Efficient Algorithms for Bounded Rigid E-Unification[⋆]

Peter Backeman and Philipp Rümmer

Uppsala University, Sweden

Abstract. Rigid E-unification is the problem of unifying two expressions modulo a set of equations, with the assumption that every variable denotes exactly one term (*rigid* semantics). This form of unification was originally developed as an approach to integrate equational reasoning in tableau-like proof procedures, and studied extensively in the late 80s and 90s. However, the fact that *simultaneous* rigid E-unification is undecidable has limited the practical relevance of the method, and to the best of our knowledge there is no tableau-based theorem prover that uses rigid E-unification. We recently introduced a new decidable variant of (simultaneous) rigid E-unification, *bounded* rigid E-unification (BREU), in which variables only represent terms from finite domains, and used it to define a first-order logic calculus. In this paper, we study the problem of computing solutions of (individual or simultaneous) BREU problems. Two new unification procedures for BREU are introduced, and compared theoretically and experimentally.

1 Introduction

The integration of efficient equality reasoning in tableaux and sequent calculi is a long-standing challenge, and has led to a wealth of theoretically intriguing, yet surprisingly few practically satisfying solutions. Among others, a family of approaches related to the (undecidable) problem of computing *simultaneous rigid* E*-unifiers* [7] have been developed, by utilising incomplete unification procedures in such a way that an overall complete first-order calculus is obtained. To the best of our knowledge, however, none of those procedures has led to competitive theorem provers.

We recently introduced *simultaneous bounded rigid* E*-unification* (BREU) [2], a new version of rigid E-unification that is bounded in the sense that variables only represent terms from finite domains, thus preserving decidability even for simultaneous E-unification problems. As demonstrated in [2], BREU can be used to design sound and complete calculi for first-order logic with equality, and to implement theorem provers that compare favourably to state-of-the-art tableau provers in terms of performance on problems with equality. In this paper we introduce two new unification algorithms for BREU problems.

[⋆] This work was partly supported by the Microsoft PhD Scholarship Programme and the Swedish Research Council.

1.1 Background and Motivating Example

We start by illustrating our approach using an example from [3, 2]:

$$\phi \quad = \quad \exists x, y, u, v. \begin{pmatrix} (a \not\approx b \ \vee \ g(x,u,v) \approx g(y, f(c), f(d))) \ \wedge \\ (c \not\approx d \ \vee \ g(u,x,y) \approx g(v, f(a), f(b))) \end{pmatrix}$$

For sake of presentation, the formula is flattened to ensure that every literal contains at most one function symbol (for more details, see [2]):

$$\begin{aligned} \phi' \quad = \quad & \forall z_1, z_2, z_3, z_4. \big(f(a) \not\approx z_1 \vee f(b) \not\approx z_2 \vee f(c) \not\approx z_3 \vee f(d) \not\approx z_4 \vee \\ & \quad \exists x, y, u, v. \forall z_5, z_6, z_7, z_8. \begin{pmatrix} g(x,u,v) \not\approx z_5 \vee g(y, z_3, z_4) \not\approx z_6 \vee \\ g(u,x,y) \not\approx z_7 \vee g(v, z_1, z_2) \not\approx z_8 \vee \\ ((a \not\approx b \vee z_5 \approx z_6) \wedge (c \not\approx d \vee z_7 \approx z_8)) \end{pmatrix}\big) \end{aligned}$$

To show that ϕ' is valid, a Gentzen-style proof (or, equivalently, a tableau) can be constructed, using free variables for x, y, u, v:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\mathcal{A}}{\ldots, g(X,U,V) \approx o_5, a \approx b \ \vdash \ o_5 \approx o_6} \qquad \cfrac{\mathcal{B}}{\ldots, g(U,X,Y) \approx o_7, c \approx d \ \vdash \ o_7 \approx o_8}}{\vdots}}{f(a) \approx o_1, f(b) \approx o_2, f(c) \approx o_3, f(d) \approx o_4 \ \vdash \ \exists x, y, u, v. \forall z_5, z_6, z_7, z_8. \ \ldots} \ (*)}{\vdots}}{\vdash \ \forall z_1, z_2, z_3, z_4. \ \ldots}$$

To finish this proof, both $\mathcal{A}$ and $\mathcal{B}$ need to be closed by applying further rules, and substituting concrete terms for the variables. In our bounded setting, we restrict the terms considered for instantiation of X, Y, U, V to the symbols that were in scope when the variables were introduced (at $(*)$ in the proof): X ranges over constants $\{o_1, o_2, o_3, o_4\}$, Y over $\{o_1, o_2, o_3, o_4, X\}$, and so on. Since the problem is flat, those sets contain representatives of all existing ground terms at point $(*)$ in the proof. We can observe that the proof can be concluded by applying the substitution $\sigma_b = \{X \mapsto o_1, Y \mapsto o_2, U \mapsto o_3, V \mapsto o_4\}$.

It has long been observed that this restricted instantiation strategy gives rise to a complete calculus for first-order logic with equality. The strategy was first introduced as *dummy instantiation* in the seminal work of Kanger [8] (in 1963, i.e., even before the introduction of unification), and later studied under the names *subterm instantiation* and *minus-normalisation* [4, 5]; the relationship to general Simultaneous Rigid E-unification (SREU) was observed in [3]. The present paper addresses the topic of solving a problem using the restricted strategy in an efficient way and makes the following main contributions:

- we define *congruence tables* and present an eager procedure for solving BREU using a SAT encoding (Sect. 4);
- we define *complemented congruence closure*, a procedure for abstract reasoning over sets of equivalence relations, and present a *lazy solving procedure* utilising this method (Sect. 5 and 6);
- we give an experimental comparison between the two methods (Sect. 7).

Further related work. For a general overview of research on equality handling in sequent calculi and related systems, as well as on SREU, we refer the reader to the detailed handbook chapter [4]. To the best of our knowledge, we are the first to develop algorithms for the BREU problem.

