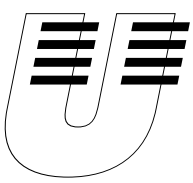

UNIVERSITÄT
KOBLENZ · LANDAU

Model Based Deduction for Database Schema Reasoning

Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt, Thomas Kleemann

5/2004

Fachberichte
INFORMATIK

Universität Koblenz-Landau
Institut für Informatik, Universitätsstr. 1, D-56070 Koblenz

E-mail: researchreports@uni-koblenz.de,
WWW: http://www.uni-koblenz.de/fb4/

Model Based Deduction for Database Schema Reasoning

Peter Baumgartner[1], Ulrich Furbach[2], Margret Gross-Hardt[2], and Thomas Kleemann[2]

[1] MPI Informatik, D-66123 Saarbrücken, Germany, baumgart@mpi-sb.mpg.de
[2] Universität Koblenz-Landau, D-56070 Koblenz, Germany,
{uli|margret|tomkl}@uni-koblenz.de

Abstract. We aim to demonstrate that automated deduction techniques, in particular those following the model computation paradigm, are very well suited for database schema/query reasoning. Specifically, we present an approach to compute completed paths for database or XPath queries. The database schema and a query are transformed to disjunctive logic programs with default negation, using a description logic as an intermediate language. Our underlying deduction system, KRHyper, then detects if a query is satisfiable or not. In case of a satisfiable query, all completed paths – those that fulfill all given constraints – are returned as part of the computed models.
The purpose of our approach is to dramatically reduce the workload on the query processor. Without the path completion, a usual XML query processor would search the database for solutions to the query.
In the paper we describe the transformation in detail and explain how to extract the solution to the original task from the computed models.
We understand this paper as a first step, that covers a basic schema/query reasoning task by model-based deduction. Due to the underlying expressive logic formalism we expect our approach to easily adapt to more sophisticated problem settings, like type hierarchies as they evolve within the XML world.

1 Introduction

Automated theorem proving is offering numerous tools and methods to be used in other areas of computer science. An extensive overview about the state of the art and its potential for applications is given in [7]. Very often there are special purpose reasoning procedures which are used to reason for different purposes like knowledge representation [16] or logic programming [10].

The most popular methods used for practical applications are resolution-based procedures or model checking algorithms. In this paper we want to demonstrate that there is a high potential for model based procedures for database schema reasoning. Model based deduction can be based very naturally on tableau calculi [13], and in particular on the developments that started with the *SATCHMO* approach [20] and that were refined later and extended in the hyper tableau calculus [6].

We start with the idea of representing a (database) schema as a description logic knowledge base. This idea as such is not new and has been put forward by [9] and [8]. However, we found that the services usually available in description logic reasoners do not allow to express all constraints that were imposed by the schema in order to solve the tasks we are looking at. Indeed, the work in [9] and [8] aims at different

purposes, where schema reasoning tasks can be reduced to satisfiability of description logic knowledge bases. We are considering the tasks of testing and optimizing certain forms of database queries as they arise in the XML world. To this end, a "pure" description logic approach was proposed before in [4]. In the present paper, the limitations of that approach are overcome by a translation of a schema into a disjunctive logic program. When given to the hyper tableau based theorem prover [24] then, the solutions to the original task are encoded in the computed models. That is, instead of answering a satisfiability question the model itself constitutes the solution. The usage of a model generation theorem prover thus is motivated by the applications requirement to enumerate models/answers rather than querying the existence of a model.

In the following section we briefly review the hyper tableau prover, and on this basis we will describe its application in the successive sections.

2 Theorem Proving with Hyper Tableau

Features. The Hyper Tableau Calculus is a clause normal form tableau calculus [6], which can be seen as a generalization of the SATCHMO-procedure [20]. Hyper tableau have been used in various applications (for examples see [3, 5]), where two aspects turned out to be of importance: The result of the theorem prover is a model (if the specification is satisfiable) and this model can be seen as the result of the prover's "computation"; it can be used by the system, where the prover is embedded, for further computation steps. The second aspect is concerned with non-monotonic negation. Although an entire discipline, namely knowledge representation, is emphasizing the necessity of non-monotonic constructs for knowledge representation, there are rarely automated theorem proving systems which are dealing with such constructs.

The hyper tableau theorem prover *KRHyper* allows application tasks to be specified by using first order logic — plus possibly non-monotonic constructs — in clausal form. While *KRHyper* can be used straightforwardly to prove theorems, it also allows the following features, which are on one hand essential for knowledge based applications, but on the other hand usually not provided by first order theorem provers:

1. Queries which have the listing of predicate extensions as answer are supported.
2. Queries may also have the different extensions of predicates in alternative models as answer.
3. Large sets of uniformly structured input facts are handled efficiently.
4. Arithmetic evaluation is supported.
5. Non-monotonic negation is supported.
6. The reasoning system can output proofs of derived facts.

More details about these features can be found in [24]. Also, we only note that with a simple transformation of the given rule set, *KRHyper* is sound, complete and terminating with respect to the *possible models* [21] of a stratified disjunctive logic program without function symbols (except constants).

A Small Example. Hyper tableau is a "bottom-up" method, which means that it generates instances of rule[3] heads from facts that have been input or previously derived. If a hyper tableau derivation terminates without having found a proof, the derived facts form a representation of a model of the input clauses.

The following example illustrates how our hyper tableau calculus based system, *KRHyper*, proceeds to generate models. Figure 1 shows four subsequent stages of a derivation for the following input clauses[4]:

$$p(a) \leftarrow \tag{1}$$
$$q(x,y) \vee r(f(z)) \vee r(x) \leftarrow p(x) \tag{2}$$
$$\leftarrow q(x,x) \tag{3}$$
$$s(x) \leftarrow p(x),\ \text{not}\, r(x) \tag{4}$$

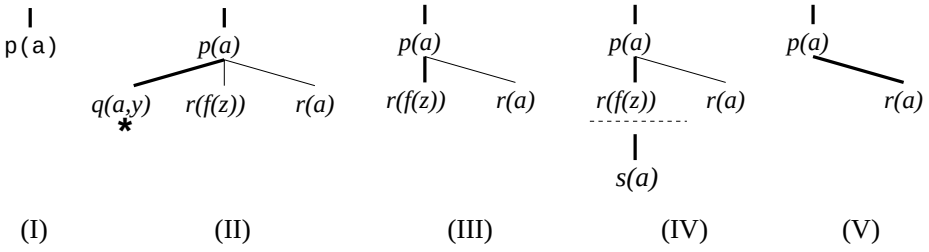

Fig. 1. Stages of a *KRHyper* derivation

KRHyper provides stratified negation as failure. The set of input clauses is partitioned into *strata*, according to the predicates in their heads: if a clause c_1 has a head predicate appearing in the scope of the negation operator "not" in the body of c_2, then c_1 is in a lower stratum than c_2. In the example, we have two strata: the lower one containing clauses (1), (2) and (3), the higher one clause (4).

