Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

Blocking and Other Enhancements for Bottom-Up Model Generation Methods

Peter Baumgartner · Renate A. Schmidt

January 18, 2019

Abstract Model generation is a problem complementary to theorem proving and is important for fault analysis and debugging of formal specifications of security protocols, programs and terminological definitions, for example. This paper discusses several ways of enhancing the paradigm of bottom-up model generation, with the two main contributions being a new range-restriction transformation and generalized blocking techniques. The range-restriction transformation refines existing transformations to range-restricted clauses by carefully limiting the creation of domain terms. The blocking techniques are based on simple transformations of the input set together with standard equality reasoning and redundancy elimination techniques, and allow for finding small, finite models. All possible combinations of the introduced techniques and a classical range-restriction technique were tested on the clausal problems of the TPTP Version 6.0.0 with an implementation based on the SPASS theorem prover using a hyperresolution-like refinement. Unrestricted domain blocking gave best results for satisfiable problems, showing that it is an indispensable technique for bottom-up model generation methods, that yields good results in combination with both new and classical range-restricting transformations. Limiting the creation of terms during the inference process by using the new range-restricting transformation has paid off, especially when using it together with a shifting transformation. The experimental results also show that classical range restriction with unrestricted blocking provides a useful complementary method. Overall, the results show bottom-up model generation methods are good for disproving theorems and generating models for satisfiable problems, but less efficient for unsatisfiable problems.

Keywords automated reasoning · model generation · blocking · first-order logic · Bernays-Schönfinkel class

P. Baumgartner
CSIRO/Data61 and Australian National University, CSIT Building (108), North Road
Canberra ACT 2601, Australia
Tel.: +61 2 6218 3717 E-mail: Peter.Baumgartner@data61.csiro.au

R. A. Schmidt
School of Computer Science, The University of Manchester
Oxford Rd, Manchester M13 9PL, UK
Tel.: +44 61 275 6163, Fax.: +44 161 275 6204
E-mail: Renate.Schmidt@manchester.ac.uk

1 Introduction

The bottom-up model generation (BUMG) paradigm encompasses a wide family of calculi and proof procedures designed to construct models of given clause sets by reading clauses as rules and applying them in a bottom-up way until saturation. Variants of (positive) hyper-resolution, grounding tableau calculi, hypertableau calculi and instantiation-based methods belong to this family. BUMG methods have been known for a long time to be useful for proving theorems and the dual task of computing models for satisfiable problems, which is recognized as important to solving mathematical problems [96,99], program synthesis [59], in software engineering [30], model checking, and other applications for fault analysis [9] and debugging of logical specifications [70,18].

This paper discusses several ways of enhancing the methods in the BUMG paradigm for first-order logic with equality (function symbols are allowed). Our presentation is focused on the use of hyperresolution, or the positive strategy or selection-based refinement in the framework of resolution and superposition defined over complete simplification orders and other refinements [75,48,4,68] as a BUMG method. We assume that interpretations are Herbrand interpretations defined by ground atom sets (which is a commonly used definition). To compute such models (i) clauses must be transformed into *range-restricted* form and (ii) these methods must be enhanced with *splitting* of positive ground clauses.

These observations are based on the following commonly exploited insights, e.g. [63, 24,53,42]. A clause is range-restricted when every variable occurring in the positive part of the clause (the head) also occurs in the negative part of the clause (the body). In this class all positive clauses (those without any negative literals) are ground. The class of range-restricted clauses is a reduction class for first-order logic; this means that every first-order formula (and also every first-order clause set) can be transformed in an equi-satisfiability and model-preserving way into a set of range-restricted clauses. In general, hyperresolution resolves n positive clauses with the main clause and the resolvent is again a positive clause. Consequently, on range-restricted clauses, hyperresolution resolves n ground, positive clauses with the main clause, always producing a positive, ground resolvent. In a calculus with splitting, the ground clauses are split into ground positive units and derivations are tree derivations. The positive units in an open branch in such a derivation can be seen to define a (partial) Herbrand model of the input clause set.

One of the contributions of the paper is the introduction to first-order logic of blocking techniques partially inspired by techniques successfully used in description and modal logic tableau-based theorem proving [50,79,2]. In this context blocking is an important mechanism to discover periodicity and merging worlds, or objects, which creates loops in the model reducing the otherwise infinite, freely generated (Herbrand) model to a finite model.[1] A common form of blocking or loop checking in this context is ancestor blocking. In [16] we introduced a first-order version of ancestor blocking and showed how blocking techniques of description and modal logic tableau-based theorem provers can be generalized to full first-order logic with equality. In the present paper this blocking method is named *subterm domain blocking*. An unrestricted version of this blocking technique, called *unrestricted blocking*, was introduced in [79] showing that it allows semantic tableau methods to decide the expressive description logic $\mathcal{ALBO}$. Unrestricted blocking was shown to be a strong technique to devise general tableau-based decision procedures for modal-type logics and expressive description

[1] Our notion of blocking should not be confused with blocking of resolution or superposition steps whose most general unifier replaces a variable by a term that can be reduced by simplification [3].

logics with complex role operators [80,82] and is an important ingredient in the tableau synthesis framework and the prover generator MetTeL [81,56].

Subterm domain blocking and unrestricted blocking drop the requirement to check for a loop in the partially constructed model and allows for the merging of terms to be undone [16, 79], which is an important departure from loop-checking. Roughly, the idea of unrestricted blocking is to add a clause $x \approx y \vee x \not\approx y$ to the input set and use it via a range-restriction transformation and hyperresolution to produce instances $s \approx t \vee s \not\approx t$ for all ground terms s and t. However these clauses are not treated as tautologies, because $\not\approx$ in the literal $x \not\approx y$ is a new predicate constrained by the clause $\neg(x \not\approx y) \vee \neg(x \approx y)$. This means $x \not\approx y$ should be thought as being a *positive* literal and $x \approx y \vee x \not\approx y$ is a positive clause. Splitting on such a clause creates one branch where $s \approx t$ holds and a second branch where it does not hold, but $s \not\approx t$ holds. In the left branch, $s \approx t$ is handled by rewriting; this means that s and t are merged (and reduced to an irreducible term with respect to all the equalities present on the current branch). We can think of the larger of the two terms as being *blocked* by the other one. When no models can be found in the left subderivation rooted by the equation $s \approx t$, standard backtracking reverts the blocking of s and t and the inference process continues with the right branch asserting $s \not\approx t$. This avoids that the two terms are equated again in subsequent steps of the derivation.

Since the clause $x \approx y \vee x \not\approx y$ (together with the 'definition' $\neg(x \not\approx y) \vee \neg(x \approx y)$ of $\not\approx$) is tautologous it can be added to any set of clauses; consequently blocking in this form is generally sound and can be widely used. Any instance of the clause is a tautology (taking into account the 'definition' of $\not\approx$), thus blocking for any specific terms s and t is sound. Even with a constraint $\mathcal{B}$ the clause $\mathcal{B} \rightarrow x \approx y \vee x \not\approx y$ preserves soundness and thus realizes a form of blocking but is triggered only when the constraint $\mathcal{B}$ holds. For instance, the constraint could trigger merging of two terms s and t only when s is a subterm of t. This generalises dynamic anywhere ancestor blocking from the description logic literature to first-order logic.

In this paper we present four different blocking techniques for varying the kinds of models that can be found. The difference between the four techniques is how restrictive the blocking is. *Unrestricted domain blocking* ensures that domain minimal models are generated. With *subterm domain blocking* or *subterm predicate blocking* larger models are produced because two terms are only merged if one is a subterm of the other. With *unrestricted predicate blocking* and *subterm predicate blocking* two terms are merged if they both belong to the extension of a unary predicate symbol, the intention being that less constrained (and thus larger) finite model can be found. This is useful for applications with types.

The second contribution of the paper is a refinement of the 'transformation to range-restricted form' introduced by Winker [96] as a trick to find small counter-examples for solving axiom independence problems. This technique has been used by Manthey and Bry [63] to build the SATCHMO prover based on hyperresolution and splitting. An improved range-restriction transformation can be found in [15]. These range-restricting transformations have the property that they force BUMG methods to enumerate the entire Herbrand universe and are therefore non-terminating except in the simplest cases. Two solutions are presented in this paper: One solution is to combine classical range-restriction transformations [96,63] with blocking. Another solution is to modify the range-restricting transformation so that new terms are created only when needed. All the BUMG enhancements introduced in this paper are defined as transformations of the clause sets and are shown to preserve satisfiability equivalence or equivalence with respect to E-satisfiability, where E is the theory of equality.

Since the enhancements are encoded on the clausal level for use with standard hyper-resolution techniques, we have implemented them using the SPASS theorem prover [94,93].

The third contribution is a comprehensive evaluation of the different techniques on the TPTP problem library [89] with an analysis of the results leading to interesting findings.

The transformations enable standard resolution theorem proving approaches to decide the Bernays-Schönfinkel class (with or without equality).[2] For this class blocking is not essential. Moreover, we claim that BUMG methods with blocking can decide fragments of first-order logic with the finite model property and can construct finite models for finitely satisfiable formulae.

The structure of the paper is as follows. Definitions of basic terminology and notation can be found in Section 2. In Section 3 we recall the characteristic properties of BUMG methods and recall the definition of hyperresolution with splitting. The main part of the paper are Sections 4 to 10. Sections 4, 5, 6 and 7 define new techniques for generating small models and generating them more efficiently. The techniques are based on a series of transformations including a refined range-restricting transformation (Section 4), its modification for equality (Section 5), instances of standard renaming and flattening (Section 6), and the introduction of blocking in various forms via amendments of the clause set and standard saturation-based equality reasoning (Section 7). The transformations are shown to be sound and complete in Section 8. One consequence of the results is a general decidability result of the Bernays-Schönfinkel class for all BUMG methods and related approaches. This is presented in Section 9. In Section 10 we present and discuss results of an evaluation carried out with our implementation in SPASS-Yarralumla on clausal problems in the TPTP library. Section 11 summarizes and discusses related work.

This paper is an extended and improved version of [16].

2 Basic Definitions

We use standard terminology from automated reasoning. We assume as given a signature $\Sigma = \Sigma_f \cup \Sigma_P$ of function symbols Σ_f and predicate symbols Σ_P of given arities. We assume Σ_f contains at least one constant symbol. As we are working (also) with equality, we assume Σ_P contains a distinguished binary predicate symbol $\approx$. Terms, atoms, literals and formulas over Σ and a given (denumerable) set of variables V are defined as usual. In particular, every term is either a variable or a *functional term* $f(t_1,\ldots,t_n)$, i.e., the application of an n-ary function symbol f to n terms $t_1,\ldots,t_n$, for some $n \geq 0$. A function symbol c of arity 0 is also called a *constant* and we just write c instead of the functional term $c()$. A functional term is called *compound* (or *proper*) if it is not a constant. An *atom* is of the form $P(t_1,\ldots,t_n)$ where P is an n-ary predicate symbol, for some $n \geq 0$. *Equations* are atoms of the form $\approx(t_1,t_2)$, which we always write using infix notation as $t_1 \approx t_2$. A *literal* is an atom or the negation of an atom. When L is a literal we write $L[t]$ to indicate that L contains a subterm t at a position left implicit.

We treat the Boolean operators $\wedge$ and $\vee$ as multi-arity operators that ignore the order of their arguments, which is justified by their associativity and commutativity properties. In other words, for instance, a disjunction $H_1 \vee \cdots \vee H_n$ of atoms is treated as the (possibly empty) multiset $\{H_1,\ldots,H_n\}$; similarly for conjunctions.

A clause is a (finite) implicitly universally quantified disjunction $\neg B_1 \vee \cdots \vee \neg B_k \vee H_1 \vee \cdots \vee H_m$ of literals, where $m,k \geq 0$. To emphasize their bottom-up operational semantics we write clauses in implication style $B_1 \wedge \cdots \wedge B_k \rightarrow H_1 \vee \cdots \vee H_m$. The part $H_1 \vee \cdots \vee H_m$ is

[2] The Bernays-Schönfinkel class with equality is sometimes referred to as the Bernays-Schönfinkel-Ramsay class.

called the *head* of the clause and $B_1 \wedge \cdots \wedge B_k$ is its *body*. We write $\bot$ for the empty head. Each H_i, if any, is called a *head atom*, and each B_j, if any, is called a *body atom*. Clauses with empty head are called *negative clauses*, and clauses with empty body are called *positive clauses*. Positive clauses are written as $H_1 \vee \cdots \vee H_m$, instead of $\rightarrow H_1 \vee \cdots \vee H_m$. Notice the empty clause is written as $\bot$.

When writing expressions such as $B \wedge \mathcal{B} \rightarrow H \vee \mathcal{H}$ we mean any clause whose head literals are H and those in the disjunction (or multiset) of literals $\mathcal{H}$, and whose body literals are B and those in the conjunction (or multiset) of literals $\mathcal{B}$.

Let F be a term, literal, head, body or clause. By var(F) we denote the set of variables occurring in F. If $\text{var}(F) = \emptyset$ then F is *ground*. As usual, a *substitution* is a mapping from variables to terms that is the identity almost everywhere. We identify a substitution σ with its homomorphic extension to terms, literals, heads, bodies and clauses. Consequently, $\sigma(F)$ denotes the result of simultaneously replacing in F every occurrence of every variable $x \in \text{var}(F)$ by $\sigma(x)$. We use the notion of (most general) unifiers in the standard way. Given two terms or atoms e_1 and e_2 we write $\sigma = \text{mgu}(e_1, e_2)$ to indicate that the substitution σ is a most general unifier of e_1 and e_2. We also need the notion of simultaneous most general unifiers: given a set $E = \{(e_1^1, e_2^1), \ldots, (e_1^n, e_2^n)\}$ of pairs of terms or atoms, where $n \geq 0$, we say that σ is a *simultaneous most general unifier of* E and write $\sigma = \text{smgu}(E)$ iff σ is a most general substitution such that $\sigma(e_1^i) = \sigma(e_2^i)$ for all $i \in \{1, \ldots, n\}$. It is well-known that a simultaneous most general unifier can always be computed (if it exists) by iterated most general unifier computation.

For a given atom $P(t_1, \ldots, t_n)$, the terms $t_1, \ldots, t_n$ are 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. For example, for a clause $\mathcal{B} \rightarrow \mathcal{H}$ the top-level terms of its body $\mathcal{B}$ are exactly the top-level terms of its body atoms.

A *(standard first-order) interpretation* I consists of a non-empty domain, denoted as |I|, and interpretation functions P^I and f^I on |I|, for every $P \in \Sigma_P$ and $f \in \Sigma_f$, as usual. A *valuation* μ is a (total) mapping from the set of variables to |I|. We form pairs (I, μ) for evaluating terms, atoms and formulas in the usual way and write, e.g., $(I, \mu)(t)$ for the result of evaluating the term t. We write $(I, \mu) \models F$ to denote the fact that (I, μ) satisfies F. If a term t does not have free variables we may write $I(t)$ instead of $(I, \mu)(t)$ without loosing anything. Similarly for atoms, formulas and clauses.

As usual, the *Herbrand universe* HU_Σ *of* Σ is the set of all ground terms of the signature Σ_f, and the *Herbrand base* HB_Σ *of* Σ is the set of all ground atoms of Σ. A *Herbrand interpretation* I is identified with a subset of ground atoms in HB_Σ, namely, those that are true in the interpretation. Using the above terminology, for every Herbrand interpretation I we have $|I| = HU_\Sigma$ and $I(t) = t$ for every $t \in HU_\Sigma$.

We work mostly, but not always, with Herbrand interpretations. Of particular interest are standard first-order interpretations that behave like Herbrand interpretations for a subset of the Herbrand universe. They are introduced as *quasi-Herbrand interpretations* below. Indeed, the purpose of our transformations is to enable BUMG methods to construct such interpretations with a domain as small as possible (and ideally a finite one).

Let us say that a set $U \subseteq HU_\Sigma$ of ground terms is *subterm-closed* if with every $t \in U$ the set U also contains every subterm of t.

Definition 1 (Quasi-Herbrand interpretation) A *quasi-Herbrand interpretation* is a standard first-order interpretation I with domain $|I| \subseteq HU_\Sigma$ such that |I| is subterm-closed and $I(t) = t$ for every $t \in |I|$. Furthermore, |I| must contain at least one constant c.

Every Herbrand interpretation is a quasi-Herbrand interpretation by taking $|\mathrm{I}| = \mathrm{HU}_\Sigma$. Unlike in Herbrand interpretations, function symbols are not always trivially interpreted as total functions in quasi-Herbrand interpretations. For instance, in the presence of a constant a, a unary function symbol f, and the domain $|\mathrm{I}| = \{\mathsf{a}, \mathsf{f}(\mathsf{a})\}$, say, one has to assign a value in the interpretation to every term. However $\mathsf{f}(\mathsf{f}(\mathsf{a}))$, for instance, cannot be evaluated to itself, as $\mathsf{f}(\mathsf{f}(\mathsf{a})) \notin |\mathrm{I}|$ and, hence, one has to have $\mathsf{f}^{\mathrm{I}}(\mathsf{f}(\mathsf{a})) = \mathsf{a}$ or $\mathsf{f}^{\mathrm{I}}(\mathsf{f}(\mathsf{a})) = \mathsf{f}(\mathsf{a})$. But still we can specify quasi-Herbrand interpretations I as sets of atoms, as with Herbrand interpretations, however in conjunction with interpretation functions f^{I} for every $f \in \Sigma_f$.

Satisfiability and validity in a Herbrand interpretation of ground literals, clauses, and clause sets are defined as expected. We write $\mathrm{I} \models \mathrm{F}$ to denote that I satisfies F, where F is a ground literal, clause or clause set (read conjunctively). For instance, for a ground clause $\mathcal{B} \to \mathcal{H}$ it holds that $\mathrm{I} \models \mathcal{B} \to \mathcal{H}$ iff $\mathrm{I} \not\models \mathcal{B}$ or $\mathrm{I} \models \mathcal{H}$. The satisfaction relation $\models$ carries over to non-ground clauses and clause sets by taking a clause as the set of its ground instances and a clause set as the union of their sets of ground instances, respectively. If M is a clause set and $\mathrm{I} \models \mathrm{M}$ we say that I *is a Herbrand model of* M.

If I is instead a quasi-Herbrand interpretation then, as the domain $|\mathrm{I}|$ consists of terms, we can still define satisfaction of non-ground clauses (and clause sets) in terms of substitutions instead of using pairs (I, μ). For instance, $\mathrm{I} \models \mathcal{B} \to \mathcal{H}$ iff for all substitutions γ with codomain $|\mathrm{I}|$ it holds that $\mathrm{I} \models (\mathcal{B} \to \mathcal{H})\gamma$.

An *E-interpretation* is a standard first-order interpretation I such that $(\mathrm{I}, \mu) \models s \approx t$ if and only if $(\mathrm{I}, \mu)(s) = (\mathrm{I}, \mu)(t)$. We use the notion of *E-(un)satisfiability* in the expected way, as (un)satisfiability with respect to E-interpretations. In particular, if M is a clause set and $\mathrm{I} \models \mathrm{M}$ we say that I *is an E-model of* M.

An E-interpretation is not necessarily a Herbrand interpretation, but E-satisfiability is the same as satisfiability in Herbrand interpretations that interpret $\approx$ as a congruence relation. In such Herbrand interpretations the following is true: for every ground term t, $t \approx t \in \mathrm{I}$, and for every ground atom A (including ground equations) whenever $\mathrm{I} \models \mathrm{A}[s]$ and $\mathrm{I} \models s \approx t$, then $\mathrm{I} \models \mathrm{A}[t]$. Alternatively, given such a Herbrand interpretation I its *induced E-interpretation* $\mathrm{I}_\approx$ is given by the domain $\mathrm{I}_\approx = |\mathrm{I}|/\!\approx$, the partitioning of $|\mathrm{I}|$ into its equivalence classes modulo $\approx$. Let $[t]_\approx$, or just $[t]$, denote the equivalence class modulo $\approx$ of a ground term $t \in |\mathrm{I}|$ $(= \mathrm{HU}_\Sigma)$. Each n-ary function $f \in \Sigma_f$ is interpreted in $\mathrm{I}_\approx$ as $f^{\mathrm{I}_\approx}([t_1], \ldots, [t_n]) = [f(t_1, \ldots, t_n)]$, and each n-ary predicate symbol $\mathrm{P} \in \Sigma_\mathrm{P}$ is interpreted in $\mathrm{I}_\approx$ as $\mathrm{P}^{\mathrm{I}_\approx}([t_1], \ldots, [t_n]) = \mathrm{P}^{\mathrm{I}}(t_1, \ldots, t_n)$. This is a well-known construction and it holds that a clause set M is satisfied by a model I iff it is E-satisfied by $\mathrm{I}_\approx$. See [36] for details.

Yet another characterization is to add to a given clause set M its equality axioms $\mathrm{EAX}(\Sigma)$, that is, the axioms expressing that $\approx$ is a congruence relation on the terms and atoms induced by the predicate symbols Σ_P and function symbols Σ_f occurring in M. It is well-known that M is E-satisfiable iff $\mathrm{M} \cup \mathrm{EAX}(\Sigma)$ is satisfiable [36].

3 BUMG Methods

Reasoners 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) for first-order logic with equality and function symbols. BUMG approaches use a forward reasoning approach where implications or clauses, $\mathcal{B} \to \mathcal{H}$, are read as rules and are repeatedly used to derive (instances of) the head $\mathcal{H}$ from (instances of) the body $\mathcal{B}$ until saturation is achieved.

