

# Fast Interprocedural Linear Two Variable Equalities

Andrea Flexeder<sup>1</sup>, Markus Müller-Olm<sup>2</sup>, Michael Petter<sup>1</sup> and Helmut Seidl<sup>1</sup>

<sup>1</sup> Technische Universität München, Boltzmannstrasse 3, 85748 Garching, Germany,  
{seidl, flexeder, petter}@in.tum.de,

<sup>2</sup> Westf. Wilhelms-Universität Münster, Mathematik und Informatik, 48149 Münster, Germany  
mmo@math.uni-muenster.de

**Abstract.** In this paper we provide an interprocedural analysis of linear two variable equalities. In contrast to the corresponding approach based on full linear algebra, the novel algorithm saves a factor of  $k^4$  in the worst-case complexity, where  $k$  is the number of variables. We also indicate how the practical runtime can be further reduced significantly. The analysis can be applied, e.g., for register coalescing, for identifying local variables and for an analysis of array index expressions.

## 1 Introduction

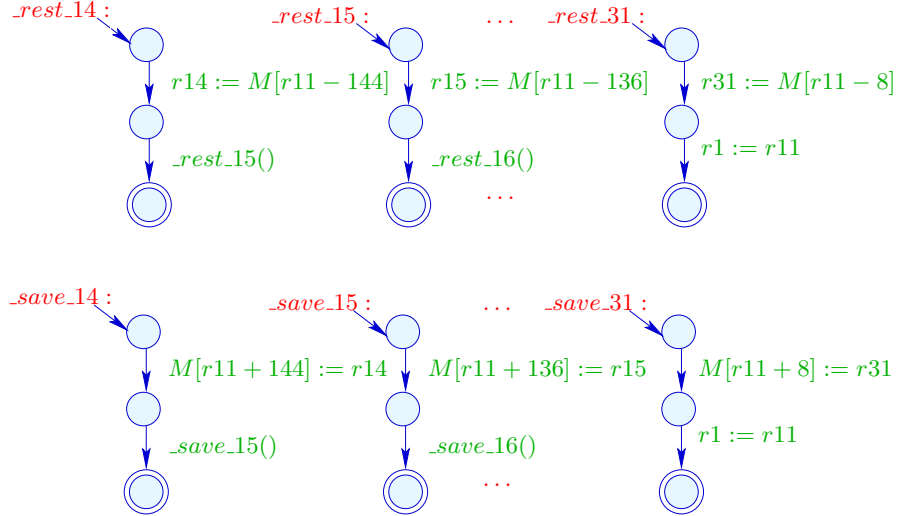
The key task when realising interprocedural analyses along the lines of the functional approach of Sharir/Pnueli [13,6], is to determine the *summary functions* for procedures which describe their effects on the abstract program state before the call. Given a complete lattice  $\mathbb{D}$  for the abstract program states, the summary functions are taken from the set of monotonic or (if we are lucky) distributive functions  $\mathbb{D} \rightarrow \mathbb{D}$ . This set is often large (if not infinite), rendering it a non-trivial task to identify a representation for summary functions which efficiently supports basic operations such as function composition or function application to values of  $\mathbb{D}$ . Examples for such efficient representations are pairs of sets in case of gen/kill bit-vector problems [5] or vector spaces of matrices in case of affine equality analyses [7].

In [10] Müller-Olm and Seidl present a fast interprocedural analysis for variable equalities that can be used e.g. for register coalescing [3]. Here, we extend this analysis first to variable differences and in a second step to *linear two variable equalities*. This extended analysis has essentially the same complexity – while providing much stronger program invariants that enable new applications. It allows us, for instance, to verify calling conventions for the stack pointer or to identify accesses to local memory locations when analysing assembler code.

It seems to be natural to assume that the stack levels before and after a procedure call are the same. This invariant, however, may be violated for code generated by common compilers such as the GNU C Compiler or the Greenhills Compiler which may rely on calls to auxiliary procedures for saving and restoring of local registers. With the aid of our analysis of variable differences we can nevertheless verify the assumption for non-auxiliary functions.

*Example 1.* According to the calling convention for the PowerPC architecture as specified, e.g., in [15], callee-save registers must be saved before and restored after a procedure call. The corresponding instructions for saving and restoring the registers  $r14$  up to  $r31$  can be organised into chains of system-level routines  $\_save\_X$  and  $\_rest\_X$  which save and restore the registers  $rX$  up to  $r31$ . The control flow graphs for these routines are shown in figure 1. In this control flow representation, memory access instructions are represented by assignments of the form  $ri := M[t]$  for memory read access instructions or  $M[t] := ri$  for memory write access instructions, respectively. Here,  $t$  is a linear expression of the form  $ri, ri + c, ri - c, ri - rj, ri + rj$ , with  $ri, rj$  processor registers and  $c$  a constant.

In our example, the auxiliary register  $r11$  holds the stack level before saving the locals extended by the required memory for saving the local registers. It is only in the procedure  $\_save\_31$  that the stack pointer register  $r1$  is set to the value of  $r11$ . Similarly for restoring the local registers, the stack pointer  $r1$  is not updated until the call of procedure  $\_rest\_31$ . Accordingly, the calling-conventions are violated both by the



**Fig. 1.** Calling sequence for saving and restoring registers.

procedures  $\_save\_X$  and the procedures  $\_rest\_X$ . Taking into account the two-variable equalities which hold between  $r11$  and the stack pointer  $r1$ , our interprocedural analysis nonetheless reveals that the stack pointer invariant holds for all other procedure calls in the program.  $\square$

In this paper we present an analysis which infers all valid variable differences. Moreover, we indicate how this analysis can be generalised further to an algorithm which infers all interprocedurally valid linear two variable equalities, i.e., all valid equalities of the form  $x_i \doteq b$  or  $x_i \doteq ax_j + b$  for constants  $a, b$ .

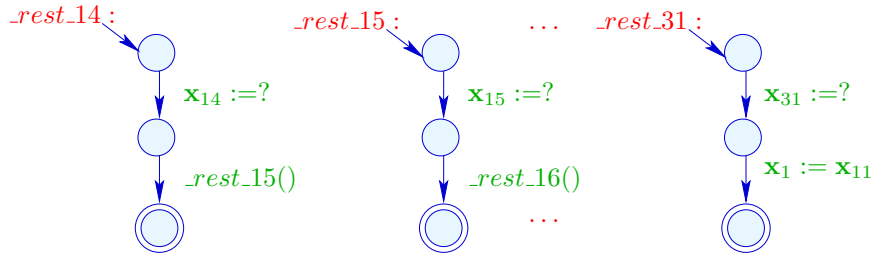
**Related Work** Certain variable equalities can be determined as a particular case of a generalised analysis of availability of expressions called *value numbering* [1]. Originally, this analysis tracks for basic blocks the symbolic expressions representing the values of the variables assigned to. Unlike in our analysis, operator symbols are left uninterpreted in value numbering. The inferred equalities between variables and terms therefore are *Herbrand* equalities. Later, the idea of inferring Herbrand equalities was generalised to arbitrary control-flow graphs by Steffen, Knoop, and Rütting [14]. Only recently, this problem has attracted fresh attention. In [4], Gulwani and Necula show that the original algorithm of Steffen, Knoop and Rütting can be turned into a polynomial time algorithm if one is interested in polynomially sized equalities between variables and terms only. Progress in a different direction was made in [8] and [11] where an analysis for Herbrand equalities that deals with negative guards and side-effect free functions is presented. It is still open whether full interprocedural analysis of Herbrand equalities is possible. However, when only assignments of variables and constants are tracked, the abstract domain can be chosen *finite* and valid equalities can be computed by an exponential-time algorithm. A less naive approach may interpret (or code) the constants as *numbers*. The problem then consists in inferring specific *affine* equalities between variables, as e.g.  $x_1 \doteq x_2 + 8$ . Therefore, in principle, the precise interprocedural analyses of [7,9] are applicable. These analyses use linear algebra for computing vector spaces of affine equalities and have a worst-case runtime of  $\mathcal{O}(n \cdot k^8)$  where  $n$  is the program size and  $k$  the number of program variables. In a former paper [10] an analysis was proposed that determines variable-variable together with variable-constant equalities, i.e. equalities of the form  $x_i \doteq x_j$  and  $x_i \doteq b$ . This algorithm improves the complexity bound by reducing the exponent to 4 in the worst case. In this paper we extend the approach from [10] to infer all interprocedurally valid variable differences and all interprocedurally valid linear two variable equalities maintaining the same complexity bound. Drawing a comparison to the best known upper bound for interprocedural copy constant propagation [5], where no equalities between variables (let alone linear two variable equality relations) are tracked and to the interprocedural linear constant propagation introduced by Reps and Sagiv in [12], the resulting bound is worse only by one factor  $k$ . In [12] the authors consider programs with instructions of the form  $x_i := a \cdot x_j + b$  and  $x_i := c$ . Similar to copy-constant propagation, their analysis runs in time  $\mathcal{O}(n \cdot k^3)$ , but only determines constant values of variables.

The remainder of the paper is organised as follows. Section 2 is devoted to the definition of programs and their collecting semantics. Then in section 3, we introduce the complete lattice of consistent equivalence relations that is central for our approach. We discuss basic operations on this lattice and their complexity in section 4. We then describe the interprocedural approach in section 5 by specifying the weakest precondition transformers. Additionally in section 6, we present the concept of introducing logical variables to allow for an effective description of procedure effects. We specify the concept of handling local variables in section 7. There, we introduce a method to further reduce complexity for the representation of procedure effects. Finally, in section 8 we provide an application for our approach when analysing assembly code and then we conclude.

## 2 Programs and their Collecting Semantics

In this section, we introduce our model of programs and define their collecting semantics. The collecting semantics forms the basis for judging soundness and precision of our analyses. We assume that programs are given as a collection of procedures  $q$  where each  $q$  is represented by a finite control flow graph  $G_q$ .

The following figure illustrates the control flow graph for the calling sequence of system functions for restoring the local registers of a procedure from example 1. Note that in our control flow representation memory access instructions are modelled via non-deterministic assignments.



**Fig. 2.** Control flow representation of Fig. 1.

Each control flow graph  $G_q$  consists of:

- a finite set  $N_q$  of *program points* of procedure  $q$ ,
- a finite set  $E_q \subseteq (N_q \times S \times N_q)$  of *edges* annotated with basic statements or procedure calls,
- a unique entry point  $s_q \in N_q$  for procedure  $q$ , and
- a unique exit point  $r_q \in N_q$  for procedure  $q$  and

We assume that there is a designated procedure *main* where program execution always starts.

Let  $\mathbf{X} = \{x_1, \dots, x_k\}$  denote the set of variables the program operates on where  $k$  is the number of variables. For the moment, we assume that all variables are global. Later, we extend our approach to deal with local variables as well. In the following, we assume that the program variables  $\mathbf{X}$  take values from the integral domain  $\mathbb{Z}$ . In order to concentrate on the essentials of the analysis, we consider simplified programs only. So, we assume that conditional branching has already been abstracted to non-deterministic branching. Each edge in the control-flow graphs is either labelled with a call  $q()$  to a procedure  $q$  or with a variable assignment  $s$  of one of the forms  $x_i := ax_j + b$ ,  $x_i := b$ , or  $x_i := ?$  for  $x_i, x_j \in \mathbf{X}$  and  $a, b \in \mathbb{Z}$ . This means that we only treat those assignments precisely where variables receive a linear term of the form  $ax_i + b$ , while more complicated forms of assignments to a variable  $x_i$  are approximated by the non-deterministic assignment  $x_i := ?$ . In this case *any* value may be assigned to the left-hand side  $x_i$ .

A program state can be represented as a vector  $x = (x_1, \dots, x_k) \in \mathbb{Z}^k$  where  $x_i \in \mathbb{Z}$  denotes the value of variable  $x_i$ . The collecting semantics assigns to each program point

the set of states that can occur at this program point in some execution of the program. In order to capture the collecting semantics, we first define the effect  $\llbracket s \rrbracket$  of an assignment  $s$  onto a set  $X \subseteq \mathbb{Z}^k$  of states:

$$\begin{aligned} \llbracket \mathbf{x}_i := ? \rrbracket X &= \{x' \mid \exists x \in X : \forall k \neq i : x'_k = x_k\} \\ \llbracket \mathbf{x}_i := b \rrbracket X &= \{x' \mid \exists x \in X : x'_i = b \wedge \forall k \neq i : x'_k = x_k\} \\ \llbracket \mathbf{x}_i := a\mathbf{x}_j + b \rrbracket X &= \{x' \mid \exists x \in X : x'_i = ax_j + b \wedge \forall k \neq i : x'_k = x_k\} \end{aligned}$$

As a second step, we describe how a procedure  $q$  transforms the set of program states before the procedure call to a set of program states after the procedure call to  $q$ . Following the approach of, e.g., [7,10], we characterise this effect of a procedure  $q$  as the least solution of a constraint system  $S$  over the complete lattice of monotone (even completely distributive) functions  $2^{\mathbb{Z}^k} \rightarrow 2^{\mathbb{Z}^k}$ :

$$\begin{aligned} S[r_q] &\supseteq \text{Id} && r_q \text{ return point of procedure } q \\ S[u] &\supseteq S[v] \circ S[s_q] && (u, q(), v) \text{ a call edge, } s_q \text{ start point of } q \\ S[u] &\supseteq S[v] \circ \llbracket s \rrbracket && (u, s, v) \text{ an assignment edge} \end{aligned}$$

Here,  $\text{Id}$  denotes the identity mapping and  $\circ$  denotes the composition of transformations. The effect of procedure  $q$  is given by the effect accumulated at the entry point  $S[s_q]$  of  $q$ . Then, the set of program states that can occur when reaching program point  $u$  is given by the least solution of the following system of inequations  $C$  over the complete lattice of sets of program states  $2^{\mathbb{Z}^k}$ :

$$\begin{aligned} C[s_{main}] &\supseteq \mathbb{Z}^k \\ C[s_q] &\supseteq C[u] && (u, q(), -) \text{ a call edge} \\ C[v] &\supseteq S[s_q](C[u]) && (u, q(), v) \text{ a call edge} \\ C[v] &\supseteq \llbracket s \rrbracket(C[u]) && (u, s, v) \text{ an assignment edge} \end{aligned}$$

Since the right-hand sides in both constraint systems denote monotonic functions, a unique least solution of these systems of inequations indeed exists.

Let us first consider the case of plain variable differences. This means that we restrict ourselves to invariants consisting of equalities of the forms  $\mathbf{x}_i \doteq b$  or  $\mathbf{x}_i \doteq \mathbf{x}_j + b$  for variables  $\mathbf{x}_i, \mathbf{x}_j$  and constants  $b \in \mathbb{Z}$ . Within our analysis for this case we only consider deterministic assignments of the form  $\mathbf{x}_i := b$  or  $\mathbf{x}_i := \mathbf{x}_j + b$  with  $b \in \mathbb{Z}$  while all other assignments are abstracted to non-deterministic assignments. Later, we will generalise our methods also to programs with assignments  $\mathbf{x}_i := a\mathbf{x}_j + b$  and arbitrary linear two variable equalities.