As noted, a rule head may be a disjunction. In hyper tableau, disjunctions are handled by exploring the alternative branches in a systematic way. This explains the tree structure in Figure 1. Backtracking can be used to generate one model after the other.

Stage (I) shows the data structure maintained by the method, also called *hyper tableau*, after the input fact (1) has been processed. One can view the calculus as attempting to construct the representation of a model, the *active branch*, shown with bold lines in the figure. At step (I), this model fragment contradicts for example with clause (2): a model containing $p(a)$ must also contain all instances of $q(a,y)$ or of $r(f(z))$ or

[3] We use an implication-style notation for clauses throughout this paper: A clause is viewed as rule *Head* $\leftarrow$ *Body*, where *Head* consists of its positive literals, combined by "$\vee$", and *Body* consists of its negative literals, combined by "," (*and*). Both the head and the body may be empty.

[4] Here and below, the letters x, y, z denote variables, while a, b, c denote constants.

$r(a)$. The model fragment is "repaired" by derivating consequences and attaching them to the hyper tableau: The corresponding instance of clause (2) is attached to the hyper tableau. Since it has a disjunctive head, the tableau splits into three branches. The first branch is inspected and proved contradictory with clause (3). This state is shown in (II).

Computation now tracks back and works on the second branch. With the clauses of the lower stratum, no further facts can be derived at this branch, which means that a model for the stratum has been found, as shown in step (III). Computation then proceeds with the next higher stratum: $s(a)$ can be derived by clause (4). Since no further facts can be derived, a model for the whole clause set has been found, represented by the facts on the active branch: $\{p(a), r(f(z)), s(a)\}$, as shown in (IV).

If desired, the procedure can backtrack again and continue to find another model, as shown in state (V). Another backtracking step then finally leads to the result, that there is no further model.

We conclude by noting that the *KRHyper* system implements the calculus by a combination of semi-naive rule evaluation with backtracking over alternative disjuncts and iterative deepening over a term weight bound. It extends the language of first order logic by stratified negation as failure and built-ins for arithmetic.

3 Flexible Database Queries for XML Data

Querying databases requires that users know the structure of a database. In fact, they have to know the *database schema* in order to formulate valid queries. In the context of complex structured data and large database schemas, knowing the complete schema is not always possible. Querying data, therefore, may be a tedious task. This section describes the application of the *KRHyper* System in order to enhance the flexibility of database queries. In particular, we focus on XML databases and address the following issues in querying XML databases:

- XML documents contain complex structured data, often rather nested. Users therefore have to navigate through these data.
- XML data usually is described by Document Type Definitions (DTDs) and more recently, XML Schema is used. Different from DTDs, XML Schema offers in some sense object oriented concepts as user defined types and aggregation as well as specialization relationships between them. During query processing and optimization this schema knowledge may be used, e.g. in order to avoid evaluation of unsatisfiable queries.
- In an XML schema so called *substitution groups* define types that can be substituted for each other, comparable to union types in other (programming) languages, though, with the difference, that types which are substitutes for each other have to be related via specialization.
- Existing querying languages like XQuery [23] offer navigational expressions on the document level in order to access parts of an XML document. These languages do not cope with type expressions. Type expressions may be helpful in order to query instances of some *general* type T, resulting in instances of type T as well as of all subtypes for T.

Let us consider an example XML document representing a university with a library and researchers working in the university. A library consists of books where each book has a title, an author and an ISBN. Researchers have a name and an associated set of publications e.g. articles, monographs or some general kind of publication without further specialization.

An XML schema itself also is an XML document listing the complex types together with their elements referring to other (complex) types. Furthermore by means of a so called restriction expression, it is possible to represent specialization relationships between types. Instead of the linear, XML based description of an XML schema, we use a more illustrative, graphical representation for the types and their relationship in a schema. An XML Schema is represented by a schema graph, where nodes represent the types and substitutiongroups of the schema and edges represent aggregation and specialization relationships between types. Starting from such a schema graph, we present an approach that allows a user to query the data, even if only parts of a database schema is known. Figure 2 shows an example schema graph.

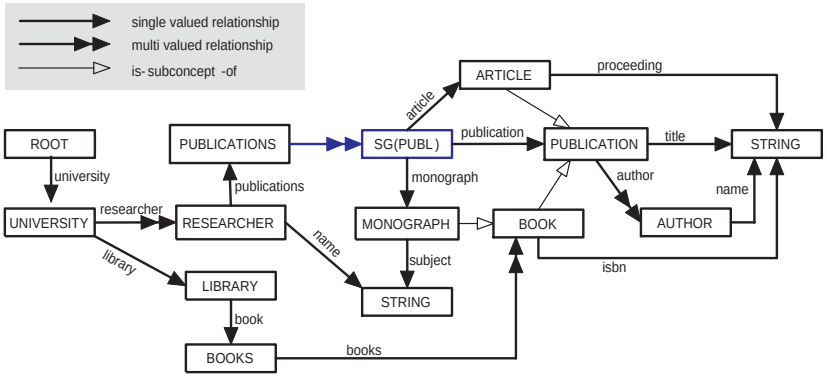

Fig. 2. Example schema graph.

This schema shows types like UNIVERSITY, PUBLICATION, AUTHOR etc. with their elements referring to other types as well as specialization relationships between types. For instance, the type UNIVERSITY has an element researcher of type RESEARCHER and an element library of type LIBRARY. Furthermore, PUBLICATION is a general type with specializations BOOK and ARTICLE. There is one substitution group SG(PUBL) contained in this schema whose general type is PUBLICATION and potential substitutes are ARTICLE and MONOGRAPH, actually specializations of PUBLICATION. In the transformation given below, we will see that substitution groups are a means to express a disjunction of disjoint concepts.

To keep the representation of schema graphs as well as the query processing simple, we only consider (XML-)*elements* describing complex data structures but do not cope

with the term of (XML)-*attributes*. Nevertheless, we will use both terms in order to refer to properties of data items.

3.1 From XML Schema to Description Logics

An XML database is described by a XML Schema. The schema is represented by a graph. The nodes of the graph are the complex type identifiers; relationships between elements and their corresponding subelement types are represented by aggregation edges and "extension"-relationships, describing the generalization relationship between complex types, are represented by so called is-a edges. XML Schema supports the modelling of multiple complex types, that are extension of the same general complex type within a substitution group, comparable to a union type in other languages.

We describe databases and schemas by means of graphs. In the following we assume a possibly infinite set L of labels. Labels will be used as edge annotations in various places.

