UNIVERSITÄT
KOBLENZ · LANDAU
Institut für Informatik

FB 4
Informatik

Tableaux Between Proving, Projection and Compilation

Christoph Wernhard

Nr. 18/2007

Arbeitsberichte aus dem Fachbereich Informatik

Universität Koblenz-Landau • Campus Koblenz • Universitätsstraße 1 • 56070 Koblenz
Telefon +49 261 287-0 • http://www.uni-koblenz.de

Die Arbeitsberichte aus dem Fachbereich Informatik dienen der Darstellung vorläufiger Ergebnisse, die in der Regel noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar. Alle Rechte vorbehalten, insbesondere die der Übersetzung, des Nachdruckes, des Vortrags, der Entnahme von Abbildungen und Tabellen – auch bei nur auszugsweiser Verwertung.

The “Arbeitsberichte aus dem Fachbereich Informatik“comprise preliminary results which will usually be revised for subsequent publication. Critical comments are appreciated by the authors. All rights reserved. No part of this report may be reproduced by any means or translated.

Arbeitsberichte des Fachbereichs Informatik

ISSN (Print): 1864-0346
ISSN (Online): 1864-0850

Herausgeber / Edited by:

Der Dekan:
Prof. Dr. Paulus

Die Professoren des Fachbereichs:
Prof. Dr. Bátori, Jun.-Prof. Dr. Beckert, Prof. Dr. Burkhardt, Prof. Dr. Diller, Prof. Dr. Ebert, Prof. Dr. Furbach, Prof. Dr. Grimm, Prof. Dr. Hampe, Prof. Dr. Harbusch, Jun.-Prof. Dr. Hass, Prof. Dr. Krause, Prof. Dr. Lautenbach, Prof. Dr. Müller, Prof. Dr. Oppermann, Prof. Dr. Paulus, Prof. Dr. Priese, Prof. Dr. Rosendahl, Prof. Dr. Schubert, Prof. Dr. Staab, Prof. Dr. Steigner, Prof. Dr. Troitzsch, Prof. Dr. von Kortzfleisch, Prof. Dr. Walsh, Prof. Dr. Wimmer, Prof. Dr. Zöbel

Kontaktdaten der Verfasser

Christoph Wernhard
Institut für Informatik
Fachbereich Informatik
Universität Koblenz-Landau
Universitätsstraße 1
D-56070 Koblenz
EMail: wernhard@uni-koblenz.de

Tableaux Between Proving, Projection and Compilation

Christoph Wernhard

Institut für Informatik, Universität Koblenz-Landau, D-56070 Koblenz, Germany,
wernhard@uni-koblenz.de

Abstract. Generalized methods for automated theorem proving can be used to compute formula transformations such as projection elimination and knowledge compilation. We present a framework based on clausal tableaux suited for such tasks. These tableaux are characterized independently of particular construction methods, but important features of empirically successful methods are taken into account, especially dependency directed backjumping and branch local operation. As an instance of that framework an adaption of DPLL is described. We show that knowledge compilation methods can be essentially improved by weaving projection elimination partially into the compilation phase.

1 Introduction

Projection elimination can be used to compute for a given formula and a set of atoms a formula which is equivalent to the first one, as far as the atoms in the given set are concerned, but does not express anything about other atoms. Projection elimination has applications in various areas of knowledge representation [11, 18, 16, 6, 17, 25, 20]. Many of the automated deduction systems that have recently been successful in applications are based on tableau methods. This includes SAT-solvers, description logic reasoners as well as systems that compute models or answer sets. Our aim is to employ such techniques to the computation of projection elimination and to knowledge compilation tasks which are closely related. We develop a notion of tableaux suited for such tasks. It is is independent of the method used for tableau construction, but important features of empirically successful methods are taken into account. This includes the possibility of destructive tableau modification, as required for dependency directed backjumping. Another feature is the proceeding with a single branch in memory which can be utilized to store the overall compilation result offline or to pass parts of the overall result to a client application as soon as they are computed. The tableau framework can be instantiated by an adaption of the Davis Putnam Logemann Loveland procedure (*DPLL*) [9] for projection elimination and knowledge compilation. Some applications of knowledge compilation also involve projection elimination [6]. We show that for those applications the efficiency of the compilation procedure can be essentially improved by weaving projection elimination partially into the compilation process, instead of performing projection elimination just on the compilation result.

2 Notation and Preliminaries

2.1 Basic Notions

We consider propositional formulas in negation normal form, constructed from literals over a denumerable set of atoms and the truth value constants $\top$ and $\bot$ with $\wedge$ and $\vee$ as binary operators. We write the positive (negative) literal with atom A as A ($\neg A$), and also, in contexts where the positive literal should be distinguished from the atom, as $+A$ ($-A$). The complement of a literal L is written as $\widetilde{L}$. If S is a set of literals then $\widetilde{S}$ is set of complements of the members of S and $\overline{S}$ is the set of all literals not in S. In certain contexts we call a set of literals a *literal scope*.

An interpretation is a set of literals containing for each atom A exactly one of $+A$ or $-A$. It represents the truth value assignment that maps the atoms of its positive (negative) members to $\top$ ($\bot$). An interpretation I is a model of the formula F, in symbols $I \models F$, if and only if F is true under the truth value assignment represented by I. A formula F_1 entails a formula F_2, in symbols $F_1 \models F_2$, if and only if for all interpretations I it holds that if $I \models F_1$ then $I \models F_2$. Two formulas F_1 and F_2 are equivalent, in symbols $F_1 \equiv F_2$, if and only if $F_1 \models F_2$ and $F_2 \models F_1$.

Unless specified otherwise, variables I, F, L, A and S, also with sub- and superscripts, range over interpretations, formulas, literals, atoms and literal scopes respectively.

2.2 Essential Literal Signature

We use three variations of the notion of signature of a formula F: $\mathcal{A}(F)$ is the set of atoms in F. $\mathcal{L}(F)$ is the set of literals in F. $\mathcal{L}_{\mathcal{E}}(F)$, the *essential literal signature* of F, is the unique smallest set of literals from which an equivalent to F can be constructed. $\mathcal{L}_{\mathcal{E}}(F)$ can be defined in a semantical way as the set of literals L for which there exists an interpretation I such that $I \models F$ and $(I - \{L\}) \cup \{\widetilde{L}\} \not\models F$. See e. g. [17] for properties and examples ($\mathcal{L}_{\mathcal{E}}$ corresponds to *DepLit* there). A related notion for clausal first order formulas (but just for atoms instead of literals) has been introduced in [25].

2.3 Projection and Forgetting for Literal Scopes

The syntax of formulas can be extended by an operator project which takes a formula and a specifier of a literal scope as arguments. (We however continue to use *formula* as specified in Sect. 2.1 and mention explicitly when the project operator is allowed.) Intuitively, project(F, S), *the projection of* formula F *onto* literal scope S is a formula which is equivalent to F relative to the literals in S, but does not express anything about other literals. Accordingly, the semantics of project operator can be defined as follows:

Definition 1 (Projection) For all formulas F (which may contain the project operator), literal scopes S and interpretations I it holds that

$$I \models \mathsf{project}(F, S) \text{ if and only if there exists an interpretation } I' \text{ such that } I' \models F \text{ and } I' \cap S \subseteq I.$$

In some contexts not the set of literals about which knowledge is retained, but its complement, the set of literals about which knowledge is "forgotten" is in the focus of interest. Thus it is convenient to introduce an additional operator for that, $\mathsf{forget}(F, S)$, the *forgetting in F about S*, that can be defined in terms of projection as

$$\mathsf{forget}(F, S) \stackrel{\text{def}}{=} \mathsf{project}(F, \overline{S}).$$

A specialization of projection onto literal scopes is projection onto *atom scopes*, sets of atoms, or equivalently onto sets of literals S such that $S = \widetilde{S}$. A definition of projection onto atom scopes for clausal first order logic was given in [25]. For propositional logic, the forgetting in a formula F about a set of atoms $\{p_1, ..., p_n\}$ corresponds to existential Boolean quantification: $\mathsf{forget}(F, \{p_1, ..., p_n\})$ is equivalent to $\exists p_1 .... \exists p_n F$, where in case of the quantification the symbols $p_1, ..., p_n$ are considered as Boolean variables instead of atoms.

Forgetting about *literal* scopes has been described previously in [17]. Our Definition 1 corresponds to the characterization of Proposition 15 in that work. In Sect. 8.1 we will discuss further related works and concepts.

Example 1 (Projection/Forgetting)

$$\mathsf{forget}((\neg p \vee q) \wedge (\neg q \vee r),\ \{+q, -q\}) \equiv \neg p \vee r.$$
$$\mathsf{forget}((\neg p \vee q) \wedge (\neg q \vee r),\ \{-q\}) \equiv (\neg p \vee q) \wedge (\neg p \vee r).$$
$$\mathsf{forget}(p \wedge q, \{+q\}) \equiv p.$$
$$\mathsf{forget}(p \vee q, \{+q\}) \equiv \top.$$

Entailment and equivalence *relative to a scope* are convenient notions defined for formulas F_1, F_2 (which may contain the project operator) and literal or atom scopes S as follows:

$$F_1 \models_S F_2 \quad \text{if and only if} \quad \mathsf{project}(F_1, S) \models \mathsf{project}(F_2, S);$$
$$F_1 \equiv_S F_2 \quad \text{if and only if} \quad \mathsf{project}(F_1, S) \equiv \mathsf{project}(F_2, S).$$

2.4 Properties of Projection

It can be shown that for all formulas of propositional logic extended by the project operator there exists an equivalent formula without that operator. We call the computation of such equivalents *projection elimination*, analogously to quantifier elimination. The following properties hold for all formulas F, F_1, F_2 (which may contain the project) operator, and literal scopes S.

