Knowledge-Base Evolution for Product and Production Planning*

Knut Hinkelmann, Manfred Meyer and Franz Schmalhofer
DFKI (German Research Center for Artificial Intelligence), Postfach 2080, 67608 Kaiserslautern, Germany
Email: {hinkelma,meyer,schmalho}@dfki.uni-kl.de

Knowledge-base evolution techniques are shown to be of critical importance for the successful application of knowledge-based systems in complex domains. By conceptualizing knowledge-base evolution as theory revision, we can take advantage of the basic findings from different research communities. Results from Inductive Logic Programming (ILP) and Explanation-Based Learning (EBL) provide a set of techniques that can be used as a foundation for obtaining new knowledge (*knowledge-base exploration*). Techniques from deductive database research might be used for testing the correctness of a knowledge base (knowledge base verification). By an interactive application of these exploration and verification techniques, domain experts and other users may similary improve the effectiveness of the knowledge base (knowledge validation). The application of such selected techniques is then discussed with respect to the specific problem of improving production parameters.

1. Introduction

It is a long held belief, that micro-worlds, such as the blocks world, sorting tasks or chess end games are the drosophila of Artificial Intelligence and Machine Learning research, where the fundamental successes are to be achieved and demonstrated. A quote by Amarel [1, p.258] highlights this view. 'These toy problems provide an excellent paradigmatic task environment in which essential aspects of the representation problem can be studied ... They are serving as drosophila of research in the general area of problem representations, and in the study of acquisition of problem solving skills'.

Although there cannot be any doubt that many successes of Machine Learning have been achieved in these micro-worlds, the utilization of these achievements in complex real world domains (e.g., the industrial applications of Machine Learning) is much more difficult than had been originally anticipated. Buchanan [10, p.5] for example, reports that except for simple classification systems, knowledge-based systems do not yet employ a learning component to construct parts of the knowledge bases from libraries of previously solved cases.

It has been pointed out only recently, that real world domains have quite different characteristics than the micro-words where new machine learning techniques are routinely demonstrated. Complexity, continuous innovations and documentation as well as incomplete and conflicting knowledge are the most eminent characteristics [37]. Because of the dynamic character of real world domains, the application of knowledge-based systems requires that the changes in the field can at least be traced (preferably predicted and discovered) by appropriately selected machine learning techniques. Such updating and revision processes are termed *knowledge base evolution*. Comparable to the human genome project which also requires additional resources, above and beyond the discovery of the genetic mechanisms with the drosophila, the ILP community must therefore also pay more attention to applications in complex real world domains.

In order to develop knowledge-base evolution techniques with respect to complex real world domains, we first analyzed the requirements of product and production planning with new materials by using the specific example of the manufacturing of bucket seats in the car industry. The results are summarized in Section 2 of this paper. Section 3 then describes a respective knowledge-base that is currently being developed by an iterative application of the CLASSIC methodology to knowledge engineering [8]. Section 4 will then show how the knowledge evolution can be understood as theory revision [33], where the knowledge-base evolution system and the user cooperate in a way, similar to an apprenticeship learning system [40].

Theory Revision has recently been proposed as a

*This research was supported by grant 413-5839-ITW9304/3 from the BMFT.

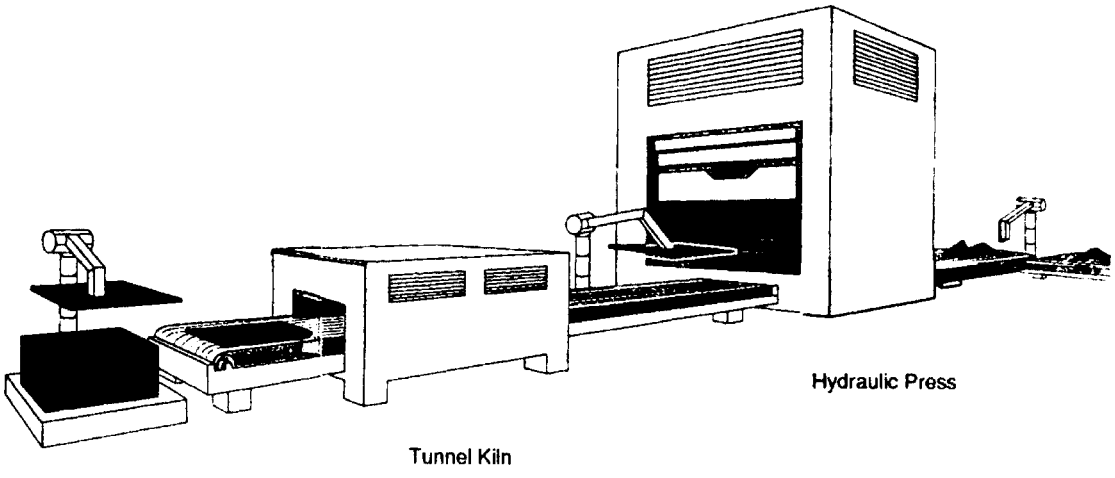

Fig. 1. The manufacturing of a bucket seat with a GMT (reprinted by permission from the Elastogran GmbH).

general framework, where Explanation-Based Learning (EBL) and Inductive Logic Programming (ILP) can be integrated [27]. For mastering the knowledge evolution requirements of the specific application, we can thus draw upon the basic research results from both EBL as well as ILP. Furthermore, exploration and verification processes will be distinguished. A continuous (interactive) improvement of a knowledge base during its entire life-time starting with the first formalizations (knowledge base seed) and still continuing along its practical use can thus be achieved [26].

Expert knowledge from the application domain is used for constraining the exploration processes, so that an efficient implementation can be obtained. Expert knowledge will be employed to determine the representation bias (also known as 'restricted hypothesis space bias') and search bias (also known as 'preference bias') of induction [32]. More specifically, domain knowledge is used to specify the representational bias and metaknowledge to determine the search bias. The paper will be concluded with a general discussion of the role of knowledge-base evolution for the quality of practical knowledge bases.

2. Product and Production Planning

In the car industry, like in other modern industries, the innovation cycles have become increasingly shorter. Driven by the objectives of environmental protection laws, hazardous manufacturing materials must be replaced by more adequate new materials. Equally important is the reduction of cost while the highest possible quality standard is being maintained. In many branches, new materials such as glass mat reinforced thermoplastics (GMT) are currently introduced and increasingly more used for manufacturing products, and thereby replacing steel and metal constructions. A GMT is a composite consisting of two components, namely a thermoplastic rein-forced by glass fiber. An example is the manu-facturing of car seats. The high security standards and other requirements (e.g., concerning wear and tear) can now be satisfied by using GMTs. For example, the rear part of a bucket seat for a car can now be manufactured with GMT, instead of more costly metal constructions.

Figure 1 shows the production process with GMTs. It consists of a preparation phase, a pressing phase and a finishing phase. In the preparation phase the raw material is put on a conveyer belt that moves it through the tunnel kiln, where it is heated. In order to avoid an undesired cooling, the material is then immediately put into the hydraulic press, where the geometry of the car seat is pressed before it is cooled off so that its form is maintained. During the finishing phase unwanted bumps must be removed.