The family of BUMG approaches includes many familiar calculi and proof procedures such as Smullyan-type semantic tableaux [86], SATCHMO [63,41], positive unit hyperresolution (PUHR) tableaux [23,24], the model generation theorem prover MGTP [37] and hypertableaux [12]. A well-established and widely known method for BUMG is hyperresolution [75]. In the framework of resolution and superposition-based on complete simplification orders, positive strategies are BUMG methods (also known as negative selection-based refinements). Relevant references are [32] for the Horn case and [48,4] for the general case. As outlined in the Introduction, BUMG can be realized by simple combinations of insights and results based on such hyperresolution-style methods. *In a nutshell, what is exploited is that any first-order formula can be effectively transformed into a class of range-restricted clauses, and on range-restricted clauses these methods are instantiation methods, the positive premises are always positive ground clauses and the conclusion is a positive ground clause. With splitting they produce models represented as sets of positive ground atoms, which is a common representation of Herbrand models (e.g. [63,42]).*

In this paper we describe methods for transforming a given clause set into another clause set so that BUMG methods benefit from the transformation. Our theoretical results are all of a semantic nature, in essence, equi-satisfiability results, which are complemented by practical experiments with a BUMG prover. Strictly speaking, we do not need to get into the details of the calculi or proof procedures behind BUMG methods for the purpose of this paper. For the sake of self-containedness, however, we summarize one BUMG method, the hyperresolution calculus extended with a splitting rule. A good in-depth overview of all aspects of the following brief exposition can be found in [92,6], see also [63,51,31,53, 42,73] where splitting is used in resolution calculi, including hyperresolution calculi. The methods introduced below can be adapted to other BUMG methods without great effort.

In its most basic form, the hyperresolution calculus consists of two inference rules, positive hyperresolution (HRes) and positive factoring (Fact):

$$\mathsf{HRes}\ \frac{\mathrm{B}_1 \wedge \ldots \wedge \mathrm{B}_n \rightarrow \mathcal{H} \qquad \mathrm{H}_1 \vee \mathcal{H}_1 \qquad \cdots \qquad \mathrm{H}_n \vee \mathcal{H}_n}{(\mathcal{H} \vee \mathcal{H}_1 \vee \cdots \vee \mathcal{H}_n)\sigma} \quad \text{if } \sigma = \mathrm{smgu}\{(\mathrm{B}_1, \mathrm{H}_1), \ldots, (\mathrm{B}_n, \mathrm{H}_n)\}$$

$$\mathsf{Fact}\ \frac{\mathrm{H}_1 \vee \mathrm{H}_2 \vee \mathcal{H}}{(\mathrm{H}_1 \vee \mathcal{H})\sigma} \text{ if } \sigma = \mathrm{mgu}(\mathrm{H}_1, \mathrm{H}_2)$$

On ground clauses, positive factoring amounts to the elimination of duplicate head atoms. In the sequel we often omit the inference rule qualifier 'positive'.

A derivation then is a (possibly infinite) sequence of clauses that starts with the given clauses, in any order, and that is extended step by step with the conclusion of an inference rule applied to premise clauses earlier in the sequence. A derivation that contains (ends with) the empty clause is called a *refutation*, which indicates that the given clause set is unsatisfiable.

Proof procedures for resolution do not directly search for derivations (say, by backtracking) and instead are *saturation-based.* That is, a clause set initially comprised of the given clauses is closed under application of the inference rules modulo deletion of redundant clauses (e.g., subsumed clauses, tautological clauses). Once the empty clause is derived a refutation can be extracted easily from the history of inference rule applications. If the empty clause is not derived then the given clause set is satisfiable. In the finite case, a (Herbrand) model can be extracted from the positive clauses of the saturated state. Specifically, for each such clause $\mathcal{H}$ it suffices to pick an atom $\mathrm{H} \in \mathcal{H}$ and assign true in the model to all ground

instances of H. While this is a crude scheme (and can be improved in practice by ordering restrictions) it is helpful for getting an understanding of the transformations below.

Most BUMG methods for first-order logic include some form of β-rule (in tableaux parlance) or splitting rule. If $C \vee D$ is a clause such that C and D are both non-empty and do not share variables then we say that $C \vee D$ is *splittable*. The terminology 'splitting $C \vee D$' refers to deriving from a clause set $M \cup \{C \vee D\}$ the two clause sets $M \cup \{C\}$ and $M \cup \{D\}$. By variable-disjointness, $M \cup \{C \vee D\}$ is equi-satisfiable with the conjunction of the two latter clause sets. Notice that derivations (or saturations) with a splitting rule enabled are tree-shaped now, in accordance with the splittings applied in the course of a derivation. Moreover, the empty clause has to be derived in every branch to make a refutation. As for model generation, it suffices to derive one branch in a saturated state and extract a model from that.

The splitting rule can be defined with arbitrary restrictions, as deemed useful, without loosing refutational completeness. For instance, the SPASS prover [94,93] implements a version that splits on positive parts of clauses only. In the hyperresolution context it can be defined (approximately) as follows:

$$\textsf{Split}\quad \frac{\mathcal{B} \to \mathcal{H}_1 \vee \mathcal{H}_2}{\mathcal{B} \to \mathcal{H}_1 \quad | \quad \mathcal{B} \to \mathcal{H}_2}$$

$$\text{if } \mathrm{var}(\mathcal{B} \to \mathcal{H}_1) \cap \mathrm{var}(\mathcal{B} \to \mathcal{H}_2) = \emptyset,\ \mathcal{H}_1 \neq \emptyset \text{ and } \mathcal{H}_2 \neq \emptyset$$

Making the Split rule mandatory, however, can be useful for model computation. In particular, if all given and derived positive, non-unit clauses $H \vee \mathcal{H}$ are splittable into a unit clause H and a rest-clause $\mathcal{H}$ then every branch not containing the empty clause in its saturated state specifies a model simply by these unit clauses. *Indeed, our transformations below make sure it is always the case that all derived positive non-unit clauses* $H \vee \mathcal{H}$ *are splittable in this way, and, moreover, that the (ground) unit clauses* H *in a saturated state not containing the empty clause specify a model of the given clause set.*

Our experiments show the splitting rule is useful for BUMG (cf. Section 10). Moreover, for our blocking transformations (considered later in Section 7), splitting on positive ground clauses (those with an empty body $\mathcal{B}$ in the Split rule) is in fact mandatory to make it effective. Most BUMG procedures support this splitting technique.

Another crucial requirement for the effective use of blocking is support of equality reasoning (for example, ordered paramodulation, rewriting or superposition [48,5,68]), in combination with simplification techniques based on orderings. We refer to [5,7,20] for general notions of redundancy in superposition-based theorem proving approaches. However, for the sake of self-containedness we include inference rules for equality below:

$$\textsf{EqRes}\quad \frac{s \approx t \wedge \mathcal{B} \to \mathcal{H}}{(\mathcal{B} \to \mathcal{H})\sigma} \text{ if } \sigma = \mathrm{mgu}(s,t)$$

$$\textsf{ParaUnitPos}\quad \frac{\mathrm{H}[s] \qquad s \approx t}{\mathrm{H}[t]}$$

$$\textsf{ParaNeg}\quad \frac{\mathrm{B}[u] \wedge \mathcal{B} \to \mathcal{H} \qquad s \approx t}{(\mathrm{B}[t] \wedge \mathcal{B} \to \mathcal{H})\sigma} \text{ if } u \text{ is not a variable and } \sigma = \mathrm{mgu}(u,s)$$

The EqRes rule eliminates a body equation $s \approx t$ whose sides can be made equal by means of a most general unifier σ. The ParaUnitPos rule replaces in a head atom (equational or otherwise) a subterm s by t in presence of a unit clause $s \approx t$. The ParaNeg rule is similar, but it targets a body atom and it uses unification to make the target term u and s equal. In both rules the right premise $s \approx t$ stands also for its symmetric version $t \approx s$.

These inference rules are slimmed-down versions of more sophisticated inference rules for equality reasoning as exhibited, e.g., in the superposition calculus. First, all inference rules can be restricted by applying them only if certain ordering restrictions are met. We refer the reader to the literature cited above. Second, the inference rule ParaUnitPos does *not* use unification. This is not necessary in our context as all positive unit clauses, derived or given, are always ground (so that unifiability coincides with syntactic equality). We also do not need to formulate the rule on non-unit clauses, as (positive) non-unit clauses are always split eagerly into unit clauses. The ParaNeg rule, however, is more general by being applicable to clauses of arbitrary shape and by using unification.[3]

To sum up, the inference rules of our prototypical BUMG method are HRes, Fact (which is actually not necessary in our context), Split, EqRes, ParaUnitPos and ParaNeg. We assume that the Split rule is applied eagerly.[4] The hypertableau calculus with equality [13, 14] features similarly structured inference rules, but includes ordering restrictions and redundancy elimination techniques.

Figure 1 shows a deliberately simple sample derivation. It exhibits an initially given clause set M, as stated, and its rr transformed version rr(M) for the derivation, proper. The rr transformation is introduced only below; how it is obtained is not important yet. The purpose of the example is to convey the flavour and some important aspects of the approach.

Referring to Figure 1, if we develop the derivation tree in a depth-first left-to-right fashion, say, then the branch with Case (1) leads to the empty clause. The derivation hence continues by branching into Case (2), Case (2.1) and then Case (2.1.1). No new clause is derivable here and the derivation stops. (Other branches are infinite.) The positive unit clauses in this saturated state are

a ≈ b	P(a)	dom(a)	Q(f(a))
	P(b)	dom(b)	Q(f(b))
	P(f(a))	dom(f(a))	R(f(a))
	P(f(b))	dom(f(b))	R(f(b))

They can be interpreted as a model of M in two ways. First, as a Herbrand model whose domain is the set of all ground terms and that assigns true to all atoms in the saturated state. Notice that $\approx$ is interpreted as a congruence relation. (Corollary 1 below has the formal result.) Second, as an E-interpretation that is derived from the quasi-Herbrand model with a (finite) domain. The (finite) domain of that quasi-Interpretation J is $|\mathsf{J}| = \{\mathsf{a}, \mathsf{b}, \mathsf{f(a)}, \mathsf{f(b)}\}$ as specified by the dom-atoms in the saturated state. The induced E-interpretation $\mathsf{J}_{\approx}$ has a domain consisting of equivalence classes determined by the congruence relation obtained from J and the interpretation function f^{J}. In this construction, ground terms not in $|\mathsf{J}|$ are put into equivalence classes by uniformly evaluating them to a

[3] Strictly speaking, the ParaNeg rule is not necessary to get a complete calculus. However, as soon as ordering restrictions are added it will become necessary.

[4] Again this is very crude. A more realistic calculus would only require that a split inference rule cannot be deferred indefinitely unless the clause it is applied to becomes redundant at some point in the derivation.

Initially given clause set M	**rr-transformed clause set rr**(M)	
$a \approx b$	$a \approx b$	$\mathsf{dom}(a)$
$P(x)$	$\mathsf{dom}(x) \rightarrow P(x)$	$x \approx y \rightarrow \mathsf{dom}(x)$
$P(x) \rightarrow R(x) \vee Q(f(x))$	$P(x) \rightarrow R(x) \vee Q(f(x))$	$x \approx y \rightarrow \mathsf{dom}(y)$
$R(b) \rightarrow \bot$	$R(b) \rightarrow \bot$	$P(x) \rightarrow \mathsf{dom}(x)$
		$Q(x) \rightarrow \mathsf{dom}(x)$
		$R(x) \rightarrow \mathsf{dom}(x)$
		$\mathsf{dom}(f(x)) \rightarrow \mathsf{dom}(x)$

Derivation

$\mathsf{dom}(b)$
$P(a)$
$P(b)$
$R(a) \vee Q(f(a))$ // Split into Case (1) – Case (2)

Case (1)	**Case (2)**
$R(a)$	$Q(f(a))$
$R(b)$	$Q(f(b))$
$\bot$ // Empty clause	$\mathsf{dom}(f(a))$
	$\mathsf{dom}(f(b))$
	$P(f(a))$
	$P(f(b))$
	$R(f(a)) \vee Q(f(f(a)))$
	// Split $R(f(a)) \vee Q(f(f(a)))$ into Case (2.1) – Case (2.2)

Case (2.1)	**Case (2.2)**
$R(f(a))$	$Q(f(f(a)))$
$R(f(b))$	$Q(f(f(b)))$
$R(f(b)) \vee Q(f(f(b)))$	...
// Split $R(f(b)) \vee Q(f(f(b)))$ into Case (2.1.1) – Case (2.1.2)	

Case (2.1.1)	**Case (2.1.2)**
// No new clause derived	$Q(f(f(a)))$
// Saturated state	$Q(f(f(b)))$
	...

Fig. 1 Sample derivation. The 'Derivation' part only lists the newly derived clauses, those extending the derivation starting with the clauses from rr(M). Likewise, a sub-case implicitly includes all clauses derived in its parent case.

default constant. In the example, if a is that constant, we get f(f(a)) $\approx$ a, f(f(b)) $\approx$ a and $|\mathsf{J}_{\approx}| = \{\{\mathsf{a}, \mathsf{b}, \mathsf{f}(\mathsf{f}(\mathsf{a})), \mathsf{f}(\mathsf{f}(\mathsf{b})), \mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a})))), \ldots\}, \{\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}), \mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))), \ldots\}\}$. Every element in $|\mathsf{J}_{\approx}|$ is represented by an element from $|\mathsf{J}|$, but not vice versa. The construction is explained in detail in Section 5.

4 Range-Restricting Transformations

Definition 2 (Range-restricted clause (set) [63]) A clause $\mathcal{B} \rightarrow \mathcal{H}$ is *range-restricted* iff $\mathrm{var}(\mathcal{H}) \subseteq \mathrm{var}(\mathcal{B})$, i.e., the body $\mathcal{B}$ contains all the variables in the clause. A clause set is *range-restricted* iff it contains only range-restricted clauses. □

This means that a positive clause $\mathcal{H}$ is range-restricted only if it is a ground clause. A negative clause $\mathcal{B} \rightarrow \bot$ is always range-restricted. Following [63], the algorithm crr (for *classical range-restriction*) below transforms a given clause set into a range-restricted set.

Algorithm crr(M)
  // Input: a clause set M
  // Output: a range-restricted version of M
  let dom = a fresh unary predicate symbol not in Σ_{P}
  // Step (1): initialization
  let c = any constant in Σ_f
  var *res* := {dom(c)} // initialized result clause set
  // Step (2): domain restriction of clauses in M:
  foreach $(\mathcal{B} \rightarrow \mathcal{H}) \in$ M **do**
    let $\{x_1, \ldots, x_k\} = \mathrm{var}(\mathcal{H}) \setminus \mathrm{var}(\mathcal{B})$
    res := *res* $\cup$ $\{\mathsf{dom}(x_1) \wedge \cdots \wedge \mathsf{dom}(x_k) \wedge \mathcal{B} \rightarrow \mathcal{H}\}$
  // Step (3): domain upward closure:
  foreach $f \in \Sigma_f$ where n is the arity of f **do**
    res := *res* $\cup$ $\{\mathsf{dom}(x_1) \wedge \cdots \wedge \mathsf{dom}(x_n) \rightarrow \mathsf{dom}(f(x_1, \ldots, x_n))\}$
  return *res*

It is not difficult to see that crr(M) is indeed range-restricted for any clause set M. In Step (3) the arity n of $f \in \Sigma_f$ can be 0, that is, constants are included. The transformation is sound and complete, that is, M is satisfiable iff crr(M) is satisfiable [63,24]. The size of crr(M) is linear in the size of M and can be computed in linear time.

Perhaps the easiest way to understand the transformation achieved by the classical range restriction algorithm crr is to imagine we use a BUMG method, for example, hyperresolution as described in Section 3. The idea is to build the model(s) during the derivation. The clause used in the initialization in Step (1) ensures that the domain of interpretation given by the domain predicate dom is non-empty. (Recall, we assume Σ_f contains at least one constant.) The loop in Step (2) turns clauses into range-restricted clauses. This is done by shielding the variables $\{x_1, \ldots, x_k\}$ in the head, that do not occur negatively, with the added dom-literals in the body. Clauses that are already range-restricted are unaffected by this step. The loop in Step (3) ensures that all elements of the Herbrand universe of the (original) clause set are added to the domain via hyperresolution inference steps. Notice that all positive non-unit clauses given in crr(M) or derived from these are splittable (because hyperresolvents on range restricted clauses are positive ground clauses), which suggests the use of the Split rule.

As a consequence a clause set M with at least one non-nullary function symbol causes hyperresolution derivations on crr(M) to be unbounded, unless M is unsatisfiable. This is a negative aspect 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 [24,23]. In particular, [23] use splitting and the δ^*-rule to generate domain minimal models. In the present research we have evaluated the combination of blocking techniques (introduced later in Section 7) with the classical range-restricting transformation crr. This has shown promising empirical results as presented in Section 10.

Next, we introduce an improved transformation to range-restricted form. Instead of enumerating the generally infinite Herbrand universe in a bottom-up fashion, the intuition is that it generates terms only as needed. A major difference to the crr transformation is that it involves term abstraction for body atoms.

Definition 3 (Term abstraction) Let $\mathrm{P}(t_1,\ldots,t_n)$ be an atom and $x_1,\ldots,x_n$ distinct, 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 $\mathrm{P}(s_1,\ldots,s_n)$ is called the *term abstraction of* $\mathrm{P}(t_1,\ldots,t_n)$. □

For example, the term abstraction of $\mathsf{P}(x,x,\mathsf{a},\mathsf{f}(x))$ is $\mathsf{P}(x,x,x_3,x_4)$.

The *range-restricting transformation* of a clause set M, denoted by rr(M), is defined by the following algorithm (explanations and an example are given afterwards).

Algorithm rr(M)
  // Input: a clause set M
  // Output: a range-restricted version of M
  let dom = a fresh unary predicate symbol not in Σ_{P}
  // Step (1): initialization
  let c = any constant in Σ_f
  var *res* := {dom(c)} // initialized result clause set
  var N := ∅ // Auxiliary set of clauses
  // Step (2): domain elements from clause bodies:
  foreach $(\mathcal{B} \to \mathcal{H}) \in \mathrm{M}$ **do**
    foreach $\mathrm{P}(t_1,\ldots,t_n) \in \mathcal{B}$ **do**
      let $\mathrm{P}(s_1,\ldots,s_n)$ = the term abstraction of $\mathrm{P}(t_1,\ldots,t_n)$
      $\mathrm{N} := \mathrm{N} \cup \{\mathrm{P}(s_1,\ldots,s_n) \to \mathsf{dom}(t_i) \mid 1 \le i \le n \text{ and } t_i \text{ is not a variable}\}$
  // Step (3): classical domain restriction of clauses in $\mathrm{M} \cup \mathrm{N}$:
  foreach $(\mathcal{B} \to \mathcal{H}) \in \mathrm{M} \cup \mathrm{N}$ **do**
    let $\{x_1,\ldots,x_k\} = \mathrm{var}(\mathcal{H}) \setminus \mathrm{var}(\mathcal{B})$
    $\mathit{res} := \mathit{res} \cup \{\mathsf{dom}(x_1) \wedge \cdots \wedge \mathsf{dom}(x_k) \wedge \mathcal{B} \to \mathcal{H}\}$
  // Step (4): domain elements from Σ_{P}-literals:
  foreach $\mathrm{P} \in \Sigma_{\mathrm{P}}$ **do**
    let $\{x_1,\ldots,x_n\}$ = n distinct variables,where n is the arity of P
    $\mathit{res} := \mathit{res} \cup \{\mathrm{P}(x_1,\ldots,x_n) \to \mathsf{dom}(x_i) \mid 1 \le i \le n\}$
  // Step (5): domain elements from Σ_f:
  foreach $f \in \Sigma_f$ **do**
    let $\{x_1,\ldots,x_n\}$ = n distinct variables,where n is the arity of f
    $\mathit{res} := \mathit{res} \cup \{\mathsf{dom}(f(x_1,\ldots,x_n)) \to \mathsf{dom}(x_i) \mid 1 \le i \le n\}$
  return *res*

The intuition of the rr transformation reveals itself if we think of what happens when using hyperresolution with splitting. The idea is again to build model(s) during the derivation, but this time terms are added to the domain only *as necessary*.

