Semantic Knowledge Partitioning

Christoph Wernhard

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

Abstract. Some operations to decompose a knowledge base (considered as a first order logic formula) in ways so that only its semantics determines the results are investigated. Intended uses include the extraction of "parts" relevant to an application, the exploration and utilizing of implicit possibilities of structuring a knowledge base and the formulation of query answers in terms of a signature demanded by an application. A semantic framework based on Herbrand interpretations is outlined. The notion of "model relative to a scope" is introduced. It underlies the partitioning operations "projection" and "forgetting" and also provides a semantic account for certain formula simplification operations. An algorithmic approach which is based on resolution and may be regarded as a variation of the SCAN algorithm is discussed.

1 Introduction

We investigate some operations to decompose a knowledge base in ways so that solely its semantics determines the results. We regard a knowledge base as represented by a logic formula and confine ourselves here to classical propositional logic and classical first order clausal logic. A decomposition operation takes a formula as input and returns one or several other formulas that are in some sense "parts" of the original one. "Semantics" can be understood in the standard sense for these logics, here however a specific notion based on Herbrand interpretations is used. This fits with automated deduction techniques and is easily accessible to a full formalization which in turn may be automatically processed.

A knowledge base can be large, can integrate further knowledge bases and is possibly designed and maintained by someone else than its user. It should be accessed through an interface of operations that are characterized purely in terms of the semantics of the knowledge base and query expressions. Some types of such operations can be straightforwardly expressed as theorem proving tasks, others require different modes of reasoning such as model generation, decision procedures, abduction, partial deduction or knowledge compilation. Semantic knowledge partitioning can be understood as such a reasoning mode. Outputs can be again inputs to reasoning in this or some other mode.

Semantic knowledge partitioning should be a means to compute the following kinds of operations:

– Extraction of an application relevant portion from a large knowledge base.
– Exploration and utilizing of implicit possibilities of structuring a knowledge base: Computing definitional dependency of predicates and functions. Computing ways of decomposition into mutually independent sentences.
– Expressing a theorem in terms of a certain signature, which characterizes for example those terms which are "understood" in some way or handled efficiently by a human or machine client.
– Removing information from an answer for security reasons.

2 Notation and Preliminaries

We consider first order formulas in Skolemized clausal form: A *formula* (or *matrix*) is a set of clauses; a *clause* is a set of literals; a *literal* is a pair of an atom and a sign. If A is an atom, we write $+A$ $(-A)$ for the positive (negative) literal with atom A. The *dual* of a literal L is written $\overline{L}$. Occasionally we use logic connectives in formulas or formula schemas, which can be understood as straightforward abbreviations of the respective matrix form.

We use a semantic framework based on Herbrand interpretations. A *function signature* Σ is a finite set of function symbols $Name/Arity$ that includes at least one constant symbol. For a syntactic object F, $\Sigma(F)$ is the set of function symbols in F. The *base* of a formula F and a function signature $\Sigma \supseteq \Sigma(F)$, in symbols $\mathcal{B}(F, \Sigma)$ is the set of ground atoms obtained by instantiating all atoms in F with terms constructed from Σ. An *interpretation* over a base $\mathcal{B}$ is a set of ground literals that contains for each $A \in \mathcal{B}$ exactly one of $+A$ or $-A$ and no other elements. To avoid notational clutter we assume an implicit function signature that includes all function symbols in formulas we consider. Thus we speak of *the base of a formula* F, in symbols $\mathcal{B}(F)$. For interpretations we assume an implicit base that is a superset of the bases of all considered formulas.

A clause C *contradicts* an interpretation I if and only if there is a ground instance C' of C such that the dual of each literal in C' is in I. An interpretation I is a *model* for a formula F, in symbols $I \models F$, if and only if F contains no clause that contradicts I. A formula F *implies* a formula G, in symbols $F \Rightarrow G$, if and only if for all interpretations I holds that if $I \models F$ then $I \models G$. Two formulas F and G are *equivalent*, in symbols $F \Leftrightarrow G$, if and only if $F \Rightarrow G$ and $G \Rightarrow F$.

Definition 1 (Essential Base). The *essential base of a formula* F, in symbols $\mathcal{E}(F)$, is the set of all ground atoms A such that there exists an interpretation I for which exactly one of $(I - \{-A\}) \cup \{+A\} \models F$ or $(I - \{+A\}) \cup \{-A\} \models F$ does hold.

For example $\mathcal{E}(p \land (q \lor \neg q)) = \{p\}$. For a propositional logic formula F, $\mathcal{E}(F)$ is the unique smallest signature such that there exists a formula equivalent to F which has this signature. Intuitively the essential base of a formula may be regarded as the set of ground atoms about which the formula does "express something."

3 Model Relative to a Scope

A *scope* is a set of ground atoms. The notion of *model relative to a scope* is the central concept in this work. It is used to specify partitioning operations semantically. It also provides semantics for certain formula simplification which do not preserve equivalence.