The pressing of the material depends on a number of parameters with complex interrelationships. The temperature of the material influences the volume per unit time which is responsible that

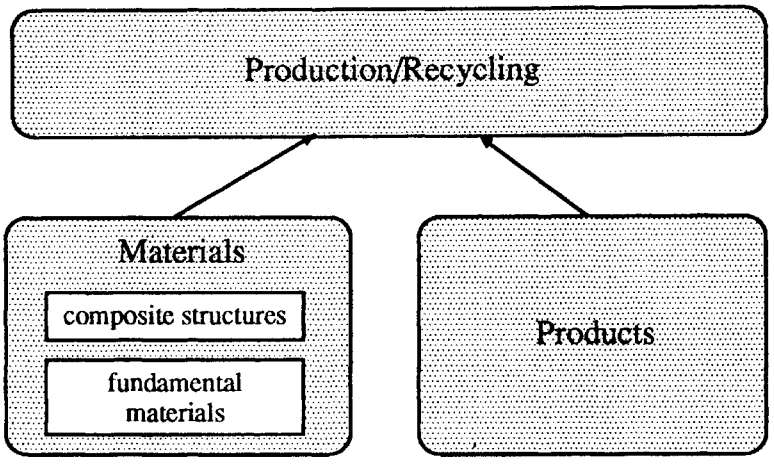

Fig. 2. Overall structure of the RPPP knowledge base.

the material reaches every part of the pressing form. As soon as the material is put into the press, the press is closed with a speed of about 800 mm/s. As soon as the press reaches the material the speed is reduced to a value between 5 and 15 mm/s. After the press is closed a constant pressing force is exerted on the material for some duration. After that, the material is left in the press for some time to cool off. The duration of cooling depends on the temperature of the material and the tool, the tool geometry, the topology of the cooling capillaries of the tool, etc.

In product and production planning, 'system development' and 'parameter optimization' are distinguished as two separate phases, which can also be called primary and secondary engineering [23]. In the primary phase, a prototype of the product and the corresponding manufacturing process is developed. In some previous research it was already shown how machine learning techniques can be applied for supporting the primary engineering phase [31]. More specifically, it was shown how an explanation based abstraction method [36] can be used for abstracting planning schemata from success cases of the real world [37]. In the secondary phase, appropiate parameters must be found for the respective primary design. In this paper, we are solely concerned with this secondary design phase. In particular, we propose a knowledge base and knowledge evolution techniques for documenting and maintaining all available information and knowledge. This knowledge concerns the various parameters and how they determine the desired characteristics of the product.

3. A Recycling-Oriented Product and Production Planning Knowledge Base

In some previous work, the selection of recyclable materials in product design and the process planning for manufacturing and recycling such products were identified as a promising application domain for knowledge base evolution. In [3] a materials knowledge base is discussed as an integral part of a declarative knowledge base for *r*ecycling-oriented *p*roduct and *p*roduction *p*lanning (RPPP). The overall structure of this knowledge base consists of a module representing the materials, a second one representing production and recycling knowledge and a third module containing products that have been manufactured from these materials (see Fig. 2).

3.1. The Materials Knowledge Base

Materials constitute the substance of production and recycling. Materials can bedivided into fundamental and composite materials. The main problem when building a knowledge base is 'finding the right way to break the domain into objects and their relationships'. One solution approach is given by the 'Knowledge Engineering Methodology for CLASSIC' [8]. This methodology suggests to formalize the domain

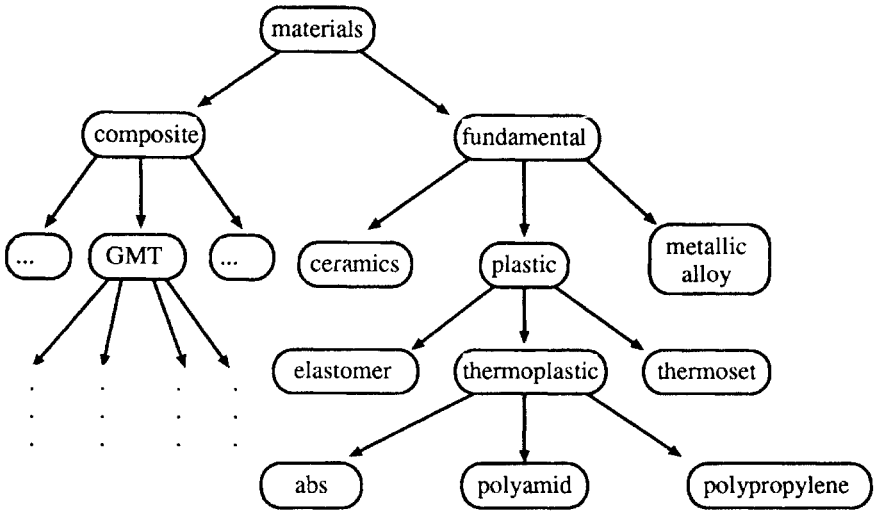

Fig. 3. A taxonomy of materials.

knowledge using some kind of terminological knowledge represen-tation in the spirit of KL-ONE [7] or a frame-like, object-centered knowledge represen-tation system using an inheritance hierarchy. The methodology consists of a sequence of design steps. We are using an iterative application of this methodology by allowing multiple iterations of two or more of the following consecutive steps:

1. Relevant object types are enumerated. As a result the relevant objects are determined to be particular plastics and composite materials, classes of such materials, qualitative and quantitative properties of the materials, numbers etc.
2. The obtained descriptions are divided into objects and properties, which are later mapped to concepts and roles. In our case, classes of materials are concepts, whereas most of the properties correspond to roles.
3. Concepts are organized into a taxonomy. This step yielded the hierarchy of the fundamental and composite materials. Part of this hierarchy is presented in Fig. 3.
4. Then, the key individuals are isolated and associated to the concepts they belong to.
5. In order to obtain the internal structure of the concepts, a list of relevant properties must be determined for each concept. These properties include intrinsic and extrinsic properties and part-of relations. In this step, the properties of the plastics have been adopted from the existing CAMPUS database [9], which contains all the plastics produced by 22 European chemical industries. An important property for GMT is the *modulus of elasticity* (*e-modulus*).
   The part-of relation is the main relation for distinguishing composite materials. A GMT consists of a thermoplastic which is reinforced with glass fibers to enhance its e-modulus. There are two types of glass fibers in the form of *papers* or *mats* and two types of thermoplastics – *polypropylene* and *polyamid*. Thus we get four types of GMTs. The e-modulus increases as the percentage of glass fibers increases.
6. In the remaining steps of the CLASSIC methodology, the restrictions of the properties for each concept are acquired in detail. As a result of this step, the particular types of possible values and the cardinality of values have been determined.

For the representation of the materials knowledge base we propose a respective hierarchical representation in a terminological representation language.

3.2. The Product Knowledge Base as Case Base

The Product Knowledge Base is a Case Base. It contains the actual parameters of the success cases of manufacturing car seats with different materials. It also represents cases, where certain quality requirements have not been satisfied by the product of the industrial manufacturing process. These

success and failure cases are denoted by e^+_{ik} and e^-_{ik}, where k is an index for referencing the specific case and i identifies that the case resulted from indus-trial experience.

In addition to these industrial cases, the results from systematic experimentation, that is performed in material sciences research institutes, should also be stored in the Product Knowledge Base. In order to determine the thermodynamic behavior of GMTs during the pressing process, researchers may for instance perform experiments, where several different parameters are systematically manipulated to determine their influence upon some criterion variable. Such scientific research may determine, 'which influences different production parameters have on the work done on the material and what kind of flow characteristics different GMT-materials show' [22]. Such experiments may investigate how the closing speed of the tools, the press force and the specific material determine the size of the pressed material. The experimental results can provide very useful information for the product engineer, who is interested in manufacturing some specific car seat. The actual data from such experiments should therefore also be stored in the case base. We denote such cases from scientific experimentation by e^+_{sk}, where the index s indicates that this result was achieved by science research and k is an index that denotes the specific experiment.

