Blocking and Other Enhancements for Bottom-Up Model Generation Methods

Peter Baumgartner[1] and Renate A. Schmidt[2]

[1] National ICT Australia (NICTA)
Peter.Baumgartner@nicta.com.au
[2] The University of Manchester
schmidt@cs.man.ac.uk

Abstract. In this paper we introduce several new improvements to the bottom-up model generation (BUMG) paradigm. Our techniques are based on non-trivial transformations of first-order problems into a certain implicational form, namely range-restricted clauses. These refine existing transformations to range-restricted form by extending the domain of interpretation with new Skolem terms in a more careful and deliberate way. Our transformations also extend BUMG with a blocking technique for detecting recurrence in models. Blocking is based on a conceptually rather simple encoding together with standard equality theorem proving and redundancy elimination techniques. This provides a general-purpose method for finding small models. The presented techniques are implemented and have been successfully tested with existing theorem provers on the satisfiable problems from the TPTP library.

1 Introduction

The bottom-up model generation (BUMG) paradigm encompasses a wide family of calculi and proof procedures that explicitly try to construct a model of a given (first-order) clause set by reading clauses as rules and applying them in a bottom-up way until completion. For instance, variants of hyperresolution and certain tableau calculi belong to this family. BUMG methods have been known for a long time to be refutationally complete. Comparably little effort has however been undertaken to exploit them for the dual task of refutational theorem proving, namely, computing models for satisfiable problems. This is somewhat surprising, as computing models is recognized as being important in software engineering, model checking, and other applications, and is becoming increasingly important for building and maintaining web ontologies. The BUMG methods we develop and study in this paper are intended to be used for consistency testing of ontologies and software specifications, and for aiding with the debugging through the generation of (counter-)models. Our techniques are partially inspired by techniques already successfully used in the area. For instance, we show how blocking techniques of description and modal logic tableau-based theorem provers can be generalized to full first-order logic. In our approach blocking is encoded on the clausal level and is combined with standard resolution techniques. In this way, suitable provers can be utilised to construct (small) models which can be easily read off from the derived clauses. Our other contributed techniques are significant improvements to the well-known "transformation to range-restricted form" as introduced in the context of the SATCHMO prover

U. Furbach and N. Shankar (Eds.): IJCAR 2006, LNAI 4130, pp. 125–139, 2006.

in the eighties [15] and later improved in e.g. [4]. The existing transformations have the disadvantage that they force BUMG methods to enumerate the entire Herbrand universe and are therefore non-terminating except in the simplest cases. Our method extends and combines the transformation introduced in [21] for reducing first-order formulae and clauses into range-restricted clauses, which was used to develop general-purpose resolution decision procedures for the Bernays-Schönfinkel class. Our approach is similar in spirit to the methods in e.g. [10,13], by capitalizing on available *first-order* (equational) automated reasoning technology.

Other methods for model computation can be classified as methods that directly search for a finite model, like the extended PUHR tableau method [7], the method in [6] and the methods in the SEM-family [22,26,17]. In contrast, MACE-style model builders [9,16, e.g.] reduce model search to testing of propositional satisfiability. Being based on translation, the MACE-style approach is conceptually related, but different, to our approach. Both SEM- and MACE-style methods search for finite models, essentially, by searching the space of interpretations with domain sizes $1, 2, \ldots$, in increasing order, until a model is found. Our method operates significantly differently, as it is *not* parametrized by a domain size. Consequently, there is no iterative deepening over the domain size, and the search for finite models works differently. This way, we address a problem often found with models computed by these methods: from a pragmatic perspective, they tend to identify too many terms. For instance, for the two unit clauses $\mathsf{P}(\mathsf{a})$ and $\mathsf{Q}(\mathsf{b})$ there is a model that identifies a and b with the same object. Such models can be counterintuitive, for instance, in a description logic setting, where unique names are often assumed. Our transformations are careful at identifying objects than the methods mentioned and thus work closer to a Herbrand semantics. The difference in operation also shows up experimentally. Our methods can solve an overlapping, but disjoint set of the satisfiable TPTP problems solvable by the state-of-the-art MACE-style model builder Paradox.

The structure of the paper is as follows. Sections 1.1 and 2 give basic definitions and recall the characteristic properties of BUMG methods. Section 3 defines new techniques for generating small models and generating them more efficiently. The techniques are based on a series of transformations which include an improved range-restricting transformation (Section 3.1), instances of standard renaming and flattening (Section 3.2), and the introduction of blocking through an encoding and standard saturation-based equality reasoning (Section 3.3). In Section 4 we present and discuss results of experiments carried out with our methods on all satisfiable TPTP problems. Details and proofs, which are omitted due to lack of space, can be found in the long version [5].

1.1 Preliminaries

We use standard terminology from automated reasoning. We assume as given a signature $\Sigma = \Sigma_f \cup \Sigma_P$ of function symbols Σ_f (including constants) and predicate symbols Σ_P. As we are working (also) with equality, we assume Σ_P contains a distinguished binary predicate symbol $\approx$, which is used in infix form. Terms, atoms, literals and formulas over Σ and a given (denumerable) set of variables V are defined as usual.

A clause is a (finite) implicitly universally quantified disjunction of literals. We write clauses in a logic-programming style, i.e. we write $H_1 \vee \cdots \vee H_m \leftarrow B_1 \wedge \cdots \wedge B_k$ rather than $H_1 \vee \cdots \vee H_m \vee \neg B_1 \vee \cdots \vee \neg B_k$, where $m, k \geq 0$. Each H_i is called a *head atom*,

and each B_j is called a *body atom*. When writing expressions like $H \vee \mathcal{H} \leftarrow B \wedge \mathcal{B}$ we mean any clause whose head literals are H and those in the disjunction of literals $\mathcal{H}$, and whose body literals are B and those in the conjunction of literals $\mathcal{B}$. A *clause set* is a finite set of clauses. A clause $\mathcal{H} \leftarrow \mathcal{B}$ is said to be *range-restricted* iff the body $\mathcal{B}$ contains all the variables in it. A clause set is range-restricted iff it contains only range-restricted clauses. For a given atom $P(t_1,\ldots,t_n)$ the terms $t_1,\ldots,t_n$ are also called the *top-level* terms of $P(t_1,\ldots,t_n)$ (P being $\approx$ is permitted). This notion generalizes to clause bodies, clause heads and clauses as expected. E.g., for a clause $\mathcal{H} \leftarrow \mathcal{B}$ the top-level terms of its body $\mathcal{B}$ are exactly the top-level terms of its body atoms. A *proper functional term* is a term which is neither a variable nor a constant.