### 3 Lattice of Conjunctions of Equalities

Our goal is to infer for every program point  $v$ , valid equalities of the form  $\mathbf{x}_i \doteq b$  or  $\mathbf{x}_i \doteq \mathbf{x}_j + b$  with  $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$  and  $b \in \mathbb{Z}$ . We denote the set of all equalities of this form by  $\mathbb{P}(\mathbf{X})$ . Equalities of the form  $\mathbf{x}_i \doteq \mathbf{x}_i$  are called *trivial*. A vector  $x \in \mathbb{Z}^k$  *satisfies* the equality  $\mathbf{x}_i \doteq b$  iff  $x_i = b$ . Likewise,  $x$  satisfies the equality  $\mathbf{x}_i \doteq \mathbf{x}_j + b$  iff  $x_i = x_j + b$ . The vector  $x$  satisfies the conjunction  $E$  of equalities iff  $x$  satisfies every equality in  $E$ . In this case, we write  $x \models E$ . Accordingly, the set  $X \subseteq \mathbb{Z}^k$  satisfies  $E$  (written:  $X \models E$ )

iff  $x \models E$  for all  $x \in X$ . Finally, we call a finite conjunction of equalities  $E$  *satisfiable* iff  $x \models E$  for some vector  $x$ . A conjunction which is not satisfiable is equivalent to **false**. Assume  $E$  is a satisfiable conjunction of equalities or **false**. Then, we say,  $E$  *implies* an equality  $e$  iff  $x \models e$  whenever  $x \models E$ . Likewise,  $E$  implies a conjunction  $E'$  iff  $E$  implies every equality in  $E'$  and we write  $E \implies E'$ . In particular,  $E$  is equivalent to  $E'$  iff  $E \implies E'$  and  $E' \implies E$ . Using these conventions, we define the lattice  $\mathbb{E}(\mathbf{X})$  as the set of equivalence classes of all satisfiable finite conjunctions of equalities from  $\mathbb{P}(\mathbf{X})$  together with the value **false**. The ordering  $\sqsubseteq$  on  $\mathbb{E}(\mathbf{X})$  is given by implication. The top element  $\top$  w.r.t. this ordering is the empty conjunction which corresponds to **true**, while the least element is **false** which we therefore denote also by  $\perp$ .

Any finite satisfiable conjunction  $E$  of equalities together with all (non-trivial) equalities  $e$  implied by  $E$  can be represented by a weighted directed graph  $\mathcal{G}(E)$  on the set  $\mathbf{X} \cup \{\mathbf{0}\}$  of program variables and a particular node  $\mathbf{0}$  representing the hard-wired value 0. An edge from  $\mathbf{x}_i$  to  $\mathbf{0}$  with weight  $b$  represents the equality  $\mathbf{x}_i \doteq b$ . For convenience,  $\mathcal{G}(E)$  then also has an edge from  $\mathbf{0}$  to  $\mathbf{x}_i$  with weight  $-b$ . Furthermore, an edge from  $\mathbf{x}_i$  to  $\mathbf{x}_j$  for  $i, j > 0$  with weight  $b$  represents the equality  $\mathbf{x}_i \doteq \mathbf{x}_j + b$ . By construction, the graph  $\mathcal{G}(E)$  is *symmetric*, i.e., for every edge from  $u$  to  $v$  with edge weight  $b$  there exists an edge from  $v$  to  $u$  with weight  $-b$ . This graph is also *transitive*, i.e., for every pair of edges  $(u, v)$  and  $(v, w)$  with edge weights  $b_1$  and  $b_2$ , respectively, there exists an edge from  $u$  to  $w$  with weight  $b_1 + b_2$ . We conclude that  $\mathcal{G}(E)$  consists of a disjoint union of complete digraphs.

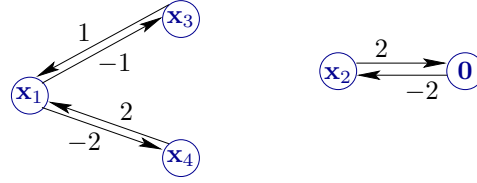
Each maximal connected component  $Q$  within  $\mathcal{G}(E)$  can be identified by a single *reference node* for which we choose  $\mathbf{0}$  if  $Q$  contains  $\mathbf{0}$  and otherwise the variable  $\mathbf{x}_i$  in  $Q$  with the least index  $i$ . For every other variable  $\mathbf{x}_j$  in  $Q$ , it then suffices to record the equality  $\mathbf{x}_j \doteq b$  if  $b$  is the weight of edge  $(\mathbf{x}_j, \mathbf{0})$  or the equality  $\mathbf{x}_j \doteq \mathbf{x}_i + b$  if  $b$  is the weight of the edge  $(\mathbf{x}_j, \mathbf{x}_i)$  and  $\mathbf{x}_i$  is the reference node of the maximal connected component of  $\mathbf{x}_j$ . The conjunction of all these equalities is still equivalent to  $E$ . It has the following syntactical properties:

- (1) If the conjunction has an equality  $\mathbf{x}_j \doteq b$ , then  $\mathbf{x}_j$  does not occur elsewhere in the conjunction.
- (2) If the conjunction has an equality  $\mathbf{x}_j \doteq \mathbf{x}_i + b$ , then  $j > i$  and  $\mathbf{x}_j$  does not occur elsewhere in the conjunction.

A conjunction with these properties is called *normalised*. Note that trivial equalities are omitted in normalised conjunctions.

*Example 2.* Figure 3 illustrates the graphical representation of the normalised conjunction  $E = (\mathbf{x}_2 \doteq 2) \wedge (\mathbf{x}_3 \doteq \mathbf{x}_1 + 1) \wedge (\mathbf{x}_4 \doteq \mathbf{x}_1 + 2)$ .

A normalised conjunction  $E$  of equalities consists of at most  $k$  equalities. Technically, such a conjunction can be represented by an array of size  $k$ . This array is indexed with the variables  $\mathbf{x}_1, \dots, \mathbf{x}_k$  where the entry for  $\mathbf{x}_i$  is given by  $t$ , if  $\mathbf{x}_i \doteq t$  occurs in  $E$  or with  $\mathbf{x}_i$ , if  $\mathbf{x}_i$  does not occur on the left-hand side of an equality in  $E$ . By abuse of notation, we will denote this array by  $E$  as well and write in algorithms  $E(\mathbf{x}_i)$  for the right-hand side  $t$  of the equality  $\mathbf{x}_i \doteq t$  in  $E$  or  $\mathbf{x}_i$  if there is no such equality in  $E$ . In order to specify an update of the right-hand side of a variable  $\mathbf{x}_i$  in  $E$ , we also write  $E(\mathbf{x}_i) \leftarrow t$  where  $t$  denotes the new right-hand side. W.r.t. this array representation, two



**Fig. 3.** Graphical representation of  $E$ .

variables  $\mathbf{x}_i, \mathbf{x}_j$  belong to the same connected component of  $\mathcal{G}(E)$  iff either both  $E(\mathbf{x}_i)$  and  $E(\mathbf{x}_j)$  yield constants, or  $E(\mathbf{x}_i) = \mathbf{x}_h + b_i$  and  $E(\mathbf{x}_j) = \mathbf{x}_h + b_j$  for the same variable  $\mathbf{x}_h$ .

**Lemma 1.** *Assume  $E$  is a finite conjunction of  $r$  equalities. Then, the following holds:*

1. *If  $E$  is satisfiable, then there exists a normalised conjunction  $E'$  equivalent to  $E$ .*
2. *There is an algorithm running in time  $\mathcal{O}(r + k^2)$  which returns the array representation of a normalised conjunction  $E'$  equivalent to  $E$  if it exists — or false if  $E$  is unsatisfiable.*

*Proof.* Assume that  $E = e_1 \sqcap \dots \sqcap e_r$  for equalities  $e_i$ . For computing the array representation of the normalised conjunction  $E$ , we start with an array  $E_0$  for the empty conjunction and then for  $i = 1, \dots, r$  determine a representation for  $E_{i-1} \wedge e_i$ . In order to implement the inductive step, assume that we are given an array representation for a normalised conjunction  $E'$  together with an equality  $e$ . We distinguish two cases.

*Case 1.*  $e$  is of the form  $\mathbf{x}_i \doteq b$  for some  $b \in \mathbb{Z}$ . First assume that  $E(\mathbf{x}_i) = b'$  for some constant  $b'$ . Then  $e$  is implied by  $E$  iff  $b = b'$ . If, on the other hand,  $b' \neq b$ , then  $E \wedge e$  is unsatisfiable and we return false. Now assume,  $E(\mathbf{x}_i) = \mathbf{x}_h + b'$ . Then the conjunction  $E \wedge e$  is equivalent to  $E \wedge (\mathbf{x}_h \doteq b - b')$ , and we obtain the normalised form for  $E \wedge e$  by substituting  $b - b'$  for all occurrences of  $\mathbf{x}_h$  in the array corresponding to  $E$ .

*Case 2.*  $e$  is of the form  $\mathbf{x}_i \doteq \mathbf{x}_j + b$ . First assume that  $E'(\mathbf{x}_i) = b_1$ . Then the conjunction  $E' \wedge e$  is equivalent to  $E' \wedge (\mathbf{x}_j \doteq b_1 - b)$  and we may proceed as in case 1. Likewise if  $E'(\mathbf{x}_j) = b_2$ , then the conjunction  $E' \wedge e$  is equivalent to  $E' \wedge (\mathbf{x}_i \doteq b_2 + b)$  and we again may proceed as in case 1. Now assume that  $E'(\mathbf{x}_i) = \mathbf{x}_{h_1} + b_1$  and  $E'(\mathbf{x}_j) = \mathbf{x}_{h_2} + b_2$ . If  $h_1 = h_2$  and  $b_1 = b_2 + b$ , then  $e$  is implied by  $E'$ . Otherwise, if  $h_1 = h_2$  and  $b_1 \neq b_2 + b$ , then the conjunction  $E' \wedge e$  is unsatisfiable, and we return false. Finally, if  $h_1 \neq h_2$ , then  $\mathbf{x}_i$  and  $\mathbf{x}_j$  belong to different connected components of the graph  $\mathcal{G}(E')$ . The new equality will therefore *join* the maximal connected components of  $\mathcal{G}(E')$  corresponding to  $h_1$  and  $h_2$ . If  $h_1 < h_2$ , then  $\mathbf{x}_{h_1}$  becomes the reference node of the new component where  $\mathbf{x}_{h_1} \doteq \mathbf{x}_{h_2} + b + (b_2 - b_1)$ . This means that we obtain the new array for  $E' \wedge e$  by substituting in  $E'$ ,  $\mathbf{x}_{h_1} + b_1 - (b + b_2)$  for every occurrence of  $\mathbf{x}_{h_2}$ . If on the other hand  $h_1 > h_2$ , then  $\mathbf{x}_{h_2}$  becomes the reference variable of the new component implying that we then obtain the array for  $E' \wedge e$  by substituting  $\mathbf{x}_{h_2} + b + (b_2 - b_1)$  for every occurrence of  $\mathbf{x}_{h_1}$ .

Overall we remark that it can be decided in time  $\mathcal{O}(1)$  if  $E' \wedge e$  is unsatisfiable. Otherwise, the array representation of the normalised conjunction  $E' \wedge e$  can be computed from the array representation of  $E'$  in time  $\mathcal{O}(1)$  if  $e$  is implied by  $E'$  and in time  $\mathcal{O}(k)$  if  $e$  is not implied. Since from all  $r$  equalities, at most  $k$  equalities may not be implied, we obtain the complexity bound  $(r + k^2)$ .  $\square$

## 4 Lattice Operations

The *greatest lower bound*  $E_1 \sqcap E_2$  is the *conjunction* of all equalities of  $E_1$  and  $E_2$ . Applying the algorithm from lemma 1, we obtain:

**Lemma 2.** *The greatest lower bound of  $n$  normalised conjunctions of equalities  $E_1 \sqcap \dots \sqcap E_n$  can be computed in time  $\mathcal{O}((n + k) \cdot k)$ .*  $\square$

We conclude that we can restrict ourselves to computing with normalised conjunctions (or false). We find:

**Lemma 3.** *The length  $h$  of every strictly increasing chain  $\text{false} \sqcap E_1 \sqcap \dots \sqcap E_h$  of conjunctions of equalities  $E_i \in \mathbb{E}(\mathbf{X})$  is bounded by  $k + 1$ .*

*Proof.* Let  $n(i)$  denote the number of connected components of the weighted directed graph  $\mathcal{G}(E_i)$  corresponding to  $E_i$ . Since  $E_i$  strictly implies  $E_{i+1}$ ,  $\mathcal{G}(E_{i+1})$  has more connected components than  $\mathcal{G}(E_i)$ , i.e.,  $n(i) < n(i + 1)$ . Since moreover,  $1 \leq n(h) \leq k + 1$ , we conclude that  $h \leq k + 1$ , and the assertion follows.  $\square$

Thus, the lattice  $\mathbb{E}(\mathbf{X})$  satisfies the ascending chain condition and therefore forms a *complete* lattice. By definition, the least upper bound of two conjunctions of equalities  $E_1, E_2$  is given by the conjunction of all equalities implied both by  $E_1$  and  $E_2$ . The next lemma provides an efficient algorithm for computing least upper bounds.

**Lemma 4.** *The least upper bound  $E_1 \sqcup E_2$  of two normalised conjunctions of equalities  $E_1, E_2$  can be computed in time  $\mathcal{O}(k \cdot \log(k))$ .*

*Proof.* Let  $E = E_1 \sqcup E_2$ . We first determine the connected components of the graph  $\mathcal{G}(E)$ . For that, we first define the *difference*  $\delta(\mathbf{x}_i)$  of variable  $\mathbf{x}_i$  w.r.t. the conjunctions  $E_1$  and  $E_2$  as  $b_1 - b_2$  if  $b_i$  is the constant offset obtained via  $E_i(\mathbf{x}_i)$ . Then, we observe:

*Observation:* Variables  $\mathbf{x}_j$  and  $\mathbf{x}_{j'}$  are in the same component of  $\mathcal{G}(E)$  iff the following two properties hold:

- $\mathbf{x}_j$  and  $\mathbf{x}_{j'}$  are in the same component of  $\mathcal{G}(E_1)$  and  $\mathcal{G}(E_2)$ ;
- $\delta(\mathbf{x}_j) = \delta(\mathbf{x}_{j'})$ .  $\square$

Using this observation, we proceed as follows. We first partition the set of variables  $\mathbf{X}$  into classes of variables which agree both in the reference nodes w.r.t.  $\mathcal{G}(E_1)$  and  $\mathcal{G}(E_2)$ . Each such class  $Q$  again is partitioned into subclasses of variables  $\mathbf{x}_j$  which additionally

agree in their differences  $\delta(\mathbf{x}_j)$ . Let  $\Pi$  be the resulting partition of variables  $\mathbf{X}$ . We define  $E$  for one equivalence class  $Q' \in \Pi$  after the other.

Assume that  $Q'$  consists of a single member  $\mathbf{x}_i$  only. If  $E_1(\mathbf{x}_i) = E_2(\mathbf{x}_i) = b$  for some constant  $b$ , we set  $E(\mathbf{x}_i) = b$ . Otherwise, we set  $E(\mathbf{x}_i) = \mathbf{x}_i$ .

Now assume that the equivalence class  $Q'$  consists of more than one variable. Then we distinguish two cases. Again  $E_1$  and  $E_2$  are given in array representation.

*Case 1.* For every  $\mathbf{x}_j \in Q'$ , both  $E_1(\mathbf{x}_j)$  and  $E_2(\mathbf{x}_j)$  yield constants. If  $\delta(\mathbf{x}_j) = 0$ , these constants are equal, and we set  $E(\mathbf{x}_j) \leftarrow E_1(\mathbf{x}_j)$  for all  $\mathbf{x}_j \in Q'$ . If on the other hand,  $\delta(\mathbf{x}_j) \neq 0$ , then we choose that variable  $\mathbf{x}_h \in Q'$  with least index  $h$  as new reference node. Let  $E_1(\mathbf{x}_j) = b_j$  for  $\mathbf{x}_j \in Q'$ . Then we define  $E(\mathbf{x}_j) \leftarrow \mathbf{x}_h - b_h + b_j$ ,  $\mathbf{x}_j \in Q'$ .

*Case 2.* For some  $i \in \{1, 2\}$ ,  $E_i(\mathbf{x}_j) = \mathbf{x}_{h_i} + b_j$  for  $\mathbf{x}_j \in Q'$ . Let  $\mathbf{x}_h$  denote the variable in  $Q'$  with least index  $h$ . Then  $\mathbf{x}_h$  becomes the new reference node of  $Q'$ . Since  $\mathbf{x}_{h_i} \doteq \mathbf{x}_h - b_h$ , we then define  $E(\mathbf{x}_j) \leftarrow \mathbf{x}_h - b_h + b_j$ .