A central property of projection that underlies applications of projection elimination in knowledge representation is that the projection of a knowledge base onto S can be used to answer queries in S:

$$F_1 \models F_2 \text{ if and only if } \mathsf{project}(F_1, \mathcal{L}_\mathcal{E}(F_2)) \models F_2.$$

Projection is independent of the syntactic structure of its formula argument:

$$\text{If } F_1 \equiv F_2 \text{ then } \mathsf{project}(F_1, S) \equiv \mathsf{project}(F_2, S).$$

The essential literal signature of a projection is a subset of its scope:

$$\mathcal{L}_\mathcal{E}(\mathsf{project}(F, S)) \subseteq S.$$

Since we have characterized projection in Definition 1 just semantically, no analogous statement can be made for the syntactic literal signature and it is permitted that a formula obtained as result of projection elimination does contain literals not in the scope of the projection. For scopes S such that $S = \widetilde{S}$ (i. e. that correspond to atom scopes), this differentiation can be neglected, since if neither $+A$ nor $-A$ are in the essential literal signature of a formula, then all occurrences of A can be eliminated by substitution with a truth value constant. The algorithms for projection elimination that we consider later on include syntactic elimination of literals not in the projection scope, also for literal scopes that do not correspond to atom scopes.

Eliminating projection to $\emptyset$ can be used to decide satisfiability:

$$F \text{ is satisfiable if and only if } \mathsf{project}(F, \emptyset) \equiv \top.$$

2.5 Elimination of Projection by Expansion

Define $F[A\backslash W]$ as formula F with atom A substituted by $W \in \{\top, \bot\}$. If F is a formula then

$$\mathsf{forget}(F, \{L\}) \equiv F[A\backslash W] \vee (\widetilde{L} \wedge F), \tag{i}$$

where A is the atom of L and $W = \top$ ($\bot$) if L is positive (negative). Since $\mathsf{forget}(F, \{L\} \cup S) \equiv \mathsf{forget}(\mathsf{forget}(F, \{L\}), S)$, rewriting of subformulas with the project and forget operator according to equivalence (i) provides an algorithm for projection elimination.

If S is an atom scope, projection elimination can of course be based on equivalence (i), but also more compactly on the following equivalence:

$$\mathsf{forget}(F, \{A\}) \equiv F[A\backslash\top] \vee F[A\backslash\bot]. \tag{ii}$$

Equivalences (i) and (ii) can be used to show that — conversely to the outline in Sect. 2.3 — projection onto literal scopes can be defined in terms of projection onto atom scopes:

$$\begin{aligned} \mathsf{forget}(F, \{+A\}) &\equiv \\ F[A\backslash\top] \vee (\neg A \wedge F) &\equiv \\ \mathsf{forget}(F \wedge A, \{A\}) \vee (\neg A \wedge F). & \end{aligned}$$

2.6 Projection Elimination by Resolution

If F is a formula in tautology-free[1] conjunctive normal form then a refinement of the expansion method can be used to eliminate the forget operator in $\mathsf{forget}(F, \{A\})$. It works as follows: Return the union of the set of all non-tautological binary resolvents of clauses in F upon literals with atom A with the set of clauses in F that do not contain a literal with atom A. To eliminate the forgetting of a *literal*, also the clauses of F which contain its complement have to be included in the result. Iterated over $\mathcal{A}(F)$ (along with simplifications) the method for atom scopes is the original Davis Putnam procedure [10]. *SCAN* [11] and *replace-by-resolvents* [25] lift this method to first order clauses.

2.7 Simplifications Preserving Equivalence Relative to a Scope

Restricted to certain special cases, the resolution method to eliminate projection onto atom scopes properly reduces the number of clauses. The method is then a formula simplification, which preserves equivalence relative to atom scopes not containing literals upon which resolvents are built. Examples of these special cases are the well known purity and *ISOL* [3] simplifications, where atoms of literals to resolve upon occur only in a single polarity or in each polarity only once. The variation of the resolution method for *literal* forgetting can be used for formula simplifications that preserve equivalence relative to *literal* scopes.

2.8 Projection Elimination and Linklessness

The following equivalences are not hard to verify from the definition of project:

$$\mathsf{project}(F_1 \vee F_2, S) \equiv \mathsf{project}(F, S) \vee \mathsf{project}(F_2, S), \tag{iii}$$

$$\mathsf{project}(L, S) \equiv L \qquad \text{if } L \text{ is a literal in } S, \tag{iv}$$

$$\mathsf{project}(L, S) \equiv \top \qquad \text{if } L \text{ is a literal not in } S. \tag{v}$$

Equivalences (iii) – (iv) suggest to perform projection elimination in linear time by pushing the project operation inward until only literals appear as argument formulas, which are either retained according to (iv) or substituted by $\top$ according to (v). However, an equivalence for the conjunction operator is still missing.

The analogy to equivalence (iii) for conjunction does not hold unrestricted. We now show that it holds when certain preconditions are met. This is the basis for applying tableau methods to projection elimination, since they can be used to compute formulas meeting these preconditions.

Definition 2 (Linkless) If F, F_1, F_2 are formulas and S is a literal scope then

(i) F_1 and F_2 are *linkless outside* S if and only if

$$\mathcal{L}(F_1) \cap \widetilde{\mathcal{L}(F_2)} \subseteq S \cap \widetilde{S};$$

[1] No clause contains a literal and its complement.

(ii) F_1 and F_2 are *essentially linkless outside* S if and only if

$$\mathcal{L}_{\mathcal{E}}(F_1) \cap \widetilde{\mathcal{L}_{\mathcal{E}}(F_2)} \subseteq S \cap \widetilde{S};$$

(iii) F is *linkless outside* S if and only if for all subformulas of F which have the form $F_1 \wedge F_2$ it holds that F_1 and F_2 are linkless outside S.

(iv) F is *linkless inside* S if and only if it is linkless outside $\overline{S}$;

(v) F is *fully linkless*, or just *linkless*, if and only if F is linkless outside $\emptyset$.

If F_1 and F_2 are linkless outside S, then they clearly are also *essentially* linkless outside S. F_1 *and* F_2 *are linkless outside* $\overline{S}$ — the ternary relation underlying *linkless inside* — is equivalent to $\mathcal{L}(F_1) \cap \widetilde{\mathcal{L}(F_2)\}} \cap (S \cup \widetilde{S}) = \emptyset$.

Theorem 1 (Conjoining Projections) *For all formulas F_1, F_2 and literal scopes S such that F_1 and F_2 are essentially linkless outside S it holds that*

$$\mathsf{project}(F_1, S) \wedge \mathsf{project}(F_2, S) \equiv \mathsf{project}(F_1 \wedge F_2, S).$$

The right to left direction of this equivalence follows easily from the definition of projection. A proof of the left to right direction can be found in the appendix.

From equivalences (iii) – (iv) and Theorem 1 follows that projection elimination can be performed in linear time on linkless formulas by just substituting literals not in the scope with $\top$:

Theorem 2 *If F is a formula that is linkless inside a literal scope S then*

$$\mathsf{forget}(F, S) \equiv F[\![S \backslash \top]\!]$$

where $F[\![S \backslash \top]\!]$ is F with all literals that are in S substituted by $\top$.

3 A Tableau Framework for Projection Elimination and Knowledge Compilation

3.1 An Integrating View on Projection Elimination, Compilation and Deciding Satisfiability

We consider tableau methods that compute for a given formula F and two literal scopes S_l and S_u a formula F' such that

1. F' is linkless outside S_l,
2. $F' \models_{S_u} F$, and
3. $F \models F'$.

Such a method can be applied to various tasks:

– *Projection elimination.* The project operator in $\mathsf{project}(F', S_l)$ can be eliminated in linear time by substitution with $\top$ according to Theorem 2. Since $\mathsf{project}(F', S_u) \equiv \mathsf{project}(F, S_u)$, if the same literal scope S is used as S_l as well as S_u then the computation of F' followed by substitution of literals not in S with $\top$ is a method to eliminate the project operator in $\mathsf{project}(F, S)$.
– *Compilation to an equivalent linkless outside a given literal scope.* Let S_l be the given literal scope and S_u be the set of all literals. F' is then linkless outside S_l and equivalent to F.
– *Compilation to a fully linkless equivalent.* This is an instance of the previous case where S_l is the empty set.
– *Deciding satisfiability.* Let S_l be the set of all literals. Eliminating the project operator in $\mathsf{project}(F', S_l)$ in linear time according to Theorem 2 (along with further well-known simplifications that eliminate truth value constants as proper subformulas) yields either $\top$ or $\bot$. S_u can be any set of literals, including the empty set. F is then satisfiable if and only if the simplified F' is $\top$.

3.2 Fwd-Tableaux and Leafy Formulas

We are interested in algorithms that are based on theorem proving methods with clausal tableaux and (considered as a special case of them) semantic trees. We use a tableau data structure called *fwd-tableau*, short for *tableau with forward labels*, which extends the standard notion of clausal tableau in two respects relevant for computing formula transformations:

– *Forward labels.* Aside from the labeling by a literal, a fwd-tableau node has a second label, the *forward label*, which is a formula. Intuitively the forward label represents a part of the computation problem which is associated with the node while being in the focus of computation and remains to be solved (i.e. will possibly be solved at a future "*forward*" point in time). For computing formula transformations the forward labels of leaf nodes at the terminal state of a computation can be included into the output.
– *And-nodes.* Along with nodes in the standard sense for clausal tableaux, fwd-tableaux can contain nodes of a second type, called *a-nodes*, short for *and-nodes*. A fwd-tableau can then be considered as an and-or tree. And-nodes have been introduced to DPLL-based model counters [1], where they make it possible just to multiply the numbers of models counted for subformulas which are independent from each other in a certain way. And-nodes have subsequently been used in a compiler to DNNF (*decomposable negation normal form*) which is based on such a model counter [14].