3.3. The Production Knowledge Base

The pressing of materials depends on a large number of parameters. There are complex relationships among these parameters, as well as between these parameters and the material and the quality requirements. As already mentioned in Section 2, there is relatively little knowledge available about which parameter values achieve the desired result. Even for an expert it is nearly impossible to find exact adjustments at once. To find the dependencies between various parameters, the product engineer usually tries several possibilities. The results of these trials are represented in the Product Knowledge Base. In the Production Knowledge Base, we will thus represent the regularities which are (supposedly) valid for the production process, in general. More specifically, we are concerned with the different parameter values for manufacturing GMT products with a hydraulic press (see Fig. 1).

The results of such scientific experiments are most often summarized by a linear equation, that is obtained by a regression analysis or by an Analysis of Variance [23]. Such an equation may for instance take the form:

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_n X_n$$

Although such numeric equations are quite useful and have a broad field of application in research and industrial practice, there are also a few disadvantages, which can be compensated by a more abstract and qualitative description. One problem lies in the fact, the each experiment yields a new equation and it may be quite difficult for any practitioner (and even researcher) to derive a set of general regularities from the various equations. Secondly, these equations hold only within certain limits.This is, however, not directly represented by the equation. For instance, increasing the pressing force beyond certain limits will not increase the surface area in the way that is predicted by the linear equation, but may instead damage the press. In other words, there is an upper and lower bound on the parameters as well as on the values of the criterion variable (e.g., the surface area).

In addition to such numerical representations, we therefore propose a more abstract and qualitative description for representing the general knowledge from the various cases. Unlike the numerical equation, we assume upper and lower bounds for the criterion variable, whose values are denoted qualitatively, like for instance by 'large', 'medium' or 'small'. In other words, there is for instance no value that is smaller than 'very small' and no value that is larger than 'very large'. As a consequence of these bounds, the *qualitative addition* operation, which we denote by $\oplus$, can no longer be a closed operation. In order to embody these limitations, we define the *qualitative addition* operation in the following way. Let A denote a set of qualitative descriptors, like a_1, a_2, a_3 ... an, which we could for instance also call a_1=very small, a_2=small, a_3= medium, ... a_n=very large. We postulate that the set A is weakly ordered. Since the cartesian product $A \times A$ contains all logically possible *qualitative additions* of the form $a \oplus b$, where a and b are in A, those that can actually be formed must constitute a

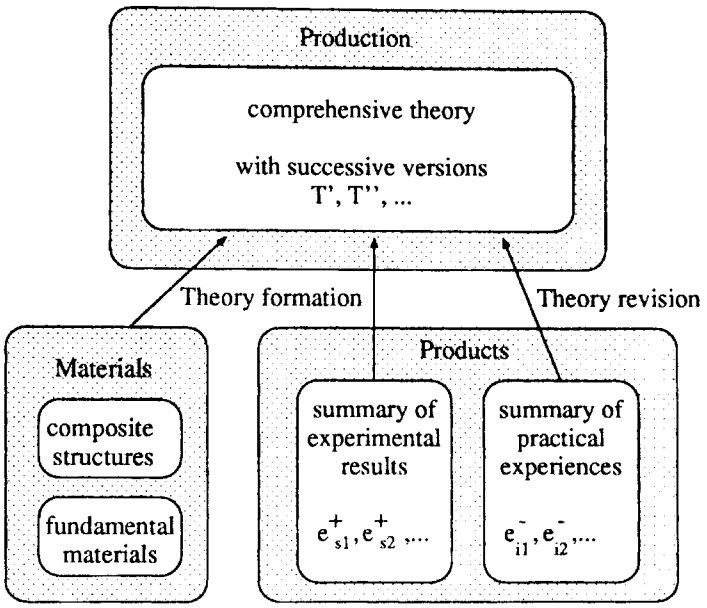

Fig. 4. Formation and revision of the production knowledge base.

subset B of $A \times A$. Thus, if (a,b) is in B, then a and b can be *qualitatively added* and so $a \oplus b$ is in A. This means that the operation $\oplus$ is a function from B into A. In order to account for the fact, that not all *qualitative additions* are possible, we define a qualitative structure $<A, \succ, B, \oplus>$, where associativity and monotonicity are somewhat modified. In order to accomplish this, we impose the following limitations on A and B: If $a \succ b$, we assert the existence of a c in A such that (c,b) is in B and $a \succ c \oplus b$. The requirements on the proposed qualitative structure, which are summarized in the following definition, provide important integrity constraints for the production knowledge base (Fig. 4).

Integrity constraints for qualitative structures. Let A be a nonempty set of qualitative descriptors (such as 'small', 'medium', 'large') or $a_1, a_2, \ldots, a_n$, $\succeq$ a binary relation on A, B a nonempty subset of $A \times A$ and $\oplus$ a binary function from B into A. The quadruple $<A, \succeq, B, \oplus>$ is a qualitative structure if the following six conditions are satisfied for all a, b, $c \in A$:

1. $<A, \succeq>$ is a weak order.
2. If $(a,b) \in B$ and $(a \oplus b, c) \in B$, then $(b,c) \in B$, $(a, b \oplus c) \in B$, and $(a \oplus b) \oplus c \succeq a \oplus (b \oplus c)$.
3. If $(a,c) \in B$ and $a \succeq b$, then $(c,b) \in B$ and $a \oplus c \succeq c \oplus b$.
4. If $a \succ b$, then there exists $d \in A$ such that $(b,d) \in B$ and $a \succeq b \oplus d$.
5. If $(a,b) \in B$, then $a \oplus b \succ a$.
6. $a_1, \ldots, a_n, \ldots \in A$ is a strictly bounded and finite standard sequence if for $n=2, \ldots$, $a_n = a_{n-1} \oplus a_1$, and it is only strictly bounded if for some $b \in A$ and for all a_n in the sequence, $b \succ a_n$.

4. Knowledge Base Evolution as Theory Revision

4.1. The Knowledge Base Evolution Scenario

Knowledge base evolution covers not only the maintenance of an existing KB [13], but also the continous improvement of the KB, its structure and content. Knowledge-base evolution operates on the KB of a knowledge-based system. Thus, for an overall description of knowledge base evolution in the RPPP context we distinguish two main units (Fig. 5): the knowledgebase itself (RPPP) and the knowledge-evolution system (KES).

The KES operates as a meta-level system on the object level KB. Reasoning in the knowledge evolution system is performed by the exploration and verification components.

- Similar to discovery systems the *knowledge explorer* scans the KB in search for interesting

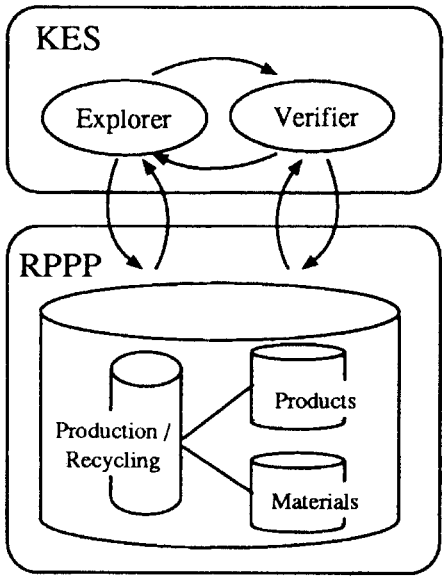

Fig. 5. The RPPP knowledge base evolution architecture.

patterns. Exploration can be seen as an iterative process starting with the generation of a pattern hypothesis, proceeding with a search for the pattern in the KB, and resulting in a possible interactive assimilation of the discovered pattern into the KB. Thus, inductive techniques play a major role for knowledge exploration.