This algorithm can be implemented by stably sorting the variables in  $\mathbf{X}$  first according to the variables occurring in the right-hand sides of  $E_1$  and  $E_2$ , respectively, and then according to the differences  $\delta(\mathbf{x}_i)$ . We thus conclude that the overall runtime is at most  $\mathcal{O}(k \cdot \log(k))$ .  $\square$

*Example 3.* Consider the array representations of the normalised conjunctions  $E_1, E_2$  as given in the left table. Then stably sorting w.r.t. the reference nodes and  $\delta$  yields the partitioning as specified in the right table:

$E_1$ and $E_2$ in normal form:			... and after partitioning:			
$\mathbf{X}$	$E_1$	$E_2$	$\mathbf{X}$	$E_1$	$E_2$	$\delta$
$\mathbf{x}_1$	$\mathbf{x}_1$	$\mathbf{x}_1$	$\mathbf{x}_1$	$\mathbf{x}_1$	$\mathbf{x}_1$	0
$\mathbf{x}_2$	$\mathbf{x}_2$	$\mathbf{x}_2$	$\mathbf{x}_2$	$\mathbf{x}_2$	$\mathbf{x}_2$	0
$\mathbf{x}_3$	$\mathbf{x}_1$	$\mathbf{x}_2 - 5$	$\mathbf{x}_4$	$\mathbf{x}_2 + 5$	$\mathbf{x}_2 + 5$	0
$\mathbf{x}_4$	$\mathbf{x}_2 + 5$	$\mathbf{x}_2 + 5$	$\mathbf{x}_6$	$\mathbf{x}_1 + 3$	$\mathbf{x}_2 + 1$	2
$\mathbf{x}_5$	$\mathbf{x}_1 + 5$	$\mathbf{x}_2$	$\mathbf{x}_7$	$\mathbf{x}_1 + 2$	$\mathbf{x}_2$	2
$\mathbf{x}_6$	$\mathbf{x}_1 + 3$	$\mathbf{x}_2 + 1$	$\mathbf{x}_3$	$\mathbf{x}_1$	$\mathbf{x}_2 - 5$	5
$\mathbf{x}_7$	$\mathbf{x}_1 + 2$	$\mathbf{x}_2$	$\mathbf{x}_5$	$\mathbf{x}_1 + 5$	$\mathbf{x}_2$	5

In order to compute  $E = E_1 \sqcup E_2$ , we successively consider each subclass. Since for variables  $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_4$  the equalities in  $E_1$  and  $E_2$  are syntactically equal, we obtain  $E(\mathbf{x}_1) \leftarrow \mathbf{x}_1, E(\mathbf{x}_2) \leftarrow \mathbf{x}_2$  and  $E(\mathbf{x}_4) \leftarrow \mathbf{x}_2 + 5$ . For the remaining two subclasses we choose new reference variables, namely  $\mathbf{x}_3$  and  $\mathbf{x}_6$ , respectively, and thus:  $E(\mathbf{x}_3) \leftarrow \mathbf{x}_3, E(\mathbf{x}_5) \leftarrow \mathbf{x}_3 + 5, E(\mathbf{x}_6) \leftarrow \mathbf{x}_6$  and  $E(\mathbf{x}_7) \leftarrow \mathbf{x}_6 - 1$ .

The normalised conjunction for  $E_1 \sqcup E_2$  therefore, is given by  $E = (\mathbf{x}_4 \doteq \mathbf{x}_2 + 5) \wedge (\mathbf{x}_5 \doteq \mathbf{x}_3 + 5) \wedge (\mathbf{x}_7 \doteq \mathbf{x}_6 - 1)$ .  $\square$

Summarising, we have constructed a complete lattice  $\mathbb{E}(\mathbf{X})$  of height  $k + 1$  and provided efficient implementations for the basic lattice operations  $\sqcup, \sqcap$  on normalised representations of conjunctions of equalities.

### Abstract Effect of Statements

Every element  $E \in \mathbb{E}(\mathbf{X})$  can be considered as description of the set  $\gamma(E)$  of the concrete states  $x \in \mathbb{Z}^k$  with  $x \models E$ . Likewise, the best description  $\alpha(X)$  of a set  $X \subseteq \mathbb{Z}^k$  of states is the conjunction of all equalities  $e$  with  $x \models e$  for all  $x \in X$ . Together,  $\alpha$  and  $\gamma$  form a Galois connection between  $(2^{\mathbb{Z}^k}, \subseteq)$ , the powerset of the set of states ordered by inclusion, and  $(\mathbb{E}(\mathbf{X}), \Rightarrow)$ .

For our analysis, we define the abstract effect  $\llbracket s \rrbracket^\sharp$  for every assignment  $s$  by:

$$\begin{aligned} \llbracket \mathbf{x}_i := ? \rrbracket^\sharp E &= \exists^\sharp \mathbf{x}_i. E \\ \llbracket \mathbf{x}_i := \mathbf{x}_i + c \rrbracket^\sharp E &= E[\mathbf{x}_i - c / \mathbf{x}_i] \\ \llbracket \mathbf{x}_i := c \rrbracket^\sharp E &= (\exists^\sharp \mathbf{x}_i. E) \wedge (\mathbf{x}_i \doteq c) \\ \llbracket \mathbf{x}_i := \mathbf{x}_j + c \rrbracket^\sharp E &= (\exists^\sharp \mathbf{x}_i. E) \wedge (\mathbf{x}_i \doteq \mathbf{x}_j + c) \text{ if } i \neq j \end{aligned}$$

Here,  $\exists^\sharp \mathbf{x}_i. E$  denotes the *abstract existential quantification*, i.e.,  $\exists^\sharp \mathbf{x}_i. E = \perp$  if  $E = \perp$ , otherwise it is the conjunction of all equalities implied by  $E$  which do not contain variable  $\mathbf{x}_i$ . The array representation of the normalised conjunction for  $E' = \exists^\sharp \mathbf{x}_i. E$  can be computed as follows. Let  $X$  denote the connected component containing  $\mathbf{x}_i$  in the graph  $\mathcal{G}(E)$ . All entries of  $E'$  for variables  $\mathbf{x}_j \notin X$  equal the corresponding entries in  $E$ . If  $X = \{\mathbf{x}_i\}$ ,  $E'$  equals  $E$ . Otherwise, we remove the variable  $\mathbf{x}_i$  from  $X$ . This means that we set  $E'(\mathbf{x}_i) \leftarrow \mathbf{x}_i$ . If  $\mathbf{x}_i$  is not the reference variable of  $X$ , then also  $E'(\mathbf{x}_j) = E(\mathbf{x}_j)$  for all remaining  $\mathbf{x}_j \in X, \mathbf{x}_j \neq \mathbf{x}_i$ . If  $\mathbf{x}_i$  is the reference variable of  $X$ , then we determine the variable  $\mathbf{x}_h \neq \mathbf{x}_i \in X$  with least index. Assume that  $E(\mathbf{x}_h) = \mathbf{x}_i + b$ . Then we set  $E'(\mathbf{x}_j) \leftarrow \mathbf{x}_h - b + b_j$  if  $E(\mathbf{x}_j) = \mathbf{x}_i + b_j$ .

Note that  $\exists^\sharp \mathbf{x}_i. E$  preserves  $\perp$  and commutes with least upper bounds. Furthermore, the given algorithm runs in time  $\mathcal{O}(k)$ .

For an assignment  $\mathbf{x}_i := \mathbf{x}_i + c$ , we observe that the value of  $\mathbf{x}_i$  before the assignment can be recovered from the value of  $\mathbf{x}_i$  after the assignment. Therefore, the conjunction after the assignment can be obtained from the conjunction  $E$  before the assignment by substituting  $\mathbf{x}_i - c$  for  $\mathbf{x}_i$ . If  $\mathbf{x}_i$  occurs on the right-hand side of equalities in  $E$ , this substitution is implemented by preserving the equality  $\mathbf{x}_i \doteq \mathbf{x}_i$  and replacing every other equality  $\mathbf{x}_j \doteq \mathbf{x}_i + b_j$  with  $\mathbf{x}_j \doteq \mathbf{x}_i - c + b_j$ . If  $\mathbf{x}_i$  only occurs on the left-hand side, i.e., in an equality  $\mathbf{x}_i \doteq \mathbf{x}_h + b_i$  or  $\mathbf{x}_i \doteq b_i$ , then we replace this equality with  $\mathbf{x}_i \doteq \mathbf{x}_h + c + b_i$  or  $\mathbf{x}_i \doteq c + b_i$ , respectively. Again, this operation is distributive and can be executed in time  $\mathcal{O}(k)$ .

For the remaining instances of assignments, we first remove variable  $\mathbf{x}_i$  from all equalities in  $E$  by means of abstract existential quantification, and then add the equality  $\mathbf{x}_i \doteq c$  or  $\mathbf{x}_i \doteq \mathbf{x}_j + c$ , respectively. Since both abstract existential quantification and conjunction with a single equality can be executed in time  $\mathcal{O}(k)$ , this transformation can be executed in time  $\mathcal{O}(k)$ , as well. Since it is composed of distributive transformations, it is also distributive.

Summarising, we found that for every assignment  $s$  the transformation  $\llbracket s \rrbracket^\sharp$  is distributive where  $\llbracket s \rrbracket^\sharp E$  for a normalised conjunction  $E$  can be computed in time  $\mathcal{O}(k)$ . Moreover, we find:

**Lemma 5.** *For a set of program states  $X \subseteq \mathbb{Z}^k$  and an arbitrary assignment  $s$ :  $\alpha(\llbracket s \rrbracket X) = \llbracket s \rrbracket^\sharp(\alpha(X))$ .  $\square$*

## 5 Interprocedural Analysis

In order to provide an interprocedural analysis of variable differences, we must provide an effective and if possible succinct representation of the effects of procedures. An obvious approach would be to tabulate the abstract effect of a procedure on its inputs. Here, we follow the approach of [10] and rely on *weakest precondition* transformers. The advantage of weakest precondition transformers is that they are *completely distributive*, i.e., commute with arbitrary conjunctions. This implies that weakest precondition transformers only need to be specified for *single* equalities alone.

We are interested in equalities of the form  $\mathbf{x}_i \doteq c$  or  $\mathbf{x}_i \doteq \mathbf{x}_j + c$  for global variables  $\mathbf{x}_i, \mathbf{x}_j$  and constants  $c$ . For computing the representation of procedures, however, we take a broader perspective and consider weakest preconditions for a slightly larger class of equalities. In particular, we introduce one extra *logical variable*  $\bullet$  and thus consider equalities of one of the following forms:

- (1)  $a \cdot \bullet + b \doteq 0$
- (2)  $\mathbf{x}_i \doteq a \cdot \bullet + b$
- (3)  $\mathbf{x}_i \doteq \mathbf{x}_j + a \cdot \bullet + b$

for global variables  $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$  and constants  $a, b \in \mathbb{Z}$ . Let us call such equalities *parametric*. Note that  $a \cdot \bullet + b \doteq 0$  is only satisfiable over  $\mathbb{Z}$  iff  $a = b = 0$  or  $a$  divides  $b$ . In the latter case it has the unique solution  $\bullet = \frac{-b}{a}$ .

Satisfiability of single equalities  $e$  of the above forms as well as of conjunctions  $E$  of such equalities again is denoted by  $Z \models e$  and  $Z \models E$ , respectively, where now  $Z \subseteq \mathbb{Z}^{k+1}$  is a set of vectors, each consisting of values for the variables  $\mathbf{x}_i$  together with one value for  $\bullet$  as component  $k+1$ . Such a vector  $z \in \mathbb{Z}^{k+1}$  is called an *extended state* which is also written as a pair  $(x, c)$  for a vector  $x \in \mathbb{Z}^k$  with values for the variables  $\mathbf{x}_i$  together with a value  $c$  for  $\bullet$ . For a satisfiable conjunction  $E$  of parametric equalities without equalities of form (1), we define a normalised form analogously to the normal form of finite conjunctions of ordinary equalities. However, if there is a satisfiable equality of form (1), then we determine the unique value  $v$  for  $\bullet$  and remove  $\bullet$  from all other equalities. In this case, the normal form is  $(\bullet \doteq v) \wedge E'$  where  $E'$  is the normal form which we have defined for conjunctions without  $\bullet$ . Let  $\mathbb{E}_\bullet(\mathbf{X})$  denote the complete lattice of equivalence classes of finite conjunctions of parametric equalities over variables from  $\mathbf{X}$ .

The concrete semantics operates on sets  $X \subseteq \mathbb{Z}^k$  and does not affect the value of the logical variable. Accordingly, we extend any completely distributive transformation  $f : \mathbb{Z}^k \rightarrow \mathbb{Z}^k$  of concrete sets of states to a completely distributive transformation  $\text{ext } f : \mathbb{Z}^{k+1} \rightarrow \mathbb{Z}^{k+1}$  of sets of extended states by defining:

$$\text{ext } f \{(x, c)\} = \{(x', c) \mid x' \in f \{x\}\}$$

For an assignment  $s$ , the WP transformer  $\llbracket s \rrbracket^\top$  applied to a single non-trivial equality  $e$  is given by:

$$\begin{aligned} \llbracket \mathbf{x}_i := ? \rrbracket^\top e &= \forall \mathbf{x}_i. e &= \begin{cases} \perp & \text{if } e \text{ contains } \mathbf{x}_i \\ e & \text{otherwise} \end{cases} \\ \llbracket \mathbf{x}_i := c \rrbracket^\top e &= e [c/\mathbf{x}_i] \\ \llbracket \mathbf{x}_i := \mathbf{x}_j + c \rrbracket^\top e &= e [\mathbf{x}_j + c/\mathbf{x}_i] \end{aligned}$$

for  $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$  and  $c \in \mathbb{Z}$ .

The weakest precondition for a non-deterministic assignment  $\mathbf{x}_i := ?$  applied to a non-trivial equality  $e$  is  $\perp$  if variable  $\mathbf{x}_i$  occurs in  $e$  because  $e$  cannot hold for multiple values of  $\mathbf{x}_i$ . In order to compute the weakest precondition for an assignment  $\mathbf{x}_i := t$ , we substitute  $t$  for every occurrence of variable  $\mathbf{x}_i$  in  $e$ . If  $\mathbf{x}_i$  occurs on the left-hand side of  $e$ , this may violate the format we have fixed for equalities. This format, though, can be restored straightforwardly by algebraic simplification. Thus, e.g.,

$$\llbracket \mathbf{x}_4 := \mathbf{x}_1 + 5 \rrbracket^\top (\mathbf{x}_4 \doteq \mathbf{x}_3 + 2) = (\mathbf{x}_1 + 5 \doteq \mathbf{x}_3 + 2) = (\mathbf{x}_3 \doteq \mathbf{x}_1 + 3).$$

We obtain:

**Lemma 6.** *For a set of extended program states  $Z \subseteq \mathbb{Z}^{k+1}$ ,  $E \in \mathbb{E}_\bullet(\mathbf{X})$  and an arbitrary assignment  $s$ :  $\text{ext } \llbracket s \rrbracket(Z) \models E$  iff  $Z \models \llbracket s \rrbracket^\top E$ .  $\square$*

By this lemma, the WP transformers provide an exact abstraction of the extended concrete transformers of the collecting semantics, i.e., the extended concrete effect function applied to a set of extended states  $Z$  satisfies the conjunction  $E$  iff  $Z$  satisfies the conjunction returned by the WP transformer for  $E$ .

In order to describe the abstract effects of whole procedures, we set up the following constraint system  $S^\top$ :

$$\begin{array}{ll} S^\top[r_q] \sqsubseteq \text{Id} & r_q \text{ exit point of procedure } q \\ S^\top[u] \sqsubseteq S^\top[s_q] \circ S^\top[v] & (u, q(), v) \text{ a call edge, } s_q \text{ entry point of } q \\ S^\top[u] \sqsubseteq \llbracket s \rrbracket^\top \circ S^\top[v] & (u, s, v) \text{ an assignment edge} \end{array}$$

Again, Id denotes the identity mapping that maps  $E$  to itself for every  $E \in \mathbb{E}_\bullet(\mathbf{X})$ . Here,  $S^\top[u]$  specifies the weakest precondition transformer for a program point  $u$  of procedure  $q$  when starting from  $u$  and reaching the procedure exit of  $q$ . All operations in this constraint system are monotonic. Therefore, it has a greatest solution. Since all occurring functions are  $\sqcap$ -distributive, composition is  $\sqcap$ -distributive as well. We obtain:

**Theorem 1.** *Assume  $Z \subseteq \mathbb{Z}^{k+1}$  is a set of extended states and  $E \in \mathbb{E}_\bullet(\mathbf{X})$ . Then, for every program point  $u$ :  $\text{ext } S[u] Z \models E$  iff  $Z \models S^\top[u] E$ .  $\square$*

The proof of theorem 1 proceeds by induction on the  $i$ -th approximation of the least fixpoint of  $S$  and the greatest fixpoint of constraint system  $S^\top$ .