Definition 3 (Fwd-Tableau) An *fwd-tableau* (short for *forward labeled tableau*) is an ordered tree with three node labeling functions: If N is a node in an fwd-tableau then

- its *literal label*, in symbols L_N, is a literal or the truth value constant $\top$.
- its *forward label*, in symbols F_N, is a formula, and
- if N is a non-leaf node, then it is either an *o-node* (short for *or-node*) or *a-node* (short for *and-node*). A leaf node is neither an o-node nor an a-node.

Unless especially noted, we only consider fwd-tableaux with a finite number of nodes.

Similar to a clausal tableau, a fwd-tableau can be considered as representation of a formula. For transformation tasks, a mapping of tableaux to formulas is convenient, which differs from the standard way, in that the forward labels of the leaf nodes are also included. The formula associated in this way with a fwd-tableau is called its *leafy formula*.

Definition 4 (Leafy Formula)

(i) If N is a node in a fwd-tableau then the *leafy formula* of N, in symbols $\mathsf{leafy}(N)$, is a formula defined as: If N is a leaf node then

$$\mathsf{leafy}(N) \stackrel{\text{def}}{=} L_N \wedge F_N;$$

if N is an o-node whose children are $N_1, ..., N_n$ then

$$\mathsf{leafy}(N) \stackrel{\text{def}}{=} L_N \wedge \bigvee_{i \in 1..n} \mathsf{leafy}(N_i);$$

if N is an a-node whose children are $N_1, ..., N_n$ then

$$\mathsf{leafy}(N) \stackrel{\text{def}}{=} L_N \wedge \bigwedge_{i \in 1..n} \mathsf{leafy}(N_i).$$

(ii) If T is a fwd-tableau then the *leafy formula* of T, in symbols $\mathsf{leafy}(T)$, is the leafy formula of the root node of T.

We now consider methods for projection elimination and knowledge compilation which work by constructing a fwd-tableau whose leafy formula satisfies the three conditions of Sect. 3.1 in the role of F'.

We do no commit to a particular tableau construction method, but instead present a set of constraints on fwd-tableau that essentially relate children and parent nodes and can be verified as invariants for a concrete calculus by showing that they are preserved by its rules.

This framework can be instantiated by variants of tableau calculi which include refinements important for practical success, such as space efficiency by working on a single branch at a time and dependency directed backtracking. In Sect. 5 we outline this for a variant of DPLL. Examples for the concepts introduced in this section are shown in Sect. 7.

3.3 Notation for Branches

Definition 5 (Root, Branch) If T is a fwd-tableau and N is a node in T then:

(i) $\mathsf{root}(T)$ denotes the root of T.

(ii) The *branch in T to N*, in symbols B_N, is the sequence of nodes defined as:

$$B_N \stackrel{\text{def}}{=} N_1 ... N_n$$

where $N_1 = \mathsf{root}(T)$, $N_n = N$ and for all N_i, N_{i+1}, $i \in 1..n-1$, N_i is the parent of N_{i+1}. For the symbolic notation "B_N" it is assumed that T is clear from the context.

(iii) We overload notation for branches: If $B_N = N_1 ... N_n$ is the branch in T to N, then B_N is also used to denote the formula defined as:

$$B_N \stackrel{\text{def}}{=} L_{N_1} \wedge ... \wedge L_{N_n}.$$

(iv) A *branch of T* is a branch in T to a leaf node.

3.4 Ensuring Linklessness of the Leafy Formula

A constraint, *linklessness-preserving*, and a property, PB-L, for determining that the construction of branch of a tableau is complete are defined such that if T is a fwd-tableau that is linklessness-preserving for a literal scope S and all leaf nodes of T satisfy $\mathsf{PB\text{-}L}_S$ then

$$\mathsf{leafy}(T) \textit{ is linkless outside } S.$$

Definition 6 (Linklessness-Preserving) A fwd-tableau is called *linklessness-preserving* for a literal scope S if and only if

(i) if N is an a-node in the fwd-tableau whose children are $N_1, ..., N_n$ then for all $i, j \in 1..n$ such that $i \neq j$ it holds that

$$(L_{N_i} \wedge F_{N_i}) \textit{ and } (L_{N_j} \wedge F_{N_j}) \textit{ are linkless outside } S,$$

(ii) if N is a node in the fwd-tableau which has an a-node as ancestor and N' is a child of N then

$$\mathcal{L}(L_{N'} \wedge F_{N'}) \subseteq \mathcal{L}(F_N) \cup (S \cap \widetilde{S}).$$

Linklessness-preserving is a constraint which is parameterized with a literal scope S. Nodes of a fwd-tableau without a-nodes trivially satisfy this constraint. The first condition of *linklessness-preserving* states that the literal and forward labels of different children of an a-node are pairwise linkless outside S. The second condition ensures that the literal signature of literal labels and forward labels of nodes below such a child is a subset of that of the child's forward label. This implies that also the leafy formulas of different children of an a-node are pairwise linkless outside S. The condition of Definition 6.ii does not constrain the literal signature of forward labels and literal labels with respect to literals in $S \cap \widetilde{S}$, thus if S is the set of all literals then it is trivially satisfied.

Definition 7 (Branch Completeness Property PB-L) If N is a node in a fwd-tableau and S is a literal scope then N satisfies the property $\mathsf{PB\text{-}L}_S$ if and only if

$$(B_N \wedge F_N) \text{ is linkless outside } S.$$

PB-L is a property that applies to a node and is parameterized with a literal scope. Since it is used as criterion for determining that the construction of branch of a tableau is complete it is called a *branch completeness property*.

3.5 Ensuring that the Leafy Formula Entails the Source Relative to a Scope

A constraint on fwd-tableaux, *upward-preserving*, which is parameterized with a literal scope, is defined, such that if T is a fwd-tableau that is upward-preserving for a literal scope S then

$$\mathsf{leafy}(T) \models_S F_{\mathsf{root}(T)}.$$

Definition 8 (Upward-Preserving) A fwd-tableau is called *upward-preserving for* a literal scope S if and only if for all nodes N in the fwd-tableau whose children are $N_1, ..., N_n$ it holds that

(i) if N is an o-node then

$$\bigvee_{i \in 1..n} (B_{N_i} \wedge \mathsf{project}(F_{N_i}, S)) \models \mathsf{project}(F_N, S),$$

(ii) if N is an a-node then

$$\bigwedge_{i \in 1..n} (B_{N_i} \wedge \mathsf{project}(F_{N_i}, S)) \models \mathsf{project}(F_N, S).$$

3.6 Ensuring that the Leafy Formula is Entailed by the Source

Two constraints on fwd-tableaux, *extensional* and *branch implied forward labels*, are defined, such that if T is a fwd-tableau such that

- T is extensional,
- $L_{\mathsf{root}(T)} = \top$, and
- T has branch implied forward labels

then

$$F_{\mathsf{root}(T)} \models \mathsf{leafy}(T).$$

Definition 9 (Extensional) A fwd-tableau T is called *extensional* if and only if for all nodes N in the fwd-tableau whose children are $N_1, ..., N_n$ it holds that

(i) if N is an o-node then

$$F_{\mathsf{root}(T)} \wedge B_N \models \bigvee_{i \in 1..n} L_{N_i},$$

(ii) if N is an a-node then

$$F_{\mathsf{root}(T)} \wedge B_N \models \bigwedge_{i \in 1..n} L_{N_i}.$$

Definition 10 (Branch Implied Forward Labels) A fwd-tableau is said to have *branch implied forward labels* if and only if for all nodes N, N' in the tableau such that N' is a child of N it holds that

$$F_N \wedge B_{N'} \models F_{N'}.$$

The *extensional* constraint only affects the *literal labels* of the tableau in relation to the forward label of the root of the tableau. As we will see in below, many familiar tableau methods for theorem proving construct tableaux which have the *extensional* property for the input formula.

The second property, *branch implied forward labels*, constrains the forward label of nodes such that they have to be implied by the branch to the node and the forward label of the parent node. Also many familiar tableau methods construct tableaux with this property. This is particularly straightforward to see for variants of the DPLL method for propositional logic where the forward label F_N of a node N is considered as obtained from the forward label of the parent node by substituting all atoms that appear in B_N with truth values according to their polarity in B_N.

The property of having *branch implied forward labels* entails the following similar property, which relates the formula label of a node to that of the tableau root instead of its parent: If N is a node in a fwd-tableau T with branch implied forward labels then

$$F_{\mathsf{root}(T)} \wedge B_N \models F_N.$$

Construction of Extensional Fwd-Tableaux. It is easy to see, that many of the familiar rules for the construction of clausal tableaux to show unsatisfiability of an input formula preserve the *extensional* property for the input formula. Examples include attaching to a leaf node a "cut" (two children with complementary literal labels), a clause from the input formula (for each literal in the clause a child labeled with that literal), or a single child that has been inferred by unit propagation from the input formula and the branch to the leaf node.

We specify abstract tableau construction operations, *Extension*, *A-Extension* and *Truncation*, which preserve the *extensional* property: If T, T' are fwd-tableaux, T is extensional and T' is obtained from T by applying an *Extension*, *A-Extension* or *Truncation* step then

$$T' \textit{ is extensional.}$$

In Definition 11 we make use of operational metaphors and describe tableau construction operations as tree modifications. These operations can obviously be understood declaratively as specifications of mappings between fwd-tableaux.

Definition 11 (Extension and Truncation) A fwd-tableau T' is obtained from a fwd-tableau T by

(i) *Extension* if and only if T' is obtained from T by labeling a leaf node N in T as an o-node and attaching children $N_1, ..., N_n$, $n \geq 1$, to N such that

$$F_{\mathsf{root}(T)} \wedge B_N \models \bigvee_{i \in 1..n} L_{N_i};$$

(ii) *A-Extension for* if and only if T' is obtained from T by labeling a leaf node N in T as an a-node and attaching children $N_1, ..., N_n$, $n \geq 1$, to N such that

$$F_{\mathsf{root}(T)} \wedge B_N \models \bigwedge_{i \in 1..n} L_{N_i};$$

(iii) *Truncation*, if and only if T' is obtained from T by removing all children from some o-node (this node is then no longer labeled as o-node).