- The *knowledge verifier* can perform verification and with appropriate user assistance also validation. It examines the KB to detect structural or functional defects. Validation and verification can also be seen as an iterative process starting with the generation of a defect suspicion, proceeding with a check for a defect w.r.t. the suspicion in the KB, and resulting in a possible defect description or repair suggestion. Here, techniques for checking integrity constraints become most relevant.

The iteration cycles can be arbitrarily interleaved, permitting evolution to consist of dual verification and exploration processes. Together they form a heuristic, approximative process that alternates focusing and processing phases and improves the KB any time a sufficient amount of knowledge for an update (i.e., assimilation or repair) is accumulated within the KES or provided by the user. For example, assume that the verifier has identified a rule whose premises cannot be satisfied in a given KB. The explorer could then try to generalize that particular rule or to complete the missing knowledge reachable from its premises. Conversely, after the explorer has discovered a pattern (e.g., a new or generalized rule) the verifier may be asked to verify the KB, focused on the assimilated pattern.

4.2. Theory Revision

The problem of building up a knowledge base (knowledge acquisition) can be seen as a two-phase process [16]: In the first phase the knowledge engineer builds an initial model (i.e., the seeding of the knowledge base). In the second phase this initial knowledge base is refined or revised into a high performance knowledge base. During the further practical use of the knowledge-base, the dynamically changing world may cause the knowledge base to become invalid in one of the following senses:

- New developments may cause new problem cases not being covered by the knowledge base. This results in the KBS not being able to solve these problems. For example, neglecting the effects of changing parameter values determined by recent experiments would leave the RPPP system incapable to find the best production process.
- Some knowledge stored in the knowledge base may become out of date and should no longer be used as it would lead to solutions that for some reasons are no longer valid in the current application environment. For example, a fluent additive that has become known to be noxious should no longer be used or be used only in closed-circle production and recycling processes.

In the first situation we have a new application case (i.e., a positive example) that is not yet derivable from the knowledge base. In the second situation, we can derive a specific solution from the knowledge base which is no longer admissible (e.g., because of new environmental protection laws). This is consequently called a negative example.

From a more formal point of view, this means that a given knowledge base *KB* has to be revised using positive examples E^+ (positive experiments to be included) and/or negative examples E^- (failing experiments to be excluded), such that all the

positive examples but none of the negative examples are covered by the resulting knowledge base KB'.

Taking the knowledge base as a Horn theory $T = F \cup R$ consisting of facts F and rules R and satisfying a set of integrity constraints IC, the *exploration task* of theory revision is to change T into T' such that $T' \vdash e\ \forall e \in E^+$ and $T' \nvdash e\ \forall e \in E^-$. The resulting theory T' must, of course, still satisfy the given integrity constraints, i.e., $IC \cup T'$ must be consistent. This integrity checking represents the *verification task* of theory revision and thus again demonstrates the *interleaved exploration* and *verification* principle.

The main task, however, remains how to obtain the revised theory T'. In principle, there are two possibilities:

- First, we can modify the rules R, for instance by using generalization or specialization techniques,
- or we can extend the set of facts F, where the additional facts can be found by abduction.

In the following section we will discuss some selected techniques from the fields of inductive logic programming and deductive databases, which could be applied within the proposed theory revision framework.

5. Selected Methods for Knowledge Base Evolution

Generalization techniques are the basic techniques of Inductive Logic Programming and also Theory Revision. Generalization operators perform two basic syntactic operations on a clause:

- apply an inverse substitution to a clause;
- remove a literal from the body of a clause.

In this section we will first review the least general generalization and generalized subsumption frameworks defined by Plotkin and Buntine, respectively, before we will then extend these techniques for the needs of theory revision in practical applications like the evolution of the RPPP knowledge bases.

5.1. Least General Generalization

Least general generalization was originally introduced by Plotkin [29]. It is the opposite of most general unification [34]; therefore it is also called *anti-unification*. Given two atomic formulas $p(f(a),x)$ and $p(f(y),b)$, unification computes their most general specialization $p(f(a),b)$ while anti-unification computes their most special generalization $p(f(y),x)$.

In addition to the generalization of literals, Plotkin also describes an algorithm for the least generalization of clauses. A clause C_1 generalizes a clause C_2 (denoted by $C_1 \leq C_2$), if C_1 subsumes C_2, i.e., there exists a substitution θ such that $C_1\theta \subseteq C_2$. This is also called θ–subsumption [11]. A generalization C_1 of a clause C_2 can thus be ob-tained by applying a θ–subsumption-based generalization operator ρ that maps a clause C_2 to a set of clauses $\rho(C_2)$ which are generalizations of C_2. Informally speaking, if clause C θ–subsumes clause D, then D can be converted to C by (1) dropping premises and (2) turning constants to variables. A clause C is a least generalization of a set of clauses S, if

1. C generalizes each clause in S: $\forall E \in S : C \leq E$
2. C is the smallest clause satisfying condition 1: $(\exists D \forall E \in S, D \leq E) \Rightarrow D \leq C$

5.2. Generalized Subsumption

The definition of generality presented so far is local to the set S of clauses. Referring to implication instead of the weaker subsumption relationship would also consider generalization w.r.t. current knowledge. In [31] a generalization relative to a set of clauses P is defined as follows: A clause C generalizes a clause D relative to a set of clauses P if there exists a substitution θ such that $P \models \forall (C\theta \rightarrow D)$. Buntine defines *generalized subsumption* of definite Horn clauses as an extension of θ–subsumption with the restriction that the corresponding clause heads must be about the same concept [11]. Informally speaking, if a clause C generally subsumes clause D, then C can be converted to D by (1) turning variables to constants or other terms, (2) adding atoms to the body, and (3) partially evaluating the body by resolving some clause in P with an atom in the body. The third

conversion process is additional to the conversion for θ–subsumption.

5.3. Generalization for Knowledge Base Evolution

The condition of *cover*ing in the definition of generalized subsumption has the effect, that generalization depends on the actual representation of the clauses. Defining generalization in terms of implication (see Plotkin's definition [30]) instead of subset-relation would be more suitable. This would lead to a combination of techniques from inductive logic programming (ILP) and explanation-based learning (EBL) [27] by using deduction when deciding the generalization of clauses.

Unfortunately, doing so, the test for generalization becomes undecidable. On the other hand, Buntine states that generalized subsumption is semidecidable, although it is guaranteed to terminate if P contains no recursion. Generalized subsumption w.r.t. a DATALOG program, however, is decidable.

Also, for practical applications, least general generalization as defined by Plotkin [29] can still be too general. Consider the least generalization of the two literals t_2 = **additive(ppn_1060, fluent)** and t_3 = **additive(r_5320, flame-retardent)** for which we get the very general term **additive(X,Y)** loosing nearly all information from and connection to the original terms that have been generalized.