With regards to semantics, we use the notions of (first-order) *satisfiability* and *E-satisfiability* in a completely standard way.

2 BUMG Methods

Proof procedures based on model generation approaches establish the satisfiability of a problem by trying to build a model for the problem. In this paper we are interested in bottom-up model generation approaches (BUMG). BUMG approaches use a forward reasoning approach where implications, or clauses, $\mathcal{H} \leftarrow \mathcal{B}$ are read as rules and are repeatedly used to derive (instances of) $\mathcal{H}$ from (instances of) $\mathcal{B}$ until a completion is found. The family of BUMG includes many familiar calculi and proof procedures, such as SATCHMO [15,12], PUHR [8,7], MGTP [11] and hyper tableaux [3]. The oldest and perhaps most widely known BUMG method is hyperresolution [19].

Hyperresolution consists of two inference rules, hyperresolution and factoring. The hyperresolution rule applies to a non-positive clause $H \leftarrow B_1 \wedge \ldots \wedge B_n$ and n positive clauses $C_1 \vee B'_1 \leftarrow \top, \ldots, C_n \vee B'_n \leftarrow \top$, and derives $(C_1 \vee \ldots \vee C_n \vee H)\sigma \leftarrow \top$, where σ is the most general unifier such that $B'_i\sigma = B_i\sigma$ for every $i \in \{1,\ldots,n\}$. The factoring rule derives the clause $(C \vee B)\sigma \leftarrow \top$ from a positive clause $C \vee B \vee B' \leftarrow \top$, where σ is the most general unifier of B and B'. A crucial requirement for the effective use of blocking (Section 3.3) is support of equality reasoning, for example, ordered paramodulation or superposition [18], in combination with simplification techniques based on orderings.

Our experiments show that a certain form of the splitting rule, or the β-rule, is quite useful for our approach. For the blocking transformation, splitting on the positive part of (ground) clauses is in fact mandatory to make it effective. This type of splitting will replace the branch of a derivation containing the positive clause $C \vee D \leftarrow \top$, say, by two copies of the branch in which the clause is replaced by $C \leftarrow \top$ and $D \leftarrow \top$, respectively, provided that C and D do not share any variables. Most BUMG procedures support this splitting technique, in particular the provers that we used do.

3 Transformations

3.1 Range-Restriction

Existing transformations to range-restricted form follow Manthey and Bry [15] (or are variations of it). The transformation can be defined by a procedure carrying out the following steps on a given set M of clauses.

(0) Initialization. Initially, let $\mathrm{crr}(M) := M$.

(1) Add a constant. Let dom be a "fresh" unary predicate symbol not in Σ_P, and let c be some constant. Extend $\mathrm{crr}(M)$ by the clause $\mathsf{dom}(c) \leftarrow$. (The constant c can be "fresh" or belong to Σ_f.)

(2) Range-restriction. For each clause $\mathcal{H} \leftarrow \mathcal{B}$ in $\mathrm{crr}(M)$, let $\{x_1, \ldots, x_k\}$ be the set of variables occurring in $\mathcal{H}$ but not in $\mathcal{B}$. Replace $\mathcal{H} \leftarrow \mathcal{B}$ by the clause

$$\mathcal{H} \leftarrow \mathcal{B} \wedge \mathsf{dom}(x_1) \wedge \cdots \wedge \mathsf{dom}(x_k).$$

(3) Enumerate the Herbrand universe. For each n-ary $f \in \Sigma_f$, add the clauses:

$$\mathsf{dom}(f(x_1, \ldots, x_n)) \leftarrow \mathsf{dom}(x_1) \wedge \cdots \wedge \mathsf{dom}(x_n).$$

We refer to the computed set $\mathrm{crr}(M)$ as the *classical range-restricting transformation* of M. It is not difficult to see that $\mathrm{crr}(M)$ is indeed range-restricted for any clause set M. The transformation is sound and complete, i.e. M is satisfiable iff $\mathrm{crr}(M)$ is satisfiable [15,8]. Clearly, the size of $\mathrm{crr}(M)$ is linear in the size of M and can be computed in linear time.

Perhaps the easiest way to understand the transformation is to imagine we use a BUMG method, e.g. hyperresolution. The idea is to build the model(s) during the derivation. The clause added in Step (1) ensures that the domain of interpretation given by the domain predicate dom is non-empty. Step (2) turns clauses into range-restricted clauses by shielding variables in the head that do not occur negatively within the added negative domain literals. Clauses that are already range-restricted are unaffected by this step. Step (3) ensures that all elements of the Herbrand universe of the (original) clause set are added to the domain via hyperresolution inference steps. As a consequence a clause set M with at least one non-nullary function symbols causes hyperresolution derivations to be unbounded for $\mathrm{crr}(M)$, unless M is unsatisfiable. This is a distinct drawback of the classical range-restricting transformation. However, the method has been shown to be useful for (domain-)minimal model generation when combined with other techniques [8,7].

In Section 4 we consider the combination of the classical range-restricting transformation crr with the blocking transformation which is introduced in Section 3.3.

Let us first turn to a new transformation to range-restricted form which aims to help avoid the brute-force enumeration of the entire Herbrand universe by BUMG approaches. The transformation involves extracting the non-variable top-level terms in an atom. Let $P(t_1, \ldots, t_n)$ be an atom and suppose $x_1, \ldots, x_n$ are fresh variables. For all $i \in \{1, \ldots, n\}$ let $s_i = t_i$, if t_i is a variable, and $s_i = x_i$, otherwise. The atom $P(s_1, \ldots, s_n)$ is called the *term abstraction of* $P(t_1, \ldots, t_n)$. Let the *abstraction substitution* α be defined by $\alpha = \{x_i \mapsto t_i \mid 1 \leq i \leq n \text{ and } t_i \text{ is not a variable}\}$. Hence, $P(s_1, \ldots, s_n)\alpha = P(t_1, \ldots, t_n)$, i.e. α reverts the term abstraction. Now, the new *range-restricting transformation*, denoted by rr, of a clause set M is the clause set obtained by carrying out the following steps (explanations and an example are given afterwards):

(0) Initialization. Initially, let $\mathrm{rr}(M) := M$.

(1) Add a constant. Same as Step (1) in the definition of crr.