The initialization of the dom-relation in Step (1) is the same as in the definition of crr. The auxiliary clauses collected in N in Step (2) cause functional terms that occur in clause bodies to be inserted into the domain. Below, after Definition 4 we explain *why* these clauses are needed. Step (3) is the classical range-restrictedness transformation applied to the given clauses and the auxiliary clauses N. 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 rr transformation consider the clause set M comprised of the single clause

$$\mathsf{P}(\mathsf{a},\mathsf{f}(x,y),x) \rightarrow \mathsf{Q}(x,\mathsf{g}(x,y)) \vee \mathsf{R}(y,z) \ . \qquad (\dagger)$$

Suppose the clause initializing *res* in Step (1) is $\mathsf{dom}(\mathsf{a})$. In Step (2) the term abstraction of the body literal of clause (†) is $\mathsf{P}(x_1,x_2,x)$. The clauses added to N are the following:

$$\begin{aligned} \mathsf{P}(x_1,x_2,x) &\rightarrow \mathsf{dom}(\mathsf{a}) \\ \mathsf{P}(x_1,x_2,x) &\rightarrow \mathsf{dom}(\mathsf{f}(x,y)) \ . \end{aligned} \qquad (\ddagger)$$

Notice that among the three clauses in $\mathrm{M} \cup \mathrm{N}$ the clauses (†) and (‡) are not range-restricted. They are however added as range-restricted clauses to *res* in Step (3), namely:

$$\begin{aligned} \mathsf{dom}(z) \wedge \mathsf{P}(\mathsf{a},\mathsf{f}(x,y),x) &\rightarrow \mathsf{Q}(x,\mathsf{g}(x,y)) \vee \mathsf{R}(y,z) \qquad (\dagger\dagger) \\ \mathsf{dom}(y) \wedge \mathsf{P}(x_1,x_2,x) &\rightarrow \mathsf{dom}(\mathsf{f}(x,y)) \ . \end{aligned}$$

Step (4) generates clauses responsible for inserting the terms that occur in the heads of clauses into the domain. That is, for each $i \in \{1,2,3\}$ and each $j \in \{1,2\}$ these clauses are added.

$$\begin{aligned} \mathsf{P}(x_1,x_2,x_3) &\rightarrow \mathsf{dom}(x_i) \\ \mathsf{Q}(x_1,x_2) &\rightarrow \mathsf{dom}(x_j) \\ \mathsf{R}(x_1,x_2) &\rightarrow \mathsf{dom}(x_j) \end{aligned}$$

Now, to explain the effect of the rr transformation assume additional clauses so that 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 (†) is derived (by hyperresolution with splitting, say). With the available clauses, $\mathsf{dom}(\mathsf{a})$ and $\mathsf{dom}(\mathsf{g}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{a})))$ are also derived. It is not necessary to insert the terms of the instance of the other head atom $\mathsf{R}(y,z)$ 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) 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 in Step (5). That is, for each $j \in \{1,2\}$ these clauses are added.

$$\begin{aligned} \mathsf{dom}(\mathsf{f}(x_1,x_2)) &\rightarrow \mathsf{dom}(x_j) \\ \mathsf{dom}(\mathsf{g}(x_1,x_2)) &\rightarrow \mathsf{dom}(x_j) \end{aligned}$$

In the example, from $\mathsf{dom}(\mathsf{g}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{a})))$ and the second of these clauses, the clause $\mathsf{dom}(\mathsf{f}(\mathsf{a},\mathsf{a}))$ is derivable. Notice that, $\mathsf{dom}(\mathsf{g}(\mathsf{a},\mathsf{a}))$ for example is not derivable. Indeed, $\mathsf{g}(\mathsf{a},\mathsf{a})$ it is not a subterm of $\mathsf{dom}(\mathsf{g}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{a})))$.

We are interested in quasi-Herbrand interpretations that mimic a given Herbrand interpretation for rr(M) as close as possible, in the following way.

Definition 4 (Induced quasi-Herbrand interpretation) Let M be a clause set and suppose that rr(M) is satisfied by a Herbrand interpretation I. Let c be the constant in Step (1) of rr (so that $\mathsf{dom}(c) \in$ rr(M)). The *quasi-Herbrand interpretation* J *induced by* M *and* I is defined as follows:

(i) $|\mathrm{J}| = \{t \mid t \text{ is a ground term and } \mathrm{I} \models \mathsf{dom}(t)\}$.

(ii) For every n-ary $f \in \Sigma_f$, the interpretation function $f^{\mathrm{J}} : |\mathrm{J}| \times \cdots \times |\mathrm{J}| \mapsto |\mathrm{J}|$ is defined as follows, for all $d_1, \ldots, d_n \in |\mathrm{J}|$:

$$f^{\mathrm{J}}(d_1, \ldots, d_n) = \begin{cases} f(d_1, \ldots, d_n) & \text{if } f(d_1, \ldots, d_n) \in |\mathrm{J}|, \text{ and} \\ c & \text{otherwise} \quad . \end{cases}$$

(iii) For every n-ary $\mathrm{P} \in \Sigma_{\mathrm{P}}$, the interpretation function $\mathrm{P}^{\mathrm{J}} : |\mathrm{J}| \times \cdots \times |\mathrm{J}| \mapsto \{\mathit{false}, \mathit{true}\}$ is defined as follows, for all $d_1, \ldots, d_n \in |\mathrm{J}|$:

$$\mathrm{P}^{\mathrm{J}}(d_1, \ldots, d_n) = \mathrm{P}^{\mathrm{I}}(d_1, \ldots, d_n) \quad .$$

□

Some comments are due. Let J be a quasi-Herbrand interpretation induced by some M and I. We wish to verify that J is indeed a quasi-Herbrand interpretation in the sense of Definition 1. For that, first, observe that |J| is subterm-closed. This is achieved by the clauses added in Step (5) in rr and the fact that I $\models$ rr(M). Second, item (ii) in Definition 4 fixes the interpretation of function symbols such that $\mathrm{J}(t) = t$, for all $t \in |\mathrm{J}|$. This can be shown by induction on the term structure, in the induction step using the fact that |J| is subterm-closed.[5] Third, |J| contains at least one constant per Step (1) in rr and item (i) in Definition 4.

Item (iii) makes J the same as I with respect to predicate symbols on the domain |J|.

The clauses added in Step (2) in the rr transformation may seem unintuitive. We are now in a position to explain why they are needed, from a completeness perspective. More precisely, if I is a Herbrand model of rr(M) then we wish to conclude that the quasi-Herbrand interpretation J induced by M and I is a model of M, cf. Proposition 1 below.

By means of an example assume a clause set M that contains the clause $\mathsf{P}(\mathsf{f}(x), y, z, z) \rightarrow \mathsf{Q}(y)$. Let c be the constant picked in Step (1) and assume as given a quasi-Herbrand interpretation J with $|\mathrm{J}| = \{\mathsf{b}, \mathsf{c}, \ldots\}$ induced by M and Herbrand model I of rr(M). That is, |J| contains at least the constants b and c and other terms unspecified for now.

Suppose, for example, that J satisfies the body of the clause instance $\mathsf{P}(\mathsf{f}(\mathsf{c}), \mathsf{c}, \mathsf{b}, \mathsf{b}) \rightarrow \mathsf{Q}(\mathsf{c})$. For completeness, we need an argument that J also satisfies the head instance $\mathsf{Q}(\mathsf{c})$.

That J satisfies $\mathsf{P}(\mathsf{f}(\mathsf{c}), \mathsf{c}, \mathsf{b}, \mathsf{b})$ can only be the case because I satisfies $\mathsf{P}(\mathsf{f}(\mathsf{c}), \mathsf{c}, \mathsf{b}, \mathsf{b})$ *or* I satisfies $\mathsf{P}(\mathsf{c}, \mathsf{c}, \mathsf{b}, \mathsf{b})$. The latter is the case if $\mathsf{f}(\mathsf{c})$ is interpreted as the default element c in J, cf. Definition 4(iii). In fact, the clauses added in Step (2) make this case impossible (for all ground instances that matter).

More precisely, the rr transformation adds in Step (2) the clause $\mathsf{P}(x_1, y, z, z) \rightarrow \mathsf{dom}(\mathsf{f}(x))$ which gets range-restricted as $\mathrm{C} = \mathsf{dom}(x) \wedge \mathsf{P}(x_1, y, z, z) \rightarrow \mathsf{dom}(\mathsf{f}(x))$ in Step (3). From Definition 4(i) and the assumption $\mathsf{c} \in |\mathrm{J}|$ it follows that $\mathrm{I} \models \mathsf{dom}(c)$. Regarding the other body literal $\mathsf{P}(x_1, y, z, z)$ of C, notice that the variable x_1 is unconstrained. This means that we can instantiate x_1 with $\mathsf{f}(\mathsf{c})$ or c as required to make $\mathsf{P}(x_1, \mathsf{c}, \mathsf{b}, \mathsf{b})$ true in I. But then, because I is a model of rr(M) it follows that I must satisfy the instantiated head $\mathsf{dom}(\mathsf{f}(c))$. Again from Definition 4(i), $\mathsf{f}(\mathsf{c}) \in |\mathrm{J}|$ follows. That is, $\mathsf{f}(\mathsf{c})$ is *not* interpreted as the default element c, as indicated above.

[5] The interpretation $\mathrm{J}(t) = c$ for terms $t \notin |\mathrm{J}|$ is irrelevant in our main application, the completeness proof of the rr transformation.

But then, by Definition 4(ii) and 4(iii) the body literal $\mathsf{P}(\mathsf{f}(\mathsf{c}),\mathsf{c},\mathsf{b},\mathsf{b})$ is evaluated to the same truth value under I and J, which is *true*, as J satisfies it. As I satisfies $\mathsf{P}(\mathsf{f}(\mathsf{c}),\mathsf{c},\mathsf{b},\mathsf{b}) \rightarrow \mathsf{Q}(\mathsf{c})$, it satisfies $\mathsf{Q}(\mathsf{c})$. Once more from Definition 4(iii) conclude that J satisfies $\mathsf{Q}(\mathsf{c})$ as well, as desired.

The following lemma and proposition give the general argument.

Lemma 1 *Let* M, I *and* J *be defined as in Definition 4.*

(i) If $t_1,\ldots,t_n \in |\mathrm{J}|$ *then* $\mathrm{J}(\mathrm{P}(t_1,\ldots,t_n)) = \mathrm{I}(\mathrm{P}(t_1,\ldots,t_n))$.
(ii) For every $\mathcal{B} \rightarrow \mathcal{H} \in \mathrm{M}$ *and substitution* γ *with codomain* $|\mathrm{J}|$*: if* $\mathrm{J} \models \mathcal{B}\gamma$ *then* $\mathrm{I} \models \mathcal{B}\gamma$.

Proof (i) Assume $t_1,\ldots,t_n \in |\mathrm{J}|$ and let $\mathrm{P} \in \Sigma_\mathrm{P}$ arbitrary. For all $1 \leq i \leq n$ let $d_i = \mathrm{J}(t_i)$. By definition of quasi-Herbrand interpretation $d_i = t_i$. From Definition 4(iii) it follows

$$\mathrm{J}(\mathrm{P}(t_1,\ldots,t_n)) = \mathrm{P}^\mathrm{J}(\mathrm{J}(t_1),\ldots,\mathrm{J}(t_n)) = \mathrm{P}^\mathrm{J}(d_1,\ldots,d_n) = \\ \mathrm{P}^\mathrm{I}(d_1,\ldots,d_n) = \mathrm{P}^\mathrm{I}(t_1,\ldots,t_n) = \mathrm{P}^\mathrm{I}(\mathrm{I}(t_1),\ldots,\mathrm{I}(t_n)) = \mathrm{I}(\mathrm{P}(t_1,\ldots,t_n)) \ .$$

(ii) Take any $\mathcal{B} \rightarrow \mathcal{H} \in \mathrm{M}$ and choose an arbitrary substitution γ with codomain $|\mathrm{J}|$. Assume $\mathrm{J} \models \mathcal{B}\gamma$. Take any body atom $\mathrm{P}(t_1,\ldots,t_n) \in \mathcal{B}$. It follows that $\mathrm{J} \models \mathrm{P}(t_1,\ldots,t_n)\gamma$. To prove the lemma statement it suffices to show $\mathrm{I} \models \mathrm{P}(t_1,\ldots,t_n)\gamma$.

The rr transformation adds in Step (2) the clauses $\mathrm{P}(s_1,\ldots,s_n) \rightarrow \mathsf{dom}(t_j)$, for all $j \in \{1 \leq i \leq n \mid \ t_i \text{ is not a variable}\}$ where $\mathrm{P}(s_1,\ldots,s_n)$ is the term abstraction of $\mathrm{P}(t_1,\ldots,t_n)$. Let $\mathrm{C}_j = \mathsf{dom}(y_1) \wedge \cdots \wedge \mathsf{dom}(y_k) \wedge \mathrm{P}(s_1,\ldots,s_n) \rightarrow \mathsf{dom}(t_j)$ be the range-restricted versions of these clauses, obtained in Step (3), where $\{y_1,\ldots,y_k\} = \mathrm{var}(t_j) \setminus \mathrm{var}(\mathrm{P}(s_1,\ldots,s_n))$.

The first subgoal is to show $\mathrm{J}(s_i\gamma) = \mathrm{J}(t_i\gamma)$ for all $i \in \{1,\ldots,n\}$. In the first case t_i is a variable. By construction $s_i = t_i$ and hence, trivially, $\mathrm{J}(s_i\gamma) = \mathrm{J}(t_i\gamma)$. In the second case t_i is not a variable and s_i is a fresh variable x_i. After extending γ by the binding $x_i \mapsto \mathrm{J}(t_i\gamma)$ it follows $x_i\gamma = s_i\gamma = \mathrm{J}(t_i\gamma)$. Notice this extension trivially preserves γ as a substitution with codomain $|\mathrm{J}|$. In particular, hence, $x_i\gamma \in |\mathrm{J}|$. By quasi-Herbrand interpretation $\mathrm{J}(x_i\gamma) = x_i\gamma$. Altogether then $\mathrm{J}(s_i\gamma) = \mathrm{J}(t_i\gamma)$. That is, in either case $\mathrm{J}(s_i\gamma) = \mathrm{J}(t_i\gamma)$ which completes the proof of the subgoal.

From $\mathrm{J}(s_i\gamma) = \mathrm{J}(t_i\gamma)$ it follows $\mathrm{J}(\mathrm{P}(t_1,\ldots,t_n)\gamma) = \mathrm{J}(\mathrm{P}(s_1,\ldots,s_n)\gamma)$ by semantics of first-order logic. From $\mathrm{J} \models \mathrm{P}(t_1,\ldots,t_n)\gamma$ above it follows $\mathrm{J} \models \mathrm{P}(s_1,\ldots,s_n)\gamma$. By construction every s_i is a variable. Because γ is a substitution with codomain $|\mathrm{J}|$ it follows trivially that $s_i\gamma \in |\mathrm{J}|$. By item (i) of this lemma then $\mathrm{I} \models \mathrm{P}(s_1,\ldots,s_n)\gamma$.

The next subgoal is to show $t_i\gamma \in |\mathrm{J}|$, for all $i \in \{1,\ldots,n\}$. The first case when t_i is a variable is trivial as γ is a substitution with codomain $|\mathrm{J}|$.

In the second case t_i is not a variable and rr(M) contains the clause $\mathrm{C}_i = \mathsf{dom}(y_1) \wedge \cdots \wedge \mathsf{dom}(y_k) \wedge \mathrm{P}(s_1,\ldots,s_n) \rightarrow \mathsf{dom}(t_i)$ as mentioned above. Recall that γ is a substitution with codomain $|\mathrm{J}|$ and that $|\mathrm{J}|$ consists of the extension of the dom-predicate in I (Definition 4(i)). It follows that $y_m\gamma \in |\mathrm{J}|$ and hence $\mathrm{I} \models \mathsf{dom}(y_m)\gamma$, for all $m \in \{1,\ldots,k\}$.

Altogether, I satisfies $(\mathsf{dom}(y_1) \wedge \cdots \wedge \mathsf{dom}(y_k) \wedge \mathrm{P}(s_1,\ldots,s_n))\gamma$, the body of the clause instance $\mathrm{C}_i\gamma$. The lemma assumes that I is a Herbrand model of rr(M) as per Definition 4. It follows that I satisfies the instantiated head atom $\mathsf{dom}(t_i\gamma)$ of $\mathrm{C}_i\gamma$. Again with Definition 4(i) conclude $t_i\gamma \in |\mathrm{J}|$. This completes the proof of the second subgoal. That is, $t_i\gamma \in |\mathrm{J}|$ for all $i \in \{1,\ldots,n\}$. With that result and $\mathrm{J} \models \mathrm{P}(t_1,\ldots,t_n)\gamma$ from above we can apply item (i) of this lemma and conclude $\mathrm{I} \models \mathrm{P}(t_1,\ldots,t_n)\gamma$, which remained to be shown. □

Proposition 1 (Completeness of range-restriction) *Let* M *be any clause set. If rr*(M) *is satisfiable then* M *is satisfiable. More specifically, if* I *is a Herbrand model of rr*(M) *then the quasi-Herbrand interpretation* J *induced by* M *and* I *is a model of* M.

Proof Let I be a Herbrand model of rr(M) and J be the quasi-Herbrand interpretation induced by M and I. We show $J \models M$. Take any clause $\mathcal{B} \rightarrow \mathcal{H} \in M$ and choose an arbitrary substitution γ with codomain $|J|$. It suffices to show $J \models (\mathcal{B} \rightarrow \mathcal{H})\gamma$. The case of $J \not\models \mathcal{B}\gamma$ being trivial, we assume $J \models \mathcal{B}\gamma$ from now on and show $J \models \mathcal{H}\gamma$.

From Lemma 1(ii) it follows that $I \models \mathcal{B}\gamma$. Let $\mathsf{dom}(x_1) \wedge \cdots \wedge \mathsf{dom}(x_k) \wedge \mathcal{B} \rightarrow \mathcal{H}$ be the range-restricted version of $\mathcal{B} \rightarrow \mathcal{H}$ as introduced in Step (3) of rr(M). Because the codomain of γ is $|J|$, from the definition of $|J|$ it follows that $I \models \mathsf{dom}(x_i)\gamma$ for all $i \in \{1, \ldots, k\}$. Together, thus, $I \models (\mathcal{B} \wedge \mathsf{dom}(x_1) \wedge \cdots \wedge \mathsf{dom}(x_k))\gamma$. As I is a Herbrand model of all clauses in rr(M) it follows that $I \models \mathcal{H}\gamma$. That is, there is a head atom of the form $P(t_1, \ldots, t_n) \in \mathcal{H}$ such that $I \models P(t_1, \ldots, t_n)\gamma$. With the clauses added in Steps (4) and (5) we conclude $I \models \mathsf{dom}(t_j\gamma)$, for all $j \in \{1, \ldots, n\}$. From the definition of $|J|$ we immediately get $t_j\gamma \in |J|$. From Definition 4(iii) $J \models P(t_1, \ldots, t_n)\gamma$ follows and hence $J \models \mathcal{H}\gamma$, as desired. □

5 Equality

The following result lays the groundwork the adaptation of the range-restriction transformation for equality.

Corollary 1 (Completeness of range-restriction wrt. E-interpretations) *Let* M *be any clause set. If rr*(M) *is E-satisfiable then* M *is E-satisfiable. More specifically, assume there is an E-model* I *of rr*(M)*. Then there is a Herbrand model* J *of rr*(M) *such that* J *interprets* $\approx$ *as a congruence relation and* $J \models s \approx t$ *whenever* $I(s) = I(t)$*, for all* $s, t \in HU_\Sigma$.

Proof We prove the second claim only, as it entails the first. Let I_H be the Herbrand interpretation such that $I_H \models rr(M) \cup E_I \cup EAX(\Sigma)$, where $E_I = \{s \approx t \mid s, t \in HU_\Sigma \text{ and } I(s) = I(t)\}$. With I given as an E-model of rr(M) it is clear that indeed I_H exists. Let

$$EAX'(\Sigma) = (EAX(\Sigma) \setminus \{x \approx x\}) \cup \{t \approx t \mid t \in HU_\Sigma\}$$

be the equality axioms with the reflexivity axiom replaced by all its ground instances. Of course, $I_H \models rr(M) \cup E_I \cup EAX'(\Sigma)$ in the Herbrand semantics. One consequence of this move is that the rr transformation does not act on the clauses in $E_I \cup EAX'(\Sigma)$ (this is straightforward to check).[6] Therefore $rr(M) \cup E_I \cup EAX'(\Sigma)$ and $rr(M \cup E_I \cup EAX'(\Sigma))$ are the same set of clauses and it follows $I_H \models rr(M \cup E_I \cup EAX'(\Sigma))$.

Let J be the quasi-Herbrand interpretation induced by $M \cup E_I \cup EAX'(\Sigma)$ and I_H. Proposition 1 implies that $J \models M \cup E_I \cup EAX'(\Sigma)$. Regarding the domain $|J| = \{t \mid t \text{ is a ground term and } I_H \models \mathsf{dom}(t)\}$, recall that $EAX'(\Sigma)$ contains the clause $t \approx t$, for every $t \in HU_\Sigma$. By the clauses added in Step (4) of the rr transformation (when P is the equality predicate $\approx$) and the fact that $I_H \models rr(M)$, it follows that $I_H \models \mathsf{dom}(t)$, for every $t \in HU_\Sigma$. In other words, $|J| = HU_\Sigma$ and so J is (also) a Herbrand interpretation.

As $J \models M \cup E_I \cup EAX'(\Sigma)$ we have in particular that $J \models E_I \cup EAX'(\Sigma)$. In other words, J interprets $\approx$ as a congruence relation that includes the ground equations in E_I, as claimed in the statement of the result. □

We emphasize that we do not actually propose to use the equality axioms in conjunction with a theorem prover (though they can of course). They serve merely as a theoretical tool to prove completeness of the transformation.

[6] The rr transformation is defined on *finite* clause sets only, but it generalizes in a straightforward way to infinite clause sets.