Definition 2 (Equality of Interpretations Relative to a Scope). For all scopes S, and interpretations I and J, *I is equal to J relative to S*, in symbols $I =_S J$, if and only if I and J assign the same values to all atoms in S.

Definition 3 (Model Relative to a Scope). For all interpretations I, formulas F and scopes S, *I is a model of F relative to S*, in symbols $I \models_S F$, if and only if there exists an interpretation I' such that $I' \models F$ and $I' =_S I$ hold.

Definition 4 (Equivalence Relative to a Scope). For all formulas F, G and scopes S, *F and G are equivalent relative to S*, in symbols $F \Leftrightarrow_S G$, if and only if for all interpretations I holds that $I \models_S F$ if and only if $I \models_S G$.

The following proposition shows simple connections between the model relationship relative to a base, the general model relationship, the essential base of a formula and the subset relationships between scopes.

Proposition 1. *For all formulas F, scopes S, S' and interpretations I it holds that 1. if $I \models F$ then $I \models_S F$, 2. if $\mathcal{E}(F) \subseteq S$ and $I \models_S F$ then $I \models F$, 3. if $S' \subseteq S$ and $I \models_S F$ then $I \models_{S'} F$, 4. if $I \models_{\mathcal{E}(F)\cap S} F$ then $I \models_S F$.*

4 Projection and Forgetting

Projection and *forgetting* are the core operations for this approach to semantic knowledge partitioning. They are defined as ternary relationships $\mathsf{project}(F, S, F')$ and $\mathsf{forget}(F, S, F')$, where F and F' are formulas and S is a scope. F and S are considered as input and F' as result. Intuitively projection means that F' is equivalent to F with respect to the atoms in S but provides no information about other atoms (like a database relation can be "projected" onto a given set of attributes). Forgetting means intuitively that F' is equivalent to F, except that F' does not provide any information about the atoms in S.

Definition 5 (Projection). For all formulas F, F' and scopes S, F' is a *projection of F onto S*, in symbols $\mathsf{project}(F, S, F')$, if and only if for all interpretations I holds that $I \models F'$ if and only if $I \models_S F$.

Definition 6 (Forgetting). For all formulas F, F' and scopes S, F' is a forgetting about S in F, in symbols $\mathsf{forget}(F, S, F')$, if and only if $\mathsf{project}(F, \mathcal{E}(F) - S, F')$.

Projection and forgetting are dual to each other in the sense that also $\mathsf{project}(F, S, F')$ is equivalent to $\mathsf{forget}(F, \mathcal{E}(F) - S, F')$. Projection and forgetting are "semantic" operations, i.e. for all formulas F, G, F', G' and scopes S it holds that if $F \Leftrightarrow G$, $\mathsf{project}(F, S, F')$ and $\mathsf{project}(G, S, G')$ then $F' \Leftrightarrow G'$ (and similarly for forget).

The following propositions show alternate characterizations of projection and forgetting in terms of essential base and relative equivalence:

Proposition 2. *For all formulas F, F' and sets of atoms S holds* $\mathsf{project}(F, S, F')$ *if and only if* $\mathcal{E}(F') \subseteq S$ and $F \Leftrightarrow_S F'$.

Proposition 3. *For all formulas F, F' and scopes S holds* $\mathsf{forget}(F, S, F')$ *if and only if the following three conditions hold: 1. $F \Leftrightarrow_{\mathcal{E}(F)-S} F'$, 2. $\mathcal{E}(F') \cap S = \emptyset$ and 3. $\mathcal{E}(F') \subseteq \mathcal{E}(F)$.*

Only the intersection of the given scope with the essential base of the input formula is relevant for the results of projection and forgetting: For all formulas F, F' and scopes S holds: $\mathsf{project}(F, S, F')$ if and only if $\mathsf{project}(F, \mathcal{E}(F) \cap S, F')$ (and similarly for forget). Since for all formulas F $\mathcal{E}(F) \subseteq \mathcal{B}(F)$, in the mutual characterizations of one operator in terms of the other instead of the $\mathcal{E}$ operator also the $\mathcal{B}$ operator can be used, which may be easier to handle computationally. For example $\mathsf{project}(F, S, F')$ holds if and only if $\mathsf{forget}(F, \mathcal{B}(F) - S, F')$ holds.

The following proposition related to proposition 3 is useful to show correctness of methods that compute forgettings if they do not introduce new predicate or function symbols to the result formula:

Proposition 4. *For all formulas F, F' and scopes S it holds that if $\mathcal{E}(F') \subseteq \mathcal{B}(F)$, $F \Leftrightarrow_{\mathcal{B}(F)-S} F'$ (which strengthens condition (1) in proposition 3) and condition (2) in that proposition are satisfied, then also condition (3) in that proposition holds.*