(2) Domain elements from clause bodies. For each clause $\mathcal{H} \leftarrow \mathcal{B}$ in M and each atom $P(t_1,\ldots,t_n)$ from $\mathcal{B}$, let $P(s_1,\ldots,s_n)$ be the term abstraction of $P(t_1,\ldots,t_n)$ and let α be the corresponding abstraction substitution. Extend $\mathrm{rr}(M)$ by the set

$$\{\mathsf{dom}(x_i)\alpha \leftarrow P(s_1,\ldots,s_n) \mid 1 \leq i \leq n \text{ and } x_i \mapsto t_i \in \alpha\}.$$

(3) Range-restriction. Same as Step (2) in the definition of crr.

(4) Domain elements from Σ_P. For each n-ary P in Σ_p, extend $\mathrm{rr}(M)$ by the set

$$\{\mathsf{dom}(x_i) \leftarrow P(x_1,\ldots,x_n) \mid i \leq i \leq n\}.$$

(5) Domain elements from Σ_f. For each n-ary f in Σ_f, extend $\mathrm{rr}(M)$ by the set

$$\{\mathsf{dom}(x_i) \leftarrow \mathsf{dom}(f(x_1,\ldots,x_n)) \mid i \leq i \leq n\}.$$

The intuition of the transformation reveals itself if we think of what happens when using hyperresolution. The idea is again to build the model(s) during the derivation, but this time terms are added to the domain only *as necessary*. Steps (1) and (3) are the same as Steps (1) and (2) in the definition of crr. The clauses added in Step (2) cause functional terms that occur negatively in the clauses to be inserted into the domain. Step (4) ensures that positively occurring functional terms are added to the domain, and Step (5) ensures that the domain is closed under subterms.

To illustrate the steps of the transformation consider the following clause.

$$\mathsf{q}(x,\mathsf{g}(x,y)) \vee \mathsf{r}(y,z) \leftarrow \mathsf{p}(\mathsf{a},\mathsf{f}(x,y),x) \qquad (\dagger)$$

The term abstraction of the body literal is $\mathsf{p}(x_1,x_2,x)$ and the abstraction substitution is $\alpha = \{x_1 \mapsto \mathsf{a}, x_2 \mapsto \mathsf{f}(x,y)\}$. The clauses added in Step (2) are thus:

$$\mathsf{dom}(\mathsf{a}) \leftarrow \mathsf{p}(x_1,x_2,x) \qquad \mathsf{dom}(\mathsf{f}(x,y)) \leftarrow \mathsf{p}(x_1,x_2,x) \qquad (\ddagger)$$

Notice that among the clauses so far the clauses (†) and (‡) are not range-restricted, but are turned into range-restricted clauses in Step (3), yielding the following.

$$\mathsf{q}(x,\mathsf{g}(x,y)) \vee \mathsf{r}(y,z) \leftarrow \mathsf{p}(\mathsf{a},\mathsf{f}(x,y),x) \wedge \mathsf{dom}(z)$$
$$\mathsf{dom}(\mathsf{f}(x,y)) \leftarrow \mathsf{p}(x_1,x_2,x) \wedge \mathsf{dom}(y)$$

Step (4) generates clauses responsible for inserting the terms that occur in the heads of clauses into the domain. I.e. for each $i \in \{1,2,3\}$ and each $j \in \{1,2\}$:

$$\mathsf{dom}(x_i) \leftarrow \mathsf{p}(x_1,x_2,x_3) \qquad \mathsf{dom}(x_j) \leftarrow \mathsf{q}(x_1,x_2) \qquad \mathsf{dom}(x_j) \leftarrow \mathsf{r}(x_1,x_2)$$

For instance, when a model assigns true to the instance $\mathsf{q}(\mathsf{a},\mathsf{g}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{a})))$ of one of the head atoms of the clause above, then $\mathsf{dom}(\mathsf{a})$ and $\mathsf{dom}(\mathsf{g}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{a})))$ will also be true. It is not necessary to insert the terms of the instance of the other head atom into the domain. The reason is that it does not matter how these (extra) terms are evaluated, or whether the atom is evaluated to true or false in order to satisfy the disjunction. The clauses added in Step (4) alone are not sufficient, however. For each term in the domain all its subterms have to be in the domain, too. This is achieved with the clauses obtained in Step (5). I.e. for each $j \in \{1,2\}$:

$$\mathsf{dom}(x_j) \leftarrow \mathsf{dom}(\mathsf{f}(x_1,x_2)) \qquad \mathsf{dom}(x_j) \leftarrow \mathsf{dom}(\mathsf{g}(x_1,x_2))$$

Proposition 3.1 (Completeness of range-restriction (wrt. E-interpretations)). *Let M be a clause set. (i) If $rr(M)$ is satisfiable then M is satisfiable. (ii) If $rr(M) \cup \{x \approx x \leftarrow \mathsf{dom}(x)\}$ is E-satisfiable then M is E-satisfiable.*

Consider the clause $\mathsf{r}(x) \leftarrow \mathsf{q}(x) \wedge \mathsf{p}(\mathsf{f}(x))$ which might be part of the translation of a modal logic formula or a description logic knowledge base. Applying Steps (2) and (3) of our transformation give us a clause,

$$\mathsf{dom}(\mathsf{f}(x)) \leftarrow \mathsf{dom}(x) \wedge \mathsf{p}(y), \qquad (*)$$

which is splittable into $\mathsf{dom}(\mathsf{f}(x)) \leftarrow \mathsf{dom}(x)$ and $\bot \leftarrow \mathsf{p}(y)$. The first split component clause is unpleasant, because it is an example of an "enumerate the Herbrand universe" clause from existing standard transformations (Step (2) in the definition of crr). Such clauses cause the entire Herbrand universe to be enumerated with BUMG approaches. One solution is to switch off splitting when using a BUMG approach, but this is not necessarily the best or the only solution. (Indeed, our experiments below demonstrate that splitting is advisable.) Before describing a solution let us analyze the problem further.