Thus, in order to overcome the problems raised by theory revision with background knowledge, namely its undecidability and its results being too general, we study two approaches in the following:

- First, we investigate how to incorporate more knowledge within the generalization process, i.e., how to *control* generalization. This will enable us to specify *when* and *where* to generalize.
- Second, we discuss how to extend the language itself by introducing new representational features for expressing generalization results. This will enable us to specify *how* to generalize and to represent the generalized term.

Finally, we will present an alternative to θ–subsumption based on terminological reasoning which preserves decidability by restricting deduction to the terminological calculus.

5.3.1. Partial Least General Generalization

The first extension is *partial least general generalization* (plgg) and allows us to only partially generalize two literals: we can say that we want to generalize two literals or terms, but can require some arguments to be fixed, i.e., that the two literals must have unifying values at that specific argument position. Thus, partial least general generalization is a combination of unification and anti-unification.

Consider again the following literals

t_1 = **additive(ppn_1060, flame-retardent)**
t_2 = **additive(ppn_1060, fluent)**
t_3 = **additive(r_5320, flame-retardent)**

and the following application:

plgg(t_1,t_2,**additive($,-)**) ⇒
additive(ppn_1060, X)

Here we only want to generalize over the fluent additives of identical materials: the generalization pattern **additive($,-)** restricts generalization to the second argument position, while the first argument, marked '**$**', cannot be generalized but has to be unifiable. Consequently, plgg can be regarded as a combination of anti-unification (for those argument positions marked '-') and unification (for the remaining argument positions). As unification can fail, plgg may fail too, but, of course, in the non-generalized argument positions only. Thus, the above generalization of t_1 and t_2 succeeds, but the generalization of t_2 and t_3 using the same generalized pattern fails.

We can also restrict generalization by requiring some arguments to be of a particular type. In this case we would use a type or sort identifier at the position of the meta-symbol **$** in the previous example. Thus, exact match as done for **$** is now replaced by sorted unification for the non-generalized argument positions.

Consider the taxonomy shown in Fig. 3. Trying to generalize t_2 and t_3 requiring the first argument of the resulting literal to be of type **novodur**[1] would fail since the least general generalization of **ppn_1060** and **r_5320** is the type **thermoplastic**, which is too general, i.e., not within the required type **novodur**:

[1]Novodur is a registered trademark of the Bayer AG.

$plgg(t_2,t_3,$**additive(novodur,_)**$) \Rightarrow$ *fail*

However, if we only require the generalized material to be of type **plastic**, the generalization of t_2 and t_3 would succeed and result in the least generalized material type **thermoplastic**:

$plgg(t_2,t_3,$**additive(plastic,_)**$) \Rightarrow$
additive(thermoplastic, X)

5.3.2. Finite Domain Generalizations

A second extension of generalization is by enumerating the occurring values in a finite domain term instead of replacing them by a variable. A logic programming extension with finite domain terms is presented in [5]. In this case anti-unification of t_1 and t_2 results in the literal

additive(ppn_1060,dom[flame-retardent,fluent])

where the second term is the finite domain containing both original constants **flame-retardent** and **fluent**. This does not induce new knowledge but only compresses the information of two literals into one. However, if we anti-unify all three terms into one we do obtain an inductive generalization. The resulting fact

additive(dom[ppn_1060,r_5320],dom[flame-retardent,fluent])

with two finite domains really represents four facts, which we get by combining each value of the first with each of the second domain. In addition to the two original clauses we get that novodur **r_5320** has the fluent additive **fluent**. In order to decide whether this hypothesis is actually true we again require validation.

5.3.3. An Alternative to θ-Subsumption Based on Terminological Reasoning

As has been shown by Plotkin, the general subsumption problem for Horn clauses is undecidable. Essentially, this negative result is due to the fact that subsumption can be reduced to logical implication. *θ*–subsumption is one approximation of the 'logical' subsumption that is based on instantiations of Herbrand terms and set inclusion. In principle, there are two ways how to get a decidable rule ordering:

- One can restrict the expressiveness of the underlying knowledge representation language such that logical implication becomes decidable, e.g., Buntine's *generalized subsumption* with restriction to DATALOG.
- Alternatively, the rule ordering can be defined using only a weak approximation of logical implication. For example, the subset test used for *θ–subsumption* is such a sound but incomplete operationalization of logical implication.

In the former case, as a side effect, the class of knowledge that can be learned will be very restricted, too. In the latter approach, the learning algorithms cannot be optimal, since they are always based on a suboptimal rule ordering. However, the class of knowledge that can be learned remains unconstrained in that case.

θ–subsumption relies on the instantiation ordering of Herbrand terms which implies additional deficits: There are 'too many' terms that are incomparable w.r.t. the instantiation ordering of Herbrand terms (e.g., $f(a,b), f(a,a), f(b,b), f(b,a)$ are all incomparable). The weakness is also indicated by the fact that there are *linear* decision procedures for the instantiation problem of Herbrand terms.

These deficits are somehow inherent to the underlying Horn logic. As an alternative, Hanschke and Meyer [17] propose a rule-formalism based on terminological logics (TL). This enables us to define a rule ordering much like *θ*–subsumption, but which is based on terminological inferences instead of instantiating Herbrand terms. As terminological reasoning formalisms are tuned to be similarly expressive while remaining tractable or at least decidable, we gain a more fine-grained rule-ordering. In particular, more rules will become comparable. Moreover, we obtain a more intuitive knowledge representation.

We will first briefly introduce the *assertional* formalism (A-box) and the *terminological* formalism (T-box) of the concept language ***ALCF*** as a prototypical representative for the family of terminological formalisms [6] that originated with KL-ONE [7]. A *terminology* of the T-box consists of a set of *concept definitions* $C = t$ where C is the newly introduced concept name and t is a concept term constructed from concept names, roles, and

attributes using the following concept forming operators: *conjunction*, *disjunction*, *negation*, *value-restriction*, and *exists-in restriction*. In an *A-box* (assertional box) concepts, roles, and attributes can be instantiated by individuals. Formally, an A-box is a *finite* set of role assertions $((i,j) : r)$, membership assertions $(i:t)$, and equalities $(i=j)$, where i and j are individual names, r is a role or attribute name, and t is a concept term. The subsumption problem for A-boxes of ***ALCF*** can be effectively decided [17].

Two individuals i and j are *directly linked* in an A-box iff the A-box contains a role assertion of the form $(i,j):R$ or $(j,i):R$. *Linked* is the transitive reflexive closure of directly linked. An A-box is called *rooted* by (individuals) $x_1,...,x_n$, $n > 0$, iff every individual in the A-box is linked to at least one of the x_i and all x_i occur in the A-box.

The rule language is now based on the terminological formalism. Its operational semantics can be based on a CLP scheme [21,38]. A *rule* takes the form

$$p_0(\underline{x}^{(0)}) \leftarrow p_1(\underline{x}^{(1)}),...,p_n(\underline{x}^{(n)}),\ A(\underline{x}^{(0)},...,\underline{x}^{(n)}).$$

where the p_i are predicate symbols with arities n_i, the $\underline{x}^{(i)}$ are tuples of individuals $(x_{i1},...,x_{im})$, and $A(\underline{x}^{(0)},...,\underline{x}^{(n)})$ is an A-box rooted by the individuals in the $x^{(i)}$. It is interpreted as a logical formula in the obvious way.