Example 1. Consider the matrix $F = \{\{\mathsf{p}(\mathsf{a})\},\ \{\neg\mathsf{p}(x),\ \mathsf{q}(x)\},\ \{\neg\mathsf{q}(x),\ \mathsf{r}(x)\}\}$ along with the function signature $\Sigma = \{\mathsf{a}/0, \mathsf{b}/0, \mathsf{c}/0\}$. The following table shows the results of forgettings about the set of all ground atoms with predicate symbol q and about the unit sets of the atoms $\mathsf{q}(\mathsf{a})$, $\mathsf{p}(\mathsf{a})$, $\mathsf{p}(\mathsf{b})$ and $\mathsf{r}(\mathsf{a})$ respectively:

$$
\begin{aligned}
\mathsf{q} &: \{\{\mathsf{p}(\mathsf{a})\}, \{\neg\mathsf{p}(x),\ \mathsf{r}(x)\}\}\\
\mathsf{q}(\mathsf{a}) &: \{\{\mathsf{p}(\mathsf{a})\}, \{\neg\mathsf{p}(\mathsf{a}),\ \mathsf{r}(\mathsf{a})\}, \{\neg\mathsf{p}(\mathsf{b}),\ \mathsf{q}(\mathsf{b})\}, \{\neg\mathsf{p}(\mathsf{c}),\ \mathsf{q}(\mathsf{c})\},\\
&\quad \{\neg\mathsf{q}(\mathsf{b}),\ \mathsf{r}(\mathsf{b})\}, \{\neg\mathsf{q}(\mathsf{c}),\ \mathsf{r}(\mathsf{c})\}\}\\
\mathsf{p}(\mathsf{a}) &: \{\{\mathsf{q}(\mathsf{a})\}, \{\neg\mathsf{p}(\mathsf{b}),\ \mathsf{q}(\mathsf{b})\}, \{\neg\mathsf{p}(\mathsf{c}),\ \mathsf{q}(\mathsf{c})\}, \{\neg\mathsf{q}(x),\ \mathsf{r}(x)\}\}\\
\mathsf{p}(\mathsf{b}) &: \{\{\mathsf{p}(\mathsf{a})\}, \{\neg\mathsf{p}(\mathsf{a}),\ \mathsf{q}(\mathsf{a})\}, \{\neg\mathsf{p}(\mathsf{c}),\ \mathsf{q}(\mathsf{c})\}, \{\neg\mathsf{q}(x),\ \mathsf{r}(x)\}\}\\
\mathsf{r}(\mathsf{a}) &: \{\{\mathsf{p}(\mathsf{a})\}, \{\neg\mathsf{p}(x),\ \mathsf{q}(x)\}, \{\neg\mathsf{q}(\mathsf{b}),\ \mathsf{r}(\mathsf{b})\}, \{\neg\mathsf{q}(\mathsf{c}),\ \mathsf{r}(\mathsf{c})\}\}
\end{aligned}
$$

Example 2. In this example a toy knowledge base in description logic is given. Forgetting is applied to the matrix of its standard translation. The results are then converted back to description logic. The input knowledge base contains the following axioms:

$$\begin{aligned} \mathsf{document} &\doteq \mathsf{text} \sqcup \mathsf{image}. \\ \mathsf{web\_document} &\doteq \mathsf{document} \sqcap \mathsf{has\_uri}. \\ \mathsf{wd}_1, \mathsf{wd}_2, \mathsf{wd}_3 &: \mathsf{web\_document}. \end{aligned}$$

The result of forgetting $\mathsf{document}$ is:

$$\begin{aligned} \mathsf{web\_document} &\doteq (\mathsf{text} \sqcup \mathsf{image}) \sqcap \mathsf{has\_uri}. \\ \mathsf{wd}_1, \mathsf{wd}_2, \mathsf{wd}_3 &: \mathsf{web\_document}. \end{aligned}$$

The result of forgetting $\mathsf{web\_document}$ is:

$$\begin{aligned} \mathsf{document} &\doteq \mathsf{text} \sqcup \mathsf{image}. \\ \mathsf{wd}_1, \mathsf{wd}_2, \mathsf{wd}_3 &: \mathsf{document} \sqcap \mathsf{has\_uri}. \end{aligned}$$

The result of forgetting $\mathsf{web\_document}(\mathsf{wd}_1)$ is:

$$\begin{aligned} \mathsf{document} &\doteq \mathsf{text} \sqcup \mathsf{image}. \\ \mathsf{wd}_1 &: \mathsf{document} \sqcap \mathsf{has\_uri}. \\ \mathsf{wd}_2, \mathsf{wd}_3 &: \mathsf{web\_document} \sqcap \mathsf{document} \sqcap \mathsf{has\_uri}. \end{aligned}$$

The result of forgetting $\mathsf{has\_uri}$ is:

$$\begin{aligned} \mathsf{document} &\doteq \mathsf{text} \sqcup \mathsf{image}. \\ \mathsf{web\_document} &\sqsubseteq \mathsf{document}. \\ \mathsf{wd}_1, \mathsf{wd}_2, \mathsf{wd}_3 &: \mathsf{web\_document}. \end{aligned}$$

5 Forgetting for Propositional Logic

For propositional formulas F and atoms A a function $\mathsf{forget}(F, A)$ such that $\mathsf{forget}(F, \{A\}, \mathsf{forget}(F, A))$ holds can be defined as $\mathsf{forget}(F, A) = F[A\backslash\mathsf{true}] \vee F[A\backslash\mathsf{false}]$ where $F[A\backslash G]$ denotes F with all occurrences of A substituted by G.