Definition 1 (XML Schema). *Let C be a set of* type names *and SG be a disjoint set of* substitution group identifiers. *An* XML schema *(or* schema *for short) is a graph $S = (C \cup SG, E_{\text{rel}}, \cup E_{\text{isa}} \cup E_{\text{SG-in}} \cup E_{\text{SG-out}}, r)$ where C and SG are the nodes, E_{rel}, E_{isa}, $E_{\text{SG-in}}$, $E_{\text{SG-out}}$ are disjoint sets of edges (representing attributes/elements edges, inheritance edges, incoming edges to SG nodes, outgoing edges from SG nodes, respectively), and $r \in C$ is the* root *of the schema. Every schema must satisfy the following properties:*

1. $(v,v) \notin E_{\text{rel}} \cup E_{\text{isa}} \cup E_{\text{SG-in}} \cup E_{\text{SG-out}}$, *for any node v.*
2. *Each edge in E_{rel} and each edge in $E_{\text{SG-out}}$ is labeled with an element from L. All other edges are not labeled.*
3. $(v,v') \in E_{\text{SG-in}}$ *if and only if $v' \in SG$ (incoming edges to nodes in SG are precisely those from $E_{\text{SG-in}}$).*
4. $(v,v') \in E_{\text{SG-out}}$ *if and only if $v \in SG$ (incoming edges to nodes in SG are precisely those from $E_{\text{SG-in}}$).*

Schema can be translated in a straightforward way to description logics as follows[5]:

Definition 2 (Schema to Description Logic). *Let $S = (C \cup SG, E_{\text{rel}}, \cup E_{\text{isa}} \cup E_{\text{SG-in}} \cup E_{\text{SG-out}}, r)$ be a schema. The* TBox for *S, $T(S)$, is defined as the smallest set of inclusion statements satisfying the following properties:*

Translation of is-a links. *If $c \in C$ and $\{d \mid (c,d) \in E_{\text{isa}}\} = \{d_1, \ldots, d_n\}$, for some $n \geq 1$, then $T(S)$ contains the inclusion $c \sqsubseteq d_1 \sqcup \cdots \sqcup d_n$.*

Translation of elements/attributes. *If $(c,d) \in E_{\text{rel}}$ and (c,d) is labeled with l, then $T(S)$ contains the inclusion $c \sqsubseteq \exists l\,.\,d$.*

Translation of substitution groups. *If $(c,v) \in E_{\text{SG-in}}$ then $T(S)$ contains the inclusion $c \sqsubseteq v$. If $v \in SG$ and $\{d \mid (v,d) \in E_{\text{SG-out}}\} = \{d_1, \ldots, d_n\}$, for some $n \geq 1$, then $T(S)$ contains the inclusion $v \sqsubseteq d_1 \sqcup \cdots \sqcup d_n$ and the inclusions $d_i \sqcap d_j \sqsubseteq \bot$, for all i, j with $1 \leq i, j \leq n$ and $i \neq j$.*

[5] We use standard description logic notation, see [2] for an introduction.

This translation conforms to concept and role formations found even in basic description logic languages like ALC. Although the translation does *not* result in an ALC TBox ($T(S)$ might contain several inclusion statements with the same concept at the left hand side), it is easy to see that the result of the transformation can easily be brought to an ALC conforming TBox (possibly cyclic).

At this point we will not discuss how to employ description logic reasoners to solve the tasks we are interested in. This discussion will be postponed after our approach based on model computation has been described.

3.2 From Description Logics to Model Computation

The following translation is the standard relational translation from description logics to predicate logic. For our purpose, it is enough to work in a restricted setting, where all inclusions in a TBox are of a particular form, which is obtained as the result of the transformation in Definition 2.

Definition 3 (Description Logic to Rules – Basic Version). *Let S be a schema. The* rules for S, $R(S)$, *is defined as the smallest set of rules satisfying the following properties:*

1. *if $c \sqsubseteq c_1 \sqcup \cdots \sqcup c_n \in T(S)$ then $R(S)$ contains the rule $c_1(x) \vee \cdots \vee c_n(x) \leftarrow c(x)$*
2. *if $c \sqsubseteq \exists l\,.\,d \in T(S)$ then $R(S)$ contains the rules $l(x, f_{c,l,d}(x)) \leftarrow c(x)$ and $d(f_{c,l,d}(x)) \leftarrow c(x)$. ($f_{c,l,d}$ is a unary function symbol whose name contains c, l and d, as indicated.)*
3. *if $c \sqcap d \sqsubseteq \bot \in T(S)$ then $R(S)$ contains the rule* false $\leftarrow c(x), d(x)$.

Using this transformation, simple graph reachability problems can be reduced easily to model computation problems. Speaking in terms of the schema graph, to compute a path, say, from a node c to a node d in a schema S, it suffices to add to $R(S)$ the fact $c(a) \leftarrow$ (for some constant a) and the rules found $\leftarrow d(x)$ and false $\leftarrow$not found, where found is a predicate symbol not occuring in $R(S)$. Each model of the thus obtained program corresponds to exactly one path from c to d in S. However, this approach works only in a satisfactory way if the schema does not contain any circle.

Example 1 (Cycle). Consider the following TBox, which can be obtained by translating some schema S containing a circle.

$$c \sqsubseteq \exists l\,.\,d \tag{5}$$

$$d \sqsubseteq \exists k\,.\,c \tag{6}$$

$$c \sqsubseteq \exists m\,.\,e \tag{7}$$

Its translation to rules gives the following program:

$$l(x, f_{c,l,d}(x)) \leftarrow c(x) \qquad d(f_{c,l,d}(x)) \leftarrow c(x) \tag{8}$$

$$k(x, f_{d,k,c}(x)) \leftarrow d(x) \qquad c(f_{d,k,c}(x)) \leftarrow d(x) \tag{9}$$

$$m(x, f_{c,m,e}(x)) \leftarrow c(x) \qquad e(f_{c,m,e}(x)) \leftarrow c(x) \tag{10}$$

Now, any Herbrand model as computed by bottom-up procedures will be infinite and contains $c(a)$, $c(f_{d,k,c}(f_{c,l,d}(a)))$, $c(f_{d,k,c}(f_{c,l,d}(f_{d,k,c}(f_{c,l,d}(a)))))$ and so on. Therefore, *KRHyper* and related procedures will not terminate.

3.3 Blocking by Transformation

In this section we give an improved transformation in order to guarantee termination of the model computation. This will be achieved by a "loop check" similar to the blocking technique known from the description logic literature. The idea is to re-use an individual already known to belong to a certain concept instead of adding a new individual to it in order to satisfy an existentially quantified role constraint. In the example, the individual a can be re-used instead of $f_{d,k,c}(f_{c,l,d}(a))$ in order to put $f_{c,l,d}(a)$ into the k-relation to some individual belonging to c. This re-use technique will be described now. It will guarantee the termination of our reasoning algorithm.