A concrete method then can be shown to construct extensional tableaux by proving that each of its rules performs instances of these abstract operations. It is easy to see that the familiar rules mentioned above are instances of *Extension*. *Truncation* can be used to model tableau construction techniques which involve destructive manipulation of the tableau under construction and seem essential for practical success. Especially certain kinds of dependency directed backtracking can be understood as a *Truncation* step followed by an *Extension* step. The *A-Extension* operation is included mainly to allow handling of a-nodes analogously to o-nodes. If all a-nodes have children with literal label $\top$, which seems to be appropriate for the envisioned applications of a-nodes, the condition of Definition 11.ii is trivially satisfied.

In the specification of rule preconditions it can be convenient to refer to the "local" forward label of a leaf node instead of "globally" to the forward label of the root node. The specification of the *Extension* operation refers to the forward label of the root node. For fwd-tableaux with branch implied formula labels also operations which instead refer to the forward label of a leaf node are instances of *Extension*. It can be shown that if T is a fwd-tableau with branch implied forward labels and T' is obtained from T by labeling a leaf node N in T as an o-node and attaching children $N_1, ..., N_n$, $n \geq 1$, to N such that

$$B_N \wedge F_N \models \bigvee_{i \in 1..n} L_{N_i};$$

then T' is obtained from T by *Extension*.

3.7 Tableaux for Projection Elimination and Compilation

The following theorem combines the constraints defined in Sect. 3.4 – 3.6 to show that if a fwd-tableau satisfies them, then the leafy formula of the fwd-tableau satisfies the three conditions of Sect. 3.1. Thus, a method that constructs

a tableau satisfying the constraints of Sect. 3.4 – 3.6 can be used for tasks such as projection elimination and knowledge compilation.

Theorem 3 (Tableaux for Projection Elimination and Compilation) *If F is a formula, S_l, S_u are literal scopes and T is a fwd-tableau such that*

1. *T is linklessness-preserving for S_l,*
2. *all leaf nodes of T satisfy $\mathsf{PB\text{-}L}_{S_l}$,*
3. *T is upward-preserving for S_u,*
4. *$F_{\mathsf{root}(T)} \models_{S_u} F$,*
5. *T is extensional,*
6. *$L_{\mathsf{root}(T)} = \top$,*
7. *T has branch implied forward labels,*
8. *$F \models F_{\mathsf{root}(T)}$*

then

1. *$\mathsf{leafy}(T)$ is linkless outside S_l,*
2. *$\mathsf{leafy}(T) \models_{S_u} F$, and*
3. *$F \models \mathsf{leafy}(T)$.*

With respect to a tableau construction method, conditions 4., 6. and 8. typically concern the initialization phase: For a given input formula F and specification of a literal scope S_u, an initial tableau consisting of a single root node with literal label $\top$ and a forward label that satisfies 4. and 6. complies with the framework. The remaining conditions with exception of 2., i. e. conditions 1., 3., 5., and 7., are typically invariants of the method, holding in all stages of the tableau construction. Condition 2. indicates when the construction of a branch is complete. Additional tools to ensure condition 5. are provided with the abstract operations *Extension* and *Truncation*.

4 Weaving Projection Elimination and Compilation

The projection to a subsignature that is relevant for an application can be an important means to compensate for the size blow-up in knowledge compilation. For example in the application of knowledge compilation to diagnosis described in [6], a given formula is compiled into an equivalent to a projection of it. Since the elimination of projection is a linear operation in knowledge compilation target formats such as DNNF and fully linkless formulas, procedures that perform compilation into an equivalent to a projection are often described as compiling into an equivalent followed by projection elimination [6, 21]. This involves construction of an intermediate formula in the compilation target format that is equivalent to the input formula. When however the *efficiency of the compilation procedure* is taken into consideration, this construction of an intermediate equivalent can be an unnecessary step, whose avoidance by weaving projection elimination partially into the compilation procedure effects essential savings in time and space requirements.

We show this by giving a family of classes of formulas, *disjoin-formulas*, for which polynomial time methods exist that perform compilation into a linkless equivalent to a projection and, on the other hand, the size of an equivalent compilation target is not polynomially bounded under the assumption $\mathsf{NP} \not\subseteq \mathsf{P/poly}$, which is considered as very likely in complexity theory.[2]

Definition 12 (Disjoin-Formula) The *disjoin-formula* over a formula $F = C_1 \wedge ... \wedge C_n$ in conjunctive normal form and a ground atom A, such that $A \notin \mathcal{A}(F)$, in symbols F_A, is following formula, which is also in conjunctive normal form:

$$(C_1 \vee A) \wedge ... \wedge (C_n \vee A).$$

Theorem 4 (Weaving Projection into Compilation) *Call an algorithm that computes for all formulas F and literal scopes S an equivalent to* $\mathsf{project}(F, S)$ *that does not contain the* project *operator and is fully linkless a* project-compile-algorithm.

If A is a ground atom and F_A^* is the class of disjoin-formulas upon A then

1. *there exists a project-compile-algorithm PC_1 and a polynomial p_1 such that for all formulas $F \in F_A^*$ and representations*[3] *of literal scopes S such that $+A \notin S$ it holds that*

$$PC_1(F, S) \text{ requires time } \leq p_1(|F| + |S|),$$

2. *unless $\mathsf{NP} \subseteq \mathsf{P/poly}$ for all project-compile-algorithms PC_2 that construct a fully linkless equivalent to their input formula as an intermediate data structure and for all polynomials p_2 there exists a formula $F \in F_A^*$ such that*

$$PC_2(F, S) \text{ requires space } > p_2(|F|).$$

An example for PC_1, that justifies item (1.) of Theorem 4, is an algorithm that checks whether its input formula F is a disjoin-formula over some atom $A \in \mathcal{A}(F)$ such that $+A \notin S$ and returns $\top$ if this is the case, and otherwise calls an arbitrary project-compile-algorithm on F and S.

Item (2.) of Theorem 4 follows since clausal entailment for arbitrary formulas in conjunctive normal form can be encoded into clausal entailment for disjoin-formulas. Clausal entailment is a linear operation for formulas which are fully

[2] $\mathsf{P/poly}$ is a non-uniform complexity class. Its relationship to knowledge compilation has been brought to attention by Kautz and Selman [15]. Cadoli et al. have generalized their results in [5]. Details and further references can be found in that work in the context of Theorem 6 which has the assumption $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ as a precondition. Failure of this assumption would imply the collapse of the polynomial hierarchy. Our proof of Theorem 4 is based on that theorem, as actually are many complexity results about knowledge compilation formats [8].

[3] As argument of an algorithm, a symbol denoting a literal scope stands for a *representation* of the literal scope. It is quietly assumed that such representations of finite sets allow to decide membership in time polynomial to their cardinality.

linkless. From Theorem 6 in [5] then follows that, unless $\mathsf{NP} \subseteq \mathsf{P/poly}$, the size of the linkless equivalent constructed by algorithms PC_2 can not be polynomially bounded by the size of the input formula, the disjoin formula F.

In practice disjoin-formulas are not expected to be important as global inputs to reasoning tasks, but it is quite plausible that they (or other types of formulas that allow efficient elimination of projection) are constructed during reasoning.

5 A Variant of DPLL for Projection Elimination and Knowledge Compilation

We outline a variant of the DPLL calculus, called DPLL-Fwd, as an exemplary instance of the tableau framework for projection elimination and knowledge compilation that has been described in Sect. 3.

Along with the input formula, DPLL-Fwd takes representations of literal scopes S_l and S_u, corresponding to the conditions of Sect. 3.1 and also Theorem 3, as parameters. DPLL-Fwd constructs a fwd-tableau of a special form, corresponding to a semantic tree: An o-node has at most two children and the literal labels of an o-node with two children are complementary.

Only a single branch of the tableau under construction is required to be explicitly represented in memory at any point of time. We call it the *active branch* and its leaf node the *active leaf*. Since the space required by a single branch is polynomially bounded by the size of the input formula, DPLL-Fwd operates in polynomial space, although the overall size of the constructed tableau is not polynomially bounded.

Decide is a rule of DPLL-Fwd that effects that the active leaf is labeled as an o-node and gets two children labeled with complementary literals, corresponding to an atomic cut. The left child becomes the new active leaf. The splitting atom, i. e. the atom of the literal labels of the two new nodes, must not already appear in a literal label on the active branch. Thus the constructed tableau is regular and a branch never contains complementary literal labels. The splitting atom A must satisfy a further condition that is complementary to the branch completeness property $\mathsf{PB\text{-}L}_{S_l}$ and intuitively means at least one of the literals $+A$ or $-A$ is not in S_l (is "to forget" for projection elimination) and $(B_N \wedge F_N)$ has a conjunctive subformula such that $+A$ occurs in one of the conjuncts and $-A$ in the other:

$$+A \notin S_l \cap \widetilde{S}_l \text{ and}$$
$$(B_N \wedge F_N) \text{ is not linkless inside } \{+A\}. \quad [4]$$

For standard DPLL algorithms (see for example [23]), the forward label of a node N can be considered as the input formula with each atom that appears as literal label in B_N substituted with $\top$ or $\bot$, depending on whether it is positively or negatively in B_N. In such a method the forward label of a node is just implicitly represented by the input formula and the branch to the node.

[4] This condition holds for $-A$ if and only if it holds for $+A$.

DPLL-Fwd permits more freedom in the computation of forward labels. In accord with the *upward-preserving* condition, the forward label of a new node can be computed from the forward label of its parent by substituting the atom of the literal label of the new node with $\top$ or $\bot$ and then performing simplifications which just preserve equivalence relative to S_u, as for example those shown in Sect. 2.7. In this way DPLL-Fwd realizes a form of weaving projection elimination into compilation, as discussed in Sect. 4.