Algorithm 1 (DNF-Forget).
INPUT: A propositional formula F and propositional atom A.
OUTPUT: A formula F' such that $\mathsf{forget}(F, \{A\}, F')$ holds.
METHOD: Convert F to non-complementary disjunctive clausal normal form (NC-DNF).[1] Remove all literals with atom A from these clauses and return the corresponding propositional formula.

It is easy to see that applying the schema $F[A\backslash\mathsf{true}] \vee F[A\backslash\mathsf{false}]$ to a formula F in NC-DNF amounts to the method of the algorithm. Correctness of the algorithm can also be easily shown by considering F converted to NC-DNF, since for all formulas F in this form and interpretations I it holds that $I \models F$ if and only if there exists a clause $C \in F$ such that $C \subseteq I$. So it is not hard to prove $F \Leftrightarrow_{\mathcal{B}(F)-\{A\}} \mathsf{forget}(F, A)$. Hence condition (1) of Proposition 3 holds. Condition (2) does obviously hold and condition (3) follows from Proposition 4.

Another way to compute forgettings for propositional formulas is algorithm $\mathsf{pattern\text{-}forgetting}$ which will be described in Sect. 7. For propositional inputs the preprocessing steps of that algorithm have no effect and can be skipped.

[1] *Non-complementary* means that no clause contains a literal and its dual.

6 Properties of and Constructions with Projection

In this section we outline some applications of projection together with the properties and constructions underlying them. We assume that the respective projections can be computed and hence write the projection operator as a binary function, as we did in the section on forgetting for propositional logic.

Justification of Knowledge Base Preprocessing. The following proposition ensures that $\mathsf{project}(F, \mathcal{B}(G))$ can be regarded as a knowledge base extract that can be used to answer all queries whose base is a subset of $\mathcal{B}(G)$:

Proposition 5. *For all formulas F and G it holds that $F \Rightarrow G$ if and only if* $\mathsf{project}(F, \mathcal{B}(G)) \Rightarrow G$.

Justification of Cheap Decomposition. A decomposable negation normal form (DNNF) is a negation normal form in which no atom is shared between any two conjuncts [6]. One of the motivations for using a DNNF as a target language for knowledge compilation is that for a DNNF projection on a set atoms can be computed in linear time [6], which can be seen from the following propositions:

Proposition 6. *For all formulas F, G and scopes S holds*
(1) $\mathsf{project}(F \lor G, S) \Leftrightarrow \mathsf{project}(F, S) \lor \mathsf{project}(G, S)$,
(2) *If* $\mathcal{B}(F) \cap \mathcal{B}(G) = \emptyset$ *then* $\mathsf{project}(F \land G, S) \Leftrightarrow \mathsf{project}(F, S) \land \mathsf{project}(G, S)$.

Preservation of Satisfiability. The next proposition implies that a formula F is satisfiable if and only if $\mathsf{project}(F, \emptyset)$ is satisfiable. This suggests that such methods for computing projection which ensure that the base of their result is a subset of the projection scope — and hence literally return true or false if the scope is empty — can be used to compute satisfiability.[2]

Proposition 7. *For all formulas F and scopes S it holds that F is satisfiable if and only if* $\mathsf{project}(F, S)$ *is satisfiable.*

Definability and Construction of Definitions. If F is formula, A a ground atom and S is a set of ground atoms not containing A, then A is called *definable in terms of S within F* if and only if F implies a formula $A \leftrightarrow D$ where $\mathcal{B}(D) \subseteq S$. Definability can be expressed using projection: A is definable in terms of S within F if and only if $\mathsf{project}(F \land A,\ S) \land \mathsf{project}(F \land \neg A,\ S)$ is unsatisfiable.

[2] Actually for a propositional clause set F, forgetting about the atoms in $\mathcal{B}(F)$ one by one using the replace-by-resolvents algorithm described in Sect. 7 combined with restricted application of the equivalence preserving simplifications *removal of subsumed clauses* and *subsumption resolution* amounts to the Davis Putnam method [7]. The *rule for eliminating atomic formulas* and, as will be shown in Sect. 7.5, the *affirmative-negative rule* both correspond to applications of replace-by-resolvents.

If definability holds for F,A and S, then definitions can be constructed in two ways: 1. $A \leftrightarrow \mathsf{project}(F \wedge A,\ S)$ and 2. $A \leftrightarrow \neg\mathsf{project}(F \wedge \neg A,\ S)$. Both constructed definitions may not be logically equivalent. Consider for example $F = (\mathsf{p} \leftrightarrow \mathsf{q} \wedge \mathsf{r}) \wedge (\mathsf{q} \vee \mathsf{r})$. These two definitions of p in terms of $\{\mathsf{q}, \mathsf{r}\}$ are constructed: 1. $\mathsf{p} \leftrightarrow \mathsf{q} \wedge \mathsf{r}$ and 2. $\mathsf{p} \leftrightarrow (\mathsf{q} \leftrightarrow \mathsf{r})$.