2 Preliminaries

We assume familiarity with classical first-order logic and Gentzen-style calculi (see e.g., [6]). Given countably infinite sets C of constants (denoted by $c, d, \dots$), V_b of bound variables (written $x, y, \dots$), and V of free variables (denoted by $X, Y, \dots$), as well as a finite set F of fixed-arity function symbols (written $f, g, \dots$), the syntactic categories of *formulae* ϕ and *terms* t are defined by

$$\phi \ ::= \ \phi \land \phi \mid \phi \lor \phi \mid \neg\phi \mid \forall x.\phi \mid \exists x.\phi \mid t \approx t \ , \qquad t \ ::= \ c \mid x \mid X \mid f(t, \dots, t) \ .$$

Note that we distinguish between constants and zero-ary functions for reasons that will become apparent later. We generally assume that bound variables x only occur underneath quantifiers $\forall x$ or $\exists x$. Semantics of terms and formulae without free variables is defined as is common.

We call constants and (free or bound) variables *atomic terms*, and all other terms *compound terms*. A *flat equation* is an equation between atomic terms, or an equation of the form $f(t_1, \dots, t_n) \approx t_0$, where $t_0, \dots, t_n$ are atomic terms. A *congruence pair* is a pair of two flat equations $(f(\bar{a}) \approx b, f(\bar{a}') \approx b')$ with $b \neq b'$.

A substitution is a mapping of variables to terms, s.t. all but finitely many variables are mapped to themselves. Symbols $\sigma, \theta, \dots$ denote substitutions, and we use post-fix notation $\phi\sigma$ or $t\sigma$ to denote application of substitutions. An *atomic substitution* is a substitution that maps variables only to atomic terms. An atomic substitution is *idempotent* if $\sigma \circ \sigma = \sigma$. We write $u[r]$ do denote that r is a sub-expression of a term or formula u, and $u[s]$ for the term or formula obtained by replacing the sub-expression r with s.

Definition 1 ([11]). *The* replacement relation $\rightarrow_E$ *induced by a set of equations E is defined by: $u[l] \rightarrow u[r]$ if $l \approx r \in E$. The relation $\leftrightarrow_E^*$ represents the reflexive, symmetric, and transitive closure of $\rightarrow_E$.*

2.1 Congruence Closure

We characterise the concept of congruence closure (CC) [9, 1] as fixed-point computation over equivalence relations between symbols. Let $S \subseteq C \cup V$ denote a finite set of constants and variables. The *equivalence closure* $Cl_{Eq}(R)$ of a binary relation $R \subseteq S^2$ is the smallest equivalence relation (ER) containing R.

Let further E be a finite set of flat equations over S (and arbitrary functions from F). Without loss of generality, we assume that every equation in E contains a function symbol; equations $a \approx b$ between constants or variables can be rewritten to $f() \approx a, f() \approx b$ by introducing a fresh zero-ary function f. The *congruence closure* $CC_E(R)$ of a relation $R \subseteq S^2$ with respect to E is the smallest ER that is consistent with the equations E, and defined as a least fixed-point over binary relations as follows:

$$\begin{aligned} CC^1_E(R) &= Cl_{Eq}\big(R \cup \{(b, b') \mid \exists\, f(\bar{a}) \approx b, f(\bar{a}') \approx b' \in E \text{ with } (\bar{a}, \bar{a}') \in R\}\big) \\ CC_E(R) &= \mu X \subseteq S^2.\ CC^1_E(R \cup X) \end{aligned}$$

where we write $(\bar{a}, \bar{a}') \in R$ for the inclusion $\{(a_1, a'_1), (a_2, a'_2), \ldots, (a_n, a'_n)\} \subseteq R$, provided $\bar{a} = (a_1, \ldots, a_n)$ and $\bar{a}' = (a'_1, \ldots, a'_n)$.

2.2 The Bounded Rigid *E*-Unification Problem

Bounded rigid E-unification is a restriction of rigid E-unification in the sense that solutions are required to be atomic substitutions s.t. variables are only mapped to smaller atomic terms according to some given partial ordering $\preceq$. This order takes over the role of an occurs-check of regular unification.

Definition 2 (BREU). *A* bounded rigid E-unification (BREU) problem *is a triple $U = (\preceq, E, e)$, with $\preceq$ being a partial order over atomic terms s.t. for all variables X the set $\{s \mid s \preceq X\}$ is finite; E is a finite set of flat formulae; and $e = s \approx t$ is an equation between atomic terms (the target equation). An atomic substitution σ is called a* bounded rigid E-unifier *of s and t if $s\sigma \leftrightarrow^*_{E\sigma} t\sigma$ and $X\sigma \preceq X$ for all variables X.*

Definition 3 (Simultaneous BREU). *A* simultaneous bounded rigid E-unification problem *is a pair $(\preceq, (E_i, e_i)_{i=1}^n)$ s.t. each triple $(\preceq, E_i, e_i)$ is a bounded rigid* E*-unification problem. A substitution σ is a* simultaneous bounded rigid E-unifier *if it is a bounded rigid* E*-unifier for each problem $(\preceq, E_i, e_i)$.*

A solution to a simultaneous BREU problem can be used in a calculus to close all branches in a proof tree. While SREU is undecidable in the general case, simultaneous BREU is decidable, in fact it is NP-complete [2]; the existence of bounded rigid E-unifiers can be decided in non-deterministic polynomial time, since it can be verified in polynomial time that a substition σ is a solution of a (possibly simultaneous) BREU problem. Hardness follows from the fact that propositional satisfiability can be reduced to BREU. Also, a number of generalisations are possible, but can be reduced to BREU as in Def. 2.

Example 4. We revisit the example introduced in Sect. 1.1, which can be captured as the following simultaneous BREU problem $(\preceq, \{(E_1, e_1), (E_2, e_2)\})$:

$$E_1 = E \cup \{a \approx b\}, \quad e_1 = o_5 \approx o_6, \qquad E_2 = E \cup \{c \approx d\}, \quad e_2 = o_7 \approx o_8,$$
$$E = \left\{ \begin{array}{l} f(a) \approx o_1, f(b) \approx o_2, f(c) \approx o_3, f(d) \approx o_4, \\ g(X, U, V) \approx o_5, g(Y, o_3, o_4) \approx o_6, g(U, X, Y) \approx o_7, g(V, o_1, o_2) \approx o_8 \end{array} \right\}$$

with $a \prec b \prec c \prec d \prec o_1 \prec o_2 \prec o_3 \prec o_4 \prec X \prec Y \prec U \prec V \prec o_5 \prec o_6 \prec o_7 \prec o_8$.