The main rationale of our rr transformation is to constrain the generation of domain elements and limit the number of inference steps. The general form of clauses produced by Step (2), followed by Step (3), is the following, where $\overline{y} \subseteq \overline{x}$, $\overline{x} \subseteq \overline{y} \cup \overline{z}$ and $\overline{u} \subseteq \overline{z}$.

$$\mathsf{dom}(f(\overline{x})) \leftarrow \mathsf{dom}(y_1) \wedge \ldots \wedge \mathsf{dom}(y_n) \wedge P(\overline{z}) \qquad \mathsf{dom}(f(\overline{u})) \leftarrow P(\overline{z})$$

Clauses of the first form are often splittable (as in the example above), and can produce clauses of the unwanted form $\mathsf{dom}(f(\overline{y})) \leftarrow \mathsf{dom}(y_1) \wedge \ldots \wedge \mathsf{dom}(y_n)$. Suppose therefore that splitting of any clause is forbidden when this splits the negative part of the clause (neither (M)SPASS nor hyper tableaux prover do this anyway). Although, compared to the classical range-restricting transformation methods, the two types of clauses above both *do* reduce the number of possible inferences, the constraining effect of the first type of clauses is a bit limited. Terms $f(\overline{s})$ are not generated, only when no fact $P(\overline{t})$ is present or has been derived. When a clause $P(\overline{t})$ is present, or as soon as such a clause is derived (for *any* (ground) terms $\overline{t}$), then terms are freely generated from terms already in the domain with f as the top symbol. Here is an example of a clause set for which the derivation is infinite on the transformation.

$$\mathsf{p}(\mathsf{b}) \leftarrow \top \qquad \mathsf{r}(x) \leftarrow \mathsf{q}(x) \wedge \mathsf{p}(\mathsf{f}(x))$$

Notice the derivation will be infinite on the classical range-restricting transformation as well, due to the generated clauses $\mathsf{dom}(\mathsf{b}) \leftarrow \top$ and $\mathsf{dom}(f(x)) \leftarrow \mathsf{dom}(x)$.

The second type of clauses, $\mathsf{dom}(f(\overline{u})) \leftarrow P(\overline{z})$, are less problematic. Here is a concrete example. For $\bot \leftarrow \mathsf{r}(x, \mathsf{f}(x))$, Step (2) produces the clause $\mathsf{dom}(\mathsf{f}(x)) \leftarrow \mathsf{r}(x,y)$. Although this clause, and the general form, still cause larger terms to be built with hyperresolution type inferences, the constraining effect is larger.

In the next two sections we discuss ways of improving the transformation further.

3.2 Shifting

The clauses introduced in Step (2) of the new transformation rr to range-restricted form insert instantations of terms occurring in the clause bodies into the domain. This is

sometimes unnecessary and can lead to non-termination of BUMG procedures. The *shifting* transformation addresses this problem. It consists of two sub-transformations, *basic shifting* and *partial flattening*.

If A is an atom $P(t_1,\ldots,t_n)$ then let $\mathsf{not\_}A$ denote the atom $\mathsf{not\_}P(t_1,\ldots,t_n)$, where $\mathsf{not\_}P$ is a fresh predicate symbol which is uniquely associated with the predicate symbol P. If P is the equality symbol $\approx$ we write $\mathsf{not\_}P$ as $\not\approx$ and use infix notation. Now, the *basic shifting transformation* of a clause set M is the clause set $\mathrm{bs}(M)$ obtained from M by carrying out the following steps.

(0) Initialization. Initially, let $\mathrm{bs}(M) := M$.

(1) Shifting deep atoms. Replace each clause in $\mathrm{bs}(M)$ of the form $\mathcal{H} \leftarrow B_1 \wedge \cdots \wedge B_m \wedge \mathcal{B}$, where each atom $B_1,\ldots,B_m$ contains at least one proper functional term and $\mathcal{B}$ contains no proper functional term, by the clause

$$\mathcal{H} \vee \mathsf{not\_}B_1 \vee \cdots \vee \mathsf{not\_}B_m \leftarrow \mathcal{B}.$$

Each of the atoms $B_1,\ldots,B_m$ is called a *shifted atom*.

(2) Shifted atoms consistency. Extend $\mathrm{bs}(M)$ by the clause set

$$\{\bot \leftarrow P(x_1,\ldots,x_n) \wedge \mathsf{not\_}P(x_1,\ldots,x_n) \mid P \text{ is the } n\text{-ary predicate symbol of a shifted atom}\}.$$

Notice that we do *not* add clauses complementary to the "shifted atoms consistency" clauses, i.e., $P(x_1,\ldots,x_n) \vee \mathsf{not\_}P(x_1,\ldots,x_n) \leftarrow \top$. They could be included but are evidently superfluous.

Let us continue the example given at the end of the previous section. We can use basic shifting to move negative occurrences of functional terms into heads. In the example, instead of the clause $(*)$ we get the following.

$$\mathsf{dom}(x) \leftarrow \mathsf{not\_p}(x) \qquad \mathsf{r}(x) \vee \mathsf{not\_p}(\mathsf{f}(x)) \leftarrow \mathsf{q}(x) \qquad (**)$$
$$\mathsf{dom}(x) \leftarrow \mathsf{r}(x) \qquad \bot \leftarrow \mathsf{not\_p}(x) \wedge \mathsf{p}(x)$$

This gets rid of the problematic clause $(*)$. Even in the presence of an additional clause, say, $\mathsf{q}(x) \leftarrow \top$, which leads to the clauses $\mathsf{dom}(\mathsf{a}) \leftarrow \top$ and $\mathsf{q}(x) \leftarrow \mathsf{dom}(x)$, termination of BUMG can be achieved. For instance, in a hyperresolution-like setting and with splitting enabled the MSPASS prover [20] splits the derived clause $\mathsf{r}(\mathsf{a}) \vee \mathsf{not\_p}(\mathsf{f}(\mathsf{a}))$, considers the case with the smaller literal $\mathsf{r}(\mathsf{a})$ first *and terminates with a model*. This is because a finite completion is found without considering the case of the bigger literal $\mathsf{not\_p}(\mathsf{f}(\mathsf{a}))$, which would have added the deeper term $\mathsf{f}(\mathsf{a})$ to the domain. The same behaviour can be achieved for example with the KRHyper BUMG prover.

As can be seen in the example, the basic shifting transformation trades the generation of new domain elements for a smaller clause body (by removing literals from it). Of course, a smaller clause body is undesirable for BUMG methods, as then the clause can be used as a premise more often. To (partially) avoid this effect, we propose an additional transformation to be performed prior to the basic shifting transformation.

For a clause set M, the *partial flattening transformation* is the clause set $\mathrm{pf}(M)$ obtained by applying the following steps.