Lossless Decomposition. A set of scopes $\{S_1, S_2, ..., S_n\}$ is a n-ary lossless decomposition of F iff $\bigwedge_{i=1..n} \mathsf{project}(F, S_i) \Rightarrow F$.

7 Computation of Forgetting by Resolution

In propositional logic the forgetting about an atom can — as an alternative to Algorithm 1 — be computed by successively replacing all clauses containing a literal with the atom to be forgotten by the resolvents upon that literal. In this section we discuss an approach to transfer this method to first order matrices. Also, instead of forgetting about a single ground atom, the method for first order logic allows to forget about all ground instances of a given pattern. The pattern has to be *linear*,[3] since a certain clause instantiation procedure is involved, which, as we will see in Sect. 7.2, is guaranteed to have finite solutions only for linear patterns.

Before we discuss the sub-procedures of the method in detail we give an overview of the overall procedure.

Algorithm 2 (Pattern Forgetting).
INPUT: A first order matrix F, a linear atom P (the "pattern") and a function signature $\Sigma \supseteq \Sigma(F) \cup \Sigma(P)$.
OUTPUT: A matrix that is the forgetting about all ground instances of P in F, or an "approximation" of the forgetting, as will be discussed in Sect. 7.6.
METHOD:

1. Preprocessing: F is assigned the result of applying *diff* expansion and *break* transformation to the input arguments.
2. Core loop: While F contains a clause C with a literal L whose atom is an instance of P, assign to F the result of eliminating C with $\mathsf{replace\text{-}by\text{-}resolvents}$ applied to F, C and L. Between the rounds of this loop equivalence preserving *simplifications* may be performed on F.
3. Postprocessing: Try to reduce F and to remove auxiliary literals introduced by the *break* transformation with *simplifications*. Output the modified F.

In the following subsections we discuss the sub-procedures of this algorithm, starting with the core procedure.

[3] An atom is *linear* if and only if it does not contain multiple occurrences of the same variable.

7.1 Replace-by-Resolvents

Algorithm 3 (Replace-by-Resolvents).
INPUT: A first order matrix F, clause C and literal L with $L \in C$ and $C \in F$.
OUTPUT: If the algorithm terminates, the matrix F with C replaced by the saturation of resolvents upon L.
METHOD: While a resolvent[4] D upon L and C with some clause from F exists such that D is not tautological and not subsumed by $F - \{C\}$ add D to F. Output $F - \{C\}$.

Theorem 1. *For all matrices F, clauses C and literals L with $L \in C$ and $C \in F$ it holds that if each resolvent upon literal L and clause C with a clause in F is tautological or subsumed by a clause in $F - \{C\}$ then, for all scopes S not containing an instance of the atom of L holds $F \Leftrightarrow_S F - \{C\}$.*

For propositional matrixes this theorem can be proved straightforwardly by properties of literals, clauses and interpretations. For first order matrixes an inductive proof over an enumeration of ground instances C_i, $i \geq 1$ of C can be given: Let F_0 be $F - \{C\}$ and F_n be $F_{n-1} \cup \{C_n\}$ for $n \geq 1$. The induction step ensures that $F_{n+1} \Leftrightarrow_S F_n$ for all $n \geq 0$ by means of the theorem for propositional matrices and the *lifting lemma*.[5]

7.2 Preprocessing Operation: *Diff* Expansion

The input matrix for a pattern forgetting operation can contain literals with atoms that are unifiable with the pattern but not instances of it. replace-by-resolvents can not be directly applied to such a matrix to forget the instances of a pattern. If the pattern is linear, however any matrix can be transformed into a form without such literals, by *diff* expansion:[6]

Algorithm 4 (Diff-Expansion).
INPUT: A first order matrix F, a linear atom P that does not share variables with F and a function signature $\Sigma \supseteq \Sigma(F) \cup \Sigma(P)$.
OUTPUT: A finite set of instances of clauses of F that has the same Herbrand expansion as F over Σ and does not contain a literal L such that the atom A of L is unifiable with P and $A \not\geq_s P$.

---

[4] *Resolvent* is understood as in [5]. We extend the notion of *literals resolved upon* from binary resolvents to resolvents: A clause C is a resolvent of clause C_1 with clause C_2 *upon* literal L_1, if $L_1 \in C_1$ and C is a binary resolvent of C_1 upon L_1 or C is a binary resolvent of a factor $C_1\sigma$ of C_1 upon $L_1\sigma$.

[5] We need here a slightly strengthened version of the lifting lemma in [5] in which we also consider a *literal resolved upon*: If C'_1, $L'_1 \in C_1$ and C'_2 are instances of C_1, $L_1 \in C_1$ and C_2 respectively, and if C' is a resolvent of C'_1 upon L'_1 and C'_2, then there is a resolvent C of C_1 upon L_1 and C_2 such that C' is an instance of C.