Disconnect is the rule of DPLL-Fwd which effects creation of and-nodes. It expresses as a rule a dynamic problem decomposition as developed by Bayardo and Pehousek [1] for DPLL-based model counting and used by Huang and Darwiche for DNNF compilation [14]. For model counting and DNNF compilation the components not only have to be pairwise linkless, but are required to have pairwise disjoint atom bases. Disconnect is applicable if the forward label of the active leaf F_N is equal modulo associativity and commutativity of the $\wedge$ operator to a propositional formula $F_1 \wedge ... \wedge F_n$ for some $n > 1$ such that for all $i, j \in 1..n$, $i \neq j$, it holds that F_i and F_j are linkless outside S_l. It effects that the active leaf is labeled as an a-node and gets a child for each of the conjuncts F_i with $\top$ as literal label and F_i (possibly further simplified) as forward label. The leftmost of the new children becomes the new active leaf.

Next is a rule that effects backtracking by assigning the next leaf node in a depth-first left-to-right ordering as active leaf. Its precondition is that the active branch is "completely constructed", which means that it satisfies the PB_{S_l} condition.

Backjump is a rule that effects a more efficient backtracking than Next in the case where the forward label of the tableau root conjoined with just fragments of the active path entails $\bot$. Backjump is based on the specification of backjumping for *Abstract DPLL* [22, 23], which formally models the main technique used in modern DPLL-based SAT solvers [2, 24, 26] for backtracking in conjunction with the generation of context related unit lemmas. An application of Backjump can be modeled as *Truncation* followed by *Extension*. This modeling allows to conclude that Backjump preserves the *extensional* property and thus can be safely included into methods that construct extensional tableaux.

For theorem proving or model computation tasks backjumping is commonly applied with respect to a branch in which siblings to the right represent possibilities to explore in future computation and siblings to the left are either not present or can be ignored since they correspond to parts of the problem that already have been solved. For transformation tasks this is different. Although siblings to the left might correspond to already transformed parts of the input formula, a backjumping step can effect that these are deleted and an improved transformation is computed with the unit lemmas made available by backjumping.

True-Up is a rule that, like the application of simplifications in the computation of forward labels, realizes a form of weaving projection elimination into compilation. It if applicable if the active leaf is the child of an o-node, has a literal label that is not in S_u, and its forward label is $\top$. It effects that the parent

of the active leaf gets $\top$ assigned as forward label, gets all its children removed (it is no longer an o-node then) and is assigned as the new active leaf. (Related preconditions apply in the case where the active leaf is the child of an *a-node*).

True-Up preserves equivalence *relative to* S_u of the leafy formula of the parent of the active node. If the literal label of the parent of the active node is, like the active node, not in S_u, then an application of True-Up can trigger another one. For empty S_u, in this way, if a model has been found — indicated by the forward label of the active node being $\top$ — the True-Up rule can be repeatedly applied until the fwd-tableau consists of just a single node with $\top$ as literal as well as forward label. This single-node tableau indicates satisfiability of the input formula. Thus the method terminates "inherently" after finding the first model: It has not to be hindered "from the outside", from backtracking and searching for alternative models. If the active node is a child of an o-node and has a sibling to its left, then True-Up can effect, similar to Backjump, that a part of the tableau under construction is thrown away.

We now sketch how termination of DPLL-Fwd can be shown. The length of branches constructed by DPLL-Fwd is finitely bounded, which follows from the regularity property, and, since children of a-nodes have the literal label $\top$ which is not subjected to the regularity condition, from a restriction on the forward labels along a branch. The following definition specifies a mapping from branches to strings and an ordering relation on these strings. Termination of DPLL-Fwd then follows since for all its rules it can be shown that the string associated with the active branch before rule application is strictly greater in terms of the ordering relation than the string associated with the active branch afterward.

Definition 13 (Ordering Branches by Outdegree Strings)

(i) If $B = N_1...N_n$ is a branch of an ordered tree then the *outdegree-right string* of B, is a sequence of pairs $\langle o_1, r_1\rangle....\langle o_n, r_n\rangle$ such that

o_i is the number of children of N_i, if $i \in 1..n-1$,
r_i is the number of children of N_i to the right of N_{i+1}, if $i \in 1..n-1$,
o_n is 0 if $F_{N_n} = \top$,
o_n is the symbol ω if $F_{N_n} \neq \top$,
r_n is 0.

(ii) If B_1, B_2 are branches of ordered trees then $B_1 >_o B_2$ holds if and only if the outdegree-right string of B_1 is lexicographically greater than outdegree-right string of B_2, where the elements of B_1 and B_2 are compared lexicographically, and the components of these elements by are compared by numerical value, considering the symbol ω greater than any number.

6 Polynomial Space and Piecemeal Output

As notes in Sect. 5, DPLL-Fwd operates with a working data structure that is polynomially bounded by the size of the input formula, although the size

of the output, the constructed tableau, is not polynomially bounded. Thus at any point of time only a piece of the tableau under construction has to be explicitly represented in memory, while the remaining piece can be considered as offline. While the polynomial space requirements are clearly beneficial for theorem proving tasks in which a yes/no answer is computed, they can also be utilized for formula transformation tasks. To this end we investigate variants of DPLL-Fwd which output pieces of their overall output as soon as they are computed, and thus fully operate in polynomial space. Such algorithms can be employed in applications in which output pieces by themselves are useful, can be piped to another program or in which it is expected that the first few output pieces provide a solution. Output pieces can for example be piped to a theorem prover: if an overall output formula F is equivalent to the disjunction of output pieces then a theorem G follows from F if and only if G follows from each of the pieces.

DPLL-Fwd (without the Disconnect rule) can be run in a way such that whenever Next is applied to a state with active leaf N the formula $(B_N \wedge F_N)$ is returned as an output piece. Finally, $(B_N \wedge F_N)$ for the active leaf N of the terminal state is returned as the last output piece. The overall output is the disjunction of those pieces.

For tableau construction methods that do not involve destructive operations such as backjumping, the combination of output pieces which each correspond to a branch of the final tableau is trivially equal to the final tableau. For methods that include destructive operations extra effort is required to show that none of the output pieces are "wrong" and no essential output piece is "missing" at termination. It may be acceptable that output pieces may become redundant in the presence of output pieces delivered at a later point of time. It can be shown for DPLL-Fwd without the Disconnect rule but with Backjump that the overall output formula satisfies the three conditions of Sect. 3.1 in the role of F'.

7 Examples

Example 2 (Fwd-Tableau and Leafy Formula) Consider the fwd-tableau in Fig. 1. Nodes are identified by numbers. For all nodes N in the tableau L_N and F_N are shown, F_N framed in boxes. For leaf nodes the boxes are bold to emphasize that they are constituents of the leafy formula. Node 1 is an a-node, which is indicated by the arc connecting its outgoing edges. Let T be this fwd-tableau. Its leafy formula is shown in Fig. 2.

Example 3 (Linklessness-Preserving) Consider the fwd-tableau in Fig. 3. Node 1 is the only a-node. Its two children, nodes 2 and 3, have $\top$ as literal label. The forward label of 2 and that of 3 are fully linkless, hence also linkless outside $S = \overline{\{p, \neg p, q, \neg q\}}$, implying the condition of Definition 6.i of *linklessness-preserving*. The condition of Definition 6.ii of follows since all literals appearing in literal and forward labels of the children of 3 also appear in the forward label of 3.

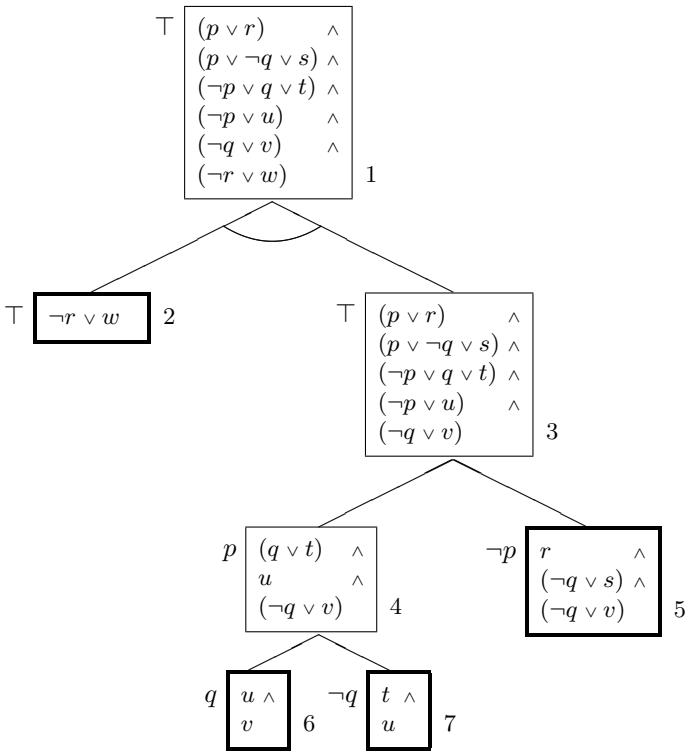

Fig. 1. A Fwd-Tableau

$$
\begin{aligned}
\mathsf{leafy}(T) = \top \wedge\ & \\
& ((\top \wedge (\neg r \vee w)) \wedge \\
& \quad (\top \wedge \\
& \qquad ((p \wedge ((q \wedge (u \wedge v)) \vee \\
& \qquad\quad (\neg q \wedge (t \wedge u)))) \vee \\
& \qquad (\neg p \wedge (r \wedge ((\neg q \vee s) \wedge (\neg q \vee v))))))).
\end{aligned}
$$

Fig. 2. Leafy Formula of the Fwd-Tableau in Fig. 1

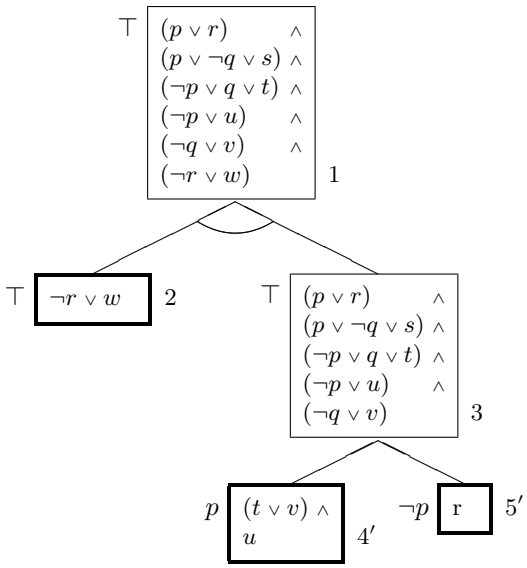

Fig. 3. A Fwd-Tableau for Forgetting About $S = \{p, \neg p, q, \neg q\}$
For a legend see Example 2

Example 4 (Branch Completeness Property PB-L$_S$) Consider the fwd-tableau in Fig. 3. Let $S = \overline{\{p, \neg p, q, \neg q\}}$. Node 3 does not satisfy PB-L$_S$, since F_3 contains for example p and $\neg p$ in different clauses. Node $4'$ satisfies PB-L$_S$ since $p \wedge ((t \vee v) \wedge u)$ is clearly linkless outside S.