The idea behind the rule ordering is essentially the same as for θ–subsumption. The only difference is that instead of searching for a substitution θ that acts as a witness for the instantiation relation, we now employ the A-box subsumption of the terminological formalism. The resulting rule ordering is called *TL-subsumption*. Assume that two rules

$$p_0(\underline{x}^{(0)}) \leftarrow p_1(\underline{x}^{(1)}),...,p_n(\underline{x}^{(n)}),\ A(\underline{x}^{(0)},...,\underline{x}^{(n)}).$$
$$q_0(\underline{y}^{(0)}) \leftarrow q_1(\underline{y}^{(1)}),...,q_m(\underline{y}^{(m)}),\ B(\underline{y}^{(0)},...,\underline{y}^{(m)}).$$

over disjoint sets of variables are given. The p-rule is more general than the q–rule w.r.t. *TL-subsumption* ($<_{TL}$) iff there is a substitution σ such that the following holds:

1. $p_0(\underline{x}^{(0)})\ \sigma$ and $q_0(\underline{y}^{(0)})$ are equal,
2. $\{p_1(\underline{x}^{(1)})\ \sigma,...,p_n(\underline{x}^{(n)})\ \sigma\} \subseteq \{q_1(\underline{y}^{(1)}),...,q_m(\underline{y}^{(m)})\}$, and
3. the A-box $A(\underline{x}^{(0)},...,\underline{x}^{(n)})\sigma$ subsumes the $B(\underline{y}^{(0)},...,\underline{y}^{(m)})$ w.r.t. $\underline{x}^{(1)}\ \sigma,...,\underline{x}^{(n)}\ \sigma$.

As the terminological formalism provides attributes and a complement operator it is possible to map Herbrand terms into the set of A-boxes that are rooted by one individual such that two Herbrand terms s and t are unifiable iff $\tilde{s}(X) \wedge \tilde{t}(Y) \wedge X = Y$ is satisfiable, where $\tilde{s}$ and $\tilde{t}$ are the images under the mapping from concept terms into first-order formulas as defined in [17]. This embedding naturally extends to a mapping from Horn rules to the rule formalism. It has been shown in [17] that TL-subsumption is at least as powerful as θ–subsumption, i.e., given two Horn rules r_1 and r_2, r_1 is more general than r_2 w.r.t. θ–subsumption iff $r_1 <_{TL} r_2$.

5.4. Abduction

Generalization as described before is applied to the clauses of a theory, i.e., facts and rules, resulting in more general rules. We have also developed a new technique for abduction. In addition to the Horn-clause theory T consisting of facts F and rules R and a set of integrity constraints IC, we also postulate a set of distinguished ground literals A called *abducibles* and a goal G which drives the abduction process.

By abduction we want to find a set of hypotheses $H \subset A$ such that we can derive the (positive) example $e \in E^+$ from $T \cup H$. In the context of theory revision $T \cup H$ gives the new theory T' which again must be consistent with IC, the set of integrity constraints:

$T \cup H \vdash e$
$T \cup H \cup IC$ is consistent

Consider the following example where we have two rules for the recyclability of polypropylenes:

recyclable(closed_circle,Plastic_Id) ←
polypropylene(Plastic_Id),
additive(Plastic_Id,flame_retardent)
recyclable(unrestricted,Plastic_Id) ←
polypropylene(Plastic_Id), pure(Plastic_Id)
polypropylene(X) ← hostalen[2] (X)
additive(ppk_1060,flame-retardent)
hostalen(ppk_1060)

The first rule expresses that a polypropylene can be recycled only in a closed circle, if it contains a

[2]Hostalen is a registered trademark of the HOECHST AG.

flame retardent agent as a fluent additive. This is because the flame retardent agent produces toxic dioxin on ultimate thermic treatment. For a pure polypropylene there is no restriction a recyclability. In two facts we also have that hostalen **ppk_1060** contains a flame retardent fluent additive.

If we declare **additive** and **pure** as abducibles and ask the query how **ppk_1060** can be recycled

?- recyclable(RecKind,ppk_1060)

we get two answers: The first, unconditional answer

RecKind = closed-circle
{}

says that **ppk_1060** can be recycled in a closed circle. The second, conditional answer

RecKind = unrestricted
{pure(ppk_1060)}

is an abductive solution: under the condition that **ppk_1060** is pure, it can be recycled unrestricted.

5.4.1. Bottom-up Abduction

In order to achieve abduction, deduction techniques can be employed in a top-down as well as in a bottom-up manner: with top-down reasoning one skips some subgoals instead of proving them if they are in the set of abducibles. If the goal only consists of abducibles, top-down reasoning stops. The set of remaining goals is the set of hypotheses explaining the toplevel goal.

On the other hand, there are a number of optimization strategies that allow query answering by bottom-up evaluation. Generalized Magic Sets rewriting is such an optimization technique that has been developed for query answering in deductive databases [2]. We have adapted this rewriting technique to achieve bottom-up abduction of Horn knowledge bases [12].

The scheme of our abduction rewriting approach is presented in Fig. 6. Given a theory and a goal we first perform a Generalized Magic Sets rewriting. In a second step we further transform this rulebase with respect to the set of abducibles. Evaluating the resulting abduction rulebase by bottom-up evaluation will compute all abductive solutions.

The transformation can be regarded as a specialization of a partially evaluated upside-down meta-interpreter originally presented by Stickel [39] (see also [19]). Compared to Stickel's approach we have a number of advantages:

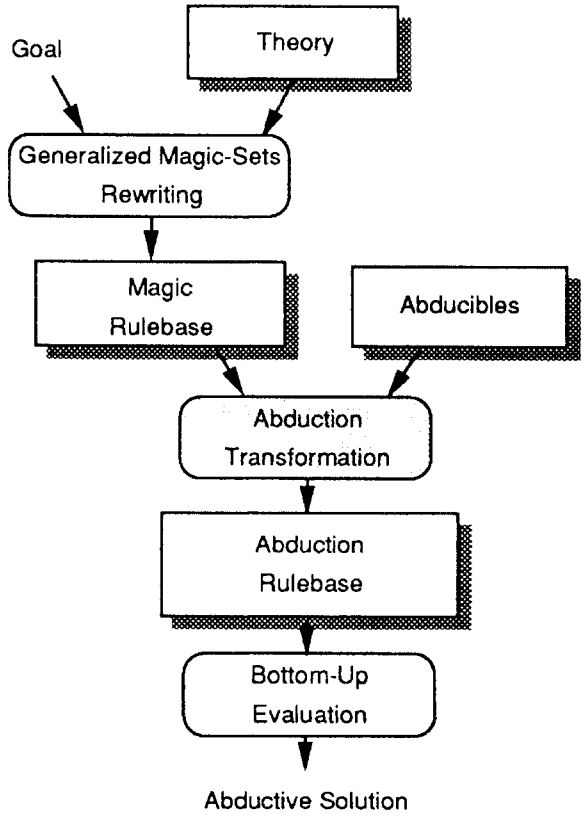

Fig. 6. Bottom-up abduction by knowledge base rewriting.

- Only the rules of the knowledge base are transformed; rewriting of the ground facts in the knowledge base is avoided. This is very important when the ground facts change frequently or if they reside on secondary storage like in deductive databases.
- There is no need for enumeration of all the possible hypotheses. Thus, the approach is applicable if the set of possible hypotheses is infinite.
- Hypotheses will be derived only if they are not already contained as facts in the knowledge base.
- By normalization meta predicates are removed, resulting in improved performance.

Most important: this set-oriented approach is usable also for large sets of facts. This is supports our objective to develop techniques suitable not only for toy examples but also for complex real

world problems with databases and large knowledge bases.