For computing a solution for constraint system  $S^\top$  an effective representation of transformers is required. As weakest precondition transformers distribute over conjunctions, it suffices to determine the results of the transformer for single equalities only. However, since  $\mathbb{Z}$  is infinite, the number of possible equalities, is infinite as well, such that we cannot simply tabulate the results for all equalities. In the next section we show how to circumvent this problem.

## 6 Effective Representation of WP Transformers

The key observation for obtaining an effective representation of WP transformers is that the WP transformers are completely determined by their values for postconditions of the forms  $\mathbf{x}_i \doteq \bullet$  or  $\mathbf{x}_i \doteq \mathbf{x}_j + \bullet$  with  $i > j$ . We call postconditions of this form *generic*. Note that generic postconditions are particular parametric postconditions. (Choose  $a = 1$  and  $b = 0$  in types (2) and (3) of parametric postconditions.) The set  $\mathbb{P}_\bullet(\mathbf{X})$  of all generic postconditions is finite and contains only  $\mathcal{O}(k^2)$  many elements. Any other equality involving globals is obtained from a generic postcondition by means of *substituting* the logical variable  $\bullet$  with a term  $a \bullet + b$  which consists of constants  $a, b$  and  $\bullet$  only. Note that  $a, b$  can be 0.

In order to recover the full WP transformer from its values for generic postconditions, we use an operator  $\text{ext}^\top$ . The operator  $\text{ext}^\top$  takes a function  $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X})$  and transforms it into a full WP transformer of type  $\mathbb{E}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X})$ . For a single equality  $e$  involving globals and  $\bullet$ , this transformer is defined by:

$$\begin{aligned} \text{ext}^\top(f^\top)(\mathbf{x}_i \doteq t) &= f^\top(\mathbf{x}_i \doteq \bullet)[t/\bullet] \\ \text{ext}^\top(f^\top)(\mathbf{x}_i \doteq \mathbf{x}_j + t) &= f^\top(\mathbf{x}_i \doteq \mathbf{x}_j + \bullet)[t/\bullet] \end{aligned}$$

for globals  $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$  and a term  $t = a \bullet + b$  for constants  $a, b$ . For equalities  $e$  only containing  $\bullet$ , we define:

$$\text{ext}^\top(f^\top)(e) = \begin{cases} \top & \text{if } f^\top(\mathbf{x}_1 \doteq \bullet) = \top \\ e & \text{otherwise} \end{cases}$$

Here, we assume that  $f^\top$  corresponds to a definitely not terminating computation if  $f^\top(\mathbf{x}_1 \doteq \bullet) = \top$ . In this case, the precondition of *any* equality should be  $\top$ . Otherwise, the precondition of the equality  $e$  should be  $e$  itself. Finally, for arbitrary conjunctions  $E = e_1 \wedge \dots \wedge e_m$ , we set

$$\text{ext}^\top(f^\top)(E) = \text{ext}^\top(f^\top)(e_1) \wedge \dots \wedge \text{ext}^\top(f^\top)(e_m)$$

Let  $f : 2^{\mathbb{Z}^k} \rightarrow 2^{\mathbb{Z}^k}$  be completely distributive. We call  $f$  *uniform* if  $f\{x\} = \emptyset$  for some vector  $x \in \mathbb{Z}^k$  implies that  $f\{x'\} = \emptyset$  for all  $x' \in \mathbb{Z}^k$ . Note that all concrete transformers which occur in this context are completely distributive and uniform. We have:

**Lemma 7.** *Let  $f : 2^{\mathbb{Z}^k} \rightarrow 2^{\mathbb{Z}^k}$  denote a concrete transformer which is completely distributive and uniform. Furthermore, let  $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X})$  denote a function where for all  $Z \subseteq \mathbb{Z}^{k+1}$  and  $e \in \mathbb{P}_\bullet(\mathbf{X})$ ,  $\text{ext } f(Z) \models e$  iff  $Z \models f^\top(e)$ . Then also*

$$\text{ext } f(Z) \models E \quad \text{iff} \quad Z \models \text{ext}^\top(f^\top)(E)$$

for all  $Z \subseteq \mathbb{Z}^{k+1}$  and  $E \in \mathbb{E}_\bullet(\mathbf{X})$ .

*Proof.* Since  $f$  and  $\text{ext}^\top(f^\top)$  are completely distributive, it suffices to consider single equalities  $e$ . We perform a case distinction on the different forms of  $e$ . First consider

an equality  $e$  which contains a single global  $\mathbf{x}_i$ , i.e., is of the form  $\mathbf{x}_i \doteq t$  for a term  $t = a \bullet + b$ . Consider the set  $Z' = \{(x, ac + b) \mid (x, c) \in Z\}$ . Then

$$\begin{aligned} \text{ext } f(Z) \models e &\text{ iff } \text{ext } f(Z') \models (\mathbf{x}_i \doteq \bullet) \\ &\text{ iff } Z' \models f^\top(\mathbf{x}_i \doteq \bullet) \\ &\text{ iff } Z \models f^\top(\mathbf{x}_i \doteq \bullet)[a \bullet + b/\bullet] \\ &\text{ iff } Z \models \text{ext}^\top(f^\top)(\mathbf{x}_i \doteq a \bullet + b) \end{aligned}$$

The proof for an equality  $e$  of the form  $\mathbf{x}_i \doteq \mathbf{x}_j + t$  for a second global  $\mathbf{x}_j$  is analogous.

Finally consider an equality  $e$  which does not contain globals  $\mathbf{x}_i$ , i.e., which only may contain constants or  $\bullet$ . We rely on the following claim:

*Claim:* Under the assumptions of the lemma for  $f$  and  $f^\top$ , one of the following statements is true:

- $f(\{x\}) = \emptyset$  for some  $x \in \mathbb{Z}^k$ . Then  $f(\{x\}) = \emptyset$  for all  $x \in \mathbb{Z}^k$ , and  $f^\top(\mathbf{x}_1 \doteq \bullet) = \top$ .
- $f(\{x\}) \neq \emptyset$  for all  $x \in \mathbb{Z}^k$ , and  $f^\top(\mathbf{x}_1 \doteq \bullet) \neq \top$ .

Before proving the claim, let us first show that the assertion of the lemma for equalities  $e$  without globals follows from the claim. First assume case 1 of the claim, i.e.,  $f(\{x\}) = \emptyset$  for all  $(x, y)$ . Then also  $\text{ext } f \{(x, c)\} = \emptyset$  for all  $x$  and  $c$ . Since  $\emptyset \models e$ , the left-hand side of the assertion is true for all  $Z$ . Now by the first case of the claim,  $f^\top(\mathbf{x}_1 \doteq \bullet) = \top$ . Hence by definition, also  $\text{ext}^\top(f^\top)(e) = \top$ , and the right-hand side of the assertion also evaluates to true for all  $Z$ .

Now assume case 2 of the claim, i.e.,  $f(\{x\}) \neq \emptyset$  for all  $x$ . Then

$$\begin{aligned} \text{ext } f(Z) \models e &\text{ iff } Z \models e \\ &\text{ iff } Z \models \text{ext}^\top(f^\top)(e) \end{aligned}$$

and the assertion follows.

It therefore remains to prove the claim. First assume that  $f(\{x\}) = \emptyset$  for some  $x$ . Then by uniformity of  $f$ , also  $\text{ext } f \{(x, c)\} = \emptyset$  for all  $(x, c)$ , i.e.,  $\text{ext } f(\mathbb{Z}^{k+1}) = \emptyset$ . Since then  $\text{ext } f(\mathbb{Z}^{k+1}) \models (\mathbf{x}_1 \doteq \bullet)$ , we conclude by the assumption on  $f$  and  $f^\top$  that  $\mathbb{Z}^{k+1} \models f^\top(\mathbf{x}_1 \doteq \bullet)$ , and therefore,  $f^\top(\mathbf{x}_1 \doteq \bullet) = \top$ .

Now assume that  $f(\{x\}) \neq \emptyset$  for all  $x$ . For a contradiction assume that  $f^\top(\mathbf{x}_1 \doteq \bullet) = \top$ . For some  $x \in \mathbb{Z}^k$  and  $x' \in f(\{x\})$ , consider the sets  $Z = \{(x, c) \mid c \in \mathbb{Z}\}$  and  $Z' = \{(x', c) \mid c \in \mathbb{Z}\}$ . Then  $Z \models f^\top(\mathbf{x}_1 \doteq \bullet)$ , and hence by the assumption on  $f$  and  $f^\top$ ,  $\text{ext } f(Z) \models (\mathbf{x}_1 \doteq \bullet)$ . Since  $Z' \subseteq \text{ext } f(Z)$ , also  $Z' \models (\mathbf{x}_1 \doteq \bullet)$ . This means that for all  $c$ ,  $x'_1 = c$ , which yields a contradiction. We conclude that  $f^\top(\mathbf{x}_1 \doteq \bullet)$  cannot be equal  $\top$  and the second statement of the claim follows. This completes the proof.  $\square$

Using the new operator  $\text{ext}^\top$ , we obtain the following modified constraint system for the weakest precondition transformers of procedures — as represented by their values on only the generic postconditions:

$$\begin{array}{ll} S^\bullet[r_q] \sqsubseteq \text{Id} & r_q \text{ exit point of procedure } q \\ S^\bullet[u] \sqsubseteq \text{ext}^\top(S^\bullet[s_q]) \circ S^\bullet[v] & (u, q(), v) \text{ a call edge, } s_q \text{ entry point of } q \\ S^\bullet[u] \sqsubseteq \llbracket s \rrbracket^\top \circ S^\bullet[v] & (u, s, v) \text{ an assignment edge} \end{array}$$

For a distinction, let us call this constraint system  $S^\bullet$ . The construction of a representation for the composition of transformers, as required for the constraints of the second and third line, must take into account that we compute with mappings from  $\mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X})$  only. This means for the constraints from the second line that we must extend the transformer for the called procedure by means of  $\text{ext}^\top$  before the composition can be performed. In general, consider a composition  $h = \text{ext}^\top(f^\top) \circ g^\top$  for completely distributive functions  $f^\top, g^\top$ . Let  $e$  denote a generic postcondition, and assume that  $e_1[t_1/\bullet] \wedge \dots \wedge e_r[t_r/\bullet]$  is a normalised conjunction for  $g^\top(e)$  where  $e_i \in \mathbb{P}_\bullet(\mathbf{X})$ . Then the value  $(\text{ext}^\top(f^\top) \circ g^\top)(e)$  is the normalised conjunction for:

$$f^\top(e_1)[t_1/\bullet] \wedge \dots \wedge f^\top(e_r)[t_r/\bullet]$$

i.e., amounts to normalising a conjunction of  $\mathcal{O}(k^2)$  equalities. According to lemma 1, this can be done in time  $\mathcal{O}(k^2)$ . Since there are at most  $\mathcal{O}(k^2)$  generic postconditions, a representation for the composition  $h$  can be computed in time  $\mathcal{O}(k^4)$ .

- Lemma 8.** 1. Assume  $S^\top[u]$  ( $u$  a program point) is the greatest solution of  $S^\top$ . Then a solution of  $S^\bullet$  is obtained by restricting each transformer  $S^\top[u]$  to generic postconditions.
2. Assume  $S^\bullet[u]$  ( $u$  a program point) is the greatest solution of  $S^\bullet$ . Then a solution of  $S^\top$  is obtained by defining  $S^\top[u] = \text{ext}^\top(S^\bullet[u])$ .  $\square$

For the first statement of the lemma, we rely on theorem 1 and lemma 7. Since both restriction and extension of transformers are monotonic operations, we conclude:

**Theorem 2.** Assume that  $Z \subseteq \mathbb{Z}^{k+1}$  and  $e \in \mathbb{P}_\bullet(\mathbf{X})$ . Then, for every program point  $u$ :  $\text{ext } S[u](Z) \models e$  iff  $Z \models S^\bullet[u](e)$ .  $\square$

The proof of this theorem is by fixpoint induction.

Our goal is to determine for every program point  $u$ , the conjunction of all equalities which are valid when reaching  $u$ . Note that these equalities may comprise of global variables only (no  $\bullet$  is needed here). For that, we require a transformation  $\text{ext}^\sharp$  which takes a weakest precondition transformer  $f^\top$  for the body of a procedure and returns the corresponding *forward* transformation  $\text{ext}^\sharp(f^\top)$  of valid equalities. For a weakest precondition transformer  $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X})$ , we define  $\text{ext}^\sharp(f^\top) : \mathbb{E}(\mathbf{X}) \rightarrow \mathbb{E}(\mathbf{X})$  as follows. Let  $E \in \mathbb{E}(\mathbf{X})$ . Then  $\text{ext}^\sharp(f^\top)(E)$  is the conjunction of all equalities  $e' = e[c/\bullet]$  for which  $E \implies (\text{ext}^\top(f^\top)(e'))$  or, equivalently,  $E \implies (f^\top(e)[c/\bullet])$ . The following lemma states that the operator  $\text{ext}^\sharp$  allows us to determine all the equalities that are valid after a procedure call from the conjunction of valid equalities before the call and the weakest precondition transformer of the called procedure.

**Lemma 9.** Let  $f : 2^{\mathbb{Z}^k} \rightarrow 2^{\mathbb{Z}^k}$  be completely distributive and uniform and let  $\text{ext } f : 2^{\mathbb{Z}^{k+1}} \rightarrow 2^{\mathbb{Z}^{k+1}}$  be the corresponding extended transformer. Let  $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X})$  be a weakest precondition transformer. Assume as in lemma 7 that  $\text{ext } f(Z) \models e$  iff  $Z \models f^\top(e)$  for all subsets  $Z \subseteq \mathbb{Z}^{k+1}$  and elements  $e \in \mathbb{P}_\bullet(\mathbf{X})$ . Assume  $X \subseteq \mathbb{Z}^k$  and  $E$  is the conjunction of all equalities  $e$  over  $\mathbf{X}$  with  $X \models e$ . Then, for every  $E' \in \mathbb{E}(\mathbf{X})$ ,  $f(X) \models E'$  iff  $\text{ext}^\sharp(f^\top)(E) \sqsubseteq E'$ .

*Proof.* It suffices to consider the case where  $E'$  is a single equality  $e'$  involving global variables  $\mathbf{x}_i$ . Then  $e' = e[c/\bullet]$  for a generic postcondition  $e$  and some constant  $c$  and we have

$$f(X) \models e' \quad \text{iff} \quad \text{ext } f(X_c) \models e$$

where  $X_c = \{(x, c) \mid x \in X\}$ . Furthermore,

$$\text{ext } f(X_c) \models e \quad \text{iff} \quad X_c \models f^\top(e)$$

by lemma 7. Then we deduce:

$$\begin{aligned} X_c \models f^\top(e) &\text{ iff } X \models f^\top(e)[c/\bullet] \\ &\text{ iff } E \sqsubseteq f^\top(e)[c/\bullet] \\ &\text{ iff } \text{ext}^\sharp(f^\top)(E) \sqsubseteq e' \end{aligned}$$

and the statement of the lemma follows.  $\square$

Using the operator  $\text{ext}^\sharp$ , we put up the following system of constraints over  $\mathbb{E}(\mathbf{X})$ :

$$\begin{array}{ll} C^\sharp[s_{main}] \sqsupseteq \top & \\ C^\sharp[s_q] \sqsupseteq C^\sharp[u] & (u, q(), -) \text{ a call edge} \\ C^\sharp[v] \sqsupseteq \text{ext}^\sharp(S^\bullet[s_q])(C^\sharp[u]) & (u, q(), v) \text{ a call edge} \\ C^\sharp[v] \sqsupseteq \llbracket s \rrbracket^\sharp(C^\sharp[u]) & (u, s, v) \text{ an assignment edge} \end{array}$$

The least solution of this constraint system precisely characterises for every program point  $u$  the conjunction of all equalities from  $\mathbb{E}(\mathbf{X})$  which are valid when program execution reaches  $u$ . Summarising, we obtain:

**Theorem 3.** *The set of all valid equalities for an interprocedural program can be computed in time  $\mathcal{O}(n \cdot k^4)$  where  $n$  is the program size and  $k$  the number of global variables.*

Note that throughout this paper the program size is defined as the sum of the number of nodes and the number of edges in the control flow representation of the program.