A unifier to this problem is sufficient to close all goals of the tree up to equational reasoning; one solution is $\sigma = \{X \mapsto o_1, Y \mapsto o_2, U \mapsto o_3, V \mapsto o_4\}$.

```
Input: BREU problem $B = (\preceq, E, s \approx t)$
1: while candidates remains do
2:     $\sigma \leftarrow$ new candidate                        // GUESSING
3:     $ER \leftarrow CC_E\{(X, X\sigma) \mid X \in S \cap V\}$     // CONGRUENCE CLOSURE
4:     if $(s,t) \in ER$ then                               // VERIFYING
5:         return $\sigma$
6:     end if
7: end while
8: return UNSAT
```

Algorithm 1: Generic search procedure for BREU

3 Solving Bounded Rigid E-Unification

Suppose $B = (\preceq, E, e)$ is a BREU problem, and $S \subseteq V \cup C$ the set of all atomic terms occurring in B ("relevant terms"). On a high level, our procedures for solving BREU problems consist of three steps: GUESSING a candidate substitution; using CONGRUENCE CLOSURE to calculate the corresponding equivalence relation; and VERIFYING that the target equation is satisfied by this relation (see Alg. 1). This schema derives from the basic observation that $s\sigma \leftrightarrow^*_{E\sigma} t\sigma$ if and only if $(s,t) \in CC_E\{(X, X\sigma) \mid X \in S \cap V\}$, provided that σ is an idempotent substitution [11]. Since an E-unifier σ with $X\sigma \preceq X$ for all $X \in V$ can be normalised to an idempotent E-unifier, search can be restricted to the latter.

This paper introduces two different methods of performing these steps; an *eager encoding* of the problem into SAT that encodes the entire procedure as a SAT-problem, and a *lazy encoding* that uses SAT to generate candidate solutions. Common to both methods is the representation of the candidate substitution.

3.1 Candidate representation

We introduce a bijection $Ind : S \to \{1, \ldots, |S|\}$, s.t. for each $s, t \in S$ we have $s \preceq t \Rightarrow Ind(s) \leq Ind(t)$; the mapping Ind will be used for the remainder of the paper. We also introduce a pseudo-integer variable[1] v_s for each $s \in S$, together with a SAT-constraint restricting the domains:

$$\bigwedge_{c \in S \cap C} v_c = Ind(c) \;\wedge \bigwedge_{X \in S \cap V} \bigvee_{\substack{t \in S \\ t \preceq X}} \big(v_X = Ind(t) \wedge v_t = Ind(t)\big) \qquad \text{(SAT DOMAIN)}$$

Any idempotent substitution σ satisfying $X\sigma \preceq X$ for the variables $X \in V$ (as in Def. 2) can be represented by $v_X = Ind(X\sigma)$, and thus gives rise to a SAT model of the domain constraint; and vice versa. A search procedure over the models is thus sufficient for solving the GUESSING step of Alg. 1. The SAT DOMAIN constraint will be used in both methods presented in this paper.

[1] A pseudo-integer variable is a bit-wise representation of an integer in the range $\{1, \ldots, n\}$ by introducing $\lceil \log n \rceil$ Boolean variables.

4 Eager Encoding of Bounded Rigid *E*-Unification

In this section we describe how to eagerly encode a (simultaneous) BREU problem into SAT based on the procedure shown in Alg. 1. We note that a fairly intricate encoding is necessary to accommodate the combination of variables, constants, and congruence reasoning. For instance, the classical Ackermann reduction can be used to encode congruence closure and constants, but is not applicable in the presence of both variables and constants.

4.1 Congruence Tables

A *congruence table* is a table where each column represents a union-find data structure in a step of the congruence closure procedure, and each row corresponds to an atomic term, the "representative" for each step. The *initial column* is defined by a substitution while every *internal column* is constrained by its previous column modulo the given set of equations. From the *final column* of the table, an equivalence relation, equal to the congruence closure of the given substitution modulo the given equations, can be extracted.

Definition 5. *Suppose E is a set of flat equations, each containing exactly one function symbol, and σ is a substitution s.t. $X\sigma \preceq X$ for all $X \in V$. As before, let $S = \{t_1, \dots, t_m\} \subseteq C \cup V$ be the relevant terms, and $\mathit{Ind}(t_i) = i$ ($i \in \{1, \dots, m\}$).*

Then a congruence table *T of size n for E and σ is a list of column vectors $[\bar{c}_1, \dots \bar{c}_n]$, with $\bar{c}_i \in \{1, \dots, |S|\}^m$, where $\bar{c}_1 = (\mathit{Ind}(t_1\sigma), \dots, \mathit{Ind}(t_m\sigma))$ and for each pair of consecutive vectors $\bar{c}_i$ and $\bar{c}_{i+1}$ and each $j \in \{1, \dots, m\}$:*

1. *if $\bar{c}_i(j)$[2] $\neq j$ then $\bar{c}_{i+1}(j) = \bar{c}_{i+1}(\bar{c}_i(j))$.*
2. *if $\bar{c}_i(j) = j$ then:*
   - *(a) $\bar{c}_{i+1}(j) = \bar{c}_{i+1}(k)$ if $k < j$, and there are equations $f(a_1, \dots, a_l) \approx b$, $f(a'_1, \dots, a'_l) \approx b' \in E$ s.t. $\bar{c}_i(\mathit{Ind}(b)) = j$ and $\bar{c}_i(\mathit{Ind}(b')) = k$, and furthermore $\bar{c}_i(\mathit{Ind}(a_h)) = \bar{c}_i(\mathit{Ind}(a'_h))$ for all $h \in \{1, \dots, l\}$.*
   - *(b) $\bar{c}_{i+1}(j) = j$ if no such pair of equations exists.*

To illustrate the definition, observe first that all entries of the first vector point upwards, i.e., $\bar{c}_1(j) \leq j$ for $j \in \{1, \dots, m\}$ (due to the definition of *Ind* in Sect. 3.1), and define a union-find forest. The rules relating consecutive vectors (union-find data structures) to each other in Def. 5 correspond to three different cases: (1) defines path shortening, stating that each term can point directly to its representative term; (2a) states that if the arguments of two function applications are equal, the results must also be equal, and enables merging of the two equivalence classes s.t. the new representative is the smaller term; and (2b) states that if no such merging is possible, a term retains its identity value. All definitions are acyclic because the property $\bar{c}_i(j) \leq j$ is preserved in all columns i (see Lem. 8 below).

[2] We write $\bar{c}(j)$ for the jth component of a vector $\bar{c}$.