(0) Initialization. Initially, let $\mathrm{pf}(M) := M$.
(1) Reflexivity. Extend $\mathrm{pf}(M)$ by the unit clause $x \approx x \leftarrow \top$.
(2) Partial flattening. For each clause $\mathcal{H} \leftarrow \mathcal{B}$ in $\mathrm{pf}(M)$, let $t_1, \ldots, t_n$ be all top-level terms occurring in the non-equational literals in the body $\mathcal{B}$ that are proper functional terms, for some $n \geq 0$. Let $x_1, ..., x_n$ be fresh variables. Replace the clause $\mathcal{H} \leftarrow \mathcal{B}[t_1, \ldots, t_n]$ by the clause

$$\mathcal{H} \leftarrow \mathcal{B}[x_1, \ldots, x_n] \wedge t_1 \approx x_1 \wedge \cdots \wedge t_n \approx x_n.$$

It should be noted that the equality symbol $\approx$ need not be interpreted as equality, but could. (Un-)satisfiability (and logical equivalence) is preserved even when reading it just as "unifiability". This is achieved by the clause $x \approx x \leftarrow \top$.

In our running example, applying the transformations pf, bs and rr, in this order, yields the following clauses (among other clauses, which are omitted because they are not relevant to the current discussion).

$$\mathsf{r}(x) \vee \mathsf{f}(x) \not\approx u \leftarrow \mathsf{q}(x) \wedge \mathsf{p}(u) \qquad \mathsf{dom}(x) \leftarrow x \not\approx y \qquad \mathsf{dom}(x) \leftarrow \mathsf{r}(x)$$
$$\bot \leftarrow x \not\approx y \wedge x \approx y \qquad \mathsf{dom}(y) \leftarrow x \not\approx y$$

Observe that the first clause is more restricted than the clause $(**)$ above because of the additional body literal $\mathsf{p}(u)$.

The reason for not extracting constants during partial flattening is that adding them to the domain will not cause non-termination of BUMG methods. It is preferable to leave them in place in the body literals because they have a stronger constraining effect than the variables introduced otherwise. Extracting top-level terms from equations has no effect at all. Consider the unit clause $\bot \leftarrow f(a) \approx b$, and its partial flattening $\bot \leftarrow x \approx b \wedge f(a) \approx x$. Applying basic shifting yields $f(a) \not\approx x \leftarrow x \approx b$, and, hyperresolution with $x \approx x \leftarrow \top$ gives $f(a) \not\approx b \leftarrow \top$. This is the same result as obtained by the transformations as defined. This explains why top-level terms of equational literals are excluded from the definition. (One could consider using "standard" flattening, i.e. recursively extracting terms, but this does not lead to any improvements over the defined transformations.)

Finally, combine basic shifting and partial flattening to give the *shifting transformation*, formally defined by $\mathrm{sh} := \mathrm{pf} \circ \mathrm{bs}$, i.e. $\mathrm{sh}(M) = \mathrm{bs}(\mathrm{pf}(M))$, for any clause set M.

Proposition 3.2 (Completeness of shifting (wrt. E-interpretations)). *Let M be a clause set. (i) If $\mathrm{sh}(M)$ is satisfiable then M is satisfiable. (ii) If $\mathrm{sh}(M)$ is E-satisfiable then M is E-satisfiable.*

3.3 Blocking

Our final transformation is intended to be a mechanism for detecting periodicity in the derived models. By definition, the *blocking transformation* of a clause set M is the clause set $\mathrm{bl}(M)$ obtained from M by carrying out the following steps.

(0) Initialization. Initially, let $\mathrm{bl}(M) := M$.

(1) Axioms describing the subterm relationship. Let sub be a "fresh" binary predicate symbol not in Σ_P. Extend $\mathrm{bl}(M)$ by $\mathsf{sub}(x,x) \leftarrow \mathsf{dom}(x)$ and, for every n-ary function symbol $f \in \Sigma_f$ and all $i \in \{1,\ldots,n\}$, add the clauses

$$\mathsf{sub}(x, f(x_1,\ldots,x_n)) \leftarrow \mathsf{sub}(x,x_i) \wedge \mathsf{dom}(x) \wedge \mathsf{dom}(f(x_1,\ldots,x_n)).$$

(2) Subterm equality case analysis. Extend $\mathrm{bl}(M)$ by these clauses.

$$x \approx y \vee x \not\approx y \leftarrow \mathsf{sub}(x,y) \qquad\qquad \leftarrow x \approx y \wedge x \not\approx y$$

The blocking transformation preserves range-restrictedness. In fact, because the dom predicate symbol is mentioned in the definition, the blocking transformation can be applied meaningfully only in combination with range-restricting transformations.

Reading $\mathsf{sub}(s,t)$ as "s is a subterm of t", the Step (1) in the blocking transformation might seem overly involved, because an apparently simpler specification of the subterm relationship for the terms of the signature Σ_f can be given. Namely:

$$\mathsf{sub}(x,x) \leftarrow \mathsf{dom}(x) \qquad\qquad \mathsf{sub}(x, f(x_1,x_2\ldots,x_n)) \leftarrow \mathsf{sub}(x,x_i)$$

for every n-ary function symbol $f \in \Sigma_f$ and all $i \in \{1,\ldots,n\}$. This clause set is range-restricted. Yet, this specification is not suitable for our purposes. For example, for a given constant a and a unary function symbol f, when just $\mathsf{dom}(\mathsf{a})$ alone has been derived, a BUMG procedure derives an infinite sequence clauses: $\mathsf{sub}(\mathsf{a},\mathsf{a})$, $\mathsf{sub}(\mathsf{a},\mathsf{f}(\mathsf{a}))$, $\mathsf{sub}(\mathsf{a},\mathsf{f}(\mathsf{f}(\mathsf{a})))$, ... This does not happen with the specification in Step (1). It ensures that conclusions of BUMG inferences involving sub are about terms currently in the domain, and the domain is always finite.

To justify the clauses added in Step (2) we continue this example and suppose an interpretation that contains $\mathsf{dom}(\mathsf{a})$ and $\mathsf{dom}(\mathsf{f}(\mathsf{a}))$. These might have been derived earlier in the run of a BUMG prover. Then, from the clauses added by blocking, the (necessarily ground) disjunction $\mathsf{f}(\mathsf{a}) \approx \mathsf{a} \vee \mathsf{f}(\mathsf{a}) \not\approx \mathsf{a} \leftarrow \top$ is derivable. Now, it is important to use a BUMG prover with support for splitting and to equip it with an appropriate search strategy. In particular, when deriving a disjunction like the one above, the $\approx$-literal should be split off and the clause set obtained in this case should be searched *first*. The reason is that the (ground) equation $\mathsf{f}(\mathsf{a}) \approx \mathsf{a}$ thereby obtained can then be used for simplification and redundancy testing purposes. For example, should $\mathsf{dom}(\mathsf{f}(\mathsf{f}(\mathsf{a})))$ be derivable now (in the current branch), then any prover based a modern, saturation-based theory of equality reasoning will be able to prove it redundant from $\mathsf{f}(\mathsf{a}) \approx \mathsf{a}$ and $\mathsf{dom}(\mathsf{a})$. Consequently, the domain will *not* be extended *explicitly*. The information that $\mathsf{dom}(\mathsf{f}(\mathsf{f}(\mathsf{a})))$ is in the domain is however implicit via the theory of equality.