Definition 4 (Description Logic to Rules – Improved Version). *Let S be a schema. The* rules for S, $R(S)$, *is defined as the smallest set of rules satisfying the following properties:*

1. *if $c \sqsubseteq c_1 \sqcup \cdots \sqcup c_n \in T(S)$ then $R(S)$ contains the rule $c_1(x) \vee \cdots \vee c_n(x) \leftarrow c(x)$*
2. *$R(S)$ contains the fact $\mathsf{equal}(x,x) \leftarrow$.*
3. *if $c \sqsubseteq \exists l\,.\,d \in T(S)$ then $R(S)$ contains the following rules:*

$$\mathsf{new}_{c,l,d}(x) \vee \mathsf{old}_{c,l,d}(x) \leftarrow c(x) \tag{1}$$

$$\mathsf{false} \leftarrow \mathsf{new}_{c,l,d}(x), \mathsf{old}_{c,l,d}(x) \tag{2}$$

$$l(x, f_{c,l,d}(x)) \leftarrow \mathsf{new}_{c,l,d}(x) \tag{3}$$

$$d(f_{c,l,d}(x)) \leftarrow \mathsf{new}_{c,l,d}(x) \tag{4}$$

$$\mathsf{l(x,z)} \leftarrow \mathsf{old}_{c,l,d}(x), c(y), l(y,z), d(z) \tag{5}$$

$$\mathsf{false} \leftarrow \mathsf{old}_{c,l,d}(x), \mathsf{not\ some}_{c,l,d} \tag{6}$$

$$\mathsf{some}_{c,l,d} \leftarrow c(x), l(x,y), d(y) \tag{7}$$

$$\mathsf{false} \leftarrow \mathsf{new}_{c,l,d}(x), \mathsf{new}_{c,l,d}(y), \mathsf{not\ equal}(x,y) \tag{8}$$

4. *if $c \sqcap d \sqsubseteq \bot \in T(S)$ then $R(S)$ contains the rule $\mathsf{false} \leftarrow c(x), d(x)$.*

Some comments are due. The difference to the previous version is the translation of inclusions of the form $c \sqsubseteq \exists l\,.\,d$. In order to explain it, suppose that the concept c is populated with some individual, say, a. That is, when constructing a model, $c(a)$ already holds true. Now, the program above distinguishes two complementary cases to satisfy the constraint $\exists l\,.\,d$ for a: either a new l-connection is made between a and some new individual in d, or an existing ("old") l-connection between some individual from c and from d is re-used. That exactly one of these cases applies is guaranteed by the rules (1) and (2). The rules (3) and (4) are responsible to establish a new connection, while the rule (5) is responsible to re-use an existing connection. To achieve the desired effect, some more constraints are needed: as said, re-using an existing connection is realized by applying the rule (5). It establishes the connection $l(x,z)$, where x stands for the object the connection is to be established from (a in the example), and z stands for the re-used object from d. However, there is no guarantee per se that the rule's subgoals $c(y)$, $l(y,z)$ and $d(z)$ are satisfied. This, however, is achieved by the rules (6) and (7): whenever the program choses to re-use an old connection, i.e. to build a model containing this choice,

by (6) and (7) this can succeed only if the mentioned subgoals are satisfiable. Finally, the rule (8) acts as a "loop check": with it, it is impossible that between individuals belonging to the concepts c and d more than one new l-connection is made. Only new l-connections cause insertion of more complex atoms[6] and thus are the only source for non-termination. With the rule (8) there is a finite bound on the complexity then for a given program.

The program above is intended to be run by a bottom-up model computation procedure like *KRHyper* (Section 2). Together with some more rules and facts obtained by further transformation steps (see Section 3.4) this yields an algorithm that is similar to usual tableau algorithms developed for description logics. On the one side, our translation and the reasoning tasks to be solved do not quite match any existing of those algorithms. This is because of the use of nonmonotonic negation to filter out nonintended models (see Section 3.4). Another difficulty we encountered with existing systems is their inability to actually output the models computed. From our application point of view this is problematic, as the answer to the tasks to be solved *is* the model (see again Section 3.4).

On the other side, it suffices for our purpose to work with TBoxes that are quite simple and do not involve constructs that are notoriously difficult to handle (like the combination of inverse roles, transitive roles and number restrictions). This allows us to use the above rather simple "loop checking" technique, which is inspired by the blocking technique developed for an ABox/TBox reasoner in [15]. We are going to explain the difference now.

The mentioned blocking technique is realized within the completion rule **R∃C** in [15]. As in our approach, a new individual $f_{c,l,d}(a)$ and two assertions about that individual $l(x, f_{c,l,d}(a))$ and $d(f_{c,l,d}(a))$ may be introduced in the completion of an ABox. The completion is restricted to cases where all three conditions hold:

1. $(\exists l\,.\,d)(a)$ is already contained in the ABox.
2. if a is a new individual then it is not blocked by an already introduced new individual c. By definition, a is blocked by c if c is introduced prior to a and the set of concepts that include c is a superset of the concepts that include a.
3. there is no b such that $l(a,b)$ and $d(b)$ are contained in the current completion of the ABox. This gives priority to the re-use of an existing l-link over a new link.

Basically the same restriction is encoded in our translation. It allows to re-use the same ideas for the correctness proof of the blocking technique. There are two differences, however: first, we allow old individuals to act as blocking individuals. This simplifies the transformation somewhat and still works in our case, where we don't have transitive roles. Second, re-using an existing link because of a blocking situation is only implicit in the model construction in [15]. In our transformation this re-use made explicit by means of the rule (5). While this does not make a difference concerning satisfiability, it makes a difference regarding the model computed. Lacking this explicit re-use the computed models would in general not readily provide an answer to the tasks to be solved.

[6] Complexity being measured as the tree depth of the atoms.

3.4 Querying the Data

Existing query languages use path queries, that navigate along the structures of the XML data. For instance, in order to access the name of all researchers of a university in *XPath* [22] we use `/university/researchers/researcher/name`. Path queries usually allow some form of "abbreviation". For instance, with `//researcher/name` we address all descendants of the "root" that are *researcher*-elements and navigate to their names. However, because path queries work directly on the XML data and not on the schema, it is not possible to query those elements from a data source, that belong to the same type or concept. In particular, in order to ask e.g. for all kinds of publications, we would have to construct the union of path queries navigating to publication, book, article and monograph, explicitly.

This problem has been addressed in [14] where concepts or type expressions, respectively, have been added to the query language. Querying instances of types or concepts basically is well known in object oriented databases. Furthermore, path expressions allow to navigate through the nested structure of the data. We assume a syntax similar to that applied in object oriented databases [19]. We aim at a query language that combines schema expressions as used in object oriented query languages with a flexible navigation mechanism as given by "abbreviated" path expressions as e.g. provided by XPath.