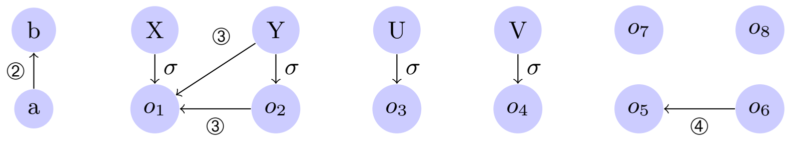

Fig. 1. Equivalence classes of different columns of Table 1

Example 6. Consider the simultaneous BREU problem and unifier σ introduced in Example 4. Table 1 shows a complete congruence table of size 4 for E_1 (the left branch) and σ; for sake of presentation, the table contains symbols t rather than their index $Ind(t)$, and in each column bold font indicates modified entries. The represented union-find forests are shown in Fig. 1, in which each edge is annotated with number of the column in which the edge was introduced. We can see that the fourth column defines an equivalence relation partitioning ER of the set of relevant terms S into seven sets. More importantly, under this equivalence relation the two terms in the target equation $e_1 = o_5 \approx_{ER} o_6$ are equal, implying that the substitution is a unifier to this sub-problem.

Definition 7. *A congruence table $T = [\bar{c}_1, \dots, \bar{c}_n]$ of size n is* complete *if for every table $T' = [\bar{c}'_1, \dots \bar{c}'_{n+1}]$ of size $n+1$, if $\bar{c}_1 = \bar{c}'_1, \dots, \bar{c}_n = \bar{c}'_n$ then $c'_{n+1} = c'_n$.*

Intuitively, a congruence table T is complete, if every additional column added would be identical to the last one.

Lemma 8. *For every congruence table $T = [\bar{c}_1, \dots, \bar{c}_n]$ of size n $\forall i \in \{1, \dots, n-1\}. \forall j \in \{1, \dots, |\bar{c}_i|\}. \bar{c}_{i+1}(j) \leq \bar{c}_i(j)$.*

Lem. 8 states that when observing a certain index of vectors of a congruence table, e.g., $\bar{c}_1(2), \bar{c}_2(2), \dots,$ the values are non-increasing. Therefore, given a set of relevant terms S, there is an upper bound b s.t. all congruence tables, with vectors of length $|S|$, with size $n \geq b$ will be complete.

Observe that every vector $\bar{c}$ in a congruence table of size n defines an equivalence relation $ER(\bar{c}) = Cl_{Eq}\{(Ind^{-1}(j), Ind^{-1}(\bar{c}(j))) \mid j \in \{1, \dots, m\}\}$. Furthermore, considering a congruence table T of size n for a set of equations E and a substitution σ, the vectors $\bar{c}_1, \dots \bar{c}_n \in T$ represent intermediate and final step of congruence closure over E and σ. This leads to the following lemma:

S	1	2	3	4
a	a	a	a	a
b	b	$\mathbf{a}$	a	a
o_1	o_1	o_1	o_1	o_1
o_2	o_2	o_2	$\mathbf{o_1}$	o_1
o_3	o_3	o_3	o_3	o_3
o_4	o_4	o_4	o_4	o_4
X	$\mathbf{o_1}$	o_1	o_1	o_1
Y	$\mathbf{o_2}$	o_2	$\mathbf{o_1}$	o_1
U	$\mathbf{o_3}$	o_3	o_3	o_3
V	$\mathbf{o_4}$	o_4	o_4	o_4
o_5	o_5	o_5	o_5	o_5
o_6	o_6	o_6	o_6	$\mathbf{o_5}$
o_7	o_7	o_7	o_7	o_7
o_8	o_8	o_8	o_8	o_8

Table 1.

Lemma 9. *Given a complete congruence table T of size n for equations E and substitution σ, it holds that $ER(\bar{c}_n) = CC_E\{(t, t\sigma) \mid t \in S\}$.*

If a BREU problem $B = (\preceq, E, s \approx t)$ has an E-unifier σ, then $(s,t) \in CC_E\{(t', t'\sigma) \mid t' \in S\}$. Therefore, with Lem. 8 and Lem. 9, it is only necessary to consider the congruence tables of a large enough size for every substitution to find a solution, and if none of them represents a solving substitution, the given BREU problem is unsatisfiable. This leads to the construction of a SAT model that encodes all possible congruence table of a certain size. However, this upper bound will in general be very pessimistic, so we introduce an iterative procedure that replaces this upper bound by checking an incompletion constraint.

4.2 Modeling Congruence Tables using SAT

In the remainder of this section we present the variables (the *congruence matrix* and the *active congruence pairs*) as well as the constraints introduced to model congruence tables for a given BREU problem $B = (\preceq, E, e)$ using SAT.

Congruence matrix. The *congruence matrix* $M \in \{1, \ldots, m\}^{m \times n}$ is a matrix of pseudo-integer variables with m rows and n columns, corresponding to the vectors $[\bar{c}_1, \ldots \bar{c}_n]$ in Def. 5. We write M^i_j for the cell in row j and column i. Intuitively, the matrix represents congruence tables of size n for a set of relevant symbols S with $|S| = m$, and cell M^i_j represents the entry $\bar{c}_i(j)$.

Active congruence pairs. The set of *congruence pairs* is the set $CP = \{(f(\bar{a}) \approx b, f(\bar{a}') \approx b') \in E^2\}$. For each column $i > 1$ in the congruence matrix, there is also a set $\{v^i_{cp} \mid cp \in CP\}$ of auxiliary Boolean variables that indicate the *active* congruence pairs $cp = (f(a_1, \ldots, a_k) \approx b, f(a'_1, \ldots, a'_k) \approx b')$, constrained by:

$$v^i_{cp} \Leftrightarrow M^{i-1}_{Ind(a_1)} = M^{i-1}_{Ind(a'_1)} \wedge \cdots \wedge M^{i-1}_{Ind(a_k)} = M^{i-1}_{Ind(a'_k)} \wedge M^{i-1}_{Ind(b)} > M^{i-1}_{Ind(b')} \qquad \text{(Table CP)}$$

Intuitively, if some v^i_{cp} is true, the congruence pair cp represents two equations in which the arguments are equal in the equivalence relation of column $i - 1$, but the results are different.