The blocking transformation was designed to realize a "loop check" for the construction of a domain, by capitalizing on available, powerful equality reasoning technology and redundancy criteria from saturation-based theorem proving. To be suitable, a resolution-based prover, for instance, should support *hyperresolution-style inference, strong equality inference e.g. superposition, splitting, and the possibility to search for split-off equations first and standard redundancy elimination techniques*. Among the well-known, current resolution theorem provers splitting is not standard, but it is available in the saturation-based prover SPASS (and the extension MSPASS) and VAMPIRE.

Unfortunately, the hyper tableau prover KRHyper does not include suitable equality inference rules. Otherwise its splitting could easily be configured to meet our needs.

The blocking transformation is inspired by a technique with the same name (and same purpose) implemented in tableau provers for description and modal logics. Indeed, when comparing these techniques in detail it becomes clear that our transformation rr$\circ$bl, when applied to a knowledge base of a description logic with the finite model property, in conjunction with a suitable BUMG method (see above), is powerful enough to simulate various forms of blocking techniques, including (dynamic and static) subset blocking and equality blocking [1]. But notice that our transformation applies to any first-order clause set, not only to clauses from the translation of description logic problems. This makes our approach more widely applicable. For instance, our approach makes it possible to extend description logics with arbitrary (first-order expressible) "rule" languages. "Rules" provide a connection to (deductive) databases and are being used to represent information that is currently not expressible in the description logics associated with OWL DL.

Proposition 3.3 (Completeness of blocking wrt. E-interpretations). *Let M be any clause set. If $bl(M)$ is E-satisfiable then M is E-satisfiable.*

Our main theoretical result is the following.

Theorem 3.4 (Completeness of the combined transformations with respect to E-interpretations). *Let M be a clause set and suppose tr is any of the transformations in $\{rr, sh \circ rr, rr \circ bl, sh \circ rr \circ bl\}$. Then: (i) $tr(M)$ is range-restricted. (ii) $tr(M)$ can be computed in quadratic time. (iii) If $tr(M) \cup \{x \approx x \leftarrow \mathsf{dom}(x)\}$ is E-satisfiable then M is E-satisfiable.*

By carefully modifying the definition of rr it is possible to compute the reductions in linear time. The reverse directions of (iii), i.e. soundness of the respective transformations, hold as well. The theorem is also true if rr is replaced by crr.

4 Experiments

We have implemented the transformations described in the previous section and carried out experiments on problems from the TPTP library, Version 3.1.1. (The implementation, in SWI-Prolog, is available from the first author's website.) Since the emphasis in this paper is on *disproving* theorems, i.e. on reporting whether a given clause set is satisfiable, we have selected for the experiments only satisfiable (clausal) problems from the TPTP suite, yet all 514 of them. The test were carried out with the BUMG systems MSPASS (Version 2.0g.1.4+) [20],[1] using ordered resolution with maximal selection of negative literals, and to a lesser extent the KRHyper theorem prover [25]. Both were run on a Linux PC with an Intel Pentium 4 3.80GHz processor and 1 GByte main memory.

Table 1 is a summary of the results of the MSPASS runs. The column with the heading "#" gives the number of problems in the listed TPTP categories. The subsequent

[1] MSPASS is an extension of the prover SPASS [24], but except for a small modification in the code we did not use any of the extra features of MSPASS. We used MSPASS because it satisfies the suitability criteria (see previous section), the source code is available, the options are documented and we are familiar with it.

Table 1. Result summary of MSPASS runs on the satisfiable clausal TPTP problems

Category	#	rr −sp	rr +sp	sh ∘ rr −sp	sh ∘ rr +sp	rr ∘ bl +sp	sh ∘ rr ∘ bl +sp	crr ∘ bl +sp
ALG	1	0	0	0	0	1	0	0
BOO	13	0	0	0	0	2	3	2
COL	5	0	0	0	0	0	0	0
GEO	1	0	0	0	0	0	0	0
GRP	25	7	7	7	8	15	14	12
KRS	8	1	1	4	8	4	6	4
LAT	1	0	0	0	0	1	1	0
LCL	4	0	1	1	1	1	1	1
MGT	10	1	1	3	4	4	5	0
MSC	1	1	1	1	1	1	1	1
NLP	236	49	79	68	96	87	160	68
NUM	1	1	1	1	1	1	1	1
PUZ	20	6	6	6	6	10	10	9
RNG	4	0	0	0	0	0	0	0
SWV	8	0	0	0	0	1	1	0
SYN	176	20	50	20	52	124	125	120
All	514	86	147	111	177	252	328	218

columns give the number of problems solved within the given time limit of five minutes (CPU time) and 300 MByte main memory consumption (which was not a bottleneck). Results are presented for the different transformations that were used. For example, sh ∘ rr ∘ bl means that shifting, the new range-restriction and blocking was used; +sp, respectively −sp, indicate whether splitting was enabled or disabled. The last column, crr ∘ bl, contains the results for the classical range-restricting transformation in combination with blocking. (For the reasons mentioned before, evaluating the classical range-restricting transformation without blocking is not of interest for satisfiable problems.) Testing the crr ∘ bl setting is interesting because it allows us to assess the significance of the shifting and our new range-restricting transformations in comparison with classical range-restriction. As can be seen from the number of problems solved, the sh, rr, and in particular, the sh ∘ rr transformations performed much better than crr in combination with bl. This demonstrates the need for *all* our new transformations. The runtimes for the problems solved spanned the whole range, from less than one second to almost all of the time allowed. It is not a mistake that no results are given for transformations with blocking but no splitting; this would not make sense.

Let us now compare the individual combinations and discuss our observations from the experiments conducted with MSPASS. Broadly, the results indicate that the performance for the combination $(\text{rr}, -\text{sp})$ was inferior to that for $(\text{rr}, +\text{sp})$ and for $(\text{sh} \circ \text{rr}, -\text{sp})$, and each of these was inferior to the performance for $(\text{sh} \circ \text{rr}, +\text{sp})$. There were only very few problems that were solved by an "inferior" combination alone. This suggests that switching splitting on is advisable, and that shifting is an effective improvement, in particular in combination with splitting. In that combination, splitting helps in particular to "forget" those atoms in the head of a clause that were introduced by shifting, which otherwise generates new domain elements.