Let A denote a set of attribute names.

Definition 5 (Query Syntax). *A* path term *is an expression of the form* $c[x]\ op_1\ a_1[x_1] \ldots op_m\ a_m[x_m]$*, where* $m \geq 0$*, c is a type name, a substitution group identifier (cf. Def. 1) or the symbol* $\top$*,* $op_i \in \{.,!\}$*,* $a_i \in A$*, and* x, x_i*, for* $i = 1, \ldots, m$ *are variables.*

A conjunctive query expression *is a conjunction of path terms* $p_1 \wedge \ldots \wedge p_n$*, where* $n \geq 1$*. A* disjunctive query expression *is a disjunction of conjunctive query expressions* $e_1 \vee \ldots \vee e_n$*, where* $n \geq 1$*.*

By simply a *query expression* we mean a disjunctive query expression, which includes the case of a conjunctive query expression as a disjunction with one element.

A path term is an expression that starts in a concept and navigates through a schema by a sequence of attributes. Variables are used to "hold" the spots during this navigation. At the instance level, a path term describes a set of paths in a data source. The result of a path term basically is a relationship, where all variables occuring in a path term are bound to elements in the XML document. Actually, there are two possibilities to traverse a schema. Firstly, by explicit navigation that specifies a path step by step. We use the "!" operator for this kind of navigation. Secondly, a path term may contain only some attributes occuring on one or several paths in a schema; then the "." operator is used.

For instance, $\mathsf{BOOK}[b]!\mathsf{title}[t]$ describes all instances b of type BOOK, with a title t as subelement. The result, basically, is a binary relation containing values for b and t. Compared to an XPath-expression, the "!" operator matches "/" and the "." operator can be compared to "//". Different from an XPath expression, in our query syntax variables can be specified and type expressions are possible. If a user does not want to specify an explicit type, the most general type $\top$ can be taken. As will be shown below (see Section 3.4), using type expressions in a query allows a user to query for different

elements described by the same general type in one simple expression. For instance, BOOK$[b]$!title$[t]$ retrieves all BOOK elements with their titles as well as all MONOGRAPH elements, because of the underlying specialization/subsumption relationship.

Furthermore, using type expressions in a query provide a query processor with a possibility to validate purely by means of the schema if a query is satisfiable. Consider another query:

$$\mathsf{PUBLICATION}[p].\mathsf{isbn}[i] \wedge \top[p].\mathsf{proceedings}[x]$$

This query asks for all instances p of PUBLICATION with their isbn and their proceedings. Actually, there are no such instances. This fact can be established automatically, as will be shown in the next section. This means it is useless to try this query on any concrete database satisfying the schema, as the result will be empty anyway.

Translating Queries to Rules. The translation of a path term p of the form $c[x]\ op_1\ a_1[x_1] \cdots op_m\ a_m[x_m]$, as introduced above, is the following list of atoms $tr(p)$:

$$\begin{array}{l} c(x), \\ tr_{sel}(x,\ op_1\ a_1[x_1]), \\ tr_{sel}(x_1,\ op_2\ a_2[x_2]), \ldots, tr_{sel}(x_{m-1},\ op_m\ a_m[x_m]) \ \ , \end{array}$$

where

$$tr_{sel}(x,\ op\ a[y]) = \begin{cases} a(x,y) & \text{if } op = \,! \\ \mathsf{role\_filler\_ref\_trans}(x,z),\ a(z,y) & \text{if } op = \,. \ \ , \\ & \text{where } z \text{ is a fresh variable} \end{cases}$$

The purpose of $tr(p)$ is to translate the path term p into a sequence of subgoals that correspond to traversing the schema as prescribed by p and thereby assigning values to the variables mentioned in p. Notice the translation distinguishes between the operators "." and "!". The former stands for the presence of an attribute immediately at the current point of the traversal and thus translates into a corresponding role filler subgoal. The latter is similar, but it allows to follow an arbitrary number of attributes first, by means of the role_filler_ref_trans relation.

(1) The translation of a conjunctive query expression $p_1 \wedge \ldots \wedge p_n$, where $n \geq 1$ and each p_i (for $i = 1,\ldots,n$) is of the form just mentioned and written as $p_i = c^i[x^i].a^i_1[x^i_1].\ldots.a^i_{m_i}[x^i_{m_i}]$ consists of the following rules:

$$\begin{array}{l} \mathsf{solution\_path}(x^1, (x^1_{m_1}, \ldots, x^n_{m_n}) \leftarrow \mathsf{role\_filler\_ref\_trans}(\mathsf{init}, x^1), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad tr(p_1), \ldots, tr(p_n) \end{array}$$

Observe that the selector x^1 mentioned in the first path term p_1, is treated in a special way. Its type c^1 is treated as the "start type", and a path from the root to it is computed in the first argument of the solution_path predicate.

The translation of a disjunctive conjunctive query expression $e_1 \vee \ldots \vee e_n$ is the union of the translation of each e_1, for $i = 1,\ldots,n$.

(2) The following rules constrain the admissible models to those that contain at least one "solution path":

$$\begin{aligned} \mathsf{false} &\leftarrow \text{ not } \mathsf{some\_solution\_path} \\ \mathsf{some\_solution\_path} &\leftarrow \mathsf{solution\_path}(x,y) \end{aligned}$$

(3) Let E_{rel} be a set of attribute names as mentioned in Definition 1. The set E_{rel} is reified by the set of rules

$$T(E_{\mathsf{rel}}) = \{\mathsf{role\_filler}(x,y) \leftarrow l(x,y) \mid l \text{ is the label of some attribute in } E_{\mathsf{rel}}\}$$

The reflexive-transitive closure of $T(E_{\mathsf{rel}})$ is obtained as follows:

$$\begin{aligned} \mathsf{role\_filler\_ref\_trans}(x,x) &\leftarrow \\ \mathsf{role\_filler\_ref\_trans}(x,z) &\leftarrow \mathsf{role\_filler}(x,y), \\ &\qquad \mathsf{role\_filler\_ref\_trans}(y,z) \end{aligned}$$