Initial column. In the initial column, we constrain each cell M^1_j to be consistent with the variables v_s introduced in Sect. 3.1 to represent solution candidates:

$$\bigwedge_{t \in S} M^1_{Ind(t)} = v_t \qquad \text{(Table Init)}$$

Internal column. In the internal columns with index $i > 1$, each cell must obey the following constraints, for every $j \in \{1, \ldots, m\}$:

$$\bigvee_{k \in \{1,\ldots,j-1\}} (M^{i-1}_j = k \wedge M^i_j = M^i_k) \vee \qquad \text{(Table Int)}$$

$$M^{i-1}_j = j \wedge \begin{pmatrix} \bigwedge_{cp \in CP} (\neg v^i_{cp} \vee M^{i-1}_{Ind(b)} \neq j) \wedge M^i_j = j \\ \vee \\ \bigvee_{cp \in CP} \big(v^i_{cp} \wedge M^{i-1}_{Ind(b)} = j \wedge \\ \qquad \bigvee_{k \in \{1,\ldots,j-1\}} (M^{i-1}_{Ind(b')} = k \wedge M^i_j = M^i_k)\big) \end{pmatrix}$$

with $cp = (f(\bar{a}) \approx b, f(\bar{a}') \approx b')$. The topmost constraint models condition (1) while the bottom constraint models condition (2) in Def. 5.

```
Input: BREU problem $B = (\preceq, E, s \approx t)$
 1: Add initial table constraint (Sat Domain, Table CP, Init, Int, Goal)
 2: while $\neg solver.isSat()$ do
 3:    Remove goal constraint (Sat Goal)
 4:    Add incompletion constraint (Table Incomp)
 5:    if $\neg solver.isSat()$ then
 6:       return UNSAT
 7:    else
 8:       Remove incompletion constraint (Table Incomp)
 9:       Add internal column and goal constraints (Table Int, Goal)
10:    end if
11: end while
12: return SAT
```

Algorithm 2: Search procedure for the table encoding of a BREU problem

Goal Constraint. The final constraint asserts that the two rows corresponding to the two terms in the target equation contain the same atomic term in the final column.

$$M^n_{Ind(s)} = M^n_{Ind(t)} \qquad \text{(Table Goal)}$$

where the target equations is $e = s \approx t$ and the table has n columns.

4.3 Eager procedure

Our eager procedure (outlined in Alg. 2) creates constraints for an initial table, and then in an iterative fashion adds columns until either a solution is found, or an incompletion constraint is not satisfied. Incompletion constraints make it unnecessary to provide an a-priori upper bound on the size of constructed tables, and instead check whether some congruence pair can be used to merge further equivalence classes in the last column:

$$\bigvee_{cp \in CP} v^{n+1}_{cp} \qquad \text{(Table Incomp)}$$

To handle a simultaneous BREU problem $B = (\preceq, (E_i, e_i)^n_{i=1})$, one table is created for each sub-problem $(\preceq, E_i, e_i)$, s.t. the variables x_t are shared. However, for many simultaneous BREU problems only a few of sub-problems are required to prove unsatisfiability. Therefore we use an iterative approach, where initially there is only a table for the first sub-problem. Once the constraints of the first table could be satisfied, the encoding is extended in an iterative fashion with tables for the other sub-problems, until either all tables are satisfied, or a subset of complete but unsatisfiable tables has been found.

5 Complemented Congruence Closure

The congruence closure algorithm (Sect. 2) efficiently decides entailment between ground equations, and can therefore be used to check (in polynomial time) whether a given substitution σ is a solution to a BREU problem: σ translates to the equivalence relation $\{(a, b) \in S^2 \mid a\sigma = b\sigma\}$ over the symbols $S \subseteq C \cup V$ occurring in the problem, and can be completed to the smallest ER solving the BREU equations via CC.

As main building block for the lazy BREU algorithm introduced in the next section, we defined a generalised version of CC that can be applied to whole *sets* of relations over S, in a manner similar to abstract interpretation (the new algorithm can indeed be identified as an abstract domain for CC, within the framework of abstract interpretation, but the details are beyond the scope of this paper). This notion of *complemented congruence closure* (CCC) can also be used as an optimisation for the SAT-based algorithm in Sect. 4, since it can often quickly rule out the existence of solutions to a BREU problem (Example 12).

CCC reasons about *disequalities* that are preserved by CC: while CC is defined as a least fixed-point over relations $R \subseteq S^2$ representing equalities between symbols (constants or variables), CCC corresponds to the computation of greatest fixed-points over relations $D \subseteq S^2$ representing disequalities between symbols. The definition of CCC is similar in shape to the one of CC in Sect. 2.1; as before, we assume that E is a finite set of flat equations over S in which each equation contains exactly one function symbol.

$$\begin{aligned} C_E^{3,1}(D) &= \left\{ (c, c') \in D \mid \begin{array}{l} c \neq c', \text{ and for all } f(\bar{a}) \approx b, f(\bar{a}') \approx b' \in E \\ \text{it holds that } D \cap \mathit{Cl}_{Eq}\{(\bar{a}, \bar{a}'), (b, c), (b', c')\} \neq \emptyset \end{array} \right\} \\ C_E^3(D) &= \nu X \subseteq S^2.\ C_E^{3,1}(D \cap X) \end{aligned}$$

The one-step function $C_E^{3,1}$ removes all pairs (c, c') (representing disequalities $c \not\approx c'$) from the relation D that can no longer be maintained, i.e., if there are equations $f(\bar{a}) \approx b$ and $f(\bar{a}') \approx b'$ s.t. in some ER (consistent with the disequalities D) it is the case that $\bar{a} \approx \bar{a}'$, $b \approx c$, and $b' \approx c'$. This criterion is expressed by checking whether the equivalence closure $\mathit{Cl}_{Eq}\{(\bar{a}, \bar{a}'), (b, c), (b', c')\}$ has some elements in common with the relation D representing assumed disequalities. The function $C_E^{3,1}$ is clearly monotonic, and can therefore be used to define C_E^3 as a greatest fixed-point over the complete lattice of binary relations; C_E^3 itself is then also monotonic.

5.1 Properties of Complemented Congruence Closure

In this and later sections, we write $R^C = S^2 \setminus R$ for the complement of a relation over S. Most importantly, we can show that CC and CCC yield the same result when starting from equivalence relations, illustrating that CCC is a strict generalisation of CC:

Theorem 10. *Suppose $R \subseteq S^2$ is an ER. Then $\mathit{CC}_E(R)^C = C_E^3(R^C)$.*

For arbitrary relations R, congruence closure $\mathit{CC}_E(R)$ will be an ER, whereas the result $C_E^3(R^C)^C$ in general is not; consider in particular the case $E = \emptyset$, in which CC_E will not have any effect beyond removing pairs (c, c) from a relation. This implies that the assumption of R being an ER is essential in the theorem.