[6] In this subsection we use the following notation: $A \geq_s B$ stands for A *is an instance of* B. $\circ$ denotes composition of substitutions, where the substitution on the left side is applied first. Application of substitution is expressed by postfix notation. $\mathsf{mgu}(A, B)$ is the most general unifier of A and B. $\mathsf{vars}(A)$ is the set of variables in A.

METHOD: While F contains a clause C with a literal L such that the atom A of L is unifiable with P and $A \not\geq_s P$ replace C in F by $C\sigma$ for all $\sigma \in$ diff-substitutions(A, P, Σ). Output the modified matrix F.

Algorithm 5 (Diff-Substitutions).
INPUT: An atom A, a linear atom P that does not share variables with A and is unifiable with A and a function signature $\Sigma \supseteq \Sigma(A) \cup \Sigma(P)$.
OUTPUT: A set of substitutions S such that 1. there is no $\sigma \in S$ such that $A\sigma$ is unifiable with P and $A\sigma \not\geq_s P$ and 2. for all clauses C with $\Sigma(C) \subseteq \Sigma$ it holds that the Herbrand expansions of C and $\{C\sigma \mid \sigma \in S\}$ over Σ are identical.
METHOD: Let S be a set of substitutions initialized to $\{\epsilon\}$.

While S contains a substitution σ such that $A\sigma$ is unifiable with P and $A\sigma \not\geq_s P$ do the following steps:

- Choose a variable X from $\mathsf{vars}(A\sigma)$ such that $(X)\mathsf{mgu}(A\sigma, P)$ is not a variable. (The existence of such a X follows from the while condition.)
- Remove σ from S.
- For each function symbol f/n in Σ generate n fresh variables $X_1, X_2, ..., X_n$ and add $\sigma \circ \{X \mapsto f(X_1, X_2, ..., X_n)\}$ to S.

Output S.

Actually in each round there is exactly one element $A\sigma \in S$ that is unifiable with P. The algorithm may produce different outputs depending on the choice of X in each round. The cardinality of the result of the algorithm applied to inputs A, P and Σ is $1 + (|\Sigma| - 1) * \sum_{X \in \mathsf{vars}(A)} |\mathsf{ocfs}((X)\mathsf{mgu}(A, P))|$ where $|\mathsf{ocfs}(T)|$ is the number of occurrences of function symbols in term T.

Example 3. Consider $F = \{\{\mathsf{p}(x, y)\}\}$, $P = \mathsf{p}(\mathsf{f}(u), \mathsf{a})$ and $\Sigma = \{\mathsf{a}/0,\ \mathsf{b}/0,\ \mathsf{f}/1\}$. A *diff* expansion is $\{\{\mathsf{p}(\mathsf{a}, x)\}, \{\mathsf{p}(\mathsf{b}, x)\}, \{\mathsf{p}(\mathsf{f}(x), \mathsf{a})\}, \{\mathsf{p}(\mathsf{f}(x), \mathsf{b})\}, \{\mathsf{p}(\mathsf{f}(x), \mathsf{f}(y))\}\}$.

For a non-linear pattern P a finite set of substitutions satisfying the output conditions of Algorithm 5 may not exist. Consider for example $A = \mathsf{p}(x, y)$, $P = \mathsf{p}(u, u)$, $\Sigma = \{\mathsf{a}/0, \mathsf{f}/1\}$. Among the required instances of clause $\{\mathsf{p}(x, y)\}$ there are all $\{\mathsf{p}(\mathsf{f}^n(\mathsf{f}(x_n)), \mathsf{f}^n(\mathsf{a})))\}$ for $n \geq 0$.

An alternative to the *diff* expansion transformation would be the use of disequality constraints, which we do not pursue here, since it has two features that we want to avoid: Symbols to be "forgotten" are retained "negatively" in the constraints and the straightforward framework of first order logic is left.

7.3 Preprocessing Operation: *Break* Transformation

replace-by-resolvents terminates if each clause contains only a single literal whose atom is weakly unifiable with L. Any matrix can be brought into this form by replacing clauses with more than one such literal with fragments of them that are linked with auxiliary definitional literals, for example if $\mathsf{p}(X)$ is the pattern, the clause $\{\neg\mathsf{p}(Y), \mathsf{p}(\mathsf{f}(Y))\}$ can be replaced by the clauses $\{\neg\mathsf{p}(Y), \neg\mathsf{break}_1(Y)\}$

and $\{\mathsf{break}_1(Y), \mathsf{p}(\mathsf{f}(Y))\}$ where break_1 is a new auxiliary predicate whose arguments are the variables shared between the two fragments of the original clause. The original and the transformed matrix are equivalent relative to scopes not containing instances of "*break* atoms", i.e. atoms of these auxiliary literals.