In particular, the rr transformed clause set may compute a (finite) quasi-Herbrand interpretation even in the presence of equality. For example, for the clause set M consisting only of $\mathsf{f(b)} \approx \mathsf{f(c)}$, its range-restricted version rr(M) has an E-model I. We get a quasi-Herbrand interpretation J induced from rr(M) and I with domain $|\mathrm{J}| = \{\mathsf{b}, \mathsf{c}, \mathsf{f(b)}, \mathsf{f(c)}\}$ such that $f^{\mathrm{J}}(f(\mathsf{b})) = c$ and $f^{\mathrm{J}}(f(\mathsf{c})) = c$. To get an E-interpretation $\mathrm{J}_{\approx}$ one first builds the congruence closure of $\{\mathsf{f(f(b))} \approx c, \mathsf{f(f(c))} \approx c\} \cup \{\mathsf{f(b)} \approx \mathsf{f(c)}\}$. The first set stems from the definition of f^{J} and the second set from $\mathrm{I} \models \mathsf{f(b)} \approx \mathsf{f(c)}$. By means of the first set any ground term (e.g., $\mathsf{f(f(f(b)))}$) is congruent to a term in $|\mathrm{J}|$, but some terms in $|\mathrm{J}|$ may be congruent as well because of the second set.

In summary, this constructs a Herbrand interpretation where $\approx$ is interpreted as a congruence relation that mimics the interpretation function f^{I} and satisfies the equations true in I. Moreover, every ground term is congruent to a term in $|\mathrm{J}|$ (which is the extension of the dom-predicate in the model I). From that, we get an induced E-interpretation $\mathrm{J}_{\approx}$ on the domain $|\,\mathrm{HU}_{\Sigma}\,|/{\approx}$ whose elements are represented by elements from $|\mathrm{J}|$. In this example we obtain

$$|\mathrm{J}_{\approx}| = \big\{\{\mathsf{b}\},\ \{\mathsf{c}, \mathsf{f(f(b))}, \mathsf{f(f(c))}, \mathsf{f(f(f(f(b))))}, \mathsf{f(f(f(f(c))))}, \ldots\},\ \{\mathsf{f(b)}, \mathsf{f(c)}, \mathsf{f(f(f(b)))}, \mathsf{f(f(f(c)))}, \ldots\}\big\}$$

and $\mathrm{J}_{\approx}$ is an E-model of M.

5.1 Modification: The myequal Predicate

When using a BUMG theorem prover with built-in equality treatment, it is important to note that one particular type of clause in the rr transformation should not be treated as a normal clause. For the equality predicate the rr transformation produces in Step (2) the clauses

$$x \approx y \rightarrow \mathsf{dom}(x) \qquad\qquad x \approx y \rightarrow \mathsf{dom}(y)\ . \tag{\#}$$

Any superposition-based theorem prover simplifies these clauses to $\mathsf{dom}(x)$: applying EqRes to $x \approx y \rightarrow \mathsf{dom}(x)$ derives $\mathsf{dom}(x)$. Both clauses (#) are now subsumed by the new clause $\mathsf{dom}(x)$ and can be deleted.

As a consequence this can lead to all negative domain literals being resolved away and all clauses containing a positive domain literal to be subsumed. This means range-restriction is undone. This is what happens, for example, in SPASS.

Since the clauses added in Step (4) really only need to be added for positively occurring predicate symbols, an easy solution involves replacing any positive occurrence of the equality predicate by a predicate symbol myequal (say), which is fresh in the signature, and adding the clauses

$$\mathsf{myequal}(x, y) \rightarrow \mathsf{dom}(x) \qquad\qquad \mathsf{myequal}(x, y) \rightarrow \mathsf{dom}(y)$$

rather than (#). In addition, the clause set needs to be extended by this 'definition' of myequal.

$$\mathsf{myequal}(x, y) \rightarrow x \approx y$$

This solution has the intended effect of adding terms occurring in positive equality literals to the domain (by Step (4) of the rr transformation), and prevents other inferences or reductions on myequal.

In other words, instead of directly deriving a unit $s \approx t$ the above solution first derives $\mathsf{myequal}(s,t)$ as an intermediate conclusion, and then $s \approx t$ from that. With $\mathsf{myequal}(s,t)$ derived, $\mathsf{dom}(s)$ and $\mathsf{dom}(t)$ follow.

It is not difficult to prove that E-satisfiability is preserved in both directions:

Proposition 2 *Let* M *be a clause set and suppose* M′ *is obtained from* M *by replacing in every clause* $\mathcal{B} \to \mathcal{H} \in \mathrm{M}$ *every equation* $s \approx t \in \mathcal{H}$ *by* $\mathsf{myequal}(s,t)$. *Then* M *is E-satisfiable iff* $\mathrm{M}' \cup \{\mathsf{myequal}(x,y) \to x \approx y\}$ *is E-satisfiable.*

Proof ($\Leftarrow$) Assume $\mathrm{M}' \cup \{\mathsf{myequal}(x,y) \to x \approx y\}$ is satisfied by an E-interpretation I. Take arbitrarily any clause $\mathcal{B} \to \mathcal{H}' \in \mathrm{M}'$ and let $\mathcal{B} \to \mathcal{H} \in \mathrm{M}$ be the clause from which $\mathcal{B} \to \mathcal{H}'$ was obtained. The clause $\mathcal{B} \to \mathcal{H}$ is a logical consequence of $\mathcal{B} \to \mathcal{H}'$ and $\mathsf{myequal}(x,y) \to x \approx y$. From $\mathrm{I} \models \mathrm{M}' \cup \{\mathsf{myequal}(x,y) \to x \approx y\}$ it follows $\mathrm{I} \models \mathcal{B} \to \mathcal{H}$. As $\mathcal{B} \to \mathcal{H}' \in \mathrm{M}'$ was chosen arbitrarily it follows that $\mathrm{I} \models \mathrm{M}$.

($\Rightarrow$) Assume M is true in an E-interpretation I. Define the E-interpretation I′ to be the same as I, but extended for the new predicate symbol myequal by $\mathsf{myequal}^{\mathrm{I}'}(d_1,d_2) := d_1 \approx^{\mathrm{I}} d_2$, for every $d_1, d_2 \in |\mathrm{I}|$. Trivially, $\mathrm{I}' \models \mathrm{M}$ (as myequal is a fresh symbol) and $\mathrm{I}' \models \mathsf{myequal}(x,y) \to x \approx y$ (by definition of $\mathsf{myequal}^{\mathrm{I}'}$). Take arbitrarily any clause $\mathcal{B} \to \mathcal{H} \in \mathrm{M}$ and let $\mathcal{B} \to \mathcal{H}' \in \mathrm{M}'$ be the clause after the described replacement. From $\mathrm{I}' \models \mathcal{B} \to \mathcal{H}$ it follows $\mathrm{I}' \models \mathcal{B} \to \mathcal{H}'$ by the semantics of first-order logic. As $\mathcal{B} \to \mathcal{H} \in \mathrm{M}$ was chosen arbitrarily it follows that $\mathrm{I} \models \mathrm{M}' \cup \{\mathsf{myequal}(x,y) \to x \approx y\}$. □

Proposition 2 ensures that the myequal predicate is semantically harmless. We should mention though an operational drawback, if superposition-style inference rules are used. Letting $\succ$ denote a suitable term ordering, the ParaUnitPos inference rule then becomes the rule that from $v[s] \approx u$ (left premise) and $s \approx t$ (right premise) derive $v[t] \approx u$, *only if* $v[s] \succ u$ *and* $s \succ t$. Now, the first restriction is lost if the left premise becomes $\mathsf{myequal}(v[t],u)$ instead. This is a drawback. A better alternative to using the myequal predicate is an inference rule that derives $\mathsf{dom}(s)$ from $s \approx t$.

5.2 Modification: Range-Restriction

We need one more modification to use the rr transformation with BUMG methods with built-in equality inference rules. The EqRes inference rule (see Section 3) can break the invariant that all positive clauses derivable from a rr transformed clause set are ground. Consider, for example, the clause $x \approx \mathsf{f}(y) \to \mathsf{P}(x) \vee \mathsf{Q}(x)$. While range-restricted, applying EqRes to it yields $\mathsf{P}(\mathsf{f}(y)) \vee \mathsf{Q}(\mathsf{f}(y))$, which is no longer range-restricted. Indeed, this clause cannot be split and processed further as intended.

The easiest fix of this problem is to replace line 17 in the rr transformation by the line

let $\{x_1, \ldots, x_k\} = \mathrm{var}(\mathcal{H})$.

That is, all variables in the head must be subject to the dom-predicate. As a result, the example clause is rr transformed into $\mathsf{dom}(x) \wedge x \approx \mathsf{f}(y) \to \mathsf{P}(x) \vee \mathsf{Q}(x)$ and EqRes instead derives the range-restricted clause $\mathsf{dom}(\mathsf{f}(y)) \to \mathsf{P}(\mathsf{f}(y)) \vee \mathsf{Q}(\mathsf{f}(y))$.

Semantically this change does not make any difference, as Corollary 1 applies without modification. Moreover, the change does not cause any operational penalties either. Consider that a (ground) unit clause H is paired with a body atom B in a clause $\mathsf{dom}(x) \wedge \mathrm{B}[x] \wedge \mathcal{B} \to \mathcal{H}$ in a HRes or ParaNeg inference (for simplicity we consider only one variable x to make the argument). By the clauses added in Steps (4) and (5) of the rr transformation, all subterms

in H are in the domain as well, by derived dom-facts. This entails that x, if assigned to a subterm of H, is in the domain. The additional body atom $\mathsf{dom}(x)$, hence, is satisfied anyway.

Notice that the classical range-restriction transformation crr needs an analogous modification.

6 Shifting Transformation

The clauses introduced in Step (2) of the new rr transformation to range-restricted form use abstraction and insert (possibly a large number of) instantiations of terms occurring in the clause bodies into the domain. These are sometimes unnecessary and can lead to non-termination of BUMG procedures.

To explain by means of an example consider the singleton clause set comprised of

$$\mathsf{Q}(x) \land \mathsf{P}(\mathsf{f}(x)) \rightarrow \mathsf{R}(x) \quad . \tag{$*$}$$

Step (2) in the rr transformation applied to the body atom $\mathsf{Q}(x)$ does not lead to a new clause and is unproblematic. Step (2) applied to the other body atom $\mathsf{P}(\mathsf{f}(x))$ adds to the set variable N a new clause of the form $\mathrm{P}(x_1) \rightarrow \mathsf{dom}(\mathsf{f}(x))$, whose range-restricted version

$$\mathsf{dom}(x) \land \mathrm{P}(x_1) \rightarrow \mathsf{dom}(\mathsf{f}(x)) \tag{$\spadesuit$}$$

is added in Step (3) to the final result.

This is already an improvement over the classical range-restriction crr which includes the less constraining clause $\mathsf{dom}(x) \rightarrow \mathsf{dom}(\mathsf{f}(x))$. Together with the clause, say, $\mathsf{dom}(\mathsf{b})$ of Step (1) a derivation from that clause set contains $\mathsf{dom}(\mathsf{f}(\mathsf{b}))$, $\mathsf{dom}(\mathsf{f}(\mathsf{f}(\mathsf{b})))$, etc, and does not terminate; the derivation from the rr transformed clause set does terminate quickly.

However, the advantage of the rr transformation is lost as soon as a clause $\mathrm{P}(t)$, for *any* ground term t, is present or has been derived. For instance, if we extend the above example $(*)$ with the clause $\mathsf{P}(\mathsf{b})$ we obtain the clause set

$$\mathsf{P}(\mathsf{b}) \qquad\qquad \mathsf{Q}(x) \land \mathsf{P}(\mathsf{f}(x)) \rightarrow \mathsf{R}(x) \quad .$$

BUMG methods now also derive $\mathsf{dom}(\mathsf{f}(\mathsf{b}))$, $\mathsf{dom}(\mathsf{f}(\mathsf{f}(\mathsf{b})))$, etc, from the rr transformation of this clause set by means of the clause $(\spadesuit)$.

The *shifting transformation* introduced in this section can address this problem (and so can the blocking transformation in Section 7 below). It consists of two sub-transformations, *basic shifting* and *partial flattening*.

6.1 Basic Shifting

If A is an atom $\mathrm{P}(t_1, \ldots, t_n)$ then let not_A denote the atom $\mathsf{not\_P}(t_1, \ldots, t_n)$, where 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 not_P as $\not\approx$ and use infix notation.

Now, the *basic shifting transformation* of a clause set M is the clause set bs(M) obtained from M by the following algorithm.

1 **Algorithm** bs(M)
2 // Input: a clause set M
3 // Output: a clause set with basic shifting applied to M
4 // Step (1): initialization
5 **var** $res := \emptyset$ // initialized result clause set
6 **var** $\Sigma_P^+ := \emptyset$ // new predicate symbols extending Σ_P
7 // Step (2): shifting deep atoms
8 **foreach** $(\mathcal{B} \rightarrow \mathcal{H}) \in M$ **do**
9 **let** $\{B_1, \ldots, B_m\} = \{B \in \mathcal{B} \mid$ some top-level term of B is a compound term$\}$
10 $\Sigma_P^+ := \Sigma_P^+ \cup \{P \mid P$ is the predicate symbol of some atom in $\{B_1, \ldots, B_m\}\}$
11 $res := res \cup \{(\mathcal{B} \setminus \{B_1, \ldots, B_m\}) \rightarrow \mathcal{H} \vee \text{not\_}B_1 \vee \cdots \vee \text{not\_}B_m\}$
12 // Step (3): shifted atoms consistency
13 **foreach** $P \in \Sigma_P^+$ where n is the arity of P **do**
14 $res := res \cup \{P(x_1, \ldots, x_n) \wedge \text{not\_}P(x_1, \ldots, x_n) \rightarrow \bot\}$
15 **return** res

In line 9 of Step (2) every atom B_i identified in a run of bs, is called a *shifted atom*. Notice that we do *not* add clauses complementary to the 'shifted atoms consistency' clauses, that is, $P(x_1, \ldots, x_n) \vee \text{not\_}P(x_1, \ldots, x_n)$. They could be included but are superfluous.

As a side effect, the clause set bs(M) has implicitly assumed the signature $\Sigma_P \cup \Sigma_P^+$ as needed for a subsequent rr transformation.

Let us consider as an example these clauses.

$$P(b) \qquad Q(x) \qquad Q(x) \wedge P(f(x)) \rightarrow R(x) \qquad (**)$$

Basic shifting moves the negative occurrences of functional terms into heads for the last clause and we get

$$P(b) \qquad Q(x) \qquad Q(x) \rightarrow R(x) \vee \text{not\_}P(f(x))$$
$$\text{not\_}P(x) \wedge P(x) \rightarrow \bot \ .$$

Applying the rr transformation we obtain the following set.

$$\begin{array}{rll} \text{dom}(b) & P(b) & R(x) \rightarrow \text{dom}(x) \\ P(x) \rightarrow \text{dom}(x) & \text{dom}(x) \rightarrow Q(x) & \text{dom}(f(x)) \rightarrow \text{dom}(x) \\ Q(x) \rightarrow \text{dom}(x) & Q(x) \rightarrow R(x) \vee \text{not\_}P(f(x)) & \\ \text{not\_}P(x) \rightarrow \text{dom}(x) & \text{not\_}P(x) \wedge P(x) \rightarrow \bot & \end{array}$$

The clauses in column one are added in Steps (1) and (2), the clauses in columns one and two are present after Step (3), and the clauses in the third column are added by Step (4) and (5).

Even on this clause set termination of BUMG can be achieved. For instance, in a hyperresolution-like mode of operation and with splitting enabled, the SPASS prover [94, 93] splits the derived clause $R(b) \vee \text{not\_}P(f(b))$, considers the case with the literal R(b) first *and terminates with a model*. This is because a finite completion (model) is found without considering the case of the bigger literal not_P(f(b)), which would have added the term f(b) to the domain. The same behaviour can be achieved, for example, with the KRHyper BUMG prover, a hypertableaux theorem prover [95].

6.2 Partial Flattening

As can be seen in the previous example, the basic shifting transformation avoids the generation of new domain elements by removing literals from a clause body. Of course, a smaller clause body affects the search space, as then the clause can be used as a premise more often. To (partially) avoid this effect, we propose an additional transformation, called *partial flattening*, to be performed prior to the basic shifting transformation. It consists of replacing all compound terms in non-equational body atoms and replacing them by equations with these variables.

We need some preliminaries before defining partial flattening formally. Let unif be a fresh binary predicate symbol and assume Σ_P has already been extended by unif. (Intuitively, unif stands for 'unifiability' and is defined by the clause $\mathsf{unif}(x, x)$.)

Definition 5 (Target term) Let $\mathrm{P}(t_1, \ldots, t_n)$ be an atom. For all $i \in \{1, \ldots, n\}$, t_i is a *target term in* $\mathrm{P}(t_1, \ldots, t_n)$ iff $\mathrm{P} \notin \{\approx, \mathsf{unif}\}$ and t_i is a compound term. For any variable x, define $\mathrm{P}(t_1, \ldots, t_n)[t_i \mapsto x] = \mathrm{P}(t_1, \ldots, t_{i-1}, x, t_{i+1}, \ldots, t_n)$. □

For a clause set M, the *partial flattening transformation* is the clause set pf(M) obtained from M by applying the following algorithm.

```
1  Algorithm pf(M)
2    // Input: a clause set M
3    // Output: a clause set with partial flattening applied to M
4    // Step (1): initialization
5    var res := {unif(x, x)} // initialized result clause set
6    // Step (2): partial flattening
7    foreach (B_1 ∧ ··· ∧ B_k → H) ∈ M do
8      var B := {B_1, ..., B_k}
9      while there is a B ∈ B and a target term t in B do
10       let x be a fresh variable
11       B := (B \ {B}) ∪ {B[t ↦ x], unif(x, t)}
12     res := res ∪ {B → H}
13   return res
```

Applied to our running example consisting of the clauses $(**)$ we get as a result of the pf transformation the clauses

$$\mathsf{unif}(x, x) \qquad \mathsf{P}(\mathsf{b}) \qquad \mathsf{Q}(x) \qquad \mathsf{Q}(x) \wedge \mathsf{P}(u) \wedge \mathsf{unif}(u, \mathsf{f}(x)) \rightarrow \mathsf{R}(x) \ .$$

Altogether, 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).[7]

$$\mathsf{Q}(x) \wedge \mathsf{P}(u) \rightarrow \mathsf{R}(x) \vee \mathsf{not\_unif}(u, \mathsf{f}(x)) \qquad \mathsf{not\_unif}(x, y) \rightarrow \mathsf{dom}(x) \qquad \mathsf{dom}(x) \rightarrow \mathsf{unif}(x, x)$$
$$\mathsf{not\_unif}(x, y) \wedge \mathsf{unif}(x, y) \rightarrow \bot \qquad \mathsf{not\_unif}(x, y) \rightarrow \mathsf{dom}(y)$$
$$\mathsf{unif}(x, y) \rightarrow \mathsf{dom}(x)$$
$$\mathsf{unif}(x, y) \rightarrow \mathsf{dom}(y)$$

[7] The clause $\mathsf{dom}(x) \rightarrow \mathsf{unif}(x, x)$ is the result of range-restriction on the clause $\mathsf{unif}(x, x)$.

Observe that the clause $\mathsf{Q}(x) \land \mathsf{P}(u) \to \mathsf{R}(x) \lor \mathsf{not\_unif}(u, \mathsf{f}(x))$ is more restricted than the above clause $\mathsf{Q}(x) \to \mathsf{R}(x) \lor \mathsf{not\_P}(\mathsf{f}(x))$ 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 does 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.

An immediate improvement of the combined pf, bs and rr transformations comes to mind. The pf transformation introduces in its Step (2) unif-body atoms that are, by design, all shifted into corresponding not_unif-head atoms. The only unif-atoms that remain in the end are those four clauses in the above example. We can hence eliminate the unif predicate from the set by applying resolution inferences exhaustively among them. After deleting tautologies we obtain a simplified, equi-satisfiable clause set.[8] In the example it looks as follows:

$$\mathsf{Q}(x) \land \mathsf{P}(u) \to \mathsf{R}(x) \lor \mathsf{not\_unif}(\mathsf{f}(x), u) \qquad \mathsf{not\_unif}(x, y) \to \mathsf{dom}(x)$$
$$\mathsf{not\_unif}(x, x) \land \mathsf{dom}(x) \to \bot \qquad \mathsf{not\_unif}(x, y) \to \mathsf{dom}(y) \ .$$

As for rr, the pf and bs transformations can be used with respect to equality as well. However, extracting top-level terms from equations has no effect at all, and this is why they are excluded from pf. Consider the unit clause $\mathsf{f}(\mathsf{a}) \approx \mathsf{b} \to \bot$, and its (otherwise) partial flattening $x \approx \mathsf{b} \land \mathsf{unif}(x, \mathsf{f}(\mathsf{a})) \to \bot$. Applying basic shifting yields $x \approx \mathsf{b} \to \mathsf{not\_unif}(x, \mathsf{f}(\mathsf{a}))$, and, hyperresolution with a derivable instance $\mathsf{b} \approx \mathsf{b}$ (or the EqRes rule)) gives $\mathsf{not\_unif}(\mathsf{b}, \mathsf{f}(\mathsf{a}))$. In terms of derivable dom-clauses this is equivalent to $\mathsf{b} \not\approx \mathsf{f}(\mathsf{a})$ as obtained by the bs transformation applied to $f(a) \approx b \to \bot$ without doing partial flattening first. This explains why top-level terms of equational literals are excluded from the definition. (One could consider using standard flattening, that is, recursively extracting terms, but this does not lead to any improvements over the defined transformations.)