The combination $(\mathrm{sh} \circ \mathrm{rr}, -\mathrm{sp})$ was inferior to $(\mathrm{rr} \circ \mathrm{bl}, +\mathrm{sp})$. Our explanation is that shifting without splitting often generates many deep and long clauses in the search space which are not redundant according to standard redundancy criteria. Nevertheless, these clauses are redundant in the sense that they are satisfied by a finite model.

The results obtained for the combinations $(\mathrm{sh} \circ \mathrm{rr}, +\mathrm{sp})$ and $(\mathrm{rr} \circ \mathrm{bl}, +\mathrm{sp})$ are incomparable. There were many problems over all categories that were solved by either approach. This confirms our expectation that the shifting and range-restriction techniques are orthogonal. Shifting tries to avoid the generation of domain elements, but it is sometimes not strong enough. Blocking, by contrast, is a strong technique, which helps to discover finite models more often, but creates a larger search space.

The combination $(\mathrm{sh} \circ \mathrm{rr}, +\mathrm{sp})$ was strictly inferior to $(\mathrm{sh} \circ \mathrm{rr} \circ \mathrm{bl}, +\mathrm{sp})$. It suggests that adding blocking to shifting is advisable. This result is somewhat surprising. We expected that the additional search space introduced by blocking renders some examples unsolvable that can be solved with shifting alone. Interestingly, not even time-wise did blocking cause a penalty when shifting alone was sufficient.

The combination $(\mathrm{rr} \circ \mathrm{bl}, +\mathrm{sp})$ was in most cases inferior to $(\mathrm{sh} \circ \mathrm{rr} \circ \mathrm{bl}, +\mathrm{sp})$. The result was not as uniform as in the previous case, though. There were some satisfiable problems that were solved with the $(\mathrm{rr} \circ \mathrm{bl}, +\mathrm{sp})$ combination *alone* (but no other combination). It is not entirely clear, why. On the other hand, there were also some problems that were not solved with $\mathrm{rr} \circ \mathrm{bl}$, but were solved with most other transformations. For these problems the search overhead when using blocking seems too big.

We also used KRHyper [25], which is an efficient implementation of the hyper tableau calculus [3]. On range-restricted clause sets, the hyper tableau calculus is closely related to resolution with maximal selection and splitting, the instance of MSPASS that we used. KRHyper, as a tableau calculus, has splitting "built-in", but it does not include equality inference rules. It therefore lacks the refinements needed to support the blocking transformation effectively. We therefore selected all satisfiable TPTP problems without equality for the tests. There are 309 of these (out of a total of 514).

The results were as follows. The performance of KRHyper for the transformation rr was inferior to $\mathrm{sh} \circ \mathrm{rr}$. The latter was better on almost all problems, over all categories. The results parallel those above obtained with MSPASS. This was expected.

Perhaps the most interesting comparison is between KRHyper equipped with the transformation $\mathrm{sh} \circ \mathrm{rr}$ and MSPASS equipped with the combination $(\mathrm{sh} \circ \mathrm{rr}, +\mathrm{sp})$. With these settings 134 problems were solved by KRHyper and 121 problems were solved by MSPASS. Specifically, there are 17 problems that were solved by KRHyper but not by MSPASS, in any combination. The rating of these problems is between 0.00 and 0.80. Most of them are from the NLP category. The reason why KRHyper performed better than MSPASS lies in its splitting strategy, which is more suitable for our purposes than the one utilized in MSPASS. (Due to lack of space we do not give details.) It would therefore be interesting to modify the way splitting is done in MSPASS so that it mimics KRHyper's splitting. Other (probably) significant differences are the non-chronological backtracking schemes employed in KRHyper and MSPASS.

Table 2 summarizes the results with respect to problem rating. The column with the heading "MSPASS" reflects how many problems were solved, among all the combinations mentioned in Table 1 except $\mathrm{crr} \circ \mathrm{bl}$. The "KRHyper additional" column says how

Table 2. Result summary wrt. problem rating

Rating	#	MSPASS	KRHyper additional	Rating	#	MSPASS	KRHyper additional
1.00	4	0		0.40	47	26	1
0.80	57	24	4	0.33	8	4	1
0.67	26	5		0.20	70	50	
0.60	44	23	10	0.17	31	10	
0.50	5	0		0.00	223	198	1

many problems were solved by KRHyper (using the transformation sh $\circ$ rr) that were not solvable in any combination with MSPASS. As far as we know, problems with rating 0.80 have so far been solved by one theorem prover only. It was notable that each problem with a rating 0.80 or 0.67 solvable by MSPASS required blocking. On the other hand, there were several unsolvable "easy" problems.

Together, this indicates that the approach presented here and the more established methods are orthogonal. This finding was confirmed by a comparison with MSPASS (in autonomous mode) and Paradox [9], a state-of-the-art MACE-style finite model builder. We ran Paradox on the same problem set, with the same time limit of five minutes (CPU time) and a limit on 400 MByte main memory consumption. There were several problems that were solved by Paradox but not with our methods. On the other side, there were 21 problems, all of the NLP category, that were be solved with our methods but not by Paradox. Each of these problems required shifting (and splitting) to be solvable by our methods. In about half of the cases blocking was essential, while the other half were solved by shifting alone. Without shifting (with or without splitting), none of these problems were solved. The runtimes varied between two and at most 15 seconds. Memory consumption was not an issue at all. By contrast, for 13 of these 21 problems Paradox was stopped prematurely because the memory limit was exceeded before the time limit was reached. We sampled some of these problems and re-ran Paradox without artificial limits. For the problem NLP049-1, for instance, about 10 million (ground) clauses were generated for a domain size of 8 elements, consuming about 1 GByte of main memory, and the underlying SAT solver did not complete its run within 15 minutes (we stopped it then). This picture seems typical for these problems. Regarding the comparison with MSPASS in autonomous mode, the differences in which problems were solvable were more pronounced.

5 Conclusions

We have presented and tested a number of enhancements for BUMG methods. An important aspect is that our enhancements exploit the strengths of readily available BUMG system without any, or only little modifications. Our techniques have the advantage over existing approaches based on transformations to range-restricted clauses that terms are added to the domain of interpretation on a "by need" basis. Moreover, we present methods that allow us to extend BUMG methods with a blocking technique, which has only been used in more the specialized setting for non-classical logics (with tree model properties). Related research in automated theorem proving has concentrated on developing

refinements of resolution, mainly ordering refinements, for deciding numerous fragments of first-order logic. These fragments are complementary to the fragments that can be decided by refinements using the techniques presented in this paper. We thus extend the set of techniques available for resolution methods to turn them into more effective and efficient (terminating) automated reasoning methods. For example, based on the results of [21] we can use our transformations to decide the Bernays-Schönfinkel (BS) class. In particular, we can show that all procedures based on hyperresolution or BUMG can decide the class of BS formulae and the class of BS clauses (with equality).