Example 5 (Upward-Preserving for the Set of All Literals) The fwd-tableau in Fig. 1 is upward-preserving for the set of all literals. Since for all formulas F it holds that $\mathsf{project}(F, \mathsf{all\text{-}literals}) \equiv F$, the project operator in the conditions for *upward-preserving* can be dropped. Consider for example node 4. $B_4 = p$. F_4 and F_3 are the boxed formulas at nodes 4 and 3 respectively. It is easy to see, that

$$p \wedge F_4 \models F_3.$$

As a second example, consider node 1, an a-node. $B_1 = \top$. F_2 is just the last clause of F_1, F_3 is the conjunction of the other clauses of F_1. It is easy to see that

$$F_2 \wedge F_3 \models F_1.$$

Example 6 (Upward-Preserving for an Atom Scope) Let S be defined as $\overline{\{p, \neg p, q, \neg q\}}$. S represents an atom scope, since $S = \widetilde{S}$. The fwd-tableau in Fig. 3 is upward-preserving for S. We show exemplarily that the condition of Definition 8.i of the definition of *upward-preserving* is satisfied for node $4'$ in relation to 3, i. e. it holds that:

$$p \wedge \mathsf{project}(F_{4'}, S) \models \mathsf{project}(F_3, S). \qquad \text{(vi)}$$

Nodes 1 to 3 are the same as in Fig. 1. In Example 5 we have seen that for the tableau of Fig. 1 it holds that

$$p \wedge F_4 \models F_3. \tag{vii}$$

$F_{4'}$ in Fig. 3 can be considered as being obtained from F_4 by applying the *ISOL* simplification (see Sect. 2.7) upon q. Thus

$$\mathsf{project}(F_4, S) \equiv \mathsf{project}(F_{4'}, S). \tag{viii}$$

Statements (vii) and (viii) imply statement (vi), which can be shown as follows:

(1) $\mathsf{project}(p \wedge F_4, S) \models \mathsf{project}(F_3, S)$.
(2) $\mathsf{project}(p, S) \wedge \mathsf{project}(F_4, S) \models \mathsf{project}(F_3, S)$.
(3) $p \wedge \mathsf{project}(F_4, S) \models \mathsf{project}(F_3, S)$.

(1) follows from (vii). (2) follows from (1) and Theorem 1, since p and F_4 are linkless outside S. (3) follows from (2) and since $p \models \mathsf{project}(p, S)$. (vi) follows from (3) and (viii).

Example 7 (Upward-Preserving for a Literal Scope) The fwd-tableau in Fig. 4 is identical to the fwd-tableau in Fig. 3, except for node $4''$. It is upward-preserving for $\overline{\{\neg p, \neg q\}}$, a scope that contains p and q just positively.

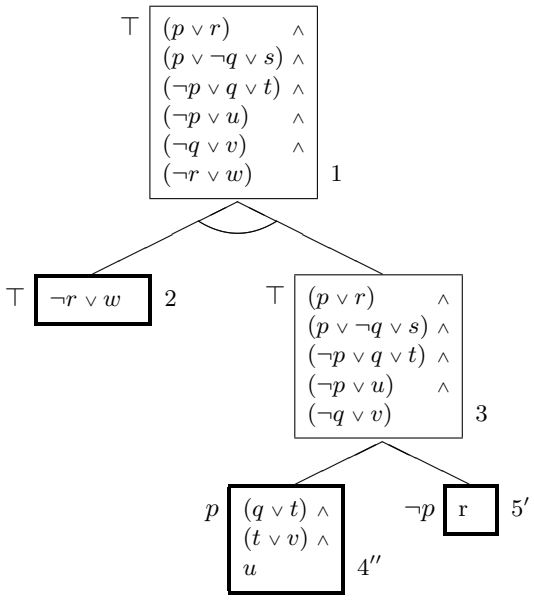

Fig. 4. A Fwd-Tableau for Forgetting About $S = \{\neg p, \neg q\}$
For a legend see Example 2

Example 8 (Branch Implied Forward Labels) It is easy to verify that the fwd-tableaux in Fig. 3 has branch implied forward labels. An example instantiation of the condition of Definition 10 where N is an o-node is $F_3 \wedge p \models F_{4'}$. An example where N is an a-node is $F_1 \models F_3$.

Example 9 (Extensional) It is easy to see that the fwd-tableau in Fig. 3 is extensional: O-nodes have two children with complementary ground literals as literal labels. Their disjunction is a tautology, such that the condition of Definition 9.i of *extensional* is trivially satisfied. Also the condition of Definition 9.ii is trivially satisfied since children of a-nodes have $\top$ as literal label.

8 Related Work

8.1 Projection/Forgetting

The term *forgetting* has been coined in [18], where it is defined for first order logic by means of agreement conditions on first order structures. Related operations for propositional logic have been investigated in the context of *local computation* [16]. The term *projection* seems to appear first in [6]. Forgetting of *literals* is defined in [17], where also many properties and applications of projection are shown. The interplay of literal projection and conjunction stated in our Theorem 1 is in [17] only shown for the rudimentary special where the conjuncts are conjunctions of literals.

A characterization of projection that applies to clausal first order logic with a Herbrand semantics has been given in [25]. In first order logic, computation of the forgetting about all ground atoms with a given predicate corresponds to elimination of an existential second order quantifier over that predicate. An algorithm for this and applications are described in [11].

Projection is related to the model theoretic concept of interpolation [13]: If F is a propositional formula and S an atom scope then the result of eliminating the projection operator in $\mathsf{project}(F, S)$ followed by substituting all atoms not in S with e. g. $\top$ is a *uniform interpolant* for F with respect to S.

8.2 DNNF Compilation

In [7, 14] adaptions of DPLL for knowledge compilation are presented. For our tableau framework we adopted two features of the DPLL-based DNNF compilers: the independent processing of conjuncts, corresponding to and-nodes, as in DPLL-based model counters [1] and the use of dependency directed backtracking to delete already constructed tableau parts. The method in [7] is described by means of pseudocode, similar to a SAT solver, with calls to auxiliary procedures that construct the output formula.

We use linkless negation normal form as a basis, in contrast to DNNF, since the linkless form is more general and also allows projection elimination in linear time. See [21] for a comparison of linkless form with DNNF.

Although in [6] it is shown that projection elimination is an important operation in the context of knowledge compilation, the methods described in [6, 7, 14] do not incorporate projection elimination into the compilation process, as suggested in our Sect. 4. Compilation to a formula which is not fully in DNNF, but decomposable *outside* a set of atoms S might be considered as a first step towards the integration of projection elimination into knowledge compilation, since it allows linear projection elimination onto atom scopes that are supersets of S. This concept is defined in [6] (called *decomposable except on atoms*), but the above relationship to projection elimination is not stated or utilized.

The methods in [7, 14] always apply DPLL exhaustively, i. e. a branch is only completely constructed if it either falsifies or implies the input formula. Our framework also permits weaker branch completeness conditions. Copies of the input formulas which are simplified with respect to a branch (but not necessarily yielding a truth value constant) are then included in the result.

Features described in [7, 14] that are not covered by our framework are caching, non-tree representations of formulas and consideration of atom orderings. The systems have been implemented and experiments have shown their practical usefulness.

8.3 Knowledge Compilation with Tableau Methods

In [21] DNNF is compared to formulas which are fully linkless. A compilation method called *semantic factoring* is investigated that works by applying the Shannon expansion to subformulas. The construction of a regular clausal tableau is shown as a second compilation method, based on the observation that a fully developed regular clausal tableau for a given formula represents an equivalent in DNNF. A notion of projection onto atom scopes is defined by means of substitution in a syntactic way that only applies to fully linkless formulas.

In [12] a procedure for equivalence preserving compilation into a form called *factored negation normal form* is described. This method is tableau-based, related to DPLL and works in polynomial space in the sense that only a single polynomial working branch has to be kept in memory while the rest of the tableau is considered offline. It seems however that this method lacks the feature that conjuncts meeting certain preconditions are compiled independently, which is achieved for semantic factoring by applying the Shannon expansion to subformulas and for DPLL-like methods by means of and-nodes. Inclusion of projection elimination into the compilation process is not considered in [21] and [12].

8.4 DPLL-Based Boolean Quantifier Elimination

A variant of DPLL for the elimination of Boolean quantifiers (which in propositional logic is the same as projection elimination for an atom scope) is described in [19]. It can be considered as incorporating projection elimination into the compilation process, but it just constructs formulas in clausal form, while our approach permits weaker restrictions, such as DNNF, linkless formulas, or also

unrestricted outputs which are obtained from requiring linklessness *outside the scope of the projection* as an internal intermediate format. The system by McMillan is tied to the Tseitin definitional encoding of the input formula and essentially depends on storing complements of generated implicants, thus it does not has a polynomially bounded working data structure.