*Proof.* We use *semi-naive* fixpoint iteration as in [2] in order to compute the least solution of the system of inequations  $S^\bullet$ . Informally, semi-naive iteration means that only individual increments for the handled values are propagated instead of whole values. For our computation of summary functions this means that only single equalities instead of whole conjunctions are propagated and thus have to be added to the computed precondition. Note that the overall costs caused by a single constraint in a semi-naive iteration are at most as big as the cost of propagating a single value of maximal size in a standard fixpoint iteration. Thus, the most costly operation of a right-hand side of a single inequation in  $S^\bullet$  which is function composition mainly contributes to the time effort estimation for computing summary functions. As stated before function composition can be done in time  $\mathcal{O}(k^4)$ . Thus, the fixpoint of  $S^\bullet$  can be computed in time  $\mathcal{O}(n \cdot k^4)$ .

In contrast, we use ordinary worklist based least fixpoint computation for constraint system  $C^\sharp$ . Here, it is not clear how to propagate only single equalities. Since the height of the lattice of conjunctions of equalities  $\mathbb{E}(\mathbf{X})$  is  $k + 1$  (lemma 3), each right-hand

side of the constraint system  $C^\#$  may be evaluated at most  $\mathcal{O}(k)$  times. In system  $C^\#$  the most expensive operation is application of the summary functions of a procedure call. This operation takes time  $\mathcal{O}(k^3)$ . Hence, the least solution of constraint system  $C^\#$  can also be computed in time  $\mathcal{O}(n \cdot k^4)$ .

Consequently, considering a whole program of size  $n$ , the estimation of the total running time of  $\mathcal{O}(n \cdot k^4)$  follows.  $\square$

## 7 Local Variables

In the following, we extend the interprocedural analysis to procedures with local variables. The concept of local variables is not only provided by high-level programming languages. Also many modern processor architectures support local registers. The calling convention of the PowerPC architecture, e.g., treats the registers  $r14$  up to  $r31$  as local variables [15]. For simplicity, we assume that every procedure  $q$  has  $l$  local variables  $\mathbf{Y} = \{y_1, \dots, y_l\}$ , while the set of global variables is still  $\mathbf{X} = \{x_1, \dots, x_k\}$ . Thus, a state is now described by a vector  $(x_1, \dots, x_k, y_1, \dots, y_l) \in \mathbb{Z}^{k+l}$ , which we identify with the pair  $(x, y)$  of vectors  $x = (x_1, \dots, x_k) \in \mathbb{Z}^k$  and  $y = (y_1, \dots, y_l) \in \mathbb{Z}^l$  of values for the global and local variables, respectively. Accordingly, the transformations  $S[u]$  are completely distributive functions from the set  $\mathbb{T} = 2^{\mathbb{Z}^{k+l}} \rightarrow 2^{\mathbb{Z}^{k+l}}$ . In order to avoid confusion between the values of the local variables of caller and callee the rules for call edges must be modified. For this purpose we introduce two auxiliary transformations. The transformation  $\text{enter} \in \mathbb{T}$  captures how a set of states propagates from the call to the start edge of the called procedure:

$$\text{enter}(X) = \{(x, y) \mid y \in \mathbb{Z}^l, \exists y' : (x, y') \in X\}$$

Here, we assume that local variables receive an arbitrary value at the beginning of their scope but other conventions can be described similarly. The second transformation  $H : \mathbb{T} \rightarrow \mathbb{T}$  adjusts the transformation computed for a called procedure to the caller:

$$H(g)(X) = \{(x', y) \mid \exists x, y' : (x, y) \in X \wedge (x', y') \in g(\text{enter}\{(x, y)\})\}$$

It ensures that local variables of the caller are left untouched by the call. The modified rules for call edges in the systems of inequations for  $S$  and  $C$  are as follows:

$$\begin{array}{ll} S[u] \supseteq S[v] \circ H(S[s_q]) & (u, q(), v) \text{ a call edge, } s_q \text{ entry point of } q \\ C[s_q] \supseteq \text{enter}(C[u]) & (u, q(), -) \text{ a call edge} \\ C[v] \supseteq H(S[s_q])(C[u]) & (u, q(), v) \text{ a call edge} \end{array}$$

In addition,  $\mathbb{Z}^k$  is replaced with  $\text{enter}(\mathbb{Z}^{k+l}) = \mathbb{Z}^{k+l}$  in the inequation for  $C[s_{\text{main}}]$ .

As in section 5 in the case of global variables only, we want to determine the weakest precondition transformers for procedures. When representing such transformers, we rely on the special logical variable  $\bullet$  for avoiding to treat each constant in postconditions separately. Recall that the local variables of the caller are not visible to the called procedure and thus are also not modified during the execution of the call. This means that the

weakest precondition of the call for a postcondition  $e$  which involves local variables of the caller and  $\bullet$  only, equals just  $e$  (provided that the call may terminate). Every other postcondition consisting of a single equality  $e$  can refer to at most one local variable of the caller. The weakest preconditions for such equalities can be deduced from the weakest preconditions of equalities involving globals and  $\bullet$  only. Thus, for the sake of determining weakest preconditions, it suffices to consider weakest preconditions for equalities containing globals together with the auxiliary variable  $\bullet$ . Due to our backward accumulation of weakest precondition transformers for procedures, we therefore consider the same set  $\mathbb{P}_\bullet(\mathbf{X})$  of generic postconditions as in the last section, i.e., only postconditions of the two forms  $\mathbf{x}_i \doteq \bullet$  or  $\mathbf{x}_i \doteq \mathbf{x}_j + \bullet$  for globals  $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$ .

For these postconditions, we now obtain preconditions from  $\mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$  which mention constants, local and global variables, as well as  $\bullet$ . For representing such preconditions as normalised conjunctions, let us adhere to the convention that we prefer local variables over globals as reference variables of connected components whenever possible. Following the approach of [10], we use an operator  $H^\top$  which takes the representation of a weakest precondition transformer  $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$  for the body of a procedure and transforms it into the representation of the effect of a call where the latter now may provide preconditions also for equalities between locals and between locals and globals.

One ingredient of this operator is the weakest precondition transformer corresponding to the transformation  $\text{enter}$  from the concrete semantics. We define  $\text{enter}^\top : \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y}) \rightarrow \mathbb{E}_\bullet(\mathbf{X})$  by

$$\text{enter}^\top(E) = \forall y_1, \dots, y_l. E$$

For a weakest precondition transformer for the body of a called procedure  $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$ , the composition  $\text{enter}^\top \circ f^\top$  thus is a transformation from  $\mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X})$  where the logical variable  $\bullet$  represents the subexpression from the postcondition which only depends on data not modified during the call. The second ingredient for obtaining the effect of a call therefore is again a (suitably adapted) operator  $\text{ext}^\top$  which now extends a transformer  $g^\top : \mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X})$  to a transformer from  $\mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y}) \rightarrow \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$ . This operator is defined as follows.

For a single equality  $e$  involving globals, we define

$$\begin{aligned} \text{ext}^\top(g^\top)(\mathbf{x}_i \doteq t) &= g^\top(\mathbf{x}_i \doteq \bullet)[t/\bullet] \\ \text{ext}^\top(g^\top)(\mathbf{x}_i \doteq \mathbf{x}_j + a \bullet + b) &= g^\top(\mathbf{x}_i \doteq \mathbf{x}_j + \bullet)[a \bullet + b/\bullet] \end{aligned}$$

for globals  $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$  and terms  $t$  of the form:

$$t ::= a \bullet + b \mid \mathbf{y}_j + a \bullet + b$$

for constants  $a, b$  and local variables  $\mathbf{y}_j$ .

For an equality  $e$  which only contains locals and  $\bullet$ , we define:

$$\text{ext}^\top(g^\top)(e) = \begin{cases} \top & \text{if } g^\top(\mathbf{x}_1 \doteq \bullet) = \top \\ e & \text{otherwise} \end{cases}$$

This definition indicates that equalities between locals of the caller are left unchanged by the called procedure — provided the called procedure terminates. (Recall that a procedure

definitely does not terminate iff its weakest precondition transformer transforms  $\mathbf{x}_1 \doteq \bullet$  into  $\top$ .) Finally for a conjunction  $E = e_1 \wedge \dots \wedge e_r$ , we define:

$$\text{ext}^\top(g^\top)(E) = \text{ext}^\top(g^\top)(e_1) \wedge \dots \wedge \text{ext}^\top(g^\top)(e_r)$$

The operator  $H^\top$  which determines the weakest precondition transformer of type  $\mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y}) \rightarrow \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$  for a call from a weakest precondition transformer  $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$  for the body of the procedure then is defined by:

$$H^\top(f^\top) = \text{ext}^\top(\text{enter}^\top \circ f^\top)$$

In order to relate the weakest precondition transformers to the concrete semantics, we extend the notions of *uniformity* and *extended* concrete transformers from section 5 to deal with locals as well. Now, a completely distributive transformer  $f : 2^{\mathbb{Z}^{k+l}} \rightarrow 2^{\mathbb{Z}^{k+l}}$  is called *uniform* if  $f\{(x, y)\} = \emptyset$  for some pair  $(x, y)$  implies  $f\{(x', y')\} = \emptyset$  for all pairs  $(x', y') \in \mathbb{Z}^{k+l}$ . The corresponding *extended* transformer  $\text{ext}(f) : 2^{\mathbb{Z}^{k+l+1}} \rightarrow 2^{\mathbb{Z}^{k+l+1}}$  for a completely distributive uniform transformer is defined by  $\text{ext}(f)\{(x, y, c)\} = \{(x', y', c) \mid (x', y') \in f(\{(x, y)\})\}$ . Analogously to lemma 7 we obtain:

**Lemma 10.** *Let  $f : 2^{\mathbb{Z}^{k+l}} \rightarrow 2^{\mathbb{Z}^{k+l}}$  denote a concrete transformer which is completely distributive and uniform. Furthermore, let  $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$  denote a weakest precondition transformer where for all  $Z \subseteq \mathbb{Z}^{k+l+1}$  and  $e \in \mathbb{P}_\bullet(\mathbf{X})$ ,  $\text{ext}(f)(Z) \models e$  iff  $Z \models f^\top(e)$ . Then also*

$$\text{ext}(H(f))(Z) \models E \quad \text{iff} \quad Z \models H^\top(f^\top)(E)$$

for all  $Z \subseteq \mathbb{Z}^{k+l+1}$  and  $E \in \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$ .

Consider the modified constraint system  $S^\top$  for the weakest precondition transformers of procedures using the operator  $H^\top$  to deal with function calls.

$$\begin{array}{ll} S^\top[r_q] \sqsubseteq \text{Id} & r_q \text{ exit point of procedure } q \\ S^\top[u] \sqsubseteq H^\top(S^\top[s_q]) \circ S^\top[v] & (u, q(), v) \text{ a call edge, } s_q \text{ entry point of } q \\ S^\top[u] \sqsubseteq \llbracket s \rrbracket^\top \circ S^\top[v] & (u, s, v) \text{ an assignment edge} \end{array}$$

Note that the composition  $H^\top(f^\top) \circ g^\top$  for completely distributive transformers  $f^\top, g^\top$  takes time  $\mathcal{O}(k^2 \cdot (k+l)^2)$ , since the weakest precondition of a generic postcondition is represented by a normalised conjunction, possibly containing globals, locals, and  $\bullet$ , and consequently contains  $\mathcal{O}(k+l)$  many equalities. Extending theorem 2 to programs involving local variables, we obtain:

**Theorem 4.** *Assume that  $Z \subseteq \mathbb{Z}^{k+l+1}$  and  $e \in \mathbb{P}_\bullet(\mathbf{X})$ . Then, for every program point  $u$ :  $\text{ext } S[u](Z) \models e$  iff  $Z \models S^\top[u](e)$ .*

*Proof.* The proof is by fixpoint induction. For  $i \geq 0$ , let  $S_i[u], S_i^\top[u]$  denote the  $i$ -th approximation to the least and greatest solutions of the constraint systems  $S$  and  $S^\top$ , respectively.

*Claim.* For every  $i \geq 0$  and every program point  $u$ , the following holds:

1.  $S_i[u]$  is uniform;
2.  $\text{ext } S_i[u](Z) \models e$  iff  $Z \models S_i^\top[u](e)$ .

We only prove the second assertion of this claim. First assume  $i = 0$ . Then  $\text{ext } S_i[u](Z) = \emptyset$ . Therefore, for every  $e$ ,  $\text{ext } S_i[u](Z) \models e$ . Since also  $S_i^\top[u](e) = \text{true}$ , the assertion holds.

Now assume  $i > 0$ . Then, in the concrete semantics,  $\text{ext } S_i[u](Z)$  is the union of sets  $S_{i-1}[v](\text{ext } \llbracket s \rrbracket_{i-1}^\sharp Z)$  for edges  $(u, s, v)$  where  $\llbracket s \rrbracket_{i-1}^\sharp$  is the transformer corresponding to the label  $s$  according to the values of unknowns from the  $(i-1)$ -th iteration. Likewise for every  $e$ ,  $S_i^\top[u](e)$  is the conjunction of the preconditions  $\llbracket s \rrbracket_{i-1}^\top(S_{i-1}^\top[v](e))$  for every edge  $(u, s, v)$  where  $\llbracket s \rrbracket_{i-1}^\top$  is the weakest precondition transformer corresponding to the label  $s$  according to the values of unknowns from the  $(i-1)$ -th iteration. Therefore, it suffices to prove for every edge  $(u, s, v)$  that  $\text{ext } S_{i-1}[v](\text{ext } \llbracket s \rrbracket_{i-1}^\sharp Z) \models e$  iff  $Z \models \llbracket s \rrbracket_i^\top(S_{i-1}^\top[v](e))$ . Consider the case of a procedure call  $s \equiv q()$ . Then

$$\begin{aligned} S_{i-1}[v](\llbracket s \rrbracket_{i-1}^\sharp Z) &= S_{i-1}[v](\text{H}(S_{i-1}[s_q]) Z) \quad \text{and} \\ \llbracket s \rrbracket_i^\top(S_{i-1}^\top[v](e)) &= \text{H}^\top(S_{i-1}^\top[s_q])(S_{i-1}^\top[v](e)). \end{aligned}$$

We have:

$$\text{ext } S_{i-1}[v](\text{ext } \text{H}(S_{i-1}[s_q]) Z) \models e \quad \text{iff} \quad \text{ext } \text{H}(S_{i-1}[s_q]) Z \models S_{i-1}^\top[v](e)$$

by induction hypothesis. Furthermore,

$$\text{ext } \text{H}(S_{i-1}[s_q]) Z \models S_{i-1}^\top[v](e) \quad \text{iff} \quad Z \models \text{H}^\top(S_{i-1}^\top[s_q])(S_{i-1}^\top[v](e))$$

by induction hypothesis for the transformers for  $s_q$  and lemma 10. Note that for the application of this lemma we rely on the complete distributivity of all transformers  $S_{i-1}[u]$  and  $S_{i-1}^\top[u]$ .

This completes the proof of the claim for the case of a function call. We omit the remaining cases of assignments or non-deterministic assignments.