Sets $C^3_E(D)$ for relations D whose complement is not an ER can be used to approximate the effect of CC, and in particular summarise the effect of applying CC to whole families of relations:

Corollary 11. *Suppose $R \subseteq S^2$ is an ER, and $D \subseteq S^2$ a relation s.t. $R \cap D = \emptyset$. Then $CC_E(R) \cap C^3_E(D) = \emptyset$.*

Example 12. Consider $S = \{c, d, e, X\}$, equations $E = \{f(X) \approx X, f(c) \approx d\}$, and the equivalence relation $R = Cl_{Eq}\{(X, c)\}$ that identifies X and c and keeps the other symbols distinct. CC on this input will also identify X and d, and thus c and d, but keep e in a separate class: $CC_E(R) = Cl_{Eq}\{(X, c), (X, d)\}$.

The complement is $R^C = \{(c, d), (d, e), (c, e), (X, d), (X, e)\}^{\leftrightarrow}$, where we write $A^{\leftrightarrow} = A \cup A^{-1}$ for the symmetric closure of a relation. CCC on R^C will remove (X, d) from the relation, since $Cl_{Eq}\{(X, c), (X, X), (d, d)\}$ is disjoint from R^C, and similarly (c, d): $C^3_E(R^C) = \{(d, e), (c, e), (X, e)\}^{\leftrightarrow} = CC_E(R)^C$.

Consider now the BREU problem $B = (\preceq, E, c \approx e)$ with $c \prec d \prec e \prec X$. Note that every substitution σ with $X\sigma \preceq X$ preserves the disequalities

$$D \;=\; \{(c, d), (d, e), (c, e)\}^{\leftrightarrow} \;=\; \bigcap_{\substack{\sigma \text{ a substitution} \\ \forall X \in V.\ X\sigma \preceq X}} \{(a, b) \in S^2 \mid a\sigma \neq b\sigma\}.$$

As before, CCC will remove (c, d) from D; but CCC will keep (c, e), because both $Cl_{Eq}\{(X, c), (X, c), (d, e)\}$ and $Cl_{Eq}\{(X, c), (X, e), (d, c)\}$ overlap with D, and similarly (d, e): $C^3_E(D) = \{(d, e), (c, e)\}^{\leftrightarrow}$. This shows that c and e are not E-unifiable, and neither are d and e.

6 Lazily Solving Bounded Rigid *E*-Unification

When dealing with large simultaneous BREU problems, e.g., containing many parallel problems as well as many equations, just constructing a monolithic SAT model (possibly containing much redundancy) as in Sect. 4 can be time-consuming, even if the subsequent solving might be fast. Our second algorithm for solving BREU problems works in the style of *lazy* SMT solving: starting from a compact SAT encoding that coarsely over-approximates the BREU problem, additional constraints are successively added, until eventually a correct E-unifier is derived, or the encoding becomes unsatisfiable. Following Alg. 1, the overall idea is to repeatedly generate candidate solutions σ, check whether the candidate is a genuine solution, and otherwise generate a *blocking constraint* that excludes (at least) this solution from the search space.

Overall procedure. Consider a simultaneous BREU problem $(\preceq, (E_i, e_i)_{i=1}^n)$. The overall procedure is shown in Alg. 3, and based on the three steps described in Sect. 3, but directly solving simultaneous BREU problems. The algorithm uses an underlying *solver* process for reasoning incrementally about the SAT encoding. The Guessing step is implemented using the Sat Domain constraints from Sect. 3.1 (line 1). When a candidate solution σ has been found, congruence closure is used to verify that σ solves each sub-problem $(\preceq, E_i, e_i)$ (line 4), executing the Congruence Closure and Verifying steps in Alg. 1.

```
1: Add domain constraints (Sat Domain)
2: while $solver.isSat()$ do
3:     $\sigma \leftarrow solver.model$
4:     if $\sigma$ solves all sub-problems then
5:         return $\sigma$
6:     else
7:         Let $(\preceq, E, e)$ be an unsolved sub-problem
8:         $D \leftarrow \{(s,t) \in S^2 \mid s\sigma \neq t\sigma\}$
9:         $D' \leftarrow minimise(D, (\preceq, E, e))$
10:        Add blocking constraint $\bigvee\{v_s = v_t \mid (s,t) \in D'\}$
11:    end if
12: end while
13: return UNSAT
```

Algorithm 3: Lazy search procedure for a simultaneous BREU problem.

```
Input: Disequality set $D$
Input: BREU problem $(\preceq, E, s \approx t)$ with $(s,t) \in C^3_E(D)$
1: Compute set $BaseD$ for $\preceq$                 // by construction, $BaseD \subseteq D$
2: for $dq \in D \setminus BaseD$ do
3:     $D' \leftarrow C^3_E(D \setminus \{dq\})$
4:     if $(s,t) \in C^3_E(D')$ then
5:         $D \leftarrow D' \cup BaseD$
6:     end if
7: end for
8: return $D$
```

Algorithm 4: Minimisation of disequality sets

Blocking constraints. Given a candidate σ that violates $(\preceq, E_i, s_i \approx t_i)$, a *blocking constraint* for σ is a formula ϕ over the solution variables $\{v_t \mid t \in S\}$ introduced in Sect. 3.1 with the property that 1. ϕ evaluates to *false* for the assignment $\{v_t \mapsto Ind(t\sigma) \mid t \in S\}$, and 2. ϕ evaluates to *true* for all genuine E-unifiers σ' and assignments $\{v_t \mapsto Ind(t\sigma') \mid t \in S\}$. In other words, ϕ excludes the incorrect solution σ, but it does not rule out any correct E-unifiers. The most straightforward blocking constraint excludes the incorrect candidate σ:

$$\bigvee_{X \in S \cap V} v_X \neq Ind(X\sigma) \tag{1}$$

This constraint leads to a correct procedure, but is inefficient since it does not generalise from the observed conflict (in SMT terminology), and does not exclude any candidates other than σ. More efficient blocking constraints can be defined by using the concept of *complemented congruence closure.* For this, observe that (1) can equivalently be expressed in terms of disequalities implied by σ:

$$\bigvee_{(s,t) \in D} v_s = v_t, \qquad D = \{(s,t) \in S^2 \mid s\sigma \neq t\sigma\} \tag{2}$$