Finally, we combine basic shifting and partial flattening to give the *shifting transformation*:

Definition 6 (Shifting) The *shifting transformation* is defined as sh := pf ∘ bs, that is, sh(M) = bs(pf(M)), for any clause set M.

Proposition 3 (Completeness of shifting) *Let* M *be any clause set. Then* M *is satisfiable iff sh*(M) *is satisfiable.*

Proof Not difficult, since bs (basic shifting) can be seen to be a structural transformation and pf (partial flattening) is a form of term abstraction. The formal argumentation is similar to the proof of Proposition 2. In particular, the interpretation function for the predicates not_P is the negation of P. □

Corollary 2 (Completeness of shifting with respect to E-interpretations) *Let* M *be any clause set. Then* M *is E-satisfiable iff sh*(M) *is E-satisfiable.*

Proof Using the same line of argument as in the proof of Corollary 1, proving preservation of E-satisfiability can be reduced to proving preservation of satisfiability by means of the equality axioms (observe that the shifting transformation does not modify the equality axioms). □

[8] This essentially uses predicate elimination [38].

7 Blocking

The final transformation introduced in this paper is called *blocking* and provides a mechanism for detecting recurrence in the derived models. The blocking transformation is 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 (for example, superposition or rewriting), splitting, the possibility to search for split-off positive equations first (explained below), and standard redundancy elimination techniques*.

The basic idea behind blocking is to add clauses that cause a case analysis of the form $s \approx t$ versus $s \not\approx t$, for (ground) terms s and t. Although such a case analysis apparently leads to a bigger search space, it provides a powerful technique to detect finite models with a BUMG prover and in practice actually leads to satisfiability being detected more quickly. This is because in the case that $s \approx t$ is assumed, this new equation leads to rewriting and representing otherwise infinitely many terms as one single term. To make this possible, the prover must support the above features, including notably splitting. Among resolution theorem provers splitting has become standard. Splitting was first available in the saturation-based prover SPASS [94,93], but is also part of VAMPIRE [74,73] and E [83]. Splitting is an integral part of the hypertableau prover E-KRHyper [13,72].

In the following we introduce four different, but closely related, blocking transformations, called *subterm domain blocking*, *subterm predicate blocking*, *unrestricted domain blocking* and *unrestricted predicate blocking*. Subterm domain blocking was introduced in the short version of this paper under the name blocking [16]. Subterm predicate blocking is inspired by and related to the blocking technique described in [50]. Unrestricted domain blocking is the first-order version of the unrestricted blocking rule introduced in [79] and used for developing terminating tableau calculi for logics with the effective finite model property [80, 81]. A logic L (or fragment $\mathcal{F}$ of first-order logic) has the *effective finite model property* if there a computable model bounding function $\mu : \mathbb{N} \to \mathbb{N}$ such that the following holds. For every formula ϕ, if ϕ is satisfiable in an L-model ($\mathcal{F}$-model) then ϕ has a finite L-model ($\mathcal{F}$-model) with domain size not exceeding $\mu(n)$, where n is the length of ϕ.

7.1 Subterm Domain Blocking

Let sub be a fresh binary predicate symbol not in Σ_P. For a clause set M, the *subterm domain blocking transformation* is the clause set sdb(M) obtained from M by applying the following algorithm.

Algorithm sdb(M)
  // Input: a clause set M
  // Output: a clause set with subterm domain blocking applied to M
  // Step (1): initialization
  var $res := \mathrm{M} \cup \{\mathsf{dom}(x) \to \mathsf{sub}(x,x)\}$ // initialized result clause set
  // Step (2): axioms describing the subterm relationship
  foreach $f \in \Sigma_f$ **do**
    for $i \in \{1,\ldots,n\}$, where n is the arity of f **do**
      $res := res \cup$
        $\{\mathsf{sub}(x,x_i) \wedge \mathsf{dom}(x) \wedge \mathsf{dom}(f(x_1,\ldots,x_n)) \to \mathsf{sub}(x,f(x_1,\ldots,x_n))\}$

11 // Step (3): subterm equality case analysis and axiom for $\not\approx$
12 $res := res \cup \{\mathsf{sub}(x,y) \rightarrow x \approx y \vee x \not\approx y,\ x \approx y \wedge x \not\approx y \rightarrow \bot\}$
13 **return** *res*

Regarding Step (3), recall from Section 6.1 that $\not\approx$ is suggestive notation for the introduced predicate symbol not_$\approx$. The subterm domain blocking transformation allows to consider whether two domain elements that are in a subterm relationship should be identified and merged, or not.

This 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 a range-restricting transformation.

Reading $\mathsf{sub}(s,t)$ as 's is a subterm of t', Step (2) 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:

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

for every n-ary function symbol $f \in \Sigma_f$ and all $i \in \{1,\ldots,n\}$. Yet, this specification, even if range-restricted, is not suitable for our purposes. The problem is that the second clause introduces compound terms.

For example, for a given constant a and a unary function symbol f, when just dom(a) alone has been derived, a BUMG procedure derives an infinite sequence of clauses:

$$\mathsf{sub(a,a)},\ \mathsf{sub(a,f(a))},\ \mathsf{sub(a,f(f(a)))},\ \ldots.$$

This does not happen with the specification in Step (2). It ensures that conclusions of BUMG inferences involving sub are about terms *currently* in the domain, without unforcedly extending it.

To justify the clauses added in Step (3) we continue this example and suppose an interpretation that contains dom(a) and dom(f(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

$$\mathsf{f(a)} \approx \mathsf{a} \vee \mathsf{f(a)} \not\approx \mathsf{a}$$

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 such as the one derived above, the $\approx$-literal should be split off and the clause set obtained in this case should be searched *first*. The reason is that inference with a positive (ground) equational unit is simplifying. For example, should dom(f(f(a))) be derivable now (in the current branch), then any prover featuring rewriting and a complete simplification ordering is able to prove it redundant from $\mathsf{f(a)} \approx \mathsf{a}$ and dom(a). Consequently, the domain is *not* to be extended *explicitly*. The information that dom(f(f(a))) is in the domain is nevertheless implicit via the theory of equality.

7.2 Subterm Predicate Blocking

Subterm domain blocking defined in the previous section applies blocking to *domain terms* where one is a proper subterm of the other. The idea of the *subterm (unary) predicate*

blocking transformation is similar, but it merges only the (sub)terms in the extension of *unary* predicate symbols different from dom in the current interpretation.

Subterm predicate blocking is defined as follows:

Algorithm spb(M)
  // Input: a clause set M
  // Output: a clause set with subterm predicate blocking applied to M
  // Step (1): initialization
  var $res := \mathrm{M} \cup \{\mathsf{dom}(x) \rightarrow \mathsf{sub}(x,x)\}$ // initialized result clause set
  // Step (2): axioms describing the subterm relationship; same as Step (2) in sdb
  foreach $f \in \Sigma_f$ **do**
    for $i \in \{1,\ldots,n\}$, where n is the arity of f **do**
      $res := res \cup$
        $\{\mathsf{sub}(x,x_i) \wedge \mathsf{dom}(x) \wedge \mathsf{dom}(f(x_1,\ldots,x_n)) \rightarrow \mathsf{sub}(x,f(x_1,\ldots,x_n))\}$
  // Step (3): subterm equality case analysis
  foreach unary $\mathrm{P} \in \Sigma_\mathrm{P}$ **do**
      $res := res \cup \{\mathsf{sub}(x,y) \wedge \mathrm{P}(x) \wedge \mathrm{P}(y) \rightarrow x \approx y \vee x \not\approx y\}$
  // Step (4): axiom for $\not\approx$
  $res := res \cup \{x \approx y \wedge x \not\approx y \rightarrow \bot\}$
  return *res*

Observe that the only difference between this transformation and the subterm domain blocking transformation lies in Step (3). The clauses

$$\mathsf{sub}(x,y) \wedge \mathrm{P}(x) \wedge \mathrm{P}(y) \rightarrow x \approx y \vee x \not\approx y$$

added here are obviously more restricted than their counterpart $\mathsf{sub}(x,y) \rightarrow x \approx y \vee x \not\approx y$ in the definition of the subterm domain blocking transformation sdb.

That subterm predicate blocking is strictly more restricted can be seen from the following example, which also helps to explain the rationale behind this transformation.

$$\mathsf{P}(\mathsf{a}) \qquad\qquad \mathsf{P}(x) \rightarrow \mathsf{Q}(\mathsf{f}(x))$$

Any BUMG prover can be expected to terminate on the range-restricted version of the transformed clause set and returns the model

$$\{\mathsf{dom}(\mathsf{a}),\ \mathsf{dom}(\mathsf{f}(\mathsf{a})),\ \mathsf{P}(\mathsf{a}),\ \mathsf{Q}(\mathsf{f}(\mathsf{a})),\ \mathsf{sub}(\mathsf{a},\mathsf{a}),\ \mathsf{sub}(\mathsf{a},\mathsf{f}(\mathsf{a}))\}.$$

Notice that the subterm predicate blocking transformation includes the clauses

$$\mathsf{sub}(x,y) \wedge \mathsf{P}(x) \wedge \mathsf{P}(y) \rightarrow x \approx y \vee x \not\approx y$$
$$\mathsf{sub}(x,y) \wedge \mathsf{Q}(x) \wedge \mathsf{Q}(y) \rightarrow x \approx y \vee x \not\approx y$$

but all BUMG inferences between these clauses and the above model lead to redundant ground conclusions $t \approx t \vee t \not\approx t$ (in this example where $t \in \{\mathsf{a},\mathsf{f}(\mathsf{a})\}$). The motivation behind these clauses is to equate the arguments of two P-literals (say) only when there are two literals $\mathsf{P}(s)$ and $\mathsf{P}(t)$ where s is a subterm of t. Conversely, if no such recurrence comes up, as in the above example, there is no reason for blocking. By contrast, the subterm domain blocking transformation sdb with its clause $\mathsf{sub}(x,y) \rightarrow x \approx y \vee x \not\approx y$ would be applicable even to distinct terms, leading to the (unnecessary) split into the cases $\mathsf{a} \approx \mathsf{f}(\mathsf{a})$ and $\mathsf{a} \not\approx \mathsf{f}(\mathsf{a})$.

From a more general perspective, the spb transformation is motivated by the application to description logic knowledge bases [50,2]. Often, such knowledge bases do not contain cyclic definitions, or only few definitions are cyclic. The subterm predicate transformation aims to apply blocking only to concepts (unary predicates) with cyclic definitions. In Section 7.5, we discuss a description logic example to highlight the differences between the various blocking transformations.

7.3 Unrestricted Domain Blocking

The two previous 'subterm' variants of the blocking transformation allow to speculatively identify terms *and their subterms*. The 'unrestricted' variants introduced next differ from both by allowing speculative identifications of *any* two terms.

For the 'domain' variant, called *unrestricted domain blocking transformation*, the definition is as follows.

1 **Algorithm** udb(M)
2 // Input: a clause set M
3 // Output: a clause set with unrestricted domain blocking applied to M
4 // Step (1): initialization
5 **var** *res* := M // initialized result clause set
6 // Step (2): equality case analysis and axiom for $\not\approx$
7 *res* := *res* $\cup$ $\{\mathsf{dom}(x) \wedge \mathsf{dom}(y) \rightarrow x \approx y \vee x \not\approx y,\ x \approx y \wedge x \not\approx y \rightarrow \bot\}$
8 **return** *res*

There is a clear trade-off between this transformation and the subterm domain blocking transformation sdb. On the one hand, the unrestricted domain blocking transformation induces a larger search space, as the bodies of the clauses $\mathsf{dom}(x) \wedge \mathsf{dom}(y) \rightarrow x \approx y \vee x \not\approx y$ are less constrained than their counterparts in the subterm domain blocking transformation. With this clause BUMG derives the clauses $s \approx t \vee s \not\approx t$, for every s,t in the derivable domain. In contrast, the clause $\mathsf{sub}(x,y) \rightarrow x \approx y \vee x \not\approx y$ from the sdb transformation derives $s \approx t \vee s \not\approx t$ only if additionally s is a subterm of t. On the other hand, the unrestricted domain blocking transformation can enable finding models with smaller domains because of the additionally derivable equations. This means fewer congruence classes on the Herbrand terms are induced by the equality relation $\approx$ and, as our evaluation shows, actually reduces effort and memory consumption as models are found quicker (if there are any), even for the crr transformation.

We claim that the termination proof in [82] for semantic ground tableau with unrestricted domain blocking for the description logic $\mathcal{ALBO}^{\mathsf{id}}$ with the expressive power similar to the two-variable fragment of first-order logic can be adapted to show BUMG with unrestricted domain blocking can return finite models, when they exist, i.e., decide finite satisfiability. For this it is essential that blocking is applied eagerly and a breadth-first or iterative deepening search strategy is used to avoid getting 'lost' in a branch of the derivation where only infinite models exist. Carrying over the results in [80] implies unrestricted domain blocking can be used in BUMG methods to return domain minimal models for logics with the effective finite model property.

7.4 Unrestricted Predicate Blocking

The definition of the last variant of blocking, the *unrestricted (unary) predicate blocking transformation*, is as follows.

1 **Algorithm** upb(M)
2 // Input: a clause set M
3 // Output: a clause set with unrestricted predicate blocking applied to M
4 // Step (1): initialization
5 **var** *res* := M // initialized result clause set
6 // Step (2): equality case analysis
7 **foreach** unary $P \in \Sigma_P$ **do**
8 $res := res \cup \{P(x) \wedge P(y) \rightarrow x \approx y \vee x \not\approx y\}$
9 // Step (3): axiom for $\not\approx$
10 $res := res \cup \{x \approx y \wedge x \not\approx y \rightarrow \bot\}$
11 **return** *res*

This transformation allows to equate any two (distinct) terms in a P-relation, if there are any. The motivation is a combination of the above, to attempt to block recurrence on P-literals if they arise, and to compute models with small domains.

7.5 Comparison on an Example

It is instructive to compare the effects on the returned models of the four blocking transformations on an example from description logics. To this end, consider the description logic knowledge base (left) and its translation into clause logic (right) in Table 1.

TBox	ABox
$P_1 \sqsubseteq \exists R.P_2$	$P_1(a)$
$P_2 \sqsubseteq \exists R.P_1$	$P_1(b)$
$P_1 \sqsubseteq \exists S.Q$	

$P_1(x) \rightarrow P_2(f(x))$	$P_1(x) \rightarrow Q(h(x))$
$P_1(x) \rightarrow R(x, f(x))$	$P_1(x) \rightarrow S(x, h(x))$
$P_2(x) \rightarrow P_1(g(x))$	$P_1(a)$
$P_2(x) \rightarrow R(x, g(x))$	$P_1(b)$

Table 1 Sample description logic knowledge base and clausal form.

Notice that the cycle in the dependency graph in the TBox (for P_1 and P_2) means that some form of blocking is needed for decidability in tableau-based description logic systems. Likewise, blocking is needed to force BUMG methods to terminate on the translated clause form. Any of the four blocking transformations defined above suffice. Table 2 summarizes the behaviour of these transformations, in terms of interesting relations in the computed model.

When comparing in detail the blocking techniques developed for description logics it becomes clear that

$$\text{the transformations } \mathrm{rr} \circ \tau \text{ and } \mathrm{sh} \circ \mathrm{rr} \circ \tau, \quad \text{for } \tau \in \{\mathrm{sdb}, \mathrm{spb}, \mathrm{udb}, \mathrm{upb}\},$$

Blocking	dom	$\approx$	P_1	P_2	Q
sdb	a, b	f(a) $\approx$ a, f(b) $\approx$ b, g(a) $\approx$ a, g(b) $\approx$ b, h(a) $\approx$ a, h(b) $\approx$ b	a, b	a, b	a, b
spb	a, b	f(a) $\approx$ a, f(b) $\approx$ b, g(a) $\approx$ a, g(b) $\approx$ b, h(a) $\approx$ a, h(b) $\approx$ b	a, b	a, b	a, b
udb	b	a $\approx$ b, f(b) $\approx$ b, g(b) $\approx$ b, h(b) $\approx$ b	b	b	b
upb	b, f(b), h(b)	a $\approx$ b, g(h(b)) $\approx$ b	b	h(b)	f(b)

Table 2 Partial truth assignments in models computed for the sample knowledge base. The transformation applied is rr$\circ\tau$ where τ is the blocking transformation in the left column. Notice that the shifting transformation sh does not have any effect in this example.

when applied to a knowledge base with finite models, in conjunction with a suitable BUMG method (see above, at the end of Section 7.1), can be refined to simulate various forms of standard blocking techniques used in description logic systems, including subset ancestor blocking and equality ancestor blocking (cf. [50, 82, 56]). Because standard loop-checking mechanisms used in description logic systems do not require backtracking, appropriate search strategies and restrictions for performing inferences and applying blocking need to be used.

An advantage of our approach to blocking as opposed to loop-checking used in mainstream description logic systems [2] (i.e., blocking without equality reasoning) is that it applies to any first-order clause set, not only to clauses from the translation of description logic problems. This makes the approach very general and 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 [1]. The specification of many natural properties of binary relations and complex statements involving binary relations are outside the scope of most current description logic systems. An example is the statement: individuals who live and work at the same location are home workers. This can be expressed as a Horn rule (clause) $\mathsf{work}(x, y) \wedge \mathsf{loc}(y) \wedge \mathsf{live}(x, z) \wedge \mathsf{loc}(z) \wedge y \approx z \rightarrow \mathsf{homeWorker}(x)$, but, with some exceptions [54, 94], is not expressible in current description logic systems.

8 Soundness and Completeness of the Transformations

Each of the blocking transformations is complete:

Proposition 4 (Completeness of blocking with respect to E-interpretations) *Let* M *be any clause set. For all* $\tau \in \{sdb, spb, udb, upb\}$*, if* τ(M) *is E-satisfiable then* M *is E-satisfiable.*

Proof Not difficult, as M $\subseteq$ τ(M) by definition. □

The converse, that is, soundness of the transformation, is easy to prove. One basically needs to observe that the clauses added in respectively steps of the blocking transformations, realize a case distinction over whether two terms are equal or not. Trivially, one of the two cases always holds.

Putting all the transformations and the corresponding results together we can state the main theoretical result of the paper.

Theorem 1 (Completeness of the combined transformations wrt. E-interpretations) *Let* M *be a clause set and suppose tr is any of the transformations in*

$$\{rr, sh \circ rr\} \cup \{rr \circ \tau, sh \circ rr \circ \tau \mid \tau \in \{sdb,\ spb,\ udb,\ upb\}\} \quad or$$
$$\{crr, sh \circ crr\} \cup \{crr \circ \tau, sh \circ crr \circ \tau \mid \tau \in \{sdb,\ spb,\ udb,\ upb\}\} \ .$$

Then:

(i) *tr*(M) *is range-restricted.*
(ii) *tr*(M) *can be computed in quadratic time.*
(iii) *If tr*(M) *is E-satisfiable then* M *is E-satisfiable.*

The reverse directions of (iii), that is, soundness of the respective transformations, hold as well. The proofs are either easy or completely standard. Also, item (iii) can be made more specific in the same way as Corollary 1.

By carefully modifying the definition of rr (by essentially structure sharing to optimize the size of the transformation and avoid unneeded duplication) it is possible to compute the reductions in linear time.

Proposition 5 *Let* M *and tr be as in the previous result. Then:*

(i) *The size of tr*(M) *is bounded by a linear function in the size of* M.
(ii) *tr*(M) *can be computed in linear time.*

9 Decidability of BS classes

Formulae in the Bernays-Schönfinkel class are conjunctions of function-free formulae of the form $\exists^*\forall^*\psi$, where ψ is free of quantifiers. A clause is a *BS clause* iff all functional terms occurring in it are constants.

From [60, §5.3] it is known that hyperresolution does not decide the Bernays-Schönfinkel class without equality directly, and neither do any of the standard refinements of resolution. However it has been shown that hyperresolution and any refinements can decide the class of range-restricted BS clauses without equality [77]. Here we assume that the language includes equality.

Theorem 2 *The class of range-restricted BS clauses (with equality), is decidable by unordered hyperresolution, or unordered resolution with selection of at least one negative literal (and paramodulation). It is also decidable by all refinements using a complete simplification ordering (and superposition).*

This means all refinements of hyperresolution (and some form of equality reasoning) combined with any translation into range-restricted clauses is a decision procedure for the BS class. Therefore:

Corollary 3 *Let* M *be any set of BS clauses, and suppose tr is a transformation in*

$$\{rr, sh \circ rr\} \cup \{rr \circ \tau, sh \circ rr \circ \tau \mid \tau \in \{sdb,\ spb,\ udb,\ upb\}\} \quad \text{and}$$
$$\{crr, sh \circ crr\} \cup \{crr \circ \tau, sh \circ rr \circ \tau \mid \tau \in \{sdb,\ spb,\ udb,\ upb\}\}.$$

Then:

(i) Hyperresolution (or resolution with selection of at least one negative literal) with paramodulation and all refinements using a superposition-based equality reasoning decide tr(M).
(ii) All BUMG methods decide M.

Since there are linear transformations of first-order formulae into clausal form, and since all the tr transformations are effective reductions of first-order clauses into range-restricted clauses, we obtain the following result.