Using our claim, we argue that

$$\begin{aligned} \text{ext } S[u] Z \models e &\text{ iff } \forall i \geq 0. \text{ext } S_i[u] Z \models e \\ &\text{ iff } \forall i \geq 0. Z \models S_i^\top[u](e) \\ &\text{ iff } Z \models S^\top[u](e) \end{aligned}$$

for all sets  $Z \subseteq \mathbb{Z}^{k+l+1}$  and equalities  $e \in \mathbb{P}_\bullet(\mathbf{X})$ . □

In order to determine for every program point  $u$ , the conjunction of all equalities (now referring to local as well as global variables) which are valid when reaching program point  $u$ , we modify the constraint system  $C^\sharp$ . For that, we require a transformation  $\text{enter}^\sharp$  which determines the conjunction of equalities at procedure entry from the conjunction of equalities before the procedure call. Since all locals are uninitialised at procedure entry, this transformation removes all equalities involving locals. Accordingly, the transformation  $\text{enter}^\sharp$  is defined by:

$$\text{enter}^\sharp(E) = \exists^\sharp \mathbf{y}_1 \dots \exists^\sharp \mathbf{y}_l. E$$

For a weakest precondition transformer  $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \rightarrow \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$  for the body of a procedure, we define  $H^\sharp(f^\top) : \mathbb{E}(\mathbf{X} \cup \mathbf{Y}) \rightarrow \mathbb{E}(\mathbf{X} \cup \mathbf{Y})$  as follows. Let  $E \in \mathbb{E}(\mathbf{X} \cup \mathbf{Y})$ . First assume that  $f^\top(\mathbf{x}_1 \dot{=} \bullet) = \top$ . Then  $H^\sharp(E) = \perp$ . Otherwise,  $H^\sharp(f^\top)(E)$  is the conjunction of all equalities which only involve locals and are implied by  $E$ , together with all equalities  $e[t/\bullet]$  for generic postconditions  $e$  and terms  $t$  of the form  $c$  or  $\mathbf{y}_j + c$  ( $c \in \mathbb{Z}, \mathbf{y}_j \in \mathbf{Y}$ ) for which  $E \implies (H^\top(f^\top)(e))[t/\bullet]$ . In particular for  $E = \perp$ , we have  $H^\sharp(f^\top)(E) = \perp$  as well. The following lemma states that the operator  $H^\sharp$  allows to determine all valid equalities after a call from the conjunction of valid equalities before the call and the weakest precondition transformer for the procedure body.

**Lemma 11.** *Assume  $Z \subseteq \mathbb{Z}^{k+l}$  and  $E = \alpha(Z)$  is the conjunction of all equalities  $e$  over  $\mathbf{X} \cup \mathbf{Y}$  with  $Z \models e$ . Let  $f : 2^{\mathbb{Z}^{k+l}} \rightarrow 2^{\mathbb{Z}^{k+l}}$  be completely distributive and uniform, and let  $f^\top$  be a weakest precondition transformer. Assume as in lemma 10 that for all  $e \in \mathbb{P}_\bullet(\mathbf{X})$ ,  $\text{ext}(f)(Z) \models e$  iff  $Z \models f^\top(e)$  for all subsets  $Z \subseteq \mathbb{Z}^{k+l+1}$  and elements  $e \in \mathbb{P}_\bullet(\mathbf{X})$ . Then, for every  $E' \in \mathbb{E}(\mathbf{X} \cup \mathbf{Y})$ ,  $H(f)(Z) \models E'$  iff  $H^\sharp(f^\top)(E) \sqsubseteq E'$ .*

We put up the following constraint system over  $\mathbb{E}(\mathbf{X} \cup \mathbf{Y})$  using the operators  $\text{enter}^\sharp$  and  $H^\sharp$ :

$$\begin{aligned} C^\sharp[s_{\text{main}}] &\sqsupseteq \text{enter}^\sharp(\top) \\ C^\sharp[s_q] &\sqsupseteq \text{enter}^\sharp(C^\sharp[u]) && (u, q(), -) \text{ a call edge} \\ C^\sharp[v] &\sqsupseteq H^\sharp(S^\top[s_q])(C^\sharp[u]) && (u, q(), v) \text{ a call edge} \\ C^\sharp[v] &\sqsupseteq \llbracket s \rrbracket^\sharp(C^\sharp[u]) && (u, s, v) \text{ an assignment edge} \end{aligned}$$

The least solution of this constraint system precisely characterises for every program point  $u$ , the conjunction of all equalities from  $\mathbb{E}(\mathbf{X} \cup \mathbf{Y})$  which are valid when program execution reaches  $u$ . Summarising, we obtain:

**Theorem 5.** *The set of all valid equalities for an interprocedural program of size  $n$  with  $k$  global variables and  $l$  local variables can be computed in time  $\mathcal{O}(n \cdot k^2 \cdot (k+l)^2)$ .*

## Practical Improvements

Typically, a procedure accesses only few global variables. Therefore, our goal is to reduce the size of the representation of the abstract effects of procedures by tracking variable differences only for those variables which are really modified. For this purpose, we define a function  $B$  which determines for a weakest precondition transformer  $f$  the set of variables whose values stay constantly unmodified when applying the transformer:

$$B(f) = \{\mathbf{x}_i \in \mathbf{X} \mid f(\mathbf{x}_i \dot{=} \bullet) = (\mathbf{x}_i \dot{=} \bullet)\}$$

Assume  $\mathbf{x}_j \in B(f)$ . Then,  $f(\mathbf{x}_i \dot{=} \mathbf{x}_j + \bullet)$  can be recovered from  $f(\mathbf{x}_i \dot{=} \bullet)$  by replacing  $\bullet$  with  $\mathbf{x}_j + \bullet$  in right-hand sides of equalities.

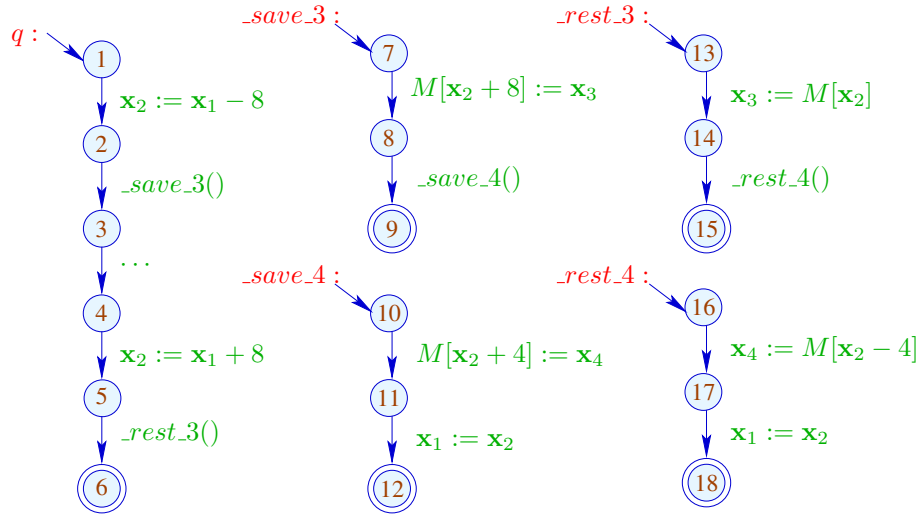
Therefore, we may proceed as follows. First we tabulate the values of  $f$  for generic postconditions of the form  $\mathbf{x}_i \dot{=} \bullet$ . From these values we can determine the set  $B(f)$ . Then it suffices to tabulate in addition only the preconditions for  $\mathbf{x}_i \dot{=} \mathbf{x}_j + \bullet$  for  $\mathbf{x}_i, \mathbf{x}_j \notin B(f)$ . This means that we can reduce the size of the table for the effect of a procedure essentially to those variables which are actually touched by this procedure.

In order to reduce also the number of generic postconditions of the form  $x_i \doteq \bullet$ , we can determine a sufficiently large subset  $B'(f)$  of  $B(f)$  by a straightforward syntactic analysis in advance. Then it suffices to determine preconditions for  $x_i \doteq \bullet$  only for variables  $x_i \notin B'(f)$ .

## 8 Example Application

As an example for our analysis, consider the code generated for a procedure  $q$  where two registers are saved and restored by means of auxiliary functions as described in the introduction. In accordance with the naming conventions of this paper, the registers corresponding to global variables of  $q$  are denoted by  $x_1, x_2, x_3, x_4$ . The resulting control flow graph is given in figure 4. For the analysis, memory writes are ignored and reads from memory into a variable are abstracted with corresponding non-deterministic assignments. The global variable  $x_1$  represents the stack pointer while the global  $x_2$  serves as an auxiliary register.

We assume that the instructions of the function body of  $q$  (indicated via dots at program point 3) do not modify the stack pointer  $x_1$ .



**Fig. 4.** Coding convention for saving and restoring registers  $x_3, x_4$ .

Within the prologue of procedure  $q$ , 8 bytes of memory are reserved for the callee-save registers, i.e.,  $x_3$  and  $x_4$ , which are saved via a call to  $\_save\_3$ . Additionally, the stack pointer  $x_1$  is modified at program point 11 in order to account for the stack growth for saving  $x_3$  and  $x_4$ . Then, after executing the body of  $q$  its callee-save registers are restored (in the call to  $\_rest\_3$ ) and additionally the memory for saving these registers is freed (program point 17).

Our goal is to verify the invariant that any execution of procedure  $q$  leaves the stack pointer unchanged, i.e., that after any call to  $q$ , the stack pointer has the same value as before the call. Note that this property is violated by the procedures `_save_3`, `_save_4`, `_rest_3` and `_rest_4`.

Let us first compute the weakest precondition of the generic postcondition  $\mathbf{x}_2 \doteq \mathbf{x}_1 + \bullet$  for the procedures `_save_3` and `_rest_3` (program points 7, 13), respectively. Note first that the memory write instructions in procedures `_save_4` and `_save_3` do not affect register equalities and thus can be ignored by the analysis. Thus, for procedure `_save_3` only the assignment to  $\mathbf{x}_1$  (in procedure `_save_4`) is relevant. Therefore, we obtain  $S^\top[7](\mathbf{x}_2 \doteq \mathbf{x}_1 + \bullet) = (\bullet \doteq 0)$  as the weakest precondition of the above generic postcondition at program point 7. For procedure `_rest_3`, i.e. at program point 13, the same precondition is computed, namely  $S^\top[13](\mathbf{x}_2 \doteq \mathbf{x}_1 + \bullet) = (\bullet \doteq 0)$ . We may infer from these preconditions that the equality  $\mathbf{x}_1 \doteq \mathbf{x}_2$  is valid upon termination of both procedures, `_save_3` and `_rest_3`.

Since it is our aim to verify that the stack pointer before and after executing the procedure body of  $q$  (i.e. at program points 1 and 6) is the same, let us now consider on the one hand the conjunction of all valid equalities and on the other hand the weakest precondition for the generic equality  $\mathbf{x}_1 \doteq \bullet$ , for every program point  $i$  of procedure  $q$ . The WP transformers and the valid conjunctions for every program point  $i$  can be looked up in the following table:

$i$	$S^\top[i](\mathbf{x}_1 \doteq \bullet)$	$C^\sharp[i]$
1	$\mathbf{x}_1 \doteq \bullet$	$\top$
2	$\mathbf{x}_2 \doteq \bullet - 8$	$\mathbf{x}_2 \doteq \mathbf{x}_1 - 8$
3	$\mathbf{x}_1 \doteq \bullet - 8$	$\mathbf{x}_2 \doteq \mathbf{x}_1$
4	$\mathbf{x}_1 \doteq \bullet - 8$	$\mathbf{x}_2 \doteq \mathbf{x}_1$
5	$\mathbf{x}_2 \doteq \bullet$	$\mathbf{x}_2 \doteq \mathbf{x}_1 + 8$
6	$\mathbf{x}_1 \doteq \bullet$	$\mathbf{x}_2 \doteq \mathbf{x}_1$

The valid conjunction of equalities for every program point of  $q$  (cf.  $C^\sharp[i]$ ) provides us with information about the relation of variables  $\mathbf{x}_1$  and  $\mathbf{x}_2$ , but does not allow us to deduce a statement about the relation of the stack pointer value at procedure start to its value at procedure exit. One way to obtain such a statement would be to instrument procedure  $q$  with a new local variable  $\mathbf{x}'_1$  that stores the initial value of  $\mathbf{x}_1$ . Then our analysis would yield the equality  $\mathbf{x}_1 \doteq \mathbf{x}'_1$  at program point 6 that witnesses preservation of the stack pointer rather directly. We note, however, that the above weakest precondition computation for procedure  $q$  (cf.  $S^\top[i](\mathbf{x}_1 \doteq \bullet)$ ) already allows us to infer the stack pointer invariant indirectly: From the fact that the WP transformer for  $\mathbf{x}_1 \doteq \bullet$  yields the same equality  $\mathbf{x}_1 \doteq \bullet$  for the start point of procedure  $q$  we see that the value  $\mathbf{x}_1$  stays the same in any execution of the procedure whatever the initial value of  $\mathbf{x}_1$  may be. Note that it is essential that our analysis interprets statements of the form  $\mathbf{x}_i \doteq \mathbf{x}_j + b$  in scenarios like this one. Otherwise it would be impossible to establish the stack invariant because of the statements that shift the value of the stack pointer.

## 9 Linear Two Variable Equalities

Now, we extend the variable difference analysis, described in the previous sections, in order to deal with general linear two variable equalities i.e., equalities of the form:  $\mathbf{x}_i \doteq b$  or  $\mathbf{x}_i \doteq a\mathbf{x}_j + b$  where  $a, b \in \mathbb{Q}$ . Let  $\mathbb{P}(\mathbf{X})$  denote the set of all equalities of this form. The more general form of invariants allows us to extend the analysis to handle precisely all linear assignments where right-hand sides contain at most one variable, i.e., which are of the form  $\mathbf{x}_i := b$  or  $\mathbf{x}_i := a\mathbf{x}_j + b$ . For simplicity, we consider global variables only, but the methods presented here can also be extended to local and global variables along the line of section 7.

Any satisfiable conjunction  $E$  of equalities from  $\mathbb{P}(\mathbf{X})$  can be brought into a normal form with the following properties:

- If  $E$  has an equality  $\mathbf{x}_i \doteq b$ , then  $\mathbf{x}_i$  does not occur in any other equality from  $E$ .
- If  $E$  has an equality  $\mathbf{x}_i \doteq a\mathbf{x}_j + b$  for  $a \neq 0$ , then  $i > j$  and  $E$  has no other equality containing  $\mathbf{x}_i$ .

In particular, this means that  $E$  consists of at most  $k$  equalities. Note that this normal form corresponds to the *reduced row-echelon form* (RREF) known from linear algebra.

As in the case of variable equalities alone, time  $\mathcal{O}(k^2 + r)$  suffices for an arbitrary conjunction of  $r$  equalities to prove that it is unsatisfiable or in case it is satisfiable, to compute its equivalent normal form.

An algorithm for computing the least upper bound of two conjunctions  $E_1, E_2$  in normal form can be obtained as a generalisation of the corresponding algorithm for variable differences. A corresponding algorithm which runs in time  $\mathcal{O}(k \cdot \log(k))$  is presented in appendix A.

### Representation of procedure effects

We are interested in program invariants of the form  $\mathbf{x}_i \doteq b$  or  $\mathbf{x}_i \doteq a \cdot \mathbf{x}_j + b$  for rational numbers  $a, b \in \mathbb{Q}$ . Again, we represent the effect of a procedure by means of its weakest precondition for postconditions of this form. In order to deal with all postconditions simultaneously which only differ in the constants  $a, b$ , we introduce *parameters*  $\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}$  and consider *generic* postconditions of the forms

$$\mathbf{a}_1\mathbf{x}_i \doteq \mathbf{b} \quad \text{or} \quad \mathbf{a}_1\mathbf{x}_i \doteq \mathbf{a}_2\mathbf{x}_j + \mathbf{b}$$

Note that there are at most  $\mathcal{O}(k^2)$  postconditions of these forms. Recall that we precisely only deal with assignments  $\mathbf{x}_j := t$  where the right-hand sides  $t$  contain at most one variable, i.e., are of the form  $t_1\mathbf{x}_j + t_0$  for  $t_0, t_1 \in \mathbb{Q}$ .