If the *break* transformation is applied before *diff* expansion, the possible generation of an exponential number of clause instances at the *diff* expansion phase is avoided.

7.4 Simplifications

In the context of theorem proving, operations that can be performed fast (typically in linear time), can be applied only a number of times that is polynomial (typically linear) in the size of the formula and reduce the refutation task in some sense while preserving satisfiability are known as *simplifications* or *reductions* [4].

Some simplifications preserve the equivalence of the input formula, for example *removal of tautological clauses*, *removal of subsumed clauses*, *subsumption resolution* and *condensation*. Other simplifications are commonly applied in theorem proving since they preserve satisfiability. A closer look at some of them however shows that they actually preserve equivalence relative to certain scopes which justifies that they can be used for knowledge base preprocessing and model generation.

For the pattern forgetting method discussed in this section simplifications play a twofold role: Equivalence preserving simplifications are applied to reduce the results and intermediate results of replace-by-resolvents. Along with equivalence preserving simplifications, simplifications that preserve all scopes that do not include *break* atoms are applied to delete *break* literals as far as possible from the final output of the forgetting algorithm.

7.5 Simplifications Preserving Equivalence Relative to Certain Scopes

We first consider two special cases of replace-by-resolvents. Let $L \in C$ be the literal occurrence to be resolved upon. L is called *pure*, if the matrix does not contain a literal that is weakly unifiable[7] with $\overline{L}$. The effect of replace-by-resolvents upon a pure L is just that C is deleted. This is a first order generalization of the *pure literal rule* (originally called *affirmative-negative rule*) of the Davis Putnam method. Another special case is that a first application of replace-by-resolvents replaces C by resolvents with partner clauses which then in turn can be removed by the purity simplification just discussed. This is a first order generalization of the *ISOL* and *ISOL** reductions in [3]. That they can be described as special cases of replace-by-resolvents shows that these simplifications do not only preserve satisfiability of the input matrix but also equivalence relative to all scopes not containing instances of the literals resolved upon.

[7] Two atoms are *weakly unifiable* if and only if they are unifiable after their variables have been standardized apart.

An *extended purity reduction rule* is given in [13]: If the matrix does not contain two clauses such that literals with a predicate P occurs only positively in one and only negatively in the other, then all clauses containing a literal with predicate P are deleted. This simplification preserves equivalence relative to all scopes that do not contain atoms with predicate P.

A further simplification is the *deletion of an isolated satisfiable clause group*: If the input matrix can be split into two disjoint subsets F_1 and F_2 such that there are no weakly unifiable literals L_1 and L_2 with L_1 in F_1 and $\overline{L_2}$ in F_2, then if F_1 is satisfiable, the clauses of F_1 are deleted. The result matrix preserves equivalence relative to all scopes not including the instance of an atom from F_1.

7.6 Correctness of the Pattern Forgetting Algorithm

Algorithm pattern-forgetting terminates for all inputs — the *diff* and *break* transformations terminate, replace-by-resolvents terminates since the *break* transformation has been applied and simplifications naturally do terminate.

If the result of applying algorithm pattern-forgetting to F and P does not contain any *break* literals, it is indeed the supposed forgetting: *diff* expansion preserves equivalence of the matrix in the sense that it preserves its Herbrand expansion. The *break* transformation preserves equivalence with respect to all scopes not including *break* atoms. For replace-by-resolvents, Theorem 1 ensures condition (1) of Proposition 3, condition (2) is obviously satisfied. That also condition (3) is satisfied can be shown with the help of Proposition 4. Finally simplifications in the postprocessing phase are either preserving equivalence or equivalence relative to all scopes not containing *break* atoms.

If *break* literals remain in the result, then condition (3.) might be violated. Depending on the application such results might nevertheless be useful.[8]

8 Related Work

Algorithm pattern-forgetting is similar to the core part of the SCAN algorithm, in which clauses are "resolved away" by C-resolution [8]. This algorithm goes back to W. Ackermann [1]. Similar algorithms and a completeness property of SCAN are described in [12, 13, 9]. SCAN was motivated by applications in correspondence theory for modal logics. Also interpolation (i.e. knowledge base preprocessing) and the computing of circumscription are shown as applications in [8].

C-factoring (C-resolution) with *constraint elimination* [9] corresponds to "standard" factoring (binary resolution) [5] if the matrix for the standard operations includes predicate substitutivity axioms for $\approx$: For each n-ary predicate Q a clause $\{\neg Q(x_1, ..., x_n),\, x_1 \not\approx y_1, ..., x_n \not\approx y_n,\, Q(y_1, ..., y_n)\}$ can be derived by binary resolution. As can easily be seen, a C-factoring (C-resolution) step then

[8] Our characterization of forgetting is not sufficient to distinguish "solutions" with residual *break* literals that are in some sense "necessary" from trivial solutions by just renaming the predicate to be forgotten. This is an issue for future work.

corresponds to a binary resolution step with such a clause followed by a factoring (binary resolution) step. A factoring (binary resolution) step corresponds to a C-factoring (C-resolution) step followed by an application of the *constraint elimination* rule.