Now let $S = (C \cup SG, E_{\mathsf{rel}}, \cup E_{\mathsf{isa}} \cup E_{\mathsf{SG\text{-}in}} \cup E_{\mathsf{SG\text{-}out}}, r)$ be a schema as in Definition 1 and q a conjunctive query expression. The *transformation of S and q* consists of the union of $R(S)$ of Definition 4, the result of the transformation step (1) applied to q, the rules from (2), the rules from (3), including $T(E_{\mathsf{rel}})$, and the facts

$$\top(x) \leftarrow \qquad\qquad \mathsf{root}(\mathsf{init}) \leftarrow$$

Examples. The query expression $\mathsf{BOOK}[b]!\mathsf{title}[t]$ from above translates into the following program:

$$\top(x) \leftarrow \tag{1}$$

$$\mathsf{root}(\mathsf{init}) \leftarrow \tag{2}$$

$$\begin{aligned} \mathsf{solution\_path}(b,t) \leftarrow\ & \mathsf{role\_filler\_ref\_trans}(\mathsf{init},b), \\ & \mathsf{BOOK}(b), \mathsf{role\_filler\_ref\_trans}(b,z), \mathsf{title}(z,t) \end{aligned} \tag{3}$$

$$\mathsf{false} \leftarrow \text{ not } \mathsf{some\_solution\_path} \tag{4}$$

$$\mathsf{some\_solution\_path} \leftarrow \mathsf{solution\_path}(x,y) \tag{5}$$

$$\mathsf{role\_filler\_ref\_trans}(x,x) \leftarrow \tag{6}$$

$$\begin{aligned} \mathsf{role\_filler\_ref\_trans}(x,z) \leftarrow\ & \mathsf{role\_filler}(x,y), \\ & \mathsf{role\_filler\_ref\_trans}(y,z) \end{aligned} \tag{7}$$

$$\mathsf{role\_filler}(x,y) \leftarrow \mathsf{university}(x,y) \tag{8}$$

$$\vdots \qquad\qquad \vdots \tag{9}$$

$$\mathsf{role\_filler}(x,y) \leftarrow \mathsf{isbn}(x,y) \tag{10}$$

Notice the rule (3) is the translation of the given (single conjunct) query expression according to the scheme (1) above. The rules starting from (6) stem from the translation of the schema graph in Figure 2 according to the scheme (3).

Now, suppose that this rule set is combined with the translation of the schema graph in Figure 2 according to Definitions 2 and 3 (or Definition 4 instead). The unique model contains

$$\begin{aligned}&\mathsf{solution\_path}(f_{\mathsf{BOOKS,books,BOOK}}(f_{\mathsf{LIBRARY,book,BOOKS}}(\\&\quad f_{\mathsf{UNIVERSITY,library,LIBRARY}}(f_{\mathsf{ROOT,university,UNIVERSITY}}(\mathsf{init})))),\\&\quad f_{\mathsf{PUBLICATION,title,STRING}}(f_{\mathsf{BOOKS,books,BOOK}}(f_{\mathsf{LIBRARY,book,BOOKS}}(\\&\qquad\qquad f_{\mathsf{UNIVERSITY,library,LIBRARY}}(f_{\mathsf{ROOT,university,UNIVERSITY}}(\mathsf{init}))))))\end{aligned}$$

Observe that the path from the ROOT concept to the BOOK concept is coded in the names of the Skolem function symbols in the first argument of solution_path. This path is extended towards a path to the title attribute in the second argument; this extension encodes, in terms of the schema graph, moving from the BOOK type to its supertype PUBLICATION and then moving to the title attribute. We note that the query expression $\mathsf{BOOK}[b].\mathsf{title}[t]$ would have given the same result.

As a second example consider $\mathsf{PUBLICATION}[p].\mathsf{isbn}[i] \wedge \top[p].\mathsf{proceedings}[x]$ from above. Its translation according to scheme (1) is

$$\begin{aligned}\mathsf{solution\_path}(p,(i,x)) \leftarrow\ &\mathsf{role\_filler\_ref\_trans}(\mathsf{init},p),\\&\mathsf{PUBLICATION}(p),\mathsf{isbn}(p,i),\\&\top(p),\mathsf{proceedings}(p,x),\end{aligned} \quad . \tag{3}$$

The rest of the transformation is the same as in the previous examples and is omitted. This time, the solution_path relation is empty, as the body of rule (3) cannot be satisfied. But then, with rules (4) and (5) this rule set is unsatisfiable. This is the expected result, because, as mentioned above, the query expression is unsatisfiable (in terms of the schema graph).

Both examples run within milliseconds on the *KRHyper* system.

To sum up, our model based approach detects if a (XPath) query is satisfiable or not. In case of a satisfiable query, a fully completed path – one that fulfills all given constraints – is returned as part of the model. The purpose of this approach is to dramatically reduce the workload on the query processor. Without the path completion, a usual XML query processor would search the database for solutions to the query.

Beyond these results there is an additional benefit of the model based approach. In case of a query that allows for multiple paths through the schema to satisfy it, the enumeration of models provides an enumeration of these paths. Because all models are unique, no path will be computed twice. Due to the "loop check" implemented in the generated set of rules and the minimality of all models all solution paths will be acyclic. Because there is only a finite number of acyclic path in a finite schema, the number of enumerated models will be finite as well.

4 Conclusion

In this paper we aimed to demonstrate that automated deduction techniques, in particular those following the model computation paradigm, are very well suited for knowledge representation purposes. We showed how a XML schema graph can be easily represented by a simple description logic. This representation was then transformed into

the predicate logic language of the first order theorem prover *KRHyper*. Guaranteeing termination by a blocking technique is part of this transformation. Based on this logical representation of a schema graph, we used model computation capabilities of the *KRHyper* system to compute XML path queries, which are formulated in a much more flexible query language.

An obvious question concerns the correctness of our approach, i.e. termination, soundness and completeness of the combination of *KRHyper* and the transformed database schema and query. While termination has been argued for in Section 3.3, it is hardly possible to establish the other two properties: there is no formal semantics of XML path queries in the literature.

Our approach is leaving the mainstream of knowledge representation research, which currently has its focus on the development of description logic (DL) systems. We want to point out that we consider the DL direction of research extremely successful: it led to a deep insight into computational properties of decidable subclasses of first order reasoning; it made clear some interesting links to non-classical logics, and, moreover, DL systems are nowadays outperforming most modal logic theorem provers. Despite these successful developments we find two reasons which motivate our approach to use a first order theorem prover for knowledge representation purposes instead of dedicated description logic systems.

First, even the key researchers in the field of description logics are stating some severe deficiencies of their systems (e.g. [12]): research into description logics focused on algorithms for investigating properties of the terminologies, and it is clear that for realistic applications the query language of description logic systems is not powerful enough. Only recently has the community investigated the extension of description logic systems towards ABox and query answering, which is not trivial [18, 17].

Second, the most advanced systems are essentially confined to classical semantics and do not offer language constructs for non-monotonic features like default negation. Although there are some results on extending DL languages with nonmonotonic features [1, 11], it seems that this direction of research is vastly unexplored. Indeed, as our investigations demonstrate, there are applications for description logics where model computation and default negation is an issue. In a wider context, it can be speculated that *Semantic Web* applications would profit from knowledge representation languages having these features.