Candidate σ	(E_1, e_1)	(E_2, e_2)	*Minimised set* D'
1: $X \mapsto X, Y \mapsto Y, U \mapsto U, V \mapsto V$	✗	(✗)	$\{(Y, o_4), (V, o_4)\} \cup \mathit{BaseD}$
2: $X \mapsto X, Y \mapsto Y, U \mapsto U, \boldsymbol{V \mapsto o_4}$	✗	(✗)	$\{(Y, o_4), (U, o_3)\} \cup \mathit{BaseD}$
3: $X \mapsto X, \boldsymbol{Y \mapsto o_4}, U \mapsto U, \boldsymbol{V \mapsto V}$	✗	(✗)	$\{(U, o_4), (V, o_4)\} \cup \mathit{BaseD}$
4: $X \mapsto X, Y \mapsto o_4, U \mapsto U, \boldsymbol{V \mapsto o_4}$	✗	(✗)	$\{(U, o_4), (U, o_3)\} \cup \mathit{BaseD}$
5: $X \mapsto X, Y \mapsto o_4, \boldsymbol{U \mapsto o_3}, V \mapsto o_4$	✗	(✗)	$\{(X, Y), (Y, a), (Y, b), (Y, o_1), (Y, o_2), (U, o_4)\} \cup \mathit{BaseD}$
6: $\boldsymbol{X \mapsto o_4}, Y \mapsto o_4, U \mapsto o_3, V \mapsto o_4$	✓	✗	$\{(X, o_2), (Y, o_2)\} \cup \mathit{BaseD}$
7: $\boldsymbol{X \mapsto o_2, Y \mapsto o_1}, U \mapsto o_3, V \mapsto o_4$	✓	✗	$\{(Y, o_2), (V, o_2)\} \cup \mathit{BaseD}$
8: $\boldsymbol{X \mapsto o_1, Y \mapsto o_2}, U \mapsto o_3, V \mapsto o_4$	✓	✓	—

Table 2. Execution of the lazy algorithm

Indeed, in order to satisfy (1), one of the disequalities in D has to be violated (since $\sigma' \neq \sigma$ implies $X\sigma' = t$ for some variable X and some $t \in S \setminus \{X\sigma\}$); and vice versa, (2) can only be satisfied by substitutions σ' different from σ.

To obtain stronger blocking constraints, we consider subsets of D in (2), but ensure that only constraints are generated that do not exclude E-unifiers of the sub-problem $(\preceq, E_i, s_i \approx t_i)$, and therefore also preserve solutions of the overall problem $(\preceq, (E_i, e_i)_{i=1}^n)$. This is the case for all constraints defined as follows:

$$\bigvee_{(s,t) \in D'} v_s = v_t, \qquad \text{(Lazy BC)}$$

$$\text{where } D' \subseteq \{(s, t) \in S^2 \mid s\sigma \neq t\sigma\} \text{ such that } (s_i, t_i) \in C^3_{E_i}(D').$$

The condition $(s_i, t_i) \in C^3_{E_i}(D')$ expresses that D' is a set of disequalities that prevents s_i and t_i from being unified. Suppose σ' is a solution candidate violating Lazy BC, which by construction implies $R \cap D' = \emptyset$ for $R = \{(s, t) \in S^2 \mid s\sigma' = t\sigma'\}$. By Corollary 11, we then have $CC_{E_i}(R) \cap C^3_{E_i}(D') = \emptyset$, and therefore $(s_i, t_i) \neq CC_{E_i}(R)$, so that σ' cannot be an E-unifier of $(\preceq, E_i, s_i \approx t_i)$.

The constraint Lazy BC is implemented in lines 8–10 in Alg. 3.

Minimisation. Greedy systematic minimisation of disequality sets D is described in Alg. 4, which successively attempts to remove elements dp from D, but preserving $(s, t) \in C^3_E(D)$. Certain disequalities $s\sigma \neq t\sigma$ are known to hold under any substitution σ, and are handled using a special set BaseD and kept in D:

$$\mathit{BaseD} \;=\; \bigcap_{\substack{\sigma \text{ a substitution} \\ \forall X \in V.\ X\sigma \preceq X}} \{(a, b) \in S^2 \mid a\sigma \neq b\sigma\}$$

BaseD can easily be derived from $\preceq$. Elimination of disequalities from BaseD is not helpful, since such disequalities are already implied by the Sat Domain constraint; at the same time, they are useful as input for CCC.

Example 13. We consider again $(\preceq, \{(E_1, e_1), (E_2, e_2)\})$ from Example 4, which is solved by the run of Alg. 3 shown in Table 2. Note that various executions

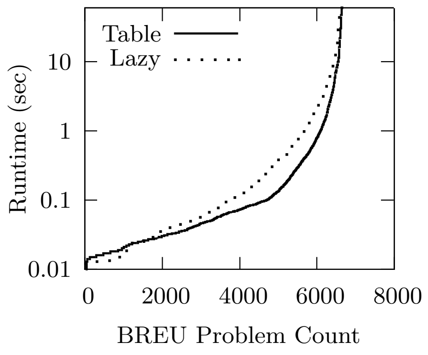

Fig. 2. Cactus plot showing the runtime distribution for the two procedures.

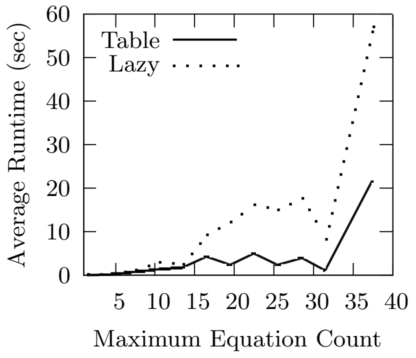

Fig. 3. Runtime dependent on the maximum number of equations in a BREU sub-problem.

	SAT	UNSAT	T/O (SAT)	T/O (UNSAT)
Table	**3769**	**2854**	0	3
Lazy	3727	2845	45	9

Table 3. Comparison of the two BREU procedures. All experiments were done on an AMD Opteron 2220 SE machine, running 64-bit Linux, heap space limited to 1.5GB.

exist, since the sets D' and the candidates σ are not uniquely determined. Sets D' directly translate to blocking constraints, for instance $\{(Y, o_4), (V, o_4)\} \cup$ *BaseD* is encoded as $v_Y = v_{o_4} \vee v_V = v_{o_4} \vee \cdots$. In iterations 1–5, the sub-problem $(\preceq, E_1, e_1)$ is violated, and used to generate a blocking constraints; in 6–7, $(\preceq, E_2, e_2)$ is used. It can be observed that the algorithm is able to derive very concise blocking constraints, and quickly focuses on interesting assignments.