We mention some apparent differences between pattern-forgetting and SCAN, but the details and practical relevance of them have not yet been worked out: Since pattern-forgetting can also be applied to matrixes without predicate substitutivity axioms, it may have a broader range of applicability. pattern-forgetting can not only be used to forget about predicates, but also to forget about instances of patterns. By *diff* expansion, in the result signature of pattern-forgetting function symbols that are to be "forgotten" do not appear, but on the other hand this result signature is related to the "absolute signature" which is input at *diff* expansion. By the introduction of *break* literals in pattern-forgetting, processing is separated into two terminating phases. Where SCAN does not terminate, the output of pattern-forgetting does include residual *break* literals. By the use of arbitrary decision procedures in simplification *deletion of an isolated satisfiable clause group* pattern-forgetting may terminate for more inputs than SCAN. In SCAN only C-resolutions between *different* clauses are allowed. The condition "tautological or subsumed by a clause in $F - \{C\}$" in Theorem 1 may be related to that restriction, but the precise relationship has not yet been worked out.

Lin and Reiter [11] describe the semantics of forgetting about ground atoms and predicates for first order logic in standard model theoretic terms. Their work is motivated by the need of an autonomous agent to "forget" knowledge about past states.

There are a number of more or less related works with different backgrounds: In the context of knowledge compilation, applications of propositional projection to knowledge base preprocessing, planning, model-based diagnosis and circuit compilation are shown [6]. Aside from disjunctive normal form and DNNF [6], ordered binary decision diagrams provide a normal form in which propositional forgetting is an operation of low complexity. Propositional forgetting corresponds to existential boolean quantification. Partition-based logical reasoning [2] is a framework for distributed theorem proving based on consequence finding in signature restricted knowledge base partitions. Projection can be considered as a special case of marginalization, one of the fundamental operations in the "local computation" framework [10, 14].

9 Next Steps

We consider the resolution based pattern forgetting algorithm as a first approach to get a deductive grip on projection/forgetting for first order logic and will investigate further techniques. One direction there are adapted model generation methods, which generate implicants of the results of projection one by one and thus can be plugged to client systems that process them one by one.

Properties and constructions like those in Sect. 6 have to be further worked out along with example scenarios.

For propositional logic $\mathsf{project}(F, S)$ corresponds to $\Diamond_S F$ in a multi-modal logic where $=_S$ (see Definition 2) is the accessibility relation for modality S. This approach to projection and the possibility of using processing methods for modal logics have to be investigated.

We intend to explore possibilities of making projection/forgetting available to other languages that can be embedded in first order logic, as we have outlined in Example 2 for a description logic.

Instead of sets of atoms as scopes, also sets of literals could be used (corresponding to the Lyndon interpolation theorem). This may be useful for knowledge base preprocessing, since the knowledge base extract could be significantly smaller and in queries often predicates are used only with a single polarity.

References

1. W. Ackermann. Untersuchungen über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110:390–413, 1935.
2. E. Amir and S. A. McIlraith. Partition-based logical reasoning. In *Princ. of Knowl. Repr. and Reasoning*, pages 389–400, 2000.
3. W. Bibel. *Deduktion — Automatisierung der Logik*. Oldenbourg, München, 1992.
4. W. Bibel and E. Eder. Methods and calculi for deduction. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artif. Int. and Logic Programming*, volume 1, chapter 3, pages 67–182. Oxford University Press, 1993.
5. C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Boston, 1973.
6. A. Darwiche. Decomposable negation normal form. *JACM*, 48(4):608–647, 2001.
7. M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM*, 7(3):201–215, 1960.
8. D. Gabbay and H. J. Ohlbach. Quantifier elimination in second-order predicate logic. In *KR'92. Princ. of Knowl. Repr. and Reasoning*, pages 425–435, 1992.
9. V. Goranko, U. Hustadt, R. A. Schmidt, and D. Vakarelov. SCAN is complete for all Sahlqvist formulae. In *Relational and Kleene-Algebraic Methods in Computer Science (RelMiCS 7)*, pages 149–162, 2004.
10. J. Kohlas, R. Haenni, and S. Moral. Propositional information systems. *Journal of Logic and Computation*, 9(5):651–681, 1999.
11. 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.
12. A. Nonnengart, H. J. Ohlbach, and A. Szałas. Quantifier elimination for second-order predicate logic. In H. Ohlbach and U. Reyle, editors, *Logic, Language and Reasoning. Essays in Honor of Dov Gabbay, Part I*, pages 159–181. Kluwer, 1999.
13. A. Szałas. On the correspondence between modal and classical logic: An automated approach. *Journal of Logic and Computation*, 3(6):605–620, 1993.
14. N. Wilson and J. Mengin. Logical deduction using the local computation framework. In *Int. Workshop on First-Order Theorem Proving, FTP'97*, RISC-Linz Report Series No. 97–50, pages 135–139. Johannes Kepler Universität, Linz, 1997.