Then, the precondition which we may obtain for a procedure for a given generic postcondition is a conjunction of equalities each of which is of one of the following types:

- (1)  $\mathbf{a}_1\mathbf{x}_{i'} \doteq c\mathbf{a}_2\mathbf{x}_{j'} + t$
- (2)  $\mathbf{a}_2\mathbf{x}_{i'} \doteq c\mathbf{a}_2\mathbf{x}_{j'} + t$
- (3)  $\mathbf{b} \doteq c_1\mathbf{a}_1 + c_2\mathbf{a}_2$
- (4)  $\mathbf{a}_1 \doteq c_2\mathbf{a}_2$
- (5)  $\mathbf{a}_2 \doteq 0$

for variables  $\mathbf{x}_{i'}, \mathbf{x}_{j'} \in \mathbf{X}$ , expression  $t$  of the form  $c_0\mathbf{b} + c_1\mathbf{a}_1 + c_2\mathbf{a}_2$  where  $c, c_0, c_1, c_2 \in \mathbb{Q}$ , and where for type 2,  $i' > j'$  whenever  $c \neq 0$ . Note that *every* conjunction of such equalities is satisfiable by choosing value 0 for  $\mathbf{b}$ ,  $\mathbf{a}_1$  and  $\mathbf{a}_2$ .

Any conjunction can be brought into RREF w.r.t. to the monomials  $\mathbf{a}_1\mathbf{x}_{i'}, \mathbf{a}_2\mathbf{x}_{j'}$ ,  $\mathbf{b}$ ,  $\mathbf{a}_1$  and  $\mathbf{a}_2$  (in this order) — which again uses only equalities of types 1 through 4. Due to the RREF, an equality of type 3 implies that no other equality has occurrences of  $\mathbf{b}$ . Moreover, we make the following two additional assumptions on the normal form:

- If an equality of type 4 is present, then also all occurrences of  $\mathbf{a}_1$  in monomials  $\mathbf{a}_1\mathbf{x}_{i'}$  are removed.
- If an equality of type 5 is present, then all monomials  $\mathbf{a}_2\mathbf{x}_{j'}$  are removed.

For every conjunction  $E$  in this normal form, we have:

- If  $\mathbf{a}_1\mathbf{x}_i \doteq t$  occurs in  $E$  for some term  $t$ , then no other equality in  $E$  contains  $\mathbf{a}_1\mathbf{x}_i$ .
- If  $\mathbf{a}_2\mathbf{x}_i \doteq c\mathbf{a}_2\mathbf{x}_j + t$  occurs in  $E$  for some term  $t$ , then no other equality in  $E$  contains  $\mathbf{a}_2\mathbf{x}_i$ . Moreover, if  $c \neq 0$ , then  $i > j$ .

These properties imply that  $E$  has at most  $k$  equalities of type 1 and at most  $k$  equalities of type 2. Since there is at most one equality of each of the types 3, 4, and 5,  $E$  has at most  $2k + 3$  equalities. Overall, we conclude that every satisfiable conjunction can be uniquely represented by a conjunction of at most  $\mathcal{O}(k)$  equalities each of which is of size  $\mathcal{O}(1)$ . Moreover, the standard algorithm for reducing to reduced row echelon form can be modified to compute for a satisfiable conjunction of  $r$  equalities a reduced conjunction of equalities in time  $\mathcal{O}(k^2 + r)$ .

*Example 4.*

Consider the following conjunction  $E$  of parametric equalities in normal form:

$$\begin{aligned} (\mathbf{a}_1\mathbf{x}_5 \doteq 2\mathbf{a}_2\mathbf{x}_3 + 3\mathbf{b} + 7\mathbf{a}_1 + 2\mathbf{a}_2) \wedge \\ (\mathbf{a}_2\mathbf{x}_4 \doteq 3\mathbf{a}_2\mathbf{x}_3 + 2\mathbf{b} - 3\mathbf{a}_1 - 1\mathbf{a}_2) \wedge \\ (\mathbf{a}_2\mathbf{x}_2 \doteq 2\mathbf{b} + 5\mathbf{a}_1 + 3\mathbf{a}_2) \end{aligned}$$

together with the equality  $e = (\mathbf{a}_1\mathbf{x}_5 \doteq 4\mathbf{a}_2\mathbf{x}_2 + 7\mathbf{b} + 9\mathbf{a}_1 + 4\mathbf{a}_2)$ .

Subtracting from  $e$  the first equality of  $E$ , we obtain:

$$0 \doteq -2\mathbf{a}_2\mathbf{x}_3 + 4\mathbf{a}_2\mathbf{x}_2 + 4\mathbf{b} + 2\mathbf{a}_1 + 2\mathbf{a}_2$$

or

$$\mathbf{a}_2\mathbf{x}_3 \doteq 2\mathbf{a}_2\mathbf{x}_2 + 2\mathbf{b} + \mathbf{a}_1 + \mathbf{a}_2$$

Using the last equality from  $E$ , the occurrence of  $\mathbf{a}_2\mathbf{x}_2$  in the right-hand can be removed which gives us:

$$\mathbf{a}_2\mathbf{x}_3 \doteq 6\mathbf{b} + 11\mathbf{a}_1 + 7\mathbf{a}_2 .$$

The resulting equality cannot be reduced further. Instead it can be used to remove occurrences of  $\mathbf{a}_2\mathbf{x}_3$  in the right-hand sides of the equalities in  $E$ . As the normal form for  $e \wedge E$  we therefore obtain:

$$\begin{aligned} (\mathbf{a}_1\mathbf{x}_5 \doteq 15\mathbf{b} + 29\mathbf{a}_1 + 16\mathbf{a}_2) \wedge \\ (\mathbf{a}_2\mathbf{x}_4 \doteq 20\mathbf{b} + 30\mathbf{a}_1 + 20\mathbf{a}_2) \wedge \\ (\mathbf{a}_2\mathbf{x}_3 \doteq 6\mathbf{b} + 11\mathbf{a}_1 + 7\mathbf{a}_2) \wedge \\ (\mathbf{a}_2\mathbf{x}_2 \doteq 2\mathbf{b} + 5\mathbf{a}_1 + 3\mathbf{a}_2) \quad \square \end{aligned}$$

The required space for the representation of summary functions is  $\mathcal{O}(k^3)$ : for each of the  $\mathcal{O}(k^2)$  generic postconditions, we provide a parametric precondition of size  $\mathcal{O}(k)$ . As we show next, the time for computing the composition operation in our representation of weakest precondition transformers is  $\mathcal{O}(k^4)$ . Consider the composition of two weakest precondition transformers  $f$  and  $g$ . For that, consider a single generic equality  $e$ . In order to compute the weakest precondition  $f(g(e))$ , we first apply  $g$  to  $e$  giving us a conjunction  $E = g(e)$  of  $\mathcal{O}(k)$  equalities. Applying  $f$  to each of the equalities in  $E$  results in  $\mathcal{O}(k)$  conjunctions each consisting of  $\mathcal{O}(k)$  equalities. The conjunction of these conjunctions thus has  $\mathcal{O}(k^2)$  equalities which can be normalised in time  $\mathcal{O}(k^2 + k^2) = \mathcal{O}(k^2)$ . By repeating this for all  $\mathcal{O}(k^2)$  generic postconditions we thus have succeeded to compute a representation for the composition of  $f$  and  $g$  in time  $\mathcal{O}(k^4)$ .

It remains to compute the strongest postcondition of a procedure call given the WP transformer  $f^\top$  for the procedure. Observe first, that, for any generic postcondition  $e$ ,  $f^\top e$  is the empty conjunction (representing true) if and only if the procedure in question has no terminating execution. In this case the strongest postcondition is always false irrespective of the equations valid at the call because the procedure never returns. Also a non-satisfiable conjunction (i.e. a conjunction equivalent to false) at the call site gives rise to postcondition false. So let us assume that  $f^\top e$  is non-empty for all generic postconditions  $e$  and that at the call to the procedure a satisfiable conjunction  $E$  is valid. Then we determine the strongest postcondition of  $E$  as the conjunction of all equalities whose weakest precondition w.r.t.  $f^\top$  is implied by  $E$ .

First, we determine the equalities of the form  $\mathbf{x}_i = b$  valid upon return from the procedure given precondition  $E$ , i.e. the variables that are constant. Let  $E' = f^\top e$  denote the weakest precondition for  $e \equiv \mathbf{a}_1 \mathbf{x}_i \doteq \mathbf{b}$  as provided by the transformer  $f^\top$ . Note that  $E'$  does not contain the parameter  $\mathbf{a}_2$ . As we are specifically interested in postconditions of the form  $\mathbf{x}_i \doteq b$  for some  $b \in \mathbb{Q}$  we fix the parameter  $\mathbf{a}_1$  to 1 and observe that a postcondition of this form can be valid for at most one value  $b$  (given that the procedure may terminate) as otherwise two contradictory equations would hold simultaneously. We determine this value (if any) as follows. We check first, if there is an equality of the form  $\mathbf{a}_1 \mathbf{x}_j \doteq t$  in  $E'$  such that the conjunction  $E$  does not contain an equality  $\mathbf{x}_j \doteq c$ . In this case  $E'$  is not implied by  $E$  for  $\mathbf{a}_1 = 1$  and some value of  $\mathbf{b}$  and, consequently,  $\mathbf{x}_i$  cannot be constant after the call. Otherwise, for every equality  $e'$  of the form  $\mathbf{a}_1 \mathbf{x}_j \doteq t$  occurring in  $E'$ ,  $E$  contains an equality  $\mathbf{x}_j \doteq c$ . This gives rise to the equation  $0 \doteq t[1/\mathbf{a}_1] - c$ . Let  $E_1$  denote the conjunction of all these equations. Furthermore, let  $E_0$  denote the (possibly empty) conjunction of equalities in  $E'$  not containing program variables  $\mathbf{x}_{j'}$ . Then the equality  $\mathbf{x}_i \doteq \mathbf{b}$  holds after the call iff  $\mathbf{b}$  satisfies the conjunction  $E_0[1/\mathbf{a}_1] \wedge E_1$ . This is a system of linear equations with at least one equation over the single variable  $\mathbf{b}$ . Thus, there is either a single solution  $b \in \mathbb{Q}$  or no solution at all. In the first case, the equation  $\mathbf{x}_i = b$  is valid after the call; in the second case  $\mathbf{x}_i$  is not constant after the call.

Having determined in this way all equalities of the form  $\mathbf{x}_i \doteq b$  it remains to determine the valid two variable equalities of the form  $\mathbf{x}_i \doteq a\mathbf{x}_j + b$ . As there can be no equalities between a constant and a non-constant variable and the two variable equalities valid between the constant variables are already implied by the equations expressing constancy, it suffices to consider two variable equalities for non-constant variables  $\mathbf{x}_i$

and  $\mathbf{x}_j$ . So assume that  $e$  is a generic postcondition of the form  $\mathbf{a}_1 \mathbf{x}_i \doteq \mathbf{a}_2 \mathbf{x}_j + \mathbf{b}$  where neither  $\mathbf{x}_i$  nor  $\mathbf{x}_j$  could be proven to be constant after the call. Let again  $E'$  denote the conjunction  $E' = f^\top e$ . As in the case of constants, we observe that there can be at most one pair of values  $(a_2, b) \in \mathbb{Q}^2$  such that  $\mathbf{x}_i \doteq a_2 \mathbf{x}_j + b$  is valid as otherwise  $\mathbf{x}_2$  would be constant after the call. In order to determine this pair (if any) we look for  $(a_2, b) \in \mathbb{Q}^2$  such that  $E'[1/\mathbf{a}_1][a_2/\mathbf{a}_2][b/\mathbf{b}]$  is implied by  $E$ . For this we first check whether for every equality  $e'$  in  $E'$  the following conditions holds:

- If  $e'$  contains exactly one program variable  $\mathbf{x}_j$ , then  $E$  contains an equality  $\mathbf{x}_j \doteq c$  for some constant  $c$ ;
- If  $e'$  contains the two program variables  $\mathbf{x}_j, \mathbf{x}_{j'}$ , then there is a program variable  $\mathbf{x}_k$  such that  $E$  contains both an equality  $\mathbf{x}_j \doteq c_1 \mathbf{x}_k + c_2$  and an equality  $\mathbf{x}_{j'} \doteq c'_1 \mathbf{x}_k + c'_2$ .

If either of this is not the case,  $E'[1/\mathbf{a}_1][a_2/\mathbf{a}_2][b/\mathbf{b}]$  is not implied by  $E$  for any pair  $(a_2, b) \in \mathbb{Q}^2$  and no non-trivial two-variable equality between  $\mathbf{x}_i$  and  $\mathbf{x}_j$  is valid after the call. Otherwise, we again put up a system of linear equations. For each equality satisfying the first condition, we extract the equation  $e'[1/\mathbf{a}][c/\mathbf{x}_j]$ . For each equality satisfying the second condition, we first consider the equation  $e'' \equiv e'[1/\mathbf{a}][c_1 \mathbf{x}_k + c_2/\mathbf{x}_j][c'_1 \mathbf{x}_k + c'_2/\mathbf{x}_{j'}]$  obtained from  $e'$  by substituting the occurrences of  $\mathbf{x}_j$  and  $\mathbf{x}_{j'}$  with the right-hand sides of the corresponding equalities in  $E$ . The equation  $e''$  now can be written as  $0 \doteq t_0 + t_1 \mathbf{x}_k$  where the terms  $t_0, t_1$  are affine combinations of the parameters  $\mathbf{a}_2, \mathbf{b}$ . From that, we extract the two equations  $0 \doteq t_0$  and  $0 \doteq t_1$ . Let  $E_1$  denote the conjunction of all these equations. Let  $E_0$  again denote the conjunction of all equalities in  $E'$  which do not contain program variables. Then the set of all pairs  $(a_2, b)$  for which  $E \implies E'[1/\mathbf{a}_1][a_2/\mathbf{a}_2][b/\mathbf{b}]$  is given by the set of solutions of the conjunction  $E_0[1/\mathbf{a}_1] \wedge E_1$ . The set of solutions to this system of equations is either empty, or consists of a single vector  $(a_2, b)$ . In the latter case, we add the equality  $e[1/\mathbf{a}_1][a_2/\mathbf{a}_2][b/\mathbf{b}]$  to the postcondition; in the former case no non-trivial equality is valid between  $\mathbf{x}_i$  and  $\mathbf{x}_j$ . Overall, the strongest postcondition can be computed in time  $\mathcal{O}(k^3)$ .

Summarising, we obtain the following generalisation of theorem 3.

**Theorem 6.** *The set of all equalities of the form  $\mathbf{x}_i \doteq a \mathbf{x}_j + b$  with  $a, b \in \mathbb{Q}$  which are valid for an interprocedural program of size  $n$  with  $k$  global variables can be computed in time  $\mathcal{O}(n \cdot k^4)$ .*

*Example 5.* Assume that we are given two arrays of data  $x\_arr, y\_arr$  whose data are accessed via a call to procedure *access* by using the global pointers  $x\_ptr$  and  $y\_ptr$ . Here we are interested in invariants for the memory addresses the variables  $x\_ptr$  and  $y\_ptr$  point to in every step of the *for*-loop. The program below is represented by the control flow graph shown in figure 5.

```

dataX_t x_arr[100];
dataY_t y_arr[100];
dataX_t *x_ptr;
dataY_t *y_ptr;

```

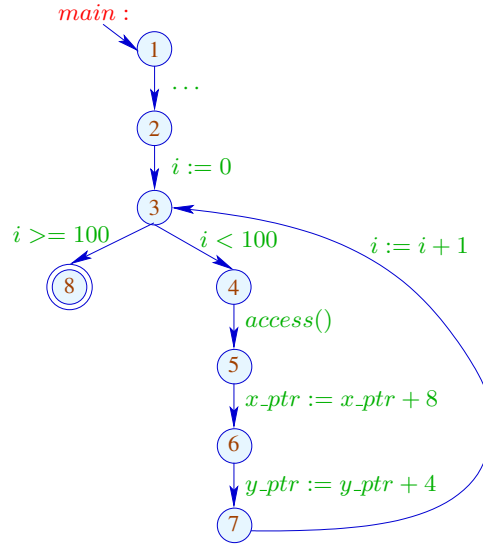


Fig. 5. Address computation

```

int main() {
  int i;
  x_ptr = &x_arr[0];
  y_ptr = &y_arr[0];

  for(i=0; i<100; i++){
    access();
    x_ptr++;
    y_ptr++;
  }
}

```

At program point 3, after the first iteration of the fixpoint algorithm the least upper bound of the two conjunctions of equalities

$$\begin{aligned}
 E_0 &= (i \doteq 0) \wedge (x\_ptr \doteq x\_arr_0) \wedge (y\_ptr \doteq y\_arr_0) \\
 E_1 &= (i \doteq 1) \wedge (x\_ptr \doteq 8 + x\_arr_0) \wedge (y\_ptr \doteq 4 + y\_arr_0)
 \end{aligned}$$

is computed. All equalities which are both implied by  $E_0$  and  $E_1$  are represented by the conjunction

$$(x\_ptr \doteq 8 \cdot i + x\_arr_0) \wedge (y\_ptr \doteq 4 \cdot i + y\_arr_0).$$

In the next round, the fixpoint iteration terminates and returns this conjunction as the invariant for program point 3.  $\square$

## 10 Conclusion

In this paper we have presented an interprocedural analysis of constant values of variables and constant differences of variables. We also extended these analyses to deal with arbitrary linear two variable equalities.