Theorem 3

(i) There is a quadratic (linear), satisfiability equivalence preserving transformation of any formula in the Bernays-Schönfinkel class, and any set of BS clauses, into a set of range-restricted BS clauses.
(ii) All procedures based on hyperresolution or BUMG using a suitable form of equality handling decide the class of BS formulae and the class of BS clauses.

In [77] a similar but different transformation is used to prove this result for hyperresolution and BS without equality. In fact, what is crucial for deciding the BS class is a grounding method. This can be achieved by any form of range-restriction and hyperresolution-like inferences. Theorem 3(ii) can therefore be strengthened to include also any instantiation-based method [39,40,22], in particular also methods using on-the-fly instantiation such as semantic Smullyan-type tableau systems, where instantiation of universally quantified variables is realized by the γ-rule.

10 Experimental Evaluation

We have implemented the transformations described in the previous sections and carried out experiments on problems from the TPTP library, Version 6.0.0 [89]. The implementation, in SWI-Prolog, is called Yarralumla (Yet another range-restriction avoiding loops under much less assumptions). Since the transformations introduced in this paper are defined for clausal problems we have selected for the experiments all the CNF problems from the TPTP suite.

In our initial research [16] we used Yarralumla with the MSPASS theorem prover, Version 2.0g.1.4 [52]. As the extra features of MSPASS have in the mean time been integrated into the SPASS theorem prover [94] and SPASS has significantly evolved since Version 2.0, for the present paper we combined Yarralumla with SPASS Version 3.8d.

For that purpose we modified the code of SPASS in a number of ways. We added one new flag to activate splitting on positive ground equality literals in positive non-Horn clauses. The main inference loop was adapted so that finding a splitting clause and applying splitting has highest priority (unchanged) followed immediately by picking a non-positive blocking clause, that is, a clause of the form $B_1 \wedge \cdots \wedge B_k \rightarrow s \approx t \vee H_1 \vee \cdots \vee H_m$ for $m \geq 0$ and $k > 0$, and performing inferences with it. The selection of splitting clauses was adapted so that positive ground clauses of the form $s \approx t \vee H_1 \vee \cdots \vee H_m$, where $m \geq 1$, are always

selected, when there are any. Moreover, the first equality literal is split upon. This adaptation ensures blocking is performed eagerly to keep the set of ground terms small. The tests with Yarralumla were performed using ordered resolution and superposition with selection of at least one negative literal, forward and backward rewriting, unlimited splitting (as described above),[9] matching replacement resolution,[10] subsumption and various other simplification rules. This means the inferences are performed in an ordered hyperresolution-style with eager splitting and forward and backward ground rewriting. The derivations constructed are thus BUMG tree derivations, the proofs produced are BUMG refutation proofs, and the models returned are BUMG models.

We also tested SPASS Version 3.8d in auto mode on the sample. In auto mode SPASS used ordered resolution with dynamic selection of negative literals (an option in SPASS). SPASS automatically turned off splitting for non-Horn clauses. Dynamic selection means typically literals were only selected if multiple maximal literals occur in a clause. This means the behaviour of SPASS in auto mode was very different to that of SPASS-Yarralumla, which always selected a literal in clauses with non-empty negative part. The changes to SPASS in SPASS-Yarralumla meant that splitting was performed eagerly and blocking clauses were targeted, which was not the case with SPASS in auto mode. We tested SPASS in auto mode only on the original files (translated from TPTP syntax to SPASS syntax).

The experiments were run on a cluster of 128 Dell PowerEdge M610 Blade Servers each with two Intel Xeon E5620 2.4 GHz processors and 48 GiB main memory each. The time limit was ten minutes (CPU time).

SPASS-Yarralumla is available from
http://www.cs.man.ac.uk/~schmidt/spass-yarralumla/.

10.1 Results

Tables 3 and 4 summarize the results for satisfiable clausal problems in the TPTP library, measuring the number of problems solved within the time limit. The columns with the heading '#' give the number of problems in the TPTP categories and the different TPTP rating ranges. The subsequent columns give the number of problems solved within the time limit. The results are presented for the different BUMG methods that were used. For example, sh ∘ rr ∘ sdb refers to the method based on the transformation defined by the new range-restriction transformation, shifting and subterm domain blocking. To evaluate the effect of the different forms of blocking the results are grouped into groups of four: new range-restriction rr, new range-restriction with shifting sh ∘ rr, classical range-restriction crr and classical range-restriction with shifting sh ∘ crr. (The groups are separated by larger gaps in the tables.) In each group the first column provides the *baseline* for that group. The last column with the heading 'auto' gives the results for runs of SPASS Version 3.8d in auto mode on the original input files. The runtimes for the problems solved spanned the whole range, from less than one second to all of the time allowed.

The best results in each group in each row are highlighted in bold font. The underlined values are the best results for all methods including SPASS in auto mode. As expected the worst results in each group were obtained for the baseline transformations without blocking. This confirms the expectation that blocking is an essential technique for BUMG methods.

[9] The splitting options in SPASS are: no splitting, unlimited splitting and a strategy performing splitting on 'optimal' literals only, but this was not used.

[10] Matching replacement resolution is also known as subsumption resolution, see [92] for a definition.

Category	#	rr	rr ∘ sdb	rr ∘ udb	rr ∘ spb	rr ∘ upb	sh ∘ rr	sh ∘ rr ∘ sdb	sh ∘ rr ∘ udb	sh ∘ rr ∘ spb	sh ∘ rr ∘ upb	crr	crr ∘ sdb	crr ∘ udb	crr ∘ spb	crr ∘ upb	sh ∘ crr	sh ∘ crr ∘ sdb	sh ∘ crr ∘ udb	sh ∘ crr ∘ spb	sh ∘ crr ∘ upb	auto
ALG	37	-	-	-	-	-	-	-	-	-	-	-	**1**	**1**	-	-	-	-	-	-	-	**3**
ANA	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
BOO	17	-	**5**	**5**	-	-	-	-	-	-	-	-	3	**6**	-	-	3	**6**	-	-	-	2
CAT	10	2	**6**	4	2	-	2	**4**	**4**	2	-	2	3	**4**	2	2	2	3	**4**	3	2	2
COL	6	-	-	-	-	-	-	-	-	-	-	-	1	**3**	-	-	-	1	**3**	-	-	-
GEO	17	-	-	-	-	-	-	-	-	-	-	-	-	**1**	-	-	-	-	-	-	-	-
GRP	85	49	55	**60**	49	50	50	53	**57**	50	51	49	55	**61**	49	50	49	52	**56**	49	50	-
HEN	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
HWC	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
HWV	39	2	2	2	2	2	2	2	2	2	2	2	**3**	**3**	2	2	2	2	**3**	2	2	-
KRS	13	1	**8**	**8**	7	5	7	**8**	**8**	**8**	**8**	-	7	**8**	7	5	-	7	7	7	5	**8**
LAT	62	-	**10**	4	-	-	-	**6**	5	-	-	-	25	**32**	-	-	-	24	**30**	-	-	-
LCL	44	1	**4**	1	2	1	2	**4**	3	2	2	-	2	**5**	-	-	1	2	**5**	1	1	-
LDA	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MGT	11	1	6	**9**	6	1	4	7	7	**8**	4	-	**7**	5	5	-	-	4	**5**	4	-	8
MSC	2	1	1	1	1	1	1	1	1	1	1	-	1	1	1	1	-	1	1	1	1	1
NLP	236	52	130	**137**	128	63	99	191	**195**	188	110	22	**125**	**125**	114	33	22	118	**125**	104	33	**198**
NUM	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
PLA	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PUZ	27	7	**8**	**8**	7	7	7	**8**	**8**	7	7	7	**8**	**8**	7	7	7	**8**	**8**	7	7	7
REL	1	-	**1**	**1**	-	-	-	**1**	**1**	-	-	-	**1**	**1**	-	-	-	**1**	**1**	-	-	-
RNG	14	1	1	1	1	1	1	1	1	1	1	1	1	**2**	1	1	1	**3**	1	1	1	-
ROB	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SCT	3	2	**3**	2	**3**	**3**	2	2	**3**	2	**3**	2	**3**	**3**	**3**	**3**	2	2	**3**	2	2	**3**
SET	12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SWC	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SWV	147	2	6	**8**	7	2	2	2	**7**	2	2	-	5	**7**	5	-	-	4	**6**	4	-	**79**
SWW	39	-	7	**8**	-	-	-	**2**	1	1	-	-	-	**4**	-	-	-	-	-	-	-	-
SYN	223	54	132	**147**	126	117	56	132	**145**	125	117	54	134	**148**	126	110	55	131	**143**	124	111	99
SYO	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TOP	19	-	2	**3**	1	-	-	**1**	**1**	-	-	-	**2**	**2**	1	-	-	1	**3**	-	-	**6**
Total	1126	176	388	**410**	345	256	236	430	**456**	402	311	140	389	**430**	324	215	142	368	**411**	310	216	417

Table 3 Number of problems solved on satisfiable problems, by TPTP categories.

Among the different blocking techniques the best results were obtained with unrestricted domain blocking in all four groups. Overall, the best result was obtained for the combination with rr and shifting, i.e., sh ∘ rr ∘ udb, solving 6.0% more problems than the second best method, crr ∘ udb using the classical range-restriction transformation without shifting, and nearly 11% more problems than the transformations rr ∘ udb and sh ∘ crr ∘ udb. This means shifting had a significant positive effect in combination with the new range-restriction transformation, but less so in combination with classical range-restriction. The positive effect of

Category	#	rr	rr ∘ sdb	rr ∘ udb	rr ∘ spb	rr ∘ upb	sh ∘ rr	sh ∘ rr ∘ sdb	sh ∘ rr ∘ udb	sh ∘ rr ∘ spb	sh ∘ rr ∘ upb	crr	crr ∘ sdb	crr ∘ udb	crr ∘ spb	crr ∘ upb	sh ∘ crr	sh ∘ crr ∘ sdb	sh ∘ crr ∘ udb	sh ∘ crr ∘ spb	sh ∘ crr ∘ upb	auto
0.00	371	112	268	**286**	256	184	154	279	**293**	267	223	80	264	**276**	241	150	80	250	**270**	227	150	231
(0.00, 0.10]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(0.10, 0.20]	32	1	20	**21**	20	2	-	6	26	**27**	**27**	7	-	**21**	19	20	1	-	18	**19**	18	-	**29**
(0.20, 0.30]	132	57	**66**	65	59	59	59	**88**	**88**	81	59	54	64	**66**	56	54	55	65	**66**	57	56	39
(0.30, 0.40]	52	4	**6**	**6**	4	4	13	21	**22**	20	13	4	5	**6**	4	4	4	4	**6**	4	4	20
(0.40, 0.50]	125	2	21	**23**	6	7	3	14	**22**	5	8	2	26	**42**	3	6	3	23	**40**	4	6	18
(0.50, 0.60]	27	-	6	**7**	-	-	1	1	**2**	**2**	1	-	3	**7**	-	-	-	-	**1**	-	-	6
(0.60, 0.70]	76	-	1	**2**	-	-	-	1	**2**	-	-	-	5	**8**	-	-	-	-	**6**	**6**	-	-
(0.70, 0.80]	83	-	-	**1**	-	-	-	**1**	**1**	-	-	-	-	-	-	-	-	-	-	-	-	**59**
(0.80, 0.90]	78	-	**1**	-	-	-	-	-	-	-	-	-	-	**4**	-	-	-	1	**2**	-	-	**15**
(0.90, 1.00]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1.00	1017†	-	-	-	-	-	-	-	-	-	-	-	-	**1**	-	-	-	-	-	-	-	-
Total	-	176	388	**410**	345	256	236	430	**456**	402	311	140	389	**430**	324	215	142	368	**411**	310	216	417

Table 4 Number of problems solved on satisfiable problems, by TPTP problem rating. Note: †1017 = 150 SAT + 28 OPN + 839 UNK

shifting could also be seen for the number of problems solved without blocking for rr and sh ∘ rr (34% improvement).[11]

The good results for crr ∘ udb show the value of classical range-restriction. In the LAT category, crr ∘ udb solved 32 problems, whereas sh ∘ rr ∘ udb solved only 5 problems. In this category the problems are from the domain of lattice theory and contain predominantly equational axioms (i.e., non-ground equational unit clauses) and one ground inequality (for the axiom to be refuted). On such problems it can be seen that crr produces instantiations of the axioms very quickly, whereas the rr favours producing clauses involving the myequal predicate and the blocking splitting decisions are less systematic because domain elements are identified on a by-need basis during the inference process, whereas for crr, all constants are in the domain from the outset, leading to blocking being performed with these immediately. On these problems crr has the advantage that blocking leads to a reduction of the size of the domain more quickly. This shows there is a trade-off between using the crr transformation and the rr transformation. The results however also showed the virtues of unrestricted domain blocking as a universal technique for BUMG. SPASS in auto mode fared very well in the SWV category, where 79 problems were solved compared to 7–8 problems for the best BUMG methods. Overall SPASS in auto mode solved 9% fewer problems than the best BUMG method sh ∘ rr ∘ udb.

Looking at the top half of Table 4 (up to difficulty rating of 0.40), the BUMG method based on sh ∘ rr ∘ udb fared best, but for problems more difficult (up to a rating of 0.70) the performance deteriorated and the method crr ∘ udb solved the highest number of problems. For problems with ratings higher than 0.70 SPASS in auto mode solved significantly more problems than the BUMG methods. One problem with rating 1.00 was solved by the crr ∘ udb method (namely, GRP741-1 in 121.86 seconds). Problems in the TPTP library with rating 1.00 have not yet been solved by any other prover.

Satisfiable	Baseline	sdb	udb	spb	upb
	rr	-0 / +212	**-4 / +238**	-0 / +169	-1 / +81
	sh ∘ rr	-1 / +195	**-5 / +225**	-2 / +168	-2 / +77
	crr	-0 / +249	**-4 / +294**	-0 / +184	-3 / +78
	sh ∘ crr	-0 / +226	**-5 / +274**	-0 / +168	-3 / +77
Unsatisfiable					
	rr	-211 / +78	-315 / **+83**	-100 / +61	**-77** / +37
	sh ∘ rr	-188 / +106	-225 / **+126**	-81 / +87	**-34** / +52
	crr	-65 / +170	-105 / **+242**	-32 / +78	**-26** / +24
	sh ∘ crr	-52 / +163	-98 / **+190**	-30 / +57	**-16** / +15

Table 5 Evaluation of blocking techniques.

Table 5 presents an evaluation of the different blocking techniques, listing the number of problems lost and the number of problems gained against the baseline methods in each group. The best results in each row of the first table (for satisfiable problems lost and gained)

[11] Since there are so many parameters interacting with each other and no categorization of the problems according to syntactic properties of the problems, it is not easy and perhaps not possible to give a precise explanation.

are highlighted in bold font. The results confirm the significant positive effect of unrestricted domain blocking for satisfiable problems. The results for unsatisfiable problems are discussed at the end of the section.

Analysis of the gain and loss of the method based on sh ∘ rr ∘ udb against the other methods gave these results: Against rr ∘ udb 66 problems were gained and 20 problems lost; against sh ∘ crr ∘ udb the gain/loss was +90/-45 and against crr ∘ udb it was +85/-59. This non-uniformity suggests each variation of range-restriction had the potential to solve some problems not solvable within the time limit by sh ∘ rr with unrestricted blocking. The biggest variation was against SPASS in auto mode, where 169 problems were gained and 130 problems were lost.

Table 6 displays how many problems were uniquely solved. We focus here only on the results for the satisfiable problems in the table and postpone the discussion for unsatisfiable problems to the end of the section. The first row lists how many problems were uniquely solved over all methods including SPASS in auto mode. Although two of the BUMG methods with unrestricted domain blocking fared better than SPASS in auto mode, the latter solved a significant number of problems that none of the BUMG could solve (namely, 115 problems, or 27.5% of the problems solved by SPASS in auto mode, or 10.2% of all satisfiable problems). This reflected the orthogonality of the underlying methods. Analogously, the relatively low number of problems uniquely solved by the BUMG methods (21 problems, i.e., 1.9% of all satisfiable problems), which is also apparent from the number of problems solved uniquely among the BUMG methods in the second row (25 or 2.2% of all satisfiable problems) can be attributed to the similarity of the underlying methods. An analysis of the number of uniquely solved problems per group of BUMG methods[12] in the third row of the table highlighted the importance of unrestricted domain blocking. While overall no problems were only solved with unary predicate blocking techniques, within the groups there were four problems solved only with unary predicate blocking.

Table 7 gives an impression of the increase in the size of the input files caused by the transformations. Although the file sizes were measured after all comments and white space were removed, variations in name lengths distort the values slightly (which can be seen in the values for shifting). The results therefore need to be interpreted cautiously. The average increase in file size does show a significant effect on the size of the problem for the new range-restriction transformations and also subterm blocking (both subterm domain blocking and subterm predicate blocking). The largest increase in size was observed for the problem SYO600-1 (13.7 fold increase), which contained 380 predicate symbols with arity up to 64, 2 constants and no non-constant function symbols. The main cause for this increase was the large number of clauses added in Step (4) of the rr transformation. For each of the 284 predicate symbols with arity 64 in the problem, 64 clauses were added in Step (4). This is a large number. In contrast for the crr transformations the increase in size was negligible, and also, generally, it was significantly lower. Despite its positive virtues this shows a downside of the rr transformation. For problems containing a large number of function symbols with high arity, Step (5) similarly adds many clauses, even though the transformation overall is still effective.

Analysis of the problems solved without any form of blocking revealed a large number belonged to the Bernays-Schönfinkel class: 131/176 (74%) for rr, 132/236 (56%) for sh ∘ rr, 133/140 (95%) for crr, and 134/142 (94%) for sh ∘ rr. Since without blocking no terms are (hypothetically) merged, these results confirm the expectation that more problems can be

[12] Using variations of rr, sh ∘ rr, crr or sh ∘ crr

	rr	rr ∘ sdb	rr ∘ udb	rr ∘ spb	rr ∘ upb	Total	sh ∘ rr	sh ∘ rr ∘ sdb	sh ∘ rr ∘ udb	sh ∘ rr ∘ spb	sh ∘ rr ∘ upb	Total	crr	crr ∘ sdb	crr ∘ udb	crr ∘ spb	crr ∘ upb	Total	sh ∘ crr	sh ∘ crr ∘ sdb	sh ∘ crr ∘ udb	sh ∘ crr ∘ spb	sh ∘ crr ∘ upb	Total	auto	Total
Satisfiable																										
All methods	-	1	-	-			-	-	-				-	11	-	-			-	4	1	-	-		**115**	136
All BUMG methods	-	1	2	-	-		-	-	6	-	-		-	**11**	-	-			-	4	1	-	-			25
All BUMG, by group	-	12	**28**	-	-	40	-	4	**25**	1	2	32	-	**57**	-	-	67		-	12	**60**	-	1	73		
Unsatisfiable																										
All methods	-	-	1	-			-	-	-	-	-		-	-	-	-			-	-	-	-	-			
All BUMG methods	**17**	4	5	2	2		3	5	3	-	3		-	3	3	2	-		-	1	3	-	-			
All BUMG, by group	**32**	15	18	23	4	92	16	24	**41**	29	14	124	4	23	**90**	10	3	130	8	31	**71**	6	2	118		
Total																									**1779**	1780
auto																										56

Table 6 Uniquely solved problems. Note: 'by group' means uniquely solved problems within each of the four groups of different variations of range-restriction: rr, sh ∘ rr, crr and sh ∘ crr.

Method	Avg
rr	1.8
rr ∘ sdb	2.4
rr ∘ udb	1.9
rr ∘ spb	2.4
rr ∘ upb	1.9
sh ∘ rr	1.7
sh ∘ rr ∘ sdb	2.3
sh ∘ rr ∘ udb	1.8
sh ∘ rr ∘ spb	2.3
sh ∘ rr ∘ upb	1.7

Method	Avg
crr	1.1
crr ∘ sdb	1.7
crr ∘ udb	1.2
crr ∘ spb	1.7
crr ∘ upb	1.2
sh ∘ crr	1.2
sh ∘ crr ∘ sdb	1.9
sh ∘ crr ∘ udb	1.3
sh ∘ crr ∘ spb	1.8
sh ∘ crr ∘ upb	1.3

Method	Avg
auto	0

Table 7 Average increase in the size of input files.

solved with the rr transformation than the crr transformation, due to the reduction of the number of terms added to the domain.

Although the main purpose of BUMG methods is *disproving* theorems and generating models for satisfiable problems, for completeness we report in Tables 8 and 9 the results for unsatisfiable clausal TPTP problems. Results of gains and losses for different blocking techniques and uniquely solved for unsatisfiable problems are included in Tables 5 and 6. Overall, the results were not as uniform as for satisfiable problems. However some general observations can be made. SPASS in auto mode fared best overall, and did so in all TPTP categories and each problem rating category. For unsatisfiable problems the drawback of BUMG methods is that clauses need to be exhaustively grounded and each branch in the derivation tree needs to be closed. The dominance of SPASS in auto mode is thus not surprising.

For the BUMG methods, a general deterioration in performance could be observed for shifting, when comparing the results for the groups with baselines sh ∘ rr and sh ∘ crr to the respective groups without shifting. This is plausible because shifting leads to fewer negative literals in clauses and more positive literals thus reducing the constraining effect and leading to more splitting. For problems with higher rating, shifting did seem to have a positive effect; for instance, in the $(0.40, 0.50]$ range, sh ∘ rr solved 70 problems whereas rr solved 32 problems.