5.4.2. Using Abduction for Generalization

As mentioned before, generalization of Horn clauses can be done in different ways, e.g., by generalizing some terms (argument positions) or by dropping entire literals (removing conditions).

Thus, the decision about which generalization operation should be applied is still a problem. Abduction can provide considerable help for making this decision. Consider again the recyclability example introduced in the last paragraph.

If we recognize the fact that hostalen **ppk_1060** can be recycled unrestricted (no matter of its pureness) then we have to revise our theory to now cover this (positive) example. By processing the query

?- recyclable(unrestricted,ppk_1060)

we obtain the abductive solution

yes
{pure(ppk_1060)}

which gives rise to generalize the **recyclable** rule for **ppk_1060** by dropping this pureness condition. Thus, we substitute the original rule

recyclable(unrestricted,Plastic_Id) ←
polypropylene(Plastic_Id), pure(Plastic_Id)

by the following generalized one:

recyclable(unrestricted,Plastic_Id) ←
polypropylene(Plastic_Id)

Although this is only one example of how abduction and generalization can cooperate in the theory revision framework, it already shows the combined potential for our application.

5.5. Knowledge Base Verification

It has already been pointed out that only those generalizations and abductive solutions are accepted which are consistent with the integrity constraints IC. Integrity constraints encode negative or disjunctive knowledge. These integrity constraints are represented as denials, i.e., clauses with an empty head. Eshghi and Kowalski use this kind of integrity constraints for their abduction procedure [15]. We can also represent them as clauses with the special atom **false** as conclusion [25].

An obvious integrity constraint is that if a material contains a fluent additive it is no longer pure. This is represented by the following rule: if a material **Pl** has a fluent additive and the same material is pure then there is an inconsistency:

false ← additive(Pl,X), pure(Pl)

Consider for example, that the following facts and rules would be contained in a knowledge base.

hostalen(ppk_1060)
novodur(r_5320)
additive(ppk_1060, flame-retardent)
polypropylene(X) ← hostalen(X)
absc(X) ← novodur(X)

In Section 5.4 we have found by abduction that Hostalen PPK 1060 could be recycled without any restriction if it was pure. So we can tentatively add this information to the knowledge base as an additional fact: **pure(ppk_1060)**.

A naive method for integrity checking would be to use a proof-finding approach and ask the query

?- false

This procedure would invoke all integrity constraints in backward direction even if they are independent from the new fact. However, it would be much more efficient to derive only those facts that are consequences of this new assertion. In [15] it is argued to do this kind of constraint checking by forward reasoning starting with the new fact. But forward reasoning from one fact alone is not sufficient. The following integrity constraint says that 'polypropylenes and ABSCs must not be components of a single composite product'.

false ← composite(Pl1,Pl2),
polypropylene(Pl1),
absc(Pl2)

Adding the new fact **composite(ppk_1060, r_5320)** would lead to an inconsistency which will

not be detected by forward chaining this fact alone. Additionally, we need to prove whether the premises **polypropylene(ppk_1060)** and **absc (r_5320)** can be satisfied.

In [25] a model-generation approach has been applied for this problem. Here, however, we regard checking of integrity constraints as a consequence-finding problem [20]. Given an update of a deductive database or a logic program, consequence finding applies only those rules that are effected by the update operation. This builds on the assumption that the database satisfied its integrity constraints prior to the update. Derivation is restricted to exactly those facts that depend on an explicitly given set of initial facts, in our case the hypotheses found by abduction.

The extended SLDNF resolution of [35] uses the clauses corresponding to the updates as top-clauses for the search space and thus achieves the effect of simplification methods investigated by [14,24,28]. The approach combines forward and backward chaining depending on whether a positive or negative literal is resolved upon.

As an alternative to this tuple-oriented method we have developed a rewriting approach [18]. It is an extension of the well-known Generalized Magic Sets rewriting technique [2], which was also the basis for bottom-up abduction in Section 5.4.1. Since this technique in some sense integrates forward and backward chaining, it seems natural to extend it for consequence finding.

By Generalized Magic Sets rewriting, information about variable bindings given by the query is propagated *down* into the bodies of the rules at compile-time. For consequence finding we do not have a query but a number of initial facts – the update information – from which to reason forward.

Thus, the input to the consequence finding transformation is a set of initial facts and the rules of the knowledge base. The transformation algorithm specializes the knowledge base by introducing additional rules and predicates. It extends the Generalized Magic Sets rewriting by an *up* propagation in addition to the usual *down* propagation. When the rewritten knowledge base is evaluated by a model-generating, bottom-up procedure, the generation of a complete minimal model is restricted to the consequences of the initial facts. Because it is a set-oriented strategy it is very efficient if facts have to be retrieved from a database.

6. Conclusion

As knowledge-based systems are brought to practical applications and knowledge bases are to be used over years, the problem of knowledge base evolution naturally comes up: the key issue is how to ensure that the knowledge base does always represent all the knowledge that is relevant for solving tasks, i.e., being '*complete*', and does not become out of date or invalid, i.e., remaining '*sound*' with respect to some specific situational context. Although this is a goal hard to achieve, it shows the direction in which knowledge base evolution research should work: to overcome the (always 'damned but nevertheless done') accumulation of 'small local hacks' causing unforeseeable consequences and to find a compromise between this ad-hoc KB modification approach and the other extreme of restarting the whole knowledge engineering work ranging from the formal specification down to the concrete representation with each KB modification.

In this paper, we have shown that knowledge base evolution can be regarded as a theory revision process. Research in Inductive Logic Programming provides us with a set of techniques that can be applied to incorporate new knowledge into the knowledge base (*knowledge base exploration*), e.g., by generalization and abduction. On the other hand, techniques from deductive database research can be used for ensuring the integrity of the knowledge base, i.e., for solving the *knowledge base verification and validation* task.

For both tasks we have developed extensions and modifications motivated by the special characteristics of the application. The generalization techniques taken from ILP have been extended towards the incorporation of meta-knowledge for guiding the generalization process (plgg) and towards additional language features for representing generalization results (e.g., finite domain terms). Additionally, we have proposed an alternative to θ–subsumption based on an extension of Horn rules incorporating termi-nological knowledge representation and reasoning (TL-subsumption). In order to get efficient evolution

techniques also for large sets of rules and facts we extended the rewriting techniques from deductive databases for abduction and integrity checking.

Further work on knowledge base evolution should not only consider developing more powerful exploration and verification methods, but should also focus on the knowledge representation language itself. It is obvious that a more powerful but still semantically clear representation formalism, as e.g., introduced for TL-subsumption, will be of great advantage for all kinds of knowledge evolution techniques. For example, introducing sorts or types as mentioned in several parts of this paper can be a first but only intermediate step: generalization within a sort lattice does already yield a more fine-grain clause ordering than simple θ-subsumption. However, extending the logic-based representation language by substituting or complementing constitutively given sorts by intensionally defined concepts and concept terms in the sense of terminological reasoning will be necessary for finding and expressing 'really least general' generalizations and thus being able to support knowledge base evolution over a long period of time.

Currently only little work is available on tailoring the knowledge representation formalism to knowledge base evolution needs [4]. But being convinced that research on this will be a key issue for the success of knowledge base evolution in the future, we will also concentrate on further improving knowledge representation approaches like TL-subsumption besides developing the evolution techniques themselves.

References

[1] S. Amarel (1983): Problems of representation in heuristic problem solving: Related issues in the development of expert systems. In M. Groner, R. Groner and W. Bischof (eds.): *Methods of Heuristics*. Erlbaum, Hillsdale, NJ, pp. 245–350.