We understand this paper as a first step, that covers a basic schema/query reasoning task by model-based deduction. Due to the underlying expressive logic formalism we expect our approach to easily adapt to more sophisticated problem settings, like type hierarchies as they evolve within the XML world. One big advantage of such a declarative approach over, say, explicitly programmed algorithms is the possibility to easily add further constraints. We intend to explore this potential in future work.

References

1. F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. In B. Nebel, C. Rich, and W. Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proc. of the Third Int. Conf.*, pages 306–317, San Mateo, California, 1992. Morgan Kaufmann.

2. F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *Description Logic Handbook*, pages 47–100. Cambridge University Press, 2002.
3. P. Baumgartner, P. Fröhlich, U. Furbach, and W. Nejdl. Semantically Guided Theorem Proving for Diagnosis Applications. In M. E. Pollack, editor, *15th Int. Joint Conf. on Artificial Intelligence (IJCAI 97)*, pages 460–465, Nagoya, 1997. Morgan Kaufmann.
4. P. Baumgartner, U. Furbach, M. Gross-Hardt, and T. Kleemann. Optimizing the Evaluation of XPath using Description Logics. In *Proc. INAP2004, 15th Int. Conf. on Applications of Declarative Programming and Knowledge Management*, Potsdam, 2004.
5. P. Baumgartner, U. Furbach, M. Gross-Hardt, and A. Sinner. **'Living Book' :- 'Deduction', 'Slicing', 'Interaction'.** – system description. In F. Baader, editor, *CADE-19 – The 19th Int. Conf. on Automated Deduction*, LNAI. Springer, 2003.
6. P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, number 1126 in LNAI. European Workshop on Logic in AI, Springer, 1996.
7. W. Bibel and P. H. Schmitt, editors. *Automated Deduction. A basis for applications*. Kluwer Academic Publishers, 1998.
8. D. Calvanese, G. D. Giacomo, and M. Lenzerini. Answering queries using views in description logics. In *Proc. DL'99, Description Logic Workshop*, 1999.
9. D. Calvanese, G. D. Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: a description logic approach. *J. of Logic and Computation*, pages 295–318, 1999.
10. J. Dix, U. Furbach, and I. Niemelä. Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations. In A. Voronkov and A. Robinson, editors, *Handbook of Automated Reasoning*, pages 1121–1234. Elsevier-Science-Press, 2001.
11. F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *ACM Trans. on Computational Logic*, 3(2):177–225, 2002.
12. D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In *Proc. of the European Knowledge Acquisition Conf. (EKAW-2000)*, Lecture Notes In Artificial Intelligence. Springer-Verlag, 2000.
13. U. Furbach. Automated deduction. In W. Bibel and P. Schmitt, editors, *Automated Deduction. A Basis for Applications*, volume I: Foundations. Calculi and Refinements. Kluwer Academic Publishers, 1998.
14. M. Gross-Hardt. Querying concepts — an approach to retrieve xml data by means of their data types. In *17. WLP - Workshop Logische Programmierung*, Technical Report. Technische Universität Dresden, 2002.
15. V. Haarslev and R. Möller. Expressive ABox Reasoning with Number Restrictions, Role Hierarchies, and Transitively Closed Roles. In *KR2000: Principles of Knowledge Representation and Reasoning*, pages 273–284. Morgan Kaufmann, 2000.
16. V. Haarslev and R. Möller. RACER system description. *Lecture Notes in Computer Science*, 2083:701, 2001.
17. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic SHIQ. In D. MacAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE-17)*, Germany, 2000. Springer Verlag.
18. I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *AAAI'2000, Proc. 17th (U.S.) National Conf. on Artificial Intelligence*, pages 399–404. AAAI Press/The MIT Press, 2000.
19. M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *SIGMOD*, 1992.
20. R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In E. Lusk and R. Overbeek, editors, *Proc. of the 9^{th} Conf. on Automated Deduction, Argonne, Illinois, May 1988*, volume 310 of *LNCS*, pages 415–434. Springer, 1988.

21. C. Sakama. Possible Model Semantics for Disjunctive Databases. In W. Kim, J.-M. Nicholas, and S. Nishio, editors, *Proc. First Int. Conf. on Deductive and Object-Oriented Databases (DOOD-89)*, pages 337–351. Elsevier Science Publishers B.V. (North–Holland) Amsterdam, 1990.
22. W3C. XPath specification. http://www.w3.org/TR/xpath, 1999.
23. W3C. XQuery 1.0: An XML query language. http://www.w3.org/TR/xquery/, 2001.
24. C. Wernhard. System Description: KRHyper. Fachberichte Informatik 14–2003, Universität Koblenz-Landau, Institut für Informatik, 2003.

Available Research Reports (since 1999):

2004

5/2004 *Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt, Thomas Kleemann.* Model Based Deduction for Database Schema Reasoning.

4/2004 *Lutz Priese.* A Note on Recognizable Sets of Unranked and Unordered Trees.

3/2004 *Lutz Priese.* Petri Net DAG Languages and Regular Tree Languages with Synchronization.

2/2004 *Ulrich Furbach, Margret Groß-Hardt, Bernd Thomas, Tobias Weller, Alexander Wolf.* Issues Management: Erkennen und Beherrschen von kommunikativen Risiken und Chancen.

1/2004 *Andreas Winter, Carlo Simon.* Exchanging Business Process Models with GXL.

2003

18/2003 *Kurt Lautenbach.* Duality of Marked Place/Transition Nets.

17/2003 *Frieder Stolzenburg, Jan Murray, Karsten Sturm.* Multiagent Matching Algorithms With and Without Coach.

16/2003 *Peter Baumgartner, Paul A. Cairns, Michael Kohlhase, Erica Melis (Eds.).* Knowledge Representation and Automated Reasoning for E-Learning Systems.

15/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, Thomas Kleemann, Christoph Wernhard.* KRHyper Inside — Model Based Deduction in Applications.

14/2003 *Christoph Wernhard.* System Description: KRHyper.

13/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, Alex Sinner.* 'Living Book' :- 'Deduction', 'Slicing', 'Interaction'..

12/2003 *Heni Ben Amor, Oliver Obst, Jan Murray.* Fast, Neat and Under Control: Inverse Steering Behaviors for Physical Autonomous Agents.

11/2003 *Gerd Beuster, Thomas Kleemann, Bernd Thomas.* MIA - A Multi-Agent Location Based Information Systems for Mobile Users in 3G Networks.

10/2003 *Gerd Beuster, Ulrich Furbach, Margret Groß-Hardt, Bernd Thomas.* Automatic Classification for the Identification of Relationships in a Metadata Repository.

9/2003 *Nicholas Kushmerick, Bernd Thomas.* Adaptive information extraction: Core technologies for information agents.