Within the BUMG groups we expected best performance for the baseline transformations, because these do not involve blocking and performing many blocking steps lead to a significant overhead. However only for the first group the rr transformation fared best. In combination with classical range-restriction crr, somewhat surprisingly, the best results were obtained with unrestricted domain blocking, the most expensive form of blocking, because it is applicable to any terms. Among the blocking techniques in each case the highest gain was obtained for unrestricted domain blocking (see Table 5). However also the greatest loss was observed for this blocking technique. The smallest loss and lowest gain was obtained for upb blocking. The high loss for udb could be a reflection of the high increase in splitting steps preventing quicker detection of contradictions. Analogously the small loss for upb could be attributable to the smallest number of additional splitting steps among the blocking techniques. The high gain for udb blocking suggests the inference process panned out significantly differently leading to solutions not found with the other techniques. This seems to be supported by the results in the third row of Table 6 according to which, with

Category	#	rr	rr ∘ sdb	rr ∘ udb	rr ∘ spb	rr ∘ upb	sh ∘ rr	sh ∘ rr ∘ sdb	sh ∘ rr ∘ udb	sh ∘ rr ∘ spb	sh ∘ rr ∘ upb	crr	crr ∘ sdb	crr ∘ udb	crr ∘ spb	crr ∘ upb	sh ∘ crr	sh ∘ crr ∘ sdb	sh ∘ crr ∘ udb	sh ∘ crr ∘ spb	sh ∘ crr ∘ upb	auto
ALG	133	**13**	10	11	10	12	**7**	6	**7**	6	6	18	18	19	19	5	5	4	**6**	4	**62**	**40**
ANA	83	7	**11**	**11**	10	7	6	**7**	6	6	6	4	7	**10**	6	4	3	4	**5**	3	3	**40**
BOO	74	38	32	28	**43**	38	38	31	26	**42**	37	33	36	**38**	33	33	33	**36**	34	33	33	**62**
CAT	52	**38**	33	33	34	**38**	29	31	**32**	29	28	34	**36**	35	35	34	27	**30**	**30**	**30**	28	**51**
COL	225	**32**	23	22	25	28	19	18	16	17	**19**	**32**	31	25	**32**	31	8	**15**	**15**	8	8	**149**
COM	14	6	6	6	6	6	**6**	**6**	5	**6**	**6**	5	5	5	5	5	4	4	4	4	4	**12**
FLD	175	24	39	44	38	**48**	19	38	**45**	38	43	23	32	**36**	35	19	17	29	**37**	29	17	**99**
GEO	187	**68**	**68**	67	67	66	71	**74**	66	72	71	45	43	**53**	45	42	45	46	**47**	44	44	**121**
GRA	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**
GRP	798	233	183	171	**240**	225	**307**	231	260	291	298	319	378	**424**	337	322	358	385	**406**	353	357	**673**
HEN	64	**41**	**41**	34	40	**41**	34	**37**	32	34	34	31	**42**	40	31	29	36	**43**	33	35	36	**62**
HWC	4	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	1	1	1	1	1	**2**
HWV	116	**44**	39	34	40	35	21	21	21	**23**	20	18	18	**20**	18	19	**15**	14	10	13	12	**72**
KRS	17	9	9	9	9	9	9	9	9	9	9	7	**9**	**9**	**9**	**9**	5	8	**9**	8	8	**9**
LAT	250	39	33	29	**40**	39	39	25	24	**43**	37	56	56	**64**	57	56	70	65	56	**71**	70	**91**
LCL	625	**44**	40	35	39	45	**19**	**19**	17	**19**	**19**	43	**44**	**44**	**44**	43	21	24	**26**	23	21	**336**
LDA	23	**3**	2	2	**3**	**3**	**3**	2	**3**	**3**	5	5	5	5	5	5	5	**6**	5	5	5	**6**
MGT	67	**41**	38	35	37	42	**52**	40	29	40	46	18	**30**	25	21	20	12	**23**	17	13	12	**61**
MSC	20	9	**11**	9	10	9	8	**10**	7	9	7	8	**10**	**10**	9	8	7	**10**	8	7	7	**14**
NLP	22	17	**18**	**18**	**18**	17	**20**	19	18	18	**20**	9	**18**	14	**18**	9	9	**18**	14	**18**	9	**22**
NUM	86	**15**	10	10	13	13	10	8	9	9	**11**	10	9	8	10	**11**	**8**	7	**8**	**8**	**8**	**36**
PLA	51	4	**5**	4	4	**5**	3	**4**	3	3	**4**	3	**4**	2	**4**	3	2	**3**	1	**3**	**3**	**31**
PUZ	72	**46**	45	37	45	45	**38**	37	36	37	**38**	**44**	43	35	43	41	**34**	**34**	30	33	33	**55**
REL	107	**4**	3	2	**4**	**4**	3	2	2	**4**	3	5	**7**	6	5	5	5	**6**	**6**	5	5	**37**
RNG	83	20	11	8	**23**	19	19	10	9	**22**	19	10	**14**	**14**	11	10	10	**14**	**14**	11	11	**48**
ROB	31	**4**	3	2	**4**	**4**	**5**	3	1	**5**	**5**	3	**5**	**14**	3	3	3	**6**	4	5	3	**14**
SCT	98	8	8	8	8	8	2	2	2	2	2	11	11	11	11	11	2	2	2	2	2	**27**
SET	450	**68**	63	57	58	66	46	50	**52**	47	45	53	50	51	**54**	53	29	36	**40**	29	28	**290**
SWC	383	**120**	107	76	100	81	75	74	75	74	**89**	**111**	106	100	105	110	99	**100**	89	99	97	**281**
SWV	855	100	72	70	92	**104**	43	45	38	**48**	45	**77**	72	71	**77**	**77**	29	31	**33**	29	28	**373**
SWW	33	-	-	-	-	-	-	-	-	-	-	1	1	1	1	1	-	-	-	-	-	**12**
SYN	621	**330**	329	321	326	328	318	**326**	318	319	317	321	321	314	319	**322**	300	**307**	**306**	300	303	**508**
SYO	17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-	1	-	1	1	**3**
TOP	5	3	3	3	3	3	1	2	**3**	1	1	2	**3**	**3**	**3**	**3**	1	1	1	1	1	**5**
Total	5842	**1431**	1298	1199	1392	1391	1273	1191	1174	1279	**1291**	1363	1468	**1500**	1409	1361	1204	**1315**	1296	1231	1203	**3665**

Table 8 Number of problems solved on unsatisfiable clausal TPTP problems.

Category	#	rr	rr ∘ sdb	rr ∘ udb	rr ∘ spb	rr ∘ upb	sh ∘ rr	sh ∘ rr ∘ sdb	sh ∘ rr ∘ udb	sh ∘ rr ∘ spb	sh ∘ rr ∘ upb	crr	crr ∘ sdb	crr ∘ udb	crr ∘ spb	crr ∘ upb	sh ∘ crr	sh ∘ crr ∘ sdb	sh ∘ crr ∘ udb	sh ∘ crr ∘ spb	sh ∘ crr ∘ upb	auto
0.00	1409	871	830	796	860	**885**	734	745	722	744	**746**	739	**769**	747	756	737	590	**654**	647	614	597	**1360**
(0.00, 0.10]	480	**244**	203	187	236	239	210	184	167	**216**	210	236	239	**245**	237	234	204	**213**	212	202	201	**477**
(0.10, 0.20]	594	**151**	145	118	148	137	133	123	112	140	**141**	151	**165**	162	164	150	145	**157**	144	**157**	144	**529**
(0.20, 0.30]	507	**98**	85	72	87	79	70	61	63	65	**72**	85	89	**94**	85	83	67	70	**71**	66	64	**382**
(0.30, 0.40]	258	**30**	24	18	**30**	24	38	30	**40**	35	36	42	55	**61**	51	44	51	**53**	46	49	51	**226**
(0.40, 0.50]	513	**32**	11	6	24	22	**70**	45	53	62	69	75	116	**139**	86	76	89	112	**125**	89	88	**344**
(0.50, 0.60]	207	3	-	1	**4**	2	13	2	**14**	9	13	22	23	**30**	20	23	27	**30**	**30**	26	27	**122**
(0.60, 0.70]	298	2	-	1	3	3	**4**	1	3	**6**	4	12	11	**13**	9	12	**19**	14	11	16	**19**	**110**
(0.70, 0.80]	324	-	-	-	-	-	1	-	-	**2**	-	1	1	**6**	1	2	**4**	**4**	**4**	**4**	**4**	**85**
(0.80, 0.90]	344	-	-	-	-	-	-	-	-	-	-	-	-	**1**	-	-	**1**	**1**	**1**	**1**	**1**	**25**
(0.90, 1.00]	259	-	-	-	-	-	-	-	-	-	-	-	**2**	-	-	-	**7**	**7**	5	**7**	**7**	**5**
1.00	1516	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Total	-	**1431**	1298	1199	1392	1391	1273	1191	1174	1279	**1291**	1363	1468	**1500**	1409	1361	1204	**1315**	1296	1231	1203	**3665**

Table 9 Result summary wrt. problem rating on unsatisfiable clausal TPTP problems. Note: †1516 = 649 UNS + 28 OPN + 839 UNK

one exception, the largest number of uniquely solved problems in each group was obtained with udb blocking. The exception was the first group, where rr led to the largest number of uniquely solved problems. Among all the BUMG methods, rr solved the largest number of problems not solved by any of the other methods. However, these results pale against the number of uniquely solved problems by SPASS in auto mode. Only one problem was solved by a BUMG method which was not solved by SPASS in auto mode.

10.2 Findings of the Experimental Evaluation

Several findings can be drawn from the results. The results have confirmed our expectation that unrestricted domain blocking is a powerful technique, which helps discover finite models more often than with the other blocking techniques. The results suggest the technique is indispensable for bottom-up model generation. Both in combination with the new range-restricting transformation, and the classical range-restricting transformation, good results have been obtained. Overall, the method based on new range-restriction, shifting and unrestricted domain blocking performed best on the sample. On satisfiable problems with higher difficulty rating this method was however gradually edged out by the method based on classical range-restriction and unrestricted domain blocking. This suggests there is a trade-off between the rr transformation, which is based on a non-trivial transformation but does restrict the creation of terms, and the simpler crr transformation, which has to rely on blocking to restrict the creation of terms.

The results for subterm domain blocking were good and often not far behind unrestricted domain blocking for satisfiable problems. In contrast, predicate blocking seems not to be effective on many problems. We attribute this to the nature of the problems in the TPTP library.

An investigation with SPASS-Yarralumla on translations of modal logic problems has revealed a different picture [78]. There, the best performance was obtained with subterm domain blocking for both satisfiable and unsatisfiable problems. Better results than for unrestricted domain blocking were also obtained with subterm predicate blocking and unrestricted predicate blocking. Better performances for subterm and predicate blocking are also expected on problems stemming from (cyclic) description logic knowledge bases. Experiments with blocking restricted by excluding a finite subset of the domain have shown better results than for unrestricted domain blocking for consistency testing on a large corpus of ontologies [56]. The better performance for restricted forms of blocking on modal and description logic problems can be attributed to mainstream modal and description logics having the finite tree model property. This means every satisfiable formula holds in a model based on a finite tree, which is not a property of first-order formulae.

The results showed BUMG methods were good for disproving theorems and generating models for satisfiable problems. For unsatisfiable problems BUMG methods were significantly less efficient than SPASS in auto mode. For theorem proving purposes a limitation of BUMG methods is that they require full grounding. It can be seen already from very small unsatisfiable examples that a complete BUMG derivation tree can be very large, whereas resolution proofs are significantly shorter.

Compared to ordered resolution (and appropriate equality reasoning) without splitting, an advantage of BUMG methods for satisfiable problems is the division of the search space into branches which are individually constructed and individually processed. As a consequence, if the right decisions are made at branching points models can be found more quickly. When the branching point decisions are less optimal the performance can deteriorate dramatically,

particularly if the search is trapped in a branch with only infinite models. This could be another explanation for the lower success rate of the BUMG methods observed for more difficult satisfiable problems. For problems where only infinite models exist, clearly other methods are better. Methods that can compute representations of infinite models include the method of [33] computing atom representations and the method of [27] using a hybrid approach based on careful enumeration and deduction on constrained clauses.

11 Discussion and Related Work

For modal, description or hybrid logics characterized by finite tree models, blocking is implemented as a loop-checking mechanism which involves the comparison of sets of formulae in order to detect periodicity in the (implicitly) constructed model. The idea is that (implicit) objects (or possible worlds) are reused when the set of formulae associated with the current (implicit) object is the same as (or a subset of) the formulae associated with another previously created (implicit) object, because the inference steps would be repetitions of inference steps previously performed for the other object [58,49,35,64,25,47,19,28]. Variations of this form of loop-checking can be found in description logic tableaux systems, referred to as predecessor equality (or subset) blocking, ancestor equality (or subset) blocking and anywhere equality (or subset) blocking [25,47,2] as well as pattern-based blocking [55]. These loop-checking techniques are sound for modal and description logics with some form of tree model property, provided the inference rules and the inference strategy ensures that objects are maximally expanded at the point that the loop-check is performed (e.g., *S4* and $\mathcal{ALC}$ with TBox axioms belong to this class). For other logics, it must be possible to undo previous merges of objects when no models can be found, and to try other merges. Examples of this form of blocking are dynamic anywhere blocking and pair-wise blocking which have been introduced for logics with inverse and functional relations [45,2,46]. Often these tableau systems do not require explicit presence of objects (or worlds) in the formulae of the tableau system and tightly integrate loop-checking into the rules and how they can be used. As a consequence the definitions of these tableau systems and the soundness and completeness proofs are very particular. How far this style of system and loop-checking can be pushed toward the boundary of undecidable modal, description and hybrid logics is open. It is also unclear if the approach can be extended beyond logics with some form of tree-model property, see the discussion in [82].

The work of [50,51,53] presents the relationship between tableau-based description logic systems and first-order resolution. In [50] it was shown how tableau decision procedures for $\mathcal{ALC}$ with TBox axioms can be step-wise simulated using a positive BUMG refinement of resolution (a negative selection refinement augmented with splitting and one-step rewriting). This requires a special encoding of the semantics of the logic and the blocking rule

$$\frac{\mathrm{M}}{\mathrm{M},\ s \approx t}$$

with the side-condition that s and t (representing two objects) are maximally expanded and indistinguishable with respect to the properties that hold for them (this exploits the Leibniz principle for all positive units with a unary predicate symbol, which suffices for $\mathcal{ALC}$ and also *S4*). This achieves the same behaviour as loop-checking based on subset (or equality) blocking, which is sufficient to decide these logics. In [31,53] the relationship is extended to a large class of modal logics (including the standard ones between multi-modal versions of

modal logic *K* and *S5*), and used as a common setting for an empirical evaluation of ordered resolution refinements and BUMG-based refinements. The link is also used to define tableau systems for novel extended modal logics defined over Kripke frames closed under relational intersection, union and converse of relations.

Blocking as introduced in this paper and previously in [16], is achieved by transforming the input clause set and extending it with blocking clauses. In semantic tableau systems unrestricted blocking can be formulated as an analytic cut rule

$$\frac{\mathrm{M}}{\mathrm{M},\ s \approx t \quad | \quad \mathrm{M},\ s \not\approx t}$$

expanded using a left-to-right strategy [79,82].[13] The rule asserts the equality $s \approx t$ of two objects (labels) and if this does not lead to finding a model (e.g., all the branches from this point onwards can be closed), then $s \not\approx t$ must hold, which is achieved by normal disjunctive backtracking. Adding this rule to any sound and complete inference system preserves soundness and completeness. The same is true for restricted versions of the rule [56]. It has been shown that if a logic admits finite filtration or has the (effective) finite model property any semantic tableau system extended with this mechanism provides a decision procedure for the logic [80,82]. These results provide the theoretical foundation for a generic platform in which tableau decision procedures can be built in a systematic and uniform way and other loop-checking mechanisms can be simulated [76,81,56]. The unrestricted blocking mechanism has been implemented in MetTeL tableau prover generator [90,91].

We argue the key principles and ideas of this platform extend to most, if not all, methods in the BUMG paradigm. Using unrestricted blocking, either as a transformation into clausal form, or as an inference rules as given above, and small adjustments to the tableau generation methodologies [80,81,76], the effort to develop terminating BUMG methods for decidable, first-order representable logics should not be considerable.

Unrestricted blocking has been used to define ground semantic tableau systems for various concrete cases of deciding: the description logics $\mathcal{ALCO}$, $\mathcal{ALCO}$ with transitive roles, $\mathcal{ALBO}$ [79,80] and $\mathcal{ALBO}^{\mathsf{id}}$ [82], intuitionistic logic [81], interrogative epistemic logics [67], modal logics with counting quantifiers [97], the description logic $\mathcal{SHOI}$ [56], a bi-intuitionistic logic [87] and a related multi-modal logic [78]. The evaluations in [56,87,78] of different variations of unrestricted blocking on description logic ontologies and modal logic problems complement those in the present paper.

Reuse of domain terms in the handling of existentially quantified formulae is common in automated model building systems for enumerating finite models. In tableau systems it can be found in the form of an adapted and extended δ-rule (i.e., a δ^*-rule) [44,61,23,70]. The δ^*-rule has the property that it avoids the use of non-constant Skolem functions and does not require the inclusion of equality in a BUMG system. The effect of δ^*-rule can be achieved with (equality-based) unrestricted blocking, but the reverse does not hold.

The nineties saw presentations of hyperresolution decision procedures (without splitting or backtracking) for various classes, which can be used for automating model building [33,26]. This work developed ways of representing models in compact form as atom representations and sometimes allows infinite models to be presented finitely. Completeness for finite model generation is achieved through enhancing the approach with an enumeration technique where the idea is reminiscent of the δ^*-rule, but splitting is not necessarily performed [71].

[13] For description or hybrid logics without an explicit equality predicate or identity relation, the rule can be appropriately reformulated using nominals, or the language of the tableau system can be extended to include the equality predicate.

A more powerful method with the functionality to compute finite representations of infinite models is described in [27].

Methods such as the extended PUHR tableau method [23], and the method for geometric logic [17] directly search for a finite model. They can be seen as BUMG methods presented as tableau calculi with a sophisticated inference rule for existential quantification. Whenever a witness term (technically, a constant) for an existentially quantified variable is sought, the idea is to attempt to re-use an existing one, to keep the domain small (and where possibly finite).

Other methods for model computation search for small finite models, by essentially searching the space of interpretations with increasing domain sizes $1, 2, \ldots$, in increasing order, until a model is found. The SEM-family [85,98,66] of procedures do that by essentially built-in exhaustive search over the interpretations for a fixed domain size. Model builders such as ANL-DP in the old style of MACE [65], including for example the method of [29], reduce model search to testing of propositional satisfiability. The approach of [10] is conceptually similar but translates into the Bernays-Schönfinkel fragment of first-order logic instead. The satisfiability test is delegated to an instance-based method (see e.g. [8,57]), which generally decide satisfiability of that class.

Being based on a translation, the MACE-style approach is conceptually related, but different to our approach, as the translation to propositional logic is parameterized by a domain size. In our approach there is no a priori requirement for iterative deepening over the domain size, and the search for finite models works differently, by making terms equal as needed or as specified by the blocking strategy.[14]

This way, we can 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 counter-intuitive, for instance, in a description logic setting where unique names are not necessarily specified or formally assumed, but it would be unnatural to equate a and b, if a represents a person, Alice say, and b represents an illness, bronchitis say.[15] Furthermore, logic programs are typically understood with respect to Herbrand semantics, and it is desirable to develop compatible model building techniques. Our transformations are more careful at identifying objects than the methods mentioned and thus work closer to a Herbrand semantics.

Winker [96] is the first work that we are aware of, which uses hyperresolution in combination with the classical range-restriction transformation to generate ground instantiations of sets of equations (in this case). As an extra 'trick' the user is allowed to successively add equations $s \approx t$ considered suitable to limit the number of instantiations and find small models by adding clauses of the form

$$\mathsf{dom}(x_1) \wedge \cdots \wedge \mathsf{dom}(x_n) \rightarrow s \approx t,$$

where $x_1, \ldots, x_n$ are the variables occurring in $s \approx t$, to the problem set. Using these tricks [96] shows that first-order theorem provers can be used to solve open problems of axiom independence of ternary Boolean algebras.

Tammet [34, Ch. 7] found a special condition and automated method of extending clause sets with equations between terms of the Herbrand universe (i.e., ground terms) to decide the

[14] Finding models with a given cardinality can be done with BUMG and unrestricted blocking, but this requires breadth-first search or an iterative deepening approach.

[15] Using superposition it is possible to get decision procedures for the unique name assumption [84].

union of the Ackermann class and the monadic class using resolution and narrowing without backtracking.

