

[Borrajo and de Antonio, 1993] D. Borrajo and A. de Antonio. Cooperation of machine learning and validation. In J. Cardeñosa and P. Meseguer, editors, *Proc. EUROVAV'93*, pages 277–292, 1993.

[Craw and Sleeman, 1993] S. Craw and D. Sleeman. Refinement in response to validation. In J. Cardeñosa and P. Meseguer, editors, *Proc. EUROVAV'93*, pages 85–99, 1993.

[Decker and Celma, 1994] H. Decker and M. Celma. A slick procedure for integrity checking in deductive databases. In Pascal Van Hentenryck, editor, *Proc. of the Eleventh Int. Conf. on Logic Programming*, pages 456–469. MIT Press, 1994.

[Fischer, 1994] Cornelia Fischer. PAntUDE - an anti-unification algorithm for expressing refined generalizations. Technical Memo TM-94-04, DFKI Kaiserslautern, 1994.

[Herfert, 1995] Michael Herfert. *Deklarative statische und dynamische Softwaremodule.* Diploma thesis, Universität Kaiserslautern, February 1995.

[Hinkelmann and Kühn, 1995] K. Hinkelmann and O. Kühn. Revising and updating a corporate memory. In *Proc. EUROVAV'95*, 1995.

[Hinkelmann *et al.*, 1994] K. Hinkelmann, M. Meyer, and F. Schmalhofer. Knowledge-Base Evolution for Product and Production Planning. *AI Communications*, 7(2):98–113, June 1994.

[Hinkelmann, 1994] Knut Hinkelmann. A consequence-finding approach for feature recognition in CAPP. In *Seventh International Conference on Industrial & Engineering Applications of Artifical Intelligence & Expert Systems (IEA/AIE'94), Austin, Texas*, pages 2–11. Gordon and Breach Science Publishers, 1994.

[Hinkelmann, 1995] Knut Hinkelmann. Knowledge-base rewriting for bottom-up abduction and integrity checking. In H. Decker, U. Geske, T. Kakas, C. Sakama, D. Seipel, and T. Urpi, editors, *Deductive Databases and Logic Programming - Abduction in Deductive Databases and Knowledge-based Systems. Proceedings of the ICLP'95 Joint Workshop.* GMD-Studien Nr. 266, Gesellschaft für Mathematik und Datenverarbeitung mbH, 1995.

[Hoppe, 1991] Thomas Hoppe. Maschinelles Lernen und Wissensvalidierung. *KI-Künstliche Intelligenz*, 5(1), 1991.

[Khabaza and Shearer, 1995] T. Khabaza and C. Shearer. Data mining with Clementine. In *Colloquium on Knowledge Discovery in Databases.* The Institution of Electrical Engineers, February 1995.

[Kowalski, 1991] Robert A. Kowalski. Logic Programming in Artificial Intelligence. In *IJCAI-91*, pages 596–603, 1991.

[Kühn *et al.*, 1994] O. Kühn, V. Becker, G. Lohse, and Ph. Neumann. Integrated Knowledge Utilization and Evolution for the Conservation of Corporate Know-How. In *IS-MICK'94: Int. Symposium on the Management of Industrial and Corporate Knowledge*, October 1994.

[Lehner, 1994] Franz Lehner. Ergebnisse einer Untersuchung zur Wartung von wissensbasierten Systemen. *Information Management*, 1994.

[Lounis, 1993] Hakim Lounis. Knowledge-based systems verification: a machine-learning-based approach. In J. Cardeñosa and P. Meseguer, editors, *Proc. EUROVAV'93*, pages 265–276, 1993.

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

[Raedt and Bruynooghe, 1993] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence.* Morgan Kaufmann, 1993.

[Richards and Mooney, 1995] B.L. Richards and R.J. Mooney. Automated refinement of first-order horn-clause domain theories. *Machine Learning*, 19(2), April 1995.

[Sadri and Kowalski, 1988] F. Sadri and R. Kowalski. A theorem-proving approach to database integrity. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 313–362. Morgan Kaufmann, Los Altos, CA, 1988.

[Sleeman, 1994] D. Sleeman. Towards a technology and a science of machine learning. *AI Communications*, 7(1), March 1994.

[Steuernagel, 1995] Ralf Steuernagel. *Offenes adaptives Engineering-Werkzeug zur automatischen Erstellung von entscheidungsunterstützenden Informationssystemen.* Forschungsberichte aus dem Institut für Werkzeugmaschinen und Betriebstechnik der Universität Karlsruhe (wbk), 1995.

[Wrobel, 1994] Stefan Wrobel. Concept formation during interactive theory revision. *Machine Learning*, 14:169–191, 1994.