8/2003 *Bernd Thomas.* Bottom-Up Learning of Logic Programs for Information Extraction from Hypertext Documents.

7/2003 *Ulrich Furbach.* AI - A Multiple Book Review.

6/2003 *Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt.* Living Books.

5/2003 *Oliver Obst.* Using Model-Based Diagnosis to Build Hypotheses about Spatial Environments.

4/2003 *Daniel Lohmann, Jürgen Ebert.* A Generalization of the Hyperspace Approach Using Meta-Models.

3/2003 *Marco Kögler, Oliver Obst.* Simulation League: The Next Generation.

2/2003 *Peter Baumgartner, Margret Groß-Hardt, Alex Sinner.* Living Book – Deduction, Slicing and Interaction.

1/2003 *Peter Baumgartner, Cesare Tinelli.* The Model Evolution Calculus.

2002

12/2002 *Kurt Lautenbach.* Logical Reasoning and Petri Nets.

11/2002 *Margret Groß-Hardt.* Processing of Concept Based Queries for XML Data.

10/2002 *Hanno Binder, Jérôme Diebold, Tobias Feldmann, Andreas Kern, David Polock, Dennis Reif, Stephan Schmidt, Frank Schmitt, Dieter Zöbel.* Fahrassistenzsystem zur Unterstützung beim Rückwärtsfahren mit einachsigen Gespannen.

9/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 4. Workshop Software Reengineering (Bad Honnef, 29./30. April 2002).

8/2002 *Richard C. Holt, Andreas Winter, Jingwei Wu.* Towards a Common Query Language for Reverse Engineering.

7/2002 *Jürgen Ebert, Bernt Kullbach, Volker Riediger, Andreas Winter.* GUPRO – Generic Understanding of Programs, An Overview.

6/2002 *Margret Groß-Hardt.* Concept based querying of semistructured data.

5/2002 *Anna Simon, Marianne Valerius.* User Requirements – Lessons Learned from a Computer Science Course.

4/2002 *Frieder Stolzenburg, Oliver Obst, Jan Murray.* Qualitative Velocity and Ball Interception.

3/2002 *Peter Baumgartner.* A First-Order Logic Davis-Putnam-Logemann-Loveland Procedure.

2/2002 *Peter Baumgartner, Ulrich Furbach.* Automated Deduction Techniques for the Management of Personalized Documents.

1/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 3. Workshop Software Reengineering (Bad Honnef, 10./11. Mai 2001).

2001

13/2001 *Annette Pook.* Schlussbericht "FUN - Funkunterrichtsnetzwerk".

12/2001 *Toshiaki Arai, Frieder Stolzenburg.* Multiagent Systems Specification by UML Statecharts Aiming at Intelligent Manufacturing.

11/2001 *Kurt Lautenbach.* Reproducibility of the Empty Marking.

10/2001 *Jan Murray.* Specifying Agents with UML in Robotic Soccer.

9/2001 *Andreas Winter.* Exchanging Graphs with GXL.

8/2001 *Marianne Valerius, Anna Simon.* Slicing Book Technology — eine neue Technik für eine neue Lehre?.

7/2001 *Bernt Kullbach, Volker Riediger.* Folding: An Approach to Enable Program Understanding of Preprocessed Languages.

6/2001 *Frieder Stolzenburg.* From the Specification of Multiagent Systems by Statecharts to their Formal Analysis by Model Checking.

5/2001 *Oliver Obst.* Specifying Rational Agents with Statecharts and Utility Functions.

4/2001 *Torsten Gipp, Jürgen Ebert.* Conceptual Modelling and Web Site Generation using Graph Technology.

3/2001 *Carlos I. Chesñevar, Jürgen Dix, Frieder Stolzenburg, Guillermo R. Simari.* Relating Defeasible and Normal Logic Programming through Transformation Properties.

2/2001 *Carola Lange, Harry M. Sneed, Andreas Winter.* Applying GUPRO to GEOS – A Case Study.

1/2001 *Pascal von Hutten, Stephan Philippi.* Modelling a concurrent ray-tracing algorithm using object-oriented Petri-Nets.

2000

8/2000 *Jürgen Ebert, Bernt Kullbach, Franz Lehner (Hrsg.).* 2. Workshop Software Reengineering (Bad Honnef, 11./12. Mai 2000).

7/2000 *Stephan Philippi.* AWPN 2000 - 7. Workshop Algorithmen und Werkzeuge für Petrinetze, Koblenz, 02.-03. Oktober 2000 .

6/2000 *Jan Murray, Oliver Obst, Frieder Stolzenburg.* Towards a Logical Approach for Soccer Agents Engineering.

5/2000 *Peter Baumgartner, Hantao Zhang (Eds.).* FTP 2000 – Third International Workshop on First-Order Theorem Proving, St Andrews, Scotland, July 2000.

4/2000 *Frieder Stolzenburg, Alejandro J. García, Carlos I. Chesñevar, Guillermo R. Simari.* Introducing Generalized Specificity in Logic Programming.

3/2000 *Ingar Uhe, Manfred Rosendahl.* Specification of Symbols and Implementation of Their Constraints in JKogge.

2/2000 *Peter Baumgartner, Fabio Massacci.* The Taming of the (X)OR.

1/2000 *Richard C. Holt, Andreas Winter, Andy Schürr.* GXL: Towards a Standard Exchange Format.

1999

10/99 *Jürgen Ebert, Luuk Groenewegen, Roger Süttenbach.* A Formalization of SOCCA.

9/99 *Hassan Diab, Ulrich Furbach, Hassan Tabbara.* On the Use of Fuzzy Techniques in Cache Memory Managament.

8/99 *Jens Woch, Friedbert Widmann.* Implementation of a Schema-TAG-Parser.

7/99 *Jürgen Ebert, and Bernt Kullbach, Franz Lehner (Hrsg.).* Workshop Software-Reengineering (Bad Honnef, 27./28. Mai 1999).

6/99 *Peter Baumgartner, Michael Kühn.* Abductive Coreference by Model Construction.

5/99 *Jürgen Ebert, Bernt Kullbach, Andreas Winter.* GraX – An Interchange Format for Reengineering Tools.

4/99 *Frieder Stolzenburg, Oliver Obst, Jan Murray, Björn Bremer.* Spatial Agents Implemented in a Logical Expressible Language.

3/99 *Kurt Lautenbach, Carlo Simon.* Erweiterte Zeitstempelnetze zur Modellierung hybrider Systeme.

2/99 *Frieder Stolzenburg.* Loop-Detection in Hyper-Tableaux by Powerful Model Generation.

1/99 *Peter Baumgartner, J.D. Horton, Bruce Spencer.* Merge Path Improvements for Minimal Model Hyper Tableaux.