Blocking has the same goal as the speculative inference method described in [62] and [21] (with improvements). The idea of allowing the speculative addition of clauses (which are not consequences) is incorporated into a DPLL($\Gamma + \mathcal{T}$) framework combining superposition and SMT solving [21]. Is is shown that two type systems relevant for software engineering (given by range-restricted theories) can be decided in this framework, when negative selection is used for resolution and superposition. Instances exemplified in [62] include replacing a clause by one that subsumes it, and adding equations for joining equivalence classes in the abstract congruence closure framework. A crucial difference in [21] to [62] is the ability to remove a speculatively added clause via native backtracking of DPLL, when this clause leads to the derivation of a contradiction. This variation means the approach of [21] is a sound blocking mechanism, just like our blocking mechanism.[16] A small difference is that our blocking mechanism replaces the speculated clause by its negation (in general, some caution is needed to avoid unneeded overhead, but if the case analysis is over ground clauses there is not an issue). In our setting, the kind of blocking behaviour as in [21] can be achieved with a small adjustment in the definition of the blocking transformations. In particular, only this clause is used in the respective 'equality case analysis' steps:

$$\mathcal{B} \rightarrow x \approx y \vee \mathsf{nonequal}(x, y),$$

where $\mathcal{B}$ is the appropriate constraint (cf. Section 7). That is, a new predicate symbol nonequal is used instead of the predicate symbol $\not\approx$ and no definitional clause is included. By not including a definition for $\mathsf{nonequal}(x, y)$ essentially the same effect is achieved: an equation $s \approx t$ is asserted and if this does not lead to finding a model the right branch is created containing the unit $\mathsf{nonequal}(s, t)$.[17] These clauses prevent blocking to s and t being applied again and they record the unsuccessful blocking attempts between terms undertaken during the derivation. The essence of what is exploited here is that the negated formula in an (analytic) cut rule does not need to be expanded if the cut rule is not essential for completeness.

12 Concluding Remarks

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 systems with only modest modifications. Our range-restriction technique is a refinement of existing transformations to range-restricted clauses in that terms are added to the domain of interpretation on a by-need basis. Moreover, we have presented methods that allow us to extend BUMG methods with blocking techniques related to loop-checking techniques with a long history in the more specialized setting of modal and description logics.

The experimental evaluation has shown blocking techniques are indispensable in BUMG methods for satisfiable problems. In particular, unrestricted domain blocking turned out to

[16] No separate proofs of soundness are needed unlike for loop-checking in modal or description logic tableau systems (cf. [56]) or the approach of Tammet [34, Ch. 7]. On the side we remark that it may be worth exploring if the subset blocking version of the PropagateEq-rule is more effective because it is can be expected to produce smaller models more quickly.

[17] This method is a slight improvement over [21] because the instance of the asserted clause that is responsible for the contradiction in the left branch is recorded, whereas [21] would only return a propositional symbol $\lceil \mathsf{nonequal}(x, y) \rceil$.

be the most powerful technique on problems from the TPTP library. Limiting the creation of terms during the inference process by using the new range-restricting transformation paid off, leading to better results. It showed best results together with the shifting transformation. The experimental results however also show that classical range restriction together with unrestricted blocking is a good complementary method. Because model generation methods are not just aimed at showing the existence of models but are built to construct and return models, when no models exist the entire search space must be traversed, which has led to inferior performance compared to saturation-based resolution.

Our bottom-up model generation approach is especially suitable for generating small models and we claim it is possible to show the approach using unrestricted domain blocking allows us to compute finite models when they exist (by adapting the proof of [81]). The models produced by subterm blocking and predicate blocking are not as small as those produced by unrestricted domain blocking. 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. Using subterm blocking or predicate blocking for the two unit clauses $\mathsf{P(a)}$ and $\mathsf{Q(b)}$, the associated model maps a and b to themselves (without merging them). More informative models are produced than those computed by unrestricted domain blocking and, for example, MACE- and SEM-style finite model searchers. From an applications perspective, this can be an advantage because larger models are more likely to be helpful to a user debugging mistakes in the formal specification of a program or protocol, or an ontology engineer trying to discover why an expected entailment does not follow from an ontology.

Research on developing resolution decision procedures has concentrated on developing ordering refinements of resolution for deciding solvable fragments of first-order logic. Fragments decidable with ordered resolution are complementary to the fragments that can be decided by refinements using the techniques presented in this paper. We have thus extended the set of techniques available for resolution methods to turn them into more effective and efficient (terminating) automated reasoning methods. In particular, we have shown that all procedures based on hyperresolution, or BUMG methods, can decide the Bernays-Schönfinkel class and the class of BS clauses with equality.

Studying how well the ideas and techniques discussed in this paper can be exploited and behave in dedicated BUMG provers, tableau-based provers and other provers (including resolution-based provers) is very important but is beyond the scope of the present paper. Our experience with another prover, Darwin [11], was 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 the presented transformations with other BUMG techniques, such as magic sets transformations [43, 88], a typed version of range-restriction [15], and minimal model computation [24, 23, 69]. Having been designed to be generic, we believe that our transformations carry over to formalisms with default negation, which could provide a possible basis for enhancements to answer-set programming systems.

Acknowledgements The second author is very grateful to Christoph Weidenbach and Uwe Waldmann for hosting her during 2010 and 2013–2014. In this time the implementation of SPASS-Yarralumla was completed and the experimental evaluation undertaken on the cluster of the Max-Planck-Institut für Informatik, Saarbrücken. We thank Uli Furbach, Dmitry Tishkovsky, Uwe Waldmann and Christoph Weidenbach for useful discussions and comments on this research, and extend our thanks to the anonymous reviewers for useful

comments on the submission. Financial support through research grants EP/F068530/1 and EP/H043748/1 of the UK Engineering and Physical Sciences Research Council (EPSRC) is gratefully acknowledged.

References

1. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *Description Logic Handbook*. Cambridge University Press, 2003.
2. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
3. L. Bachmair and N. Dershowitz. Critical pair criteria for completion. *Journal of Symbolic Computation*, 6(1):1–18, 1988.
4. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.
5. L. Bachmair and H. Ganzinger. Equational reasoning in saturation-based theorem proving. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction—A Basis for Applications*, pages 353–397. Kluwer, 1998.
6. L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*. North Holland, 2001.
7. L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 19–99. Elsevier, 2001.
8. P. Baumgartner. Logical engineering with instance-based methods. In F. Pfenning, editor, *Automated Deduction: CADE-21*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 404–409. Springer, 2007.
9. P. Baumgartner, P. Fröhlich, U. Furbach, and W. Nejdl. Semantically guided theorem proving for diagnosis applications. In M. Pollack, editor, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence: IJCAI 1997*, pages 460–465. Morgan Kaufmann, 1997.
10. P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli. Computing finite models by reduction to function-free clause logic. *Journal of Applied Logic*, 7(1):58–74, March 2009.
11. P. Baumgartner, A. Fuchs, and C. Tinelli. Implementing the model evolution calculus. *International Journal of Artificial Intelligence Tools*, 15(1):21–52, 2006.
12. P. Baumgartner, U. Furbach, and I. Niemelä. Hyper tableaux. In J. J. Alferes, L. M. Pereira, and E. Orlowska, editors, *Logics in Artificial Intelligence: JELIA 1996*, volume 1126 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1996.
13. P. Baumgartner, U. Furbach, and B. Pelzer. Hyper tableaux with equality. In F. Pfenning, editor, *Automated Deduction: CADE-21*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 492–507. Springer, 2007.
14. P. Baumgartner, U. Furbach, and B. Pelzer. The hyper tableaux calculus with equality and an application to finite model computation. *Journal of Logic and Computation*, 20(1):77–109, February 2010.
15. P. Baumgartner, U. Furbach, and F. Stolzenburg. Computing answers with model elimination. *Artificial Intelligence*, 90(1–2):135–176, 1997.
16. P. Baumgartner and R. A. Schmidt. Blocking and other enhancements for bottom-up model generation methods. In U. Furbach and N. Shankar, editors, *Automated Reasoning: IJCAR 2006*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 125–139. Springer, 2006.
17. M. Bezem. Disproving distributivity in lattices using geometry logic. In *Proceedings of CADE-20 Workshop on Disproving*, 2005.
18. J. C. Blanchette and T. Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In M. Kaufmann and L. C. Paulson, editors, *Interactive Theorem Proving: ITP 2010*, volume 6172 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2010.
19. T. Bolander and T. Bräuner. Tableau-based decision procedures for hybrid logic. *Journal of Logic and Computation*, 16(6):737–763, 2006.
20. M. P. Bonacina and J. Hsiang. On the modelling of search in theorem proving: Towards a theory of strategy analysis. *Information and Computation*, 147:171–208, 1998.
21. M. P. Bonacina, C. Lynch, and L. M. de Moura. On deciding satisfiability by theorem proving with speculative inferences. *Journal of Automated Reasoning*, 47(2):161–189, 2011.
22. M. P. Bonacina and D. A. Plaisted. Semantically-guided goal-sensitive reasoning: Inference system and completeness. *Journal of Automated Reasoning*, 59(2):165–218, 2017.
23. F. Bry and S. Torge. A deduction method complete for refutation and finite satisfiability. In J. Dix, L. Fariñas del Cerro, and U. Furbach, editors, *Logics in Artificial Intelligence: JELIA 1998*, volume 1489 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1998.

24. F. Bry and A. Yahya. Positive unit hyperresolution tableaux for minimal model generation. *Journal of Automated Reasoning*, 25(1):35–82, 2000.
25. M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.
26. R. Caferra, A. Leitsch, and N. Peltier. *Automated Model Building*, volume 31 of *Applied Logic*. Springer, 2004.
27. R. Caferra and N. Peltier. Combining enumeration and deductive techniques in order to increase the class of constructible infinite models. *Journal of Symbolic Computation*, 29(2):177–211, 2000.
28. M. Cialdea Mayer and S. Cerrito. Nominal substitution at work with the global and converse modalities. In L. Beklemishev, V. Goranko, and V. Shehtman, editors, *Advances in Modal Logic*, volume 8, pages 57–74. College Publications, 2010.
29. K. Claessen and N. Sörensson. New techniques that improve MACE-style finite model building. In P. Baumgartner and C. G. Fermüller, editors, *Proceedings of CADE-19 Workshop on Model Computation*, 2003.
30. L. M. de Moura and N. Bjørner. Bugs, moles and skeletons: Symbolic reasoning for software development. In *Automated Reasoning: IJCAR 2010*, volume 6173 of *Lecture Notes in Computer Science*, pages 400–411. Springer, 2010.
31. H. De Nivelle, R. A. Schmidt, and U. Hustadt. Resolution-based methods for modal logics. *Logic Journal of the IGPL*, 8(3):265–292, 2000.
32. N. Dershowitz. A maximal-literal unit strategy for Horn clauses. In S. Kaplan and M. Okada, editors, *Proceedings of the 2nd Workshop on Conditional and Typed Rewriting Systems (CTRS)*, volume 516 of *Lecture Notes in Computer Science*, pages 14–25. Springer, 1991.
33. C. G. Fermüller and A. Leitsch. Hyperresolution and automated model building. *Journal of Logic and Computation*, 6(2):173–203, 1996.
34. C. G. Fermüller, A. Leitsch, T. Tammet, and N. Zamov. *Resolution Method for the Decision Problem*, volume 679 of *Lecture Notes in Computer Science*. Springer, 1993.
35. M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese Library*. Reidel, 1983.
36. M. Fitting. *First-order logic and automated reasoning*. Graduate texts in computer science. Springer, 1996.
37. M. Fujita, J. Slaney, and F. Bennett. Automatic generation of some results in finite algebra. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence: IJCAI 1995*, pages 52–57. Morgan Kaufmann, 1995.
38. D. M. Gabbay and H. J. Ohlbach. Quantifier elimination in second-order predicate logic. *South African Computer Journal*, 7:35–43, 1992. Also published in B. Nebel, C. Rich, W. R. Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning: KR 1992*, pages 425–436. Morgan Kaufmann, 1992.
39. H. Ganzinger and K. Korovin. New directions in instantiation-based theorem proving. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science: LICS 2003*, pages 55–64. IEEE Computer Society Press, 2003.
40. H. Ganzinger and K. Korovin. Integrating equational reasoning into instantiation-based theorem proving. In *Computer Science Logic: CSL 2004*, volume 3210 of *Lecture Notes in Computer Science*, pages 71–84. Springer, 2004.
41. T. Geisler, S. Panne, and H. Schütz. Satchmo: The compiling and functional variants. *Journal of Automated Reasoning*, 18(2):227–236, 1997.
42. L. Georgieva, U. Hustadt, and R. A. Schmidt. Hyperresolution for guarded formulae. *Journal of Symbolic Computation*, 36(1–2):163–192, 2003.
43. 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*, volume 1249 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 1997.
44. J. Hintikka. Model minimization: An alternative to circumscription. *Journal of Automated Reasoning*, 4(1):1–13, 1988.
45. I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.
46. I. Horrocks and U. Sattler. A tableau decision procedure for SHOIQ. *Journal of Automated Reasoning*, 39(3):249–276, 2007.
47. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.
48. J. Hsiang and M. Rusinowitch. Proving refutational completeness of theorem proving strategies: The transfinite semantic tree method. *Journal of the ACM*, 38(3):559–587, 1991.
49. G. E. Hughes and M. J. Cresswell. *An Introduction to Modal Logic*. Routledge, 1968.

50. U. Hustadt and R. A. Schmidt. On the relation of resolution and tableaux proof systems for description logics. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence: IJCAI 1999*, pages 110–115. Morgan Kaufmann, 1999.
51. U. Hustadt and R. A. Schmidt. Issues of decidability for description logics in the framework of resolution. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *Lecture Notes in Artificial Intelligence*, pages 191–205. Springer, 2000.
52. U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: TABLEAUX 2000*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 67–71. Springer, 2000.
53. U. Hustadt and R. A. Schmidt. Using resolution for testing modal satisfiability and building models. *Journal of Automated Reasoning*, 28(2):205–232, 2002.
54. U. Hustadt, R. A. Schmidt, and C. Weidenbach. MSPASS: Subsumption testing with SPASS. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of International Workshop on Description Logics: DL 1999*, pages 136–137. Linköping University, 1999.
55. M. Kaminski and G. Smolka. Terminating tableau systems for hybrid logic with difference and converse. *Journal of Logic, Language and Information*, 18(4):437–464, 2009.
56. M. Khodadadi, R. A. Schmidt, and D. Tishkovsky. A refined tableau calculus with controlled blocking for the description logic $\mathcal{SHOI}$. In D. Galmiche and D. Larchey-Wendling, editors, *Automated Reasoning with Analytic Tableaux and Related Methods: TABLEAUX 2013*, volume 8123 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2013.
57. K. Korovin. Instantiation-based automated reasoning: From theory to practice. In R. A. Schmidt, editor, *Automated Deduction: CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 163–166. Springer, 2009.
58. S. Kripke. Semantical considerations of modal logic I: Normal modal propositional calculi. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.
59. K. R. M. Leino and A. Milicevic. Program extrapolation with Jennisys. In *Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications: OOPSLA 2012*, pages 411–430. ACM, 2012.
60. A. Leitsch. *The Resolution Calculus*. EATCS Texts in Theoretical Computer Science. Springer, 1997.
61. S. Lorenz. A tableaux prover for domain minimization. *Journal of Automated Reasoning*, 13(3):375–390, 1994.
62. C. Lynch. Unsound theorem proving. In *Computer Science Logic: CSL 2004*, volume 3210 of *Lecture Notes in Computer Science*, pages 473–487. Springer, 2004.
63. R. Manthey and F. Bry. SATCHMO: A theorem prover implemented in Prolog. In E. Lusk and R. Overbeek, editors, *Automated Deduction: CADE-9*, volume 310 of *Lecture Notes in Computer Science*, pages 415–434. Springer, 1988.
64. F. Massacci. Single step tableaux for modal logics: Computational properties, complexity and methodology. *Journal of Automated Reasoning*, 24(3):319–364, 2000.
65. 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.
66. W. McCune. Mace4 reference manual and guide. Technical Memorandum 264, Argonne National Laboratory, 2003.
67. S. Minica, M. Khodadadi, D. Tishkovsky, and R. A. Schmidt. Synthesising and implementing tableau calculi for interrogative epistemic logics. In P. Fontaine, R. A. Schmidt, and S. Schulz, editors, *PAAR-2012: Proceedings of the Third Workshop on Practical Aspects of Automated Reasoning*, volume 21 of *EPiC Series*, pages 109–123. EasyChair, 2013.
68. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 371–443. Elsevier, 2001.
69. F. Papacchini and R. A. Schmidt. A tableau calculus for minimal modal model generation. *Electronic Notes in Theoretical Computer Science*, 278(3):159–172, 2011.
70. F. Papacchini and R. A. Schmidt. Computing minimal models modulo subset-simulation for propositional modal logics. In P. Fontaine, C. Ringeissen, and R. A. Schmidt, editors, *Proceedings of the 9th International Symposium on Frontiers of Combining Systems: FroCoS 2013*, volume 8152 of *Lecture Notes in Artificial Intelligence*, pages 279–294. Springer, 2013.
71. N. Peltier. A calculus combining resolution and enumeration for building finite models. *Journal of Symbolic Computation*, 36(1-2):49–77, 2003.
72. B. Pelzer and C. Wernhard. System description: E-KRHyper. In F. Pfenning, editor, *Automated Deduction: CADE-21*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 508–513. Springer, 2007.
73. A. Riazanov. *Implementing an efficient theorem prover*. PhD thesis, Department of Computer Science, The University of Manchester, UK, 2003.

74. A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2-3):91–110, 2002.
75. J. A. Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1(3):227–234, 1965.
76. R. A. Schmidt. A new methodology for developing deduction methods. *Annals of Mathematics and Artificial Intelligence*, 55(1–2):155–187, 2009.
77. R. A. Schmidt and U. Hustadt. Solvability with resolution of problems in the Bernays-Schönfinkel class. Presented at Dagstuhl Seminar 05431, 2006, and ARW 2006 in Bristol, 2005.
78. R. A. Schmidt, J. G. Stell, and D. Rydeheard. Axiomatic and tableau-based reasoning for Kt(H,R). In R. Goré and A. Kurucz, editors, *Advances in Modal Logic, Volume 10*, pages 478–497. College Publications, 2014.
79. R. A. Schmidt and D. Tishkovsky. Using tableau to decide expressive description logics with role negation. In K. Aberer, K.-S. Choi, N. Fridman Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *The Semantic Web: ISWC 2007 + ASWC 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 438–451. Springer, 2007.
80. R. A. Schmidt and D. Tishkovsky. A general tableau method for deciding description logics, modal logics and related first-order fragments. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning: IJCAR 2008*, volume 5195 of *Lecture Notes in Computer Science*, pages 194–209. Springer, 2008.
81. R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. *Logical Methods in Computer Science*, 7(2):1–32, 2011.
82. R. A. Schmidt and D. Tishkovsky. Using tableau to decide description logics with full role negation and identity. *ACM Transactions on Computational Logic*, 15(1), 2014.
83. S. Schulz. System description: E 1.8. In K. McMillan, A. Middeldorp, and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning: LPAR 2013*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.
84. S. Schulz and M. P. Bonacina. On handling distinct objects in the superposition calculus. In B. Konev and S. Schulz, editors, Proceedings of the 5th International Workshop on the Implementation of Logics: IWIL 2005, http://www4.in.tum.de/~schulz/PAPERS/SB-IWIL-2005.ps.gz, 2005.
85. J. Slaney. FINDER (finite domain enumerator): Notes and guide. Technical Report TR-ARP-1/92, Australian National University, 1992.
86. R. M. Smullyan. *First Order Logic*. Springer, Berlin, 1971.
87. J. G. Stell, R. A. Schmidt, and D. Rydeheard. A bi-intuitionistic modal logic: Foundations and automation. *Journal of Logical and Algebraic Methods in Programming*, 85(4):500–519, 2016.
88. M. E. Stickel. Upside-down meta-interpretation of the model elimination theorem-proving procedure for deduction and abduction. *Journal of Automated Reasoning*, 13(2):189–210, 1994.
89. G. Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009.
90. D. Tishkovsky, R. A. Schmidt, and M. Khodadadi. MetTeL2: Towards a tableau prover generation platform. In P. Fontaine, R. A. Schmidt, and S. Schulz, editors, *PAAR-2012: Proceedings of the Third Workshop on Practical Aspects of Automated Reasoning*, volume 21 of *EPiC Series*, pages 149–162. EasyChair, 2012.
91. D. Tishkovsky, R. A. Schmidt, and M. Khodadadi. The tableau prover generator MetTeL2. In L. F. del Cerro, A. Herzig, and J. Mengin, editors, *Proceedings of the 13th European Conference on Logics in Artificial Intelligence: JELIA 2012*, volume 7519 of *Lecture Notes in Computer Science*, pages 492–495. Springer, 2012.
92. C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1965–2013. North Holland, 2001.
93. C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS version 3.5. In R. A. Schmidt, editor, *Automated Deduction: CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.
94. C. Weidenbach, R. A. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System description: SPASS version 3.0. In F. Pfenning, editor, *Automated Deduction: CADE-21*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 514–520. Springer, 2007.
95. C. Wernhard. System description: KRHyper. In *Proceedings of CADE-19 Workshop on Model Computation*, 2003.
96. S. Winker. Generation and verification of finite models and counterexamples using an automated theorem prover answering two open questions. *Journal of the ACM*, 29(2):273–284, 1982.
97. M. Zawidzki. *Deductive systems and decidability problem for hybrid logics*. PhD thesis, Faculty of Philosophy and History, University of Lodz, 2013.

98. H. Zhang. SEM: A system for enumerating models. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence: IJCAI 1995*, pages 298–303. Morgan Kaufmann, 1995.
99. H. Zhang and J. Zhang. Mace4 and SEM: A comparison of finite model generators. In M. P. Bonacina and M. E. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Artificial Intelligence*, pages 102–131. Springer, 2013.