Our approach is especially suitable for generating small models and we believe the approach allows us to compute finite models when they exist. The generated models do not need to be Herbrand models. It follows from how the transformations work that the generated models are quasi-Herbrand models, in the following sense. Whenever $\mathsf{dom}(s)$ and $\mathsf{dom}(t)$ hold in the (Herbrand) model constructed by the BUMG method, then (as in Herbrand interpretations) the terms s and t are mapped to themselves in the associated (possibly non-Herbrand) model. Reconsidering the example in the Introduction of the two unit clauses $\mathsf{P}(\mathsf{a})$ and $\mathsf{Q}(\mathsf{b})$, the associated model will map a and b to themselves, regardless as to which transformations are applied (as long as it includes rr). In this way, more informative models are produced than those computed by, e.g., MACE- and SEM-style finite model searchers.

We have implemented the approach and tested it with existing first-order logic theorem provers. The results demonstrate that our transformations are quite effective and many difficult TPTP problems can now be solved by BUMG methods, especially resolution with maximal selection or hyperresolution in MSPASS, and KRHyper. However, the results are far from conclusive, and we plan to develop and evaluate variants of our transformations, and experiment with alternative splitting strategies (particularly for MSPASS). Studying how well the ideas and techniques discussed in this paper can be exploited and behave in BUMG provers, and also tableau-based provers and other provers (including resolution-based provers) is very important but is beyond the scope of the present paper. We have started experimenting with another prover, Darwin [2], and first results are very encouraging. An in-depth comparison and analysis of BUMG approaches with our techniques and MACE-style or SEM-style model generation would also be of interest. Another source for future work is to combine our transformations with available BUMG techniques and improvements, such as magic sets transformations [14,23], a typed version of range-restriction [4], and minimal model computation. We speculate that our transformations carry over to the case with default negation, thus advancing, for example, answer-set programming beyond its current limitations.

Acknowledgement. Thanks to U. Furbach for comments on the paper and discussions.

References

1. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
2. P. Baumgartner, A. Fuchs, and C. Tinelli. DARWIN. `http://goedel.cs.uiowa.edu/Darwin/`.
3. P. Baumgartner, U. Furbach, and I. Niemelä. Hyper tableaux. In *Proc. JELIA 96*, vol. 1126 of *LNAI*. Springer, 1996.

4. P. Baumgartner, U. Furbach, and F. Stolzenburg. Computing answers with model elimination. *Artificial Intelligence*, 90(1–2):135–176, 1997.
5. P. Baumgartner and R. A. Schmidt. Blocking and other enhancements for bottom-up model generation methods. Technical report, National ICT Australia, `http://www.nicta.com.au/director/research/publications/technical_reports.cfm`, 2006.
6. M. Bezem. Disproving distributivity in lattices using geometry logic. In *Proc. CADE-20 Workshop on Disproving*, 2005.
7. F. Bry and S. Torge. A deduction method complete for refutation and finite satisfiability. In *Logics in Artificial Intelligence: JELIA'98*, vol. 1489 of *LNCS*, pp. 1–17. Springer, 1998.
8. F. Bry and A. Yahya. Positive unit hyperresolution tableaux for minimal model generation. *J. Automated Reasoning*, 25(1):35–82, 2000.
9. K. Claessen and N. Sörensson. New techniques that improve MACE-style finite model building. In *Proc. CADE-19 Workshop on Model Computation*, 2003.
10. C. Fermüller and A. Leitsch. Model building by resolution. In *Computer Science Logic: CSL'92*, vol. 702 of *LNCS*, pp. 134–148. Springer, 1993.
11. M. Fujita, J. Slaney, and F. Bennett. Automatic generation of some results in finite algebra. In *Proc. IJCAI'95*, pp. 52–57. Morgan Kaufmann, 1995.
12. T. Geisler, S. Panne, and H. Schütz. Satchmo: The compiling and functional variants. *J. Automated Reasoning*, 18(2):227–236, 1997.
13. L. Georgieva, U. Hustadt, and R. A. Schmidt. Computational space efficiency and minimal model generation for guarded formulae. In *Logic for Programming, Artificial Intelligence, and Reasoning: LPAR 2001*, vol. 2250 of *LNAI*, pp. 85–99. Springer, 2001.
14. R. Hasegawa, K. Inoue, Y. Ohta, and M. Koshimura. Non-horn magic sets to incorporate top-down inference into bottom-up theorem proving. In *Automated Deduction: CADE-14*, vol. 1249 of *LNCS*, pp. 176–190. Springer, 1997.
15. R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In *Proc. CADE'88*, vol. 310 of *LNCS*, pp. 415–434. Springer, 1988.
16. W. McCune. A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems. Technical Report MCS-TM-194, ANL, 1994.
17. W. McCune. Mace4 reference manual and guide. Technical Memorandum 264, Argonne National Laboratory, 2003.
18. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In J. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier and MIT Press, 2001.
19. J. A. Robinson. Automatic deduction with hyper-resolution. *Internat. J. Computer Math.*, 1(3):227–234, 1965.
20. R. A. Schmidt. MSPASS. `http://www.cs.man.ac.uk/~schmidt/mspass/`.
21. R. A. Schmidt and U. Hustadt. Solvability with resolution of problems in the Bernays-Schönfinkel class. Presented at Dagstuhl Seminar 05431 and ARW 2006, Bristol, 2005.
22. J. Slaney. FINDER (finite domain enumerator): Notes and guide. Technical Report TR-ARP-1/92, Australian National University, 1992.
23. M. E. Stickel. Upside-down meta-interpretation of the model elimination theorem-proving procedure for deduction and abduction. *J. Automated Reasoning*, 13(2):189–210, 1994.
24. C. Weidenbach. SPASS. `http://spass.mpi-sb.mpg.de`.
25. C. Wernhard. System description: KRHyper. In *Proc. CADE-19 Workshop on Model Computation: Principles, Algorithms, Applications Systems*, 2003.
26. H. Zhang. Sem: a system for enumerating models. In *Proc. IJCAI'95*, pp. 298–303. Morgan Kaufmann, 1995.