7 Experiments

We implemented both procedures as described in Sect. 4 and Sect. 6 and integrated them into the ePrincess theorem prover (based on [10]) using the calculus presented in [2].[3] The Sat4j solver was used to reason about the propositional encoding used in the procedures. To measure the performance of the two methods, we used randomly selected benchmarks from TPTP v.6.1.0 to generate BREU problems: when constructing a proof for a TPTP problem, ePrincess repeatedly extracts and attempts to solve BREU problems in order to close the constructed proof. ePrincess was instrumented to output and collected those BREU problems, so that altogether 6626 instances were in the end available for benchmarking. Those 6626 BREU problems were then separately processed by the *Table* and *Lazy* procedure, with a timeout of 60s.

[3] Found at `http://user.it.uu.se/~petba168/breu/`

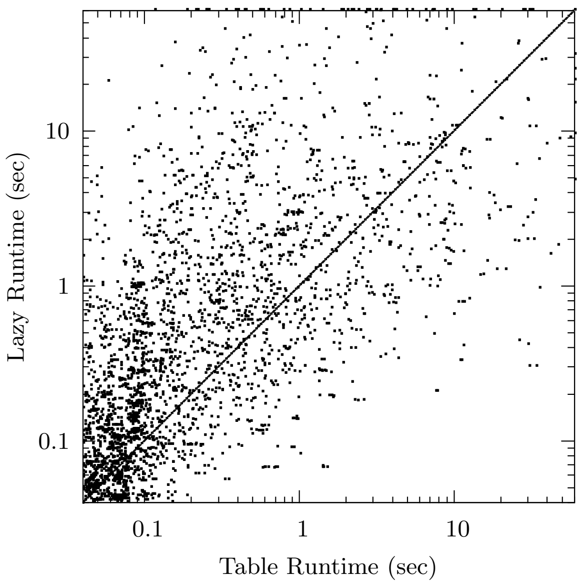

Fig. 4. Runtime comparison of the lazy and table procedures

7.1 Results and Discussion

The two procedures can handle most of the BREU problems generated. Table 3 tells us that the table procedure can solve all but three, while the lazy time-outs on slightly above 50. However, the three problems which the table method could not handle where all solved by the lazy method. The fact that almost all BREU problems could be solved indicates the efficiency of the two BREU procedures, but also that the BREU problems generated by ePrincess are not excessively large (which can be considered a strength of the calculus implemented by ePrincess [2]).

The cactus plot in Fig. 2 shows the distribution of runtime needed by either procedure to solve the BREU problems. It can be observed that more than half of the problems can be solved in less than 0.1s, and most of the problems in less than 1s. Fig. 3 shows that increasing complexity of BREU problems $(\preceq, (E_i, e_i)_{i=1}^n)$, measured in terms of the maximum number of equations in any BREU sub-problem (E_i, e_i), also leads to increased solving time. The graph illustrates that the lazy procedure is more sensitive to this form of complexity than the table procedure. The high runtime for equation count > 35 corresponds to timeouts. In contrast, we found that neither procedure is very sensitive to the number of sub-problems that a BREU problem consists of.

From Fig. 2 and Fig. 3, it can be seen that the table procedure is on average a bit faster than the lazy procedure. The scatter plot in Fig. 4 gives a more detailed comparison of runtime, and shows that the correlation of runtime of the procedures is in fact quite weak, but there is a slight trend towards shorter runtime of the table method. Note that this is a comparison between procedures for solving BREU problems, for an evaluation of the overall performance of ePrincess on TPTP problems we refer the reader to [2].

On average, the lazy procedure produces 4.3 blocking clauses before finding an E-unifier, or proving that no unifier exists. The major bottleneck of the lazy method lies in the minimisation step of blocking constraints. The procedure spends most of its time in this part, and could be improved by creating a more efficient algorithm for CCC. For the table method, most of the runtime is spent in SAT solving, in particular in calls concluding with UNSAT.

8 Conclusion

In this paper we have presented two different procedures for solving the BREU problem. Both of them are shown to be efficient and usable in an automated theorem proving environment. Apart from further improving the proposed procedures, in future work we plan to consider the combination of BREU with other theories, in particular arithmetic.

Acknowledgements We thank the anonymous referees for helpful feedback.

References

1. Bachmair, L., Tiwari, A., Vigneron, L.: Abstract congruence closure. J. Autom. Reasoning 31(2), 129–168 (2003)
2. Backeman, P., Rümmer, P.: Theorem proving with bounded rigid E-Unification. In: CADE. LNCS, Springer (2015), to appear
3. Degtyarev, A., Voronkov, A.: What you always wanted to know about rigid E-Unification. J. Autom. Reasoning 20(1), 47–80 (1998)
4. Degtyarev, A., Voronkov, A.: Equality reasoning in sequent-based calculi. In: Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press (2001)
5. Degtyarev, A., Voronkov, A.: Kanger's Choices in Automated Reasoning. Springer (2001)
6. Fitting, M.C.: First-Order Logic and Automated Theorem Proving. Graduate Texts in Computer Science, Springer-Verlag, Berlin, 2nd edn. (1996)
7. Gallier, J.H., Raatz, S., Snyder, W.: Theorem proving using rigid e-unification equational matings. In: LICS. pp. 338–346. IEEE Computer Society (1987)
8. Kanger, S.: A simplified proof method for elementary logic. In: Siekmann, J., Wrightson, G. (eds.) Automation of Reasoning 1: Classical Papers on Computational Logic 1957-1966, pp. 364–371. Springer, Berlin, Heidelberg (1983), originally appeared in 1963
9. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM 27, 356–364 (April 1980)
10. Rümmer, P.: A constraint sequent calculus for first-order logic with linear integer arithmetic. In: LPAR. LNCS, Springer (2008)
11. Tiwari, A., Bachmair, L., Rueß, H.: Rigid E-Unification revisited. In: CADE. pp. 220–234. CADE-17, Springer-Verlag, London, UK, UK (2000)