[2] Catriel Beeri and Raghu Ramakrishnan (October 1991): On the power of magic. *Journal of Logic Programming*, 10: 255–299.

[3] H. Boley, U. Buhrmann and C. Kremer (January 1994): Towards a sharable knowledge base on recyclable plastics. To appear in: *TMS'94 Symposium on Knowledge-Based Applications in Material Science and Engineering*, Feb/Mar 1994. San Francisco, USA.

[4] Harold Boley (1993): Towards Evolvable Knowledge Representation for Industrial Applications. In Knut Hinkelmann and Armin Laux (eds.): *DFKI-Workshop on Knowledge Representation Techniques*. Kaiserslautern, number D-93-11.

[5] Harold Boley (March 1994): Finite Domains and Exclusions as First-Class Citizens. In Roy Dyckhoff (ed.): *Fourth International Workshop on Extensions of Logic Programming, St. Andrews, Scotland, 1993, Preprints and Proceedings*. LNAI, Springer.

[6] Alexander Borgida, Ronald Brachman, Deborah McGuinness and Lori Resnick (1989): CLASSIC: A structural data model for objects. In *International Conference on Management on Data*. ACM SIGMOD.

[7] R.J. Brachman and J.G. Schmolze (1985): An overview of the KL-ONE knowledge representation system. *Cognitive Science* 9(2): 171–216.

[8] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick and Alexander Borgida (June 1990): Living with CLASSIC: When and How to Use a KL-ONE-Like Language. In *Principles of Semantic Networks*. J. Sowa Morgan Kaufmann Publishers Inc.

[9] H. Breuer, G. Dupp and J. Schmitz (1990): Einheitliche Werkstoffdatenbank – eine Idee setzt sich durch. *Kunststoffe* 80: 11.

[10] B.G. Buchanan (1989): Can machine learning offer anything to expert systems? *Machine Learning* 3(4): 251–254.

[11] W. Buntine (1988): Generalized subsumption and its applications on induction and redundancy. *Artificial Intelligence* 36: 149–176.

[12] Gerhard Burgun and Knut Hinkelmann (1994): Knowledge base rewriting for bottom-up abduction (in preparation).

[13] Frans Coenen and Trevor Bench-Capon (1993): *Maintenance of Knowledge-based Systems*. Academic Press.

[14] Hendrik Decker (April 1986): Integrity enforcement on deductive databases. In Larry Kerschberg (ed.): *Proceedings from the 1st International Conference on Expert Database Systems*. Charleston, South Carolina. The Benjamin/Cummings Publishing Company, Inc., pp. 381–395.

[15] Kave Eshghi and Robert Kowalski (1989): Abduction compared with negation by failure. In *6th International Conference on Logic Programming (ICLP'89)*.

[16] Allen Ginsberg, Sholom M. Weiss and Peter Politakis (1988): Automatic knowledge base refinement for classification systems. *Artificial Intelligence* 35: 197–226.

[17] Philipp Hanschke and Manfred Meyer (August 1992): An Alternative to θ-Subsumption Based on Terminological Reasoning. In Celine Rouveirol (ed.): *Workshop on Logical Approaches to Machine Learning, ECAI 92, Vienna*.

[18] Knut Hinkelmann (1994): A consequence-finding approach for feature recognition in CAPP. In *Seventh International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE'94)* (forthcoming).

[19] Katsumi Inoue, Yoshihiko Ohta, Ryuzo Hasegawa and Makoto Nakashima (1993): Bottom-up Abduction by Model Generation. In *Proc. of the 13th IJCAI*, pp. 102–108.

[20] Katsumi Inoue (1991): Consequence-finding based on ordered linear resolution. In *Proc. of the 12th IJCAI*. Sidney, Australia.

[21] Joxan Jaffar and Jean-Louis Lassez (January 1987): Constraint logic programming. In *Proc. POPL-87*. Munich, Germany. ACM, pp. 111–119.

[22] Christian Kissinger (1993): Einfluß verschiedener Verarbeitungsparameter aus die Plattengröße und die Formänderungsarbeit 2-dimensionaler verpresster GMT-Halbzeuge. Technical Report 93-58. Institut für Verbundwerkstoffe GmbH.

[23] J. Krottmaier (1991): *Versuchsplanung: Der Weg zur Qualität des Jahres 2000*. Verlag Industrielle Organisation Zürich.

[24] John W. Lloyd, E.A. Sonenberg and Rodney W. Topor (1987): Integrity constraint checking in stratified databases. *Journal of Logic Programming* 4: 331–343.

[25] Rainer Manthey and Francois Bry (1987): SATCHMO: a theorem prover implemented in prolog. In *Conference on Automated Deduction, CADE*.

[26] Manfred Meyer (August 1994): Issues in Concurrent Knowledge Engineering: Knowledge Sharing and Knowledge Evolution. In Michael Sobolewski (ed.): *Proceedings First International Conference on Concurrent Engineering, Research and Applications (CERA'94), Pittsburgh*. IEEE Computer Press.

[27] Raymond J. Mooney and John M. Zelle (1994): Integrating ILP and EBL. *SIGART Bulletin* 5(1): 12–21. Special Section on Inductive Logic Programming.

[28] Jean-Marie Nicolas (1982): Logic for improving integrity checking in relational data bases. *Acta Informatica* 18: 227–253.

[29] Gordon D. Plotkin (1970): A note on inductive generalization. In B. Meltzer and D. Michie (eds.): *Machine Intelligence*, vol. 5. Elsevier North-Holland, New York, pp. 153–163.

[30] Gordon D. Plotkin (1971): *Automatic Methods of Inductive Inference*. PhD thesis, University of Edinburgh.

[31] Thomas Reinartz and Franz Schmalhofer (June 1994): An integration of knowledge acquisition techniques and EBL for real-world production planning. *Knowledge Acquisition Journal*.

[32] Larry Rendell (1986): A general framework for induction and a study of selective induction. *Machine Learning* 1(1): 177–226.

[33] Bradley Richards and Raymond J. Mooney (March 1991): First-order theory revision. Technical Report AI 91-155. The University of Texas at Austin, Artificial Intelligence Laboratory.

[34] J.A. Robinson (1965): A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery* 12: 23–41.

[35] Fariba Sadri and Robert Kowalski (1988): A theorem-proving approach to database integrity. In Jack Minker (ed.): *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann Publishers, Inc., Los Altos, CA, pp. 313–362.

[36] F. Schmalhofer and B. Tschaitschian (June 1993): The acquisition of a procedure schema from text and experiences. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, pp. 883–888.

[37] Franz Schmalhofer, Thomas Reinartz and Bidjan Tschaitschian (1994): A unified approach to learning in complex real world domains. *Applied Artificial Intelligence* (in press).

[38] G. Smolka (May 1989): *Logic Programming over Polymorphically Order-Sorted Types*. PhD thesis, University of Kaiserslautern, Germany.

[39] Mark E. Stickel (July 1991): Upside-down meta-interpretation for the model-elimination theorem-proving procedure for deduction and abduction. Technical Report TR-664, ICOT.

[40] G. Tecuci and Y. Kodratoff (1990): Apprenticeship learning in imperfect domain theories. In Y. Kodratoff and R.S. Michalski (eds.): *Machine Learning: An artificial intelligence approach*, vol. 3. Morgan Kaufmann, San Mateo, CA, pp. 514–551.