In [20] it is shown that the use of interpolants extracted from proofs can be used to approximate the costly operation of projection elimination in certain applications of symbolic model checking.

9 Conclusion

We have specified abstract requirements for generalized proving procedures employed in formula transformation tasks such as the computation of projection and knowledge compilation. We defined an extension of clausal tableaux suited for such tasks. The abstract requirements were related with an interplay of structural and semantical properties of these tableaux.

As an example instance of this framework an adaption of DPLL for formula transformation tasks that is specified by means of tableau rules has been outlined. It covers refinements which indicate success in practice. Yet by being declarative and embedded into the tableau framework it provides an accessible basis for further refinements, variations and comparisons to related techniques.

We outlined issues of utilizing tableau methods which work in polynomial space for formula transformation tasks with exponential output size.

Our framework shows the interplay of projection elimination and certain types of knowledge compilation. Actually projection elimination can be an important operation for knowledge compilation: the projection to an application relevant subsignature is a means to compensate somewhat for the size blow-up due to compilation. We have shown that the current practice of applying the projection operation to compilation results can be essentially improved by weaving projection into the compilation process.

A main issue for future work is the application of our framework to other languages which are important in knowledge representation and have tableau-based processing techniques. This includes classical first order logic, modal and description logics and languages with non-classical semantics such as minimal model semantics which is used for first order model generation [4]. As sketched in [25], projection can be applied to express dependency relationships between concepts and to control the vocabulary of answers. To use this in practice, projection elimination tasks have to be performed embedded within other computations. Investigation how these computations as a whole can be performed efficiently is another issue for future work.

References

1. R. J. Bayardo and J. D. Pehoushek. Counting models using connected components. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*

and Twelfth Conference on Innovative Applications of Artificial Intelligence, pages 157–162. AAAI Press / The MIT Press, 2000.

2. R. J. Bayardo and R. C. Schrag. Using CSP look-back techniques to solve real-world sat instances. In *Proceedings AAAI-97*, pages 203–208, 1997.
3. W. Bibel. *Deduktion — Automatisierung der Logik*. Oldenbourg, München, 1992.
4. F. Bry and A. H. Yahya. Positive unit hyperresolution tableaux and their application to minimal model generation. *Journal of Automated Reasoning*, 25(1):35–82, 2000.
5. M. Cadoli, F. M. Donini, and M. Schaerf. Is intractability of nonmonotonic reasoning a real drawback? *Artificial Intelligence*, 88(1–2):215–251, 1996.
6. A. Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001.
7. A. Darwiche. New advances in compiling CNF to decomposable negation normal form. In R. L. de Mántaras and L. Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI 2004)*, pages 328–332. IOS Press, 2004.
8. A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.
9. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7), 1962.
10. M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM*, 7(3):201–215, 1960.
11. D. Gabbay and H. J. Ohlbach. Quantifier elimination in second-order predicate logic. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 425–435, San Mateo, California, 1992. Morgan Kaufmann.
12. R. Hähnle, N. V. Murray, and E. Rosenthal. Normal forms for knowledge compilation. In M.-S. Hacid, Z. W. Raś, and S. Tsumoto, editors, *Foundations of Intelligent Systems (ISMIS'05)*, volume 3488 of *LNAI*, pages 304–313. Springer, 2005.
13. E. Hoogland. *Definability and Interpolation — Model-theoretic Investigations*. PhD thesis, University of Amsterdam, Amsterdam, 2001.
14. J. Huang and A. Darwiche. DPLL with a trace: From SAT to knowledge compilation. In L. P. Kaelbling and A. Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 156–162. Professional Book Center, 2005.
15. H. A. Kautz and B. Selman. Forming concepts for fast inference. In *Proceedings AAAI-92*, pages 786–793, San Jose, California, 1992.
16. J. Kohlas, R. Haenni, and S. Moral. Propositional information systems. *Journal of Logic and Computation*, 9(5):651–681, 1999.
17. J. Lang, P. Liberatore, and P. Marquis. Propositional independence — formula-variable independence and forgetting. *Journal of Artificial Intelligence Research*, 18:391–443, 2003.
18. F. Lin and R. Reiter. Forget It! In R. Greiner and D. Subramanian, editors, *Working Notes, AAAI Fall Symposium on Relevance*, pages 154–159, Menlo Park, California, 1994. American Association for Artificial Intelligence.
19. K. L. McMillan. Applying sat methods in unbounded symbolic model checking. In E. Brinksma and K. G. Larsen, editors, *Computer Aided Verification — 14th International Conference, CAV 2002*, volume 2404 of *LNCS*, pages 250–264. Springer, 2002.

20. K. L. McMillan. Applications of craig interpolants in model checking. In N. Halbwachs and L. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems — 11th International Conference, TACAS 2005*, volume 3440 of *LNCS*, pages 1–12. Springer, 2005.
21. N. V. Murray and E. Rosenthal. Tableaux, path dissolution and decomposable negation normal form for knowledge compilation. In *Proceedings TABLEAUX 2003*, volume 2796 of *LNAI*, pages 165–180. Springer, 2003.
22. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Abstract DPLL and Abstract DPLL modulo theories. In F. Baader and A. Voronkov, editors, *Logic Programming and Automated Reasoning — 11th International Conference, LPAR 2004*, volume 3452 of *LNAI*, pages 36–50. Springer, 2005.
23. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, Nov. 2006.
24. J. P. M. Silva and K. A. Sakallah. GRASP — a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '96)*, pages 220–227. IEEE Computer Society, 1996.
25. C. Wernhard. Semantic knowledge partitioning. In J. J. Alferes and J. Leite, editors, *Logics in Artificial Intelligence — 9th European Conference, JELIA 04*, volume 3229 of *LNAI*, pages 552–564. Springer, 2004.
26. L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In A. Voronkov, editor, *Automated Deduction — CADE-18*, volume 2392 of *LNAI*, pages 295–313. Springer, 2002.

Appendix

Proof of Theorem 1. We show that if F_1, F_2 are formulas and S is a literal scope then

$$\text{if } F_1 \text{ and } F_2 \text{ are essentially linkless outside } S \text{ then}$$
$$\mathsf{project}(F_1, S) \wedge \mathsf{project}(F_2, S) \models \mathsf{project}(F_1 \wedge F_2, S).$$

We use the following additional notation: If I is an interpretation, A an atom, L a literal and S a set of literals then

- $I.A$ is defined as the literal with atom A which is contained in I;
- $I[L] \stackrel{\text{def}}{=} (I - \{\widetilde{L}\}) \cup \{L\}$;
- $I[S] \stackrel{\text{def}}{=} (I - \widetilde{S}) \cup S$.

We present the proof in a formal style, where justifications follow the numbered lines that show proof steps in symbolic form.

(1) F_1 and F_2 are linkless outside S.
(2) $I \models \mathsf{project}(F_1, S)$.
(3) $I \models \mathsf{project}(F_2, S)$.
(4) $J_1 \models F_1$.
(5) $J_1 \cap S \subseteq I$.
(6) $J_2 \models F_2$.
(7) $J_2 \cap S \subseteq I$.
(8) $J_1' \models F_1$.
(9) $J_2' \models F_2$.
(10) $J_1' \cap S \subseteq I$.
(11) $J_1' = J_2'$.
(12) $J_1' \models F_1 \wedge F_2$.
(13) $I \models \mathsf{project}(F_1 \wedge F_2, S)$.

Assume (1) and let I be an interpretation such that (2) and (3) hold. Let J_1, J_2 be interpretations satisfying (4) – (7). The existence of such interpretations follows from (2) and (3) respectively with the definition of project. In the following we show the construction of interpretations J_1' and J_2' that satisfy (8), (9), (10) and are actually equal, which is expressed by (11). (12) follows from (11), (9) and (8). (13), which concludes the proof, follows from (12) and (10) with the definition of project.

(14) If $J_X.A \neq J_Y.A$ and 1
$J_X.A \notin \mathcal{L}_{\mathcal{E}}(F_X)$ and 2
$(\widetilde{J_X.A} \in \mathcal{L}_{\mathcal{E}}(F_Y)$ or $J_X.A \neq I.A)$ 3
then
$J_X'.A \stackrel{\text{def}}{=} \widetilde{J_X.A}$ 4
else
$J_X'.A \stackrel{\text{def}}{=} J_X.A.$ 5

For all $X, Y \in \{1, 2\}$ with $X \neq Y$ let J_X' be the interpretation such that (14) holds for all ground atoms A.

Subproof of (8) and (9). Let $M \stackrel{\text{def}}{=} \{J_1.A \mid J_1'.A = \widetilde{J_1.A}\}$. Then $J_1' = J_1[\widetilde{M}]$. If A is a ground atom such that $J_1'.A = \widetilde{J_1.A}$, the condition of line 2 in (14) for $X = 1$ must be true, thus $J_1.A \notin \mathcal{L}_\mathcal{E}(F_1)$. Hence $M \cap \mathcal{L}_\mathcal{E}(F_1) = \emptyset$. (8) then follows from (4) and the fact that for all interpretations I, formulas F and literal scopes S it holds that

$$\text{if } I \models F \text{ and } S \cap \mathcal{L}_\mathcal{E}(F) = \emptyset \text{ then } I[\widetilde{S}] \models F.$$

(9) follows by the same arguments with subscripts 1 and 2 switched.

Subproof of (10). For all ground atoms A it holds that if $J_1'.A \neq J_1.A$ then the condition of line 1 in (14) for $X = 1$ must be true, hence $J_1.A = \widetilde{J_2.A}$, hence $J_1'.A = J_2.A$. Thus $J_1' \subseteq J_1 \cup J_2$. (10) then follows from (5) and (7).