We used parametric weakest precondition transformers to represent procedure effects. The presented techniques can be seen as non-trivial generalisations of the analysis of variable equalities proposed in [10]. While inferring much stronger invariants, the algorithms still have the same worst-case time complexity  $\mathcal{O}(n \cdot k^4)$  where  $n$  is the program size and  $k$  the number of variables. We also indicated how the running time of the analysis can be improved by taking into account that many procedures may access only very few variables.

Various applications of our analysis can be envisioned. Information on definite equalities between variables can be used, e.g., during register allocation for coalescing of registers [10]. Variable differences can be used for verifying whether a low-level code adheres to the calling conventions for the stack pointer or for identifying more advanced concepts such as local variables, whereas general two-variable equalities may be useful to infer relationships between iteration variables and pointer variables.

## References

1. B. Alpern, M. Wegman, and F. K. Zadeck. Detecting equality of variables in programs. In *15th ACM Symp. on Principles of Programming Languages (POPL)*, pages 1–11, 1988.
2. C. Fecht and H. Seidl. Propagating differences: An efficient new fixpoint algorithm for distributive constraint systems. pages 90–104, 1998.
3. L. George and A. W. Appel. Iterated register coalescing. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 18(3):300–324, 1996.
4. S. Gulwani and G. C. Necula. A polynomial-time algorithm for global value numbering. In *11th Int. Static Analysis Symposium (SAS)*, pages 212–227. Springer, LNCS 3148, 2004.
5. S. Horwitz, T. W. Reps, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *22nd ACM Symp. on Principles of Programming Languages (POPL)*, pages 49–61, 1995.
6. J. Knoop and B. Steffen. The interprocedural coincidence theorem. In *Compiler Construction (CC)*, pages 125–140. LNCS 541, Springer-Verlag, 1992.
7. M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. In *31st ACM Symp. on Principles of Programming Languages (POPL)*, pages 330–341, 2004.
8. M. Müller-Olm, O. Rüthing, and H. Seidl. Checking herbrand equalities and beyond. In *Verification Meets Model-Checking and Abstract Interpretation (VMCAI)*, pages 79–96, 2005.
9. M. Müller-Olm and H. Seidl. Analysis of modular arithmetic. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(5), 2007.
10. M. Müller-Olm and H. Seidl. Upper adjoints for fast inter-procedural variable equalities. In *17th European Symposium on Programming (ESOP)*, 2008.

11. M. Müller-Olm, H. Seidl, and B. Steffen. Interprocedural herbrand equalities. In *14th European Symposium on Programming (ESOP)*, pages 31–45, 2005.
12. S. Sagiv, T. W. Reps, and S. Horwitz. Precise interprocedural dataflow analysis with applications to constant propagation. *Theor. Comput. Sci.*, 167(1&2):131–170, 1996.
13. M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. In *Program Flow Analysis: Theory and Application*, pages 189–234, 1981.
14. B. Steffen, J. Knoop, and O. Rütting. The value flow graph: A program representation for optimal program transformations. In *3rd European Symp. on Programming (ESOP)*, pages 389–405. Springer-Verlag, LNCS 432, 1990.
15. S. Zucker and K. Karhi. *System V Application Binary Interface: PowerPC Processor Supplement*, 1995. [http://refspecs.freestandards.org/elf/elfspec\\_ppc.pdf](http://refspecs.freestandards.org/elf/elfspec_ppc.pdf).

## A Appendix

Here, we present an algorithm for computing the least upper bound of satisfiable normalised conjunctions  $E_1, E_2$  of two variable equalities as introduced in section 9. In the first step, we extend the conjunctions  $E_1, E_2$  by trivial equalities  $\mathbf{x}_i \doteq \mathbf{x}_i$  such that in both  $E_1$  and  $E_2$  every variable occurs exactly once as a left-hand side. As in the case of variable differences alone, we successively partition the set of variables.

Let  $X_0$  denote the set of variables  $\mathbf{x}_i$  where the right-hand sides in  $E_1$  and  $E_2$  coincide. Then the least upper bound of  $E_1$  and  $E_2$  will have the conjunction  $E'_0 = \bigwedge_{\mathbf{x}_i \in X_0, t_i \neq \mathbf{x}_i} (\mathbf{x}_i \doteq t_i)$  if  $t_i$  is the right-hand side for  $\mathbf{x}_i$  in  $E_1$ .

Let  $X_1$  denote the set of variables  $\mathbf{x}_i$  where the right-hand sides in  $E_1$  and  $E_2$  are just constants  $c_i^{(i)}$  but do not coincide, i.e.,  $c_i^{(1)} \neq c_i^{(2)}$ . Assume that  $h$  is the least variable index with this property. This means that for every such variable  $\mathbf{x}_i \in X_1$  with  $i \neq h$ , the system of equations:

$$c_i^{(1)} = \mathbf{a}c_h^{(1)} + \mathbf{b} \quad c_i^{(2)} = \mathbf{a}c_h^{(2)} + \mathbf{b}$$

has a unique solution  $\mathbf{a} = a_i, \mathbf{b} = b_i$ . Then, the least upper bound of  $E_1$  and  $E_2$  will have the conjunction  $E'_1 = \bigwedge_{\mathbf{x}_h \neq \mathbf{x}_i \in X_1} (\mathbf{x}_i \doteq a_i \mathbf{x}_h + b_i)$ .

Let  $X_2$  denote the set of variables  $\mathbf{x}_i$  where the right-hand sides in  $E_1$  are constants  $c_i$  but the right-hand sides in  $E_2$  contain occurrences of variables, i.e., the equalities in  $E_2$  are of the form  $\mathbf{x}_i \doteq a_i \mathbf{x}_{h_i} + b_i$ . Then we define a partition  $\Pi_2$  on the set  $X_2$  where variables  $\mathbf{x}_i, \mathbf{x}_j$  are in the same equivalence class iff the following two conditions are satisfied:

- the variables on the right-hand side in  $E_2$  coincide, i.e.,  $h_i = h_j$ ;
- $\frac{c_i - b_i}{a_i} = \frac{c_j - b_j}{a_j}$ .

For one such class  $Q \in \Pi_2$ , let  $\mathbf{x}_h$  denote the variable with least index. Then, we obtain for every other variable  $\mathbf{x}_i \neq \mathbf{x}_h$  in  $Q$  the equality  $\mathbf{x}_i \doteq \frac{a_i}{a_h} \mathbf{x}_h + (b_i - \frac{a_i b_h}{a_h})$ . Let  $E'_{2,Q}$  denote the conjunction  $\bigwedge_{\mathbf{x}_h \neq \mathbf{x}_i \in Q} (\mathbf{x}_i \doteq \frac{a_i}{a_h} \mathbf{x}_h + (b_i - \frac{a_i b_h}{a_h}))$ . Then the least upper bound of  $E_1$  and  $E_2$  has the conjunction  $E'_2 = \bigwedge_{Q \in \Pi_2} E'_{2,Q}$ .

Let  $E'_3$  denote the conjunction analogous to  $E'_2$  but with the roles of  $E_1$  and  $E_2$  exchanged.

Now let  $X_4$  denote the set of variables  $\mathbf{x}_i$  occurring as left-hand sides both in  $E_1$  and  $E_2$  where the corresponding right-hand sides both contain variables, i.e.,  $E_1$  and  $E_2$  have the equalities  $\mathbf{x}_i \doteq a_i \mathbf{x}_{h_i} + b_i$  and  $\mathbf{x}_i \doteq a'_i \mathbf{x}_{h'_i} + b'_i$ , respectively. On the set  $X_4$  we define a partition  $\Pi_4$  where variables  $\mathbf{x}_i, \mathbf{x}_j$  are put into the same equivalence class iff the following three conditions are satisfied:

1. the variables occurring in the right-hand sides w.r.t.  $E_1$  and  $E_2$  coincide, i.e.,  $h_i = h_j$  and  $h'_i = h'_j$ ;
2.  $\frac{a'_i}{a_i} = \frac{a'_j}{a_j}$ ;
3.  $\frac{b_i - b'_i}{a_i} = \frac{b_j - b'_j}{a_j}$ .

For one such class  $Q \in \Pi_4$ , let again  $\mathbf{x}_h$  denote the variable with least index. As for equivalence classes of the set  $X_2$ , we obtain for every other variable  $\mathbf{x}_i \neq \mathbf{x}_h$  in  $Q$  the equality  $\mathbf{x}_i \doteq \frac{a_i}{a_h} \mathbf{x}_h + (b_i - \frac{a_i b_h}{a_h})$ . Let  $E'_{4,Q}$  denote the conjunction  $\bigwedge_{\mathbf{x}_h \neq \mathbf{x}_i \in Q} (\mathbf{x}_i \doteq \frac{a_i}{a_h} \mathbf{x}_h + (b_i - \frac{a_i b_h}{a_h}))$ . Then the least upper bound of  $E_1$  and  $E_2$  has the conjunction  $E'_4 = \bigwedge_{Q \in \Pi_4} E'_{4,Q}$ .

This completes the enumeration of equalities both implied by  $E_1$  and  $E_2$ . The remaining equalities are not in the same equivalence class w.r.t.  $E_1$  and  $E_2$  and thus would only account for trivial equalities  $\mathbf{x}_i \doteq \mathbf{x}_i$  in  $E_1 \sqcup E_2$ . Accordingly, we define the least upper bound  $E_1 \sqcup E_2$  as the conjunction

$$E_1 \sqcup E_2 = E'_0 \wedge E'_1 \wedge E'_2 \wedge E'_3 \wedge E'_4$$

Note that the sets  $X_0, \dots, X_4$  are disjoint. The required equivalence classes can again be computed in time  $\mathcal{O}(k \cdot \log(k))$ . Since the remaining computation can be done in time  $\mathcal{O}(k)$ , we conclude that the least upper bound of two reduced conjunctions can be computed in time  $\mathcal{O}(k \cdot \log(k))$ . By definition, the least upper bound of two conjunctions of equalities  $E_1 \sqcup E_2$  is given by the conjunction of all equalities implied both by  $E_1$  and  $E_2$ . For this purpose consider the following example:

*Example 6.*

Assume given the conjunctions  $E_1$  and  $E_2$  in normal form enhanced with trivial equalities:

$$\begin{aligned} E_1 = & (\mathbf{x}_1 \doteq \mathbf{x}_1) \wedge (\mathbf{x}_2 \doteq \mathbf{x}_2) \wedge (\mathbf{x}_3 \doteq \mathbf{x}_1) \wedge (\mathbf{x}_4 \doteq 3\mathbf{x}_2 + 5) \wedge (\mathbf{x}_5 \doteq 3\mathbf{x}_1 + 15) \\ & \wedge (\mathbf{x}_6 \doteq \mathbf{x}_1 + 3) \wedge (\mathbf{x}_7 \doteq \mathbf{x}_1 + 2) \wedge (\mathbf{x}_8 \doteq 7\mathbf{x}_1 + 15) \wedge (\mathbf{x}_9 \doteq 0) \wedge \\ & (\mathbf{x}_{10} \doteq 2) \wedge (\mathbf{x}_{11} \doteq 1) \wedge (\mathbf{x}_{12} \doteq 3) \end{aligned}$$

$$\begin{aligned} E_2 = & (\mathbf{x}_1 \doteq \mathbf{x}_1) \wedge (\mathbf{x}_2 \doteq \mathbf{x}_2) \wedge (\mathbf{x}_3 \doteq \mathbf{x}_2 - 5) \wedge (\mathbf{x}_4 \doteq 3\mathbf{x}_2 + 5) \wedge (\mathbf{x}_5 \doteq 3\mathbf{x}_2) \\ & \wedge (\mathbf{x}_6 \doteq \mathbf{x}_2 + 1) \wedge (\mathbf{x}_7 \doteq \mathbf{x}_2) \wedge (\mathbf{x}_8 \doteq 21\mathbf{x}_2 - 20) \wedge (\mathbf{x}_9 \doteq 1) \wedge \\ & (\mathbf{x}_{10} \doteq 4) \wedge (\mathbf{x}_{11} \doteq 2\mathbf{x}_1 - 3) \wedge (\mathbf{x}_{12} \doteq 4\mathbf{x}_1 - 5) \end{aligned}$$

In order to compute  $E_1 \sqcup E_2$ , we successively consider the sets  $X_0, \dots, X_4$  of variables as constructed by our algorithm. The first class  $X_0$  consists of the variables  $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_4$  since for these, the right-hand sides are syntactically equal w.r.t. to  $E_1$  and  $E_2$ . By eliminating the trivial equalities, we obtain for  $E'_0$  the equality  $\mathbf{x}_4 \doteq 3\mathbf{x}_2 + 5$ .

The second set  $X_1$ , speaking about variable constant equalities only, is given by the set  $\{\mathbf{x}_9, \mathbf{x}_{10}\}$ . According to our algorithm, the conjunction  $E'_1$  therefore only consists of the equality  $\mathbf{x}_{10} \doteq 2\mathbf{x}_9 + 2$ .

Considering the set  $X_2 = \{\mathbf{x}_{11}, \mathbf{x}_{12}\}$ , we obtain  $E'_2 = \mathbf{x}_{12} \doteq 2\mathbf{x}_{11} + 1$ .

The last set  $X_4$  consists of the variables  $\mathbf{x}_3, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8$ . According to our algorithm, this set is further partitioned into the equivalence classes  $Q_1 = \{\mathbf{x}_3, \mathbf{x}_5\}$ ,  $Q_2 = \{\mathbf{x}_6, \mathbf{x}_7\}$ , and the singleton class  $Q_3 = \{\mathbf{x}_8\}$ . For the first two of these classes, we choose the new reference variables  $\mathbf{x}_3$  and  $\mathbf{x}_6$ , respectively, and thus obtain:  $E'_4 = (\mathbf{x}_5 \doteq 3\mathbf{x}_3 + 15) \wedge (\mathbf{x}_7 \doteq \mathbf{x}_6 - 1)$ . Since the right-hand sides for variable  $\mathbf{x}_8$  in  $E_1$  and  $E_2$  do not coincide they are not in the same equivalence class w.r.t. to the least upper bound of  $E_1$  and  $E_2$ . Thus, this class does not contribute any non-trivial equality.

Summarising, the least upper bound  $E_1 \sqcup E_2$  is given by the normalised conjunction:

$$(\mathbf{x}_4 \doteq 3\mathbf{x}_2 + 5) \wedge (\mathbf{x}_5 \doteq 3\mathbf{x}_3 + 15) \wedge (\mathbf{x}_7 \doteq \mathbf{x}_6 - 1) \wedge (\mathbf{x}_{10} \doteq 2\mathbf{x}_9 + 2) \wedge (\mathbf{x}_{12} \doteq 2\mathbf{x}_{11} + 1).$$

□