Subproof of (11). For all ground atoms A it holds that if $J_1.A = J_2.A$ then the condition of line 1 in (14) fails for $X = 1$ as well as for $X = 2$, hence $J_1'.A = J_1.A = J_2.A = J_2'.A$.

Otherwise $J_1.A \neq J_2.A$. We show that in this case the condition of (14) (i. e. lines 1–3) is satisfied exactly in one of the cases of $X = 1$ or $X = 2$. This implies (11): Assume lines 1-3 of (14) are satisfied for $X = 1$ but not for $X = 2$. From (14) then follows for $X = 1$ that $J_1'.A = \widetilde{J_1.A} = J_2.A$ and for $X = 2$ that $J_2'.A = J_2.A$. Thus $J_1'.A = J_2'.A$. The same argument also applies to switched subscripts 1 and 2.

(15) $J_1.A \neq J_2.A$.
(16) $J_1.A \notin (S \cap \widetilde{S})$.
(17) If $\widetilde{J_1.A} \in \mathcal{L}_\mathcal{E}(F_2)$ then $J_1.A \notin \mathcal{L}_\mathcal{E}(F_1)$.

Assume (15). (16) follows from (15), (5) and (7). (17) from (16) and (1).

(18) $\mathsf{cond}(1, 2, A)$ iff
(19) $J_1.A \notin \mathcal{L}_\mathcal{E}(F_1)$ and ($\widetilde{J_1.A} \in \mathcal{L}_\mathcal{E}(F_2)$ or $J_1.A \neq I.A$) iff
(20) ($\widetilde{J_1.A} \in \mathcal{L}_\mathcal{E}(F_2)$ and $J_1.A \notin \mathcal{L}_\mathcal{E}(F_1)$) or
($J_1.A \neq I.A$ and $J_1.A \notin \mathcal{L}_\mathcal{E}(F_1)$) iff
(21) $\widetilde{J_1.A} \in \mathcal{L}_\mathcal{E}(F_2)$ or ($J_1.A \neq I.A$ and $J_1.A \notin \mathcal{L}_\mathcal{E}(F_1)$) iff
(22) $J_2.A \in \mathcal{L}_\mathcal{E}(F_2)$ or ($J_2.A = I.A$ and $\widetilde{J_2.A} \notin \mathcal{L}_\mathcal{E}(F_1)$) iff
(23) $\neg\mathsf{cond}(2, 1, A)$.

Define $\mathsf{cond}(X, Y, A)$ as abbreviation for the condition of (14) under the assumption (15), e. g.

$$\mathsf{cond}(1, 2, A) \stackrel{\text{def}}{=} J_1.A \notin \mathcal{L}_\mathcal{E}(F_1) \textit{ and } (J_1.A \neq I.A \textit{ or } \widetilde{J_1.A} \in \mathcal{L}_\mathcal{E}(F_2)).$$

For all ground atoms A it holds that (18) – (23) are equivalent to each other: Equivalence of (19) to (18) follows by unfolding cond. (20) is logically equivalent to (19). Equivalence of (21) to (20) follows from (17). (22) is equivalent to (21)

since by (15) $J_1.A = \widetilde{J_2.A}$. (23) is equivalent to (22) by folding into cond. Thus we have shown that under assumption (15) (and the initial assumptions (1) – (3)) the condition of (14) is satisfied exactly in one of the cases of $X = 1$ or $X = 2$, which implies (11).

□

Bisher erschienen

Arbeitsberichte aus dem Fachbereich Informatik
(http://www.uni-koblenz.de/fb4/publikationen/arbeitsberichte)

Christoph Wernhard: Tableaux Between Proving, Projection and Compilation, Arbeitsberichte aus dem Fachbereich Informatik 18/2007

Ulrich Furbach, Claudia Obermaier: Knowledge Compilation for Description Logics, Arbeitsberichte aus dem Fachbereich Informatik 17/2007

Fernando Silva Parreiras, Steffen Staab, Andreas Winter: TwoUse: Integrating UML Models and OWL Ontologies, Arbeitsberichte aus dem Fachbereich Informatik 16/2007

Rüdiger Grimm, Anastasia Meletiadou: Rollenbasierte Zugriffskontrolle (RBAC) im Gesundheitswesen, Arbeitsberichte aud dem Fachbereich Informatik 15/2007

Ulrich Furbach, Jan Murray, Falk Schmidsberger, Frieder Stolzenburg: Hybrid Multiagent Systems with Timed Synchronization-Specification and Model Checking, Arbeitsberichte aus dem Fachbereich Informatik 14/2007

Björn Pelzer, Christoph Wernhard: System Description:“E-KRHyper“, Arbeitsberichte aus dem Fachbereich Informatik, 13/2007

Ulrich Furbach, Peter Baumgartner, Björn Pelzer: Hyper Tableaux with Equality, Arbeitsberichte aus dem Fachbereich Informatik, 12/2007

Ulrich Furbach, Markus Maron, Kevin Read: Location based Informationsystems, Arbeitsberichte aus dem Fachbereich Informatik, 11/2007

Philipp Schaer, Marco Thum: State-of-the-Art: Interaktion in erweiterten Realitäten, Arbeitsberichte aus dem Fachbereich Informatik, 10/2007

Ulrich Furbach, Claudia Obermaier: Applications of Automated Reasoning, Arbeitsberichte aus dem Fachbereich Informatik, 9/2007

Jürgen Ebert, Kerstin Falkowski: A First Proposal for an Overall Structure of an Enhanced Reality Framework, Arbeitsberichte aus dem Fachbereich Informatik, 8/2007

Lutz Priese, Frank Schmitt, Paul Lemke: Automatische See-Through Kalibrierung, Arbeitsberichte aus dem Fachbereich Informatik, 7/2007

Rüdiger Grimm, Robert Krimmer, Nils Meißner, Kai Reinhard, Melanie Volkamer, Marcel Weinand, Jörg Helbach: Security Requirements for Non-political Internet Voting, Arbeitsberichte aus dem Fachbereich Informatik, 6/2007

Daniel Bildhauer, Volker Riediger, Hannes Schwarz, Sascha Strauß, „grUML – Eine UML-basierte Modellierungssprache für T-Graphen“, Arbeitsberichte aus dem Fachbereich Informatik, 5/2007

Richard Arndt, Steffen Staab, Raphaël Troncy, Lynda Hardman: Adding Formal Semantics to MPEG-7: Designing a Well Founded Multimedia Ontology for the Web, Arbeitsberichte aus dem Fachbereich Informatik, 4/2007

Simon Schenk, Steffen Staab: Networked RDF Graphs, Arbeitsberichte aus dem Fachbereich Informatik, 3/2007

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik, 2/2007

Anastasia Meletiadou, J. Felix Hampe: Begriffsbestimmung und erwartete Trends im IT-Risk-Management, Arbeitsberichte aus dem Fachbereich Informatik, 1/2007

„Gelbe Reihe“

(http://www.uni-koblenz.de/fb4/publikationen/gelbereihe)

Lutz Priese: Some Examples of Semi-rational and Non-semi-rational DAG Languages. Extended Version, Fachberichte Informatik 3-2006

Kurt Lautenbach, Stephan Philippi, and Alexander Pinl: Bayesian Networks and Petri Nets, Fachberichte Informatik 2-2006

Rainer Gimnich and Andreas Winter: Workshop Software-Reengineering und Services, Fachberichte Informatik 1-2006

Kurt Lautenbach and Alexander Pinl: Probability Propagation in Petri Nets, Fachberichte Informatik 16-2005

Rainer Gimnich, Uwe Kaiser, and Andreas Winter: 2. Workshop "Reengineering Prozesse" – Software Migration, Fachberichte Informatik 15-2005

Jan Murray, Frieder Stolzenburg, and Toshiaki Arai: Hybrid State Machines with Timed Synchronization for Multi-Robot System Specification, Fachberichte Informatik 14-2005

Reinhold Letz: FTP 2005 – Fifth International Workshop on First-Order Theorem Proving, Fachberichte Informatik 13-2005

Bernhard Beckert: TABLEAUX 2005 – Position Papers and Tutorial Descriptions, Fachberichte Informatik 12-2005

Dietrich Paulus and Detlev Droege: Mixed-reality as a challenge to image understanding and artificial intelligence, Fachberichte Informatik 11-2005

Jürgen Sauer: 19. Workshop Planen, Scheduling und Konfigurieren / Entwerfen, Fachberichte Informatik 10-2005

Pascal Hitzler, Carsten Lutz, and Gerd Stumme: Foundational Aspects of Ontologies, Fachberichte Informatik 9-2005

Joachim Baumeister and Dietmar Seipel: Knowledge Engineering and Software Engineering, Fachberichte Informatik 8-2005

Benno Stein and Sven Meier zu Eißen: Proceedings of the Second International Workshop on Text-Based Information Retrieval, Fachberichte Informatik 7-2005

Andreas Winter and Jürgen Ebert: Metamodel-driven Service Interoperability, Fachberichte Informatik 6-2005

Joschka Boedecker, Norbert Michael Mayer, Masaki Ogino, Rodrigo da Silva Guerra, Masaaki Kikuchi, and Minoru Asada: Getting closer: How Simulation and Humanoid League can benefit from each other, Fachberichte Informatik 5-2005

Torsten Gipp and Jürgen Ebert: Web Engineering does profit from a Functional Approach, Fachberichte Informatik 4-2005

Oliver Obst, Anita Maas, and Joschka Boedecker: HTN Planning for Flexible Coordination Of Multiagent Team Behavior, Fachberichte Informatik 3-2005

Andreas von Hessling, Thomas Kleemann, and Alex Sinner: Semantic User Profiles and their Applications in a Mobile Environment, Fachberichte Informatik 2-2005

Heni Ben Amor and Achim Rettinger: Intelligent Exploration for Genetic Algorithms – Using Self-Organizing Maps in Evolutionary Computation, Fachberichte Informatik 1-2005