Fast Interprocedural Linear Two-Variable Equalities

ANDREA FLEXEDER, Technische Universität München
MARKUS MÜLLER-OLM, Westfälische Wilhelms-Universität Münster
MICHAEL PETTER, Technische Universität München
HELMUT SEIDL, Technische Universität München

In this paper we provide an interprocedural analysis of linear two-variable equalities. The novel algorithm has a worst-case complexity of $\mathcal{O}(n \cdot k^4)$, where k is the number of variables and n is the program size. Thus, it saves a factor of k^4 in comparison to a related algorithm based on full linear algebra. 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 thus for interprocedurally observing stack pointer modifications as well as for an analysis of array index expressions, when analysing low-level code.

Categories and Subject Descriptors: F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs; D.2.4 [**Software Engineering**]: Software/Program Verification; D.3.4 [**Programming Languages**]: Processors—*Compilers*; *Optimization*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra

General Terms: Algorithms, Languages, Performance, Verification

Additional Key Words and Phrases: Abstract Interpretation, Interprocedural Analysis, Linear Equalities, Linear Two-Variable Equalities, Low-level Code Analysis, Static Analysis, Summary Functions, Variable Differences

ACM Reference Format:
Flexeder, A.; Müller-Olm, M; Petter, M.; Seidl, H. 2011. Fast Interprocedural Linear Two-Variable Equalities ACM Trans. Program. Lang. Syst. V, N, Article A (January YYYY), 32 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

The key task when realising interprocedural analyses along the lines of the functional approach [Sharir and Pnueli 1981; Knoop and Steffen 1992] 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 $\mathbb{D} \to \mathbb{D}$. 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 [Horwitz et al. 1995] or vector spaces of matrices in case of affine equality analyses [Müller-Olm and Seidl 2004].

Müller-Olm and Seidl [2008] present a fast interprocedural analysis for variable equalities that can be used, e.g., for register coalescing [George and Appel 1996]. 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 worst-case complexity, i.e., $\mathcal{O}(n \cdot k^4)$ where n is the program size and k the number of variables, while providing much stronger program invariants that enable new applications.

Author's addresses: A. Flexeder, M. Petter, H. Seidl, Technische Universität München, Institut für Informatik, Lehrstuhl II, Boltzmannstr. 3, 85748 Garching, Germany; M. Müller-Olm, Institut für Informatik, Fachbereich Mathematik und Informatik, Westfälische Wilhelms-Universität Münster, Einsteinstr. 62, 48149 Münster, Germany.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

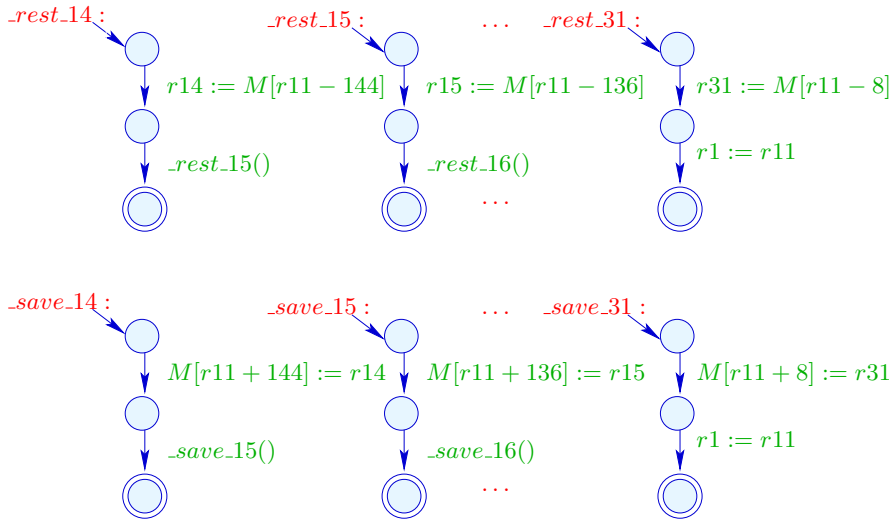


Fig. 1: Calling sequence for saving and restoring registers.

It allows us, for instance, to verify calling conventions for the stack pointer or to identify accesses to local memory locations when analysing low-level 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 local registers. With the aid of our analysis of variable differences we can nevertheless verify the assumption for non-auxiliary functions.

Example 1.1. According to the calling convention for the PowerPC architecture as specified, e.g., in [Zucker and Karhi 1995], caller-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.

The auxiliary register $r11$ holds the stack level before saving the locals, increased by the amount of 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 procedures `_save_X` and the procedures `_rest_X`. Since our interprocedural analysis tracks the two-variable equalities which hold between $r11$ and the stack pointer, it is able to reveal that the stack pointer invariant holds for all other procedure calls in the program. □

In this paper we present an analysis which infers all valid variable differences, i.e., all valid equalities of the form $\mathbf{x}_i \doteq c$ or $\mathbf{x}_i \doteq \mathbf{x}_j + c$ for some constant $c \in \mathbb{Z}$. 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 $\mathbf{x}_i \doteq b$ or $\mathbf{x}_i \doteq a\mathbf{x}_j + b$ for constants $a, b \in \mathbb{Q}$.

Related Work. Certain variable equalities can be determined as a particular case of a generalised analysis of availability of expressions called *value numbering* [Alpern et al. 1988]. Originally, this analysis tracks for basic blocks the symbolic expressions representing the values of the variables assigned to. In contrast to our analysis, value numbering leaves operator symbols uninterpreted. 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 et al. [1990]. Only recently, this problem has attracted fresh attention. In [Gulwani and Necula 2004], the authors show that the algorithm of Steffen, Knoop and Rüthing 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 [Müller-Olm et al. 2005] and [Müller-Olm et al. 2005] 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. Therefore, in principle, the precise interprocedural analyses of [Müller-Olm and Seidl 2004; Müller-Olm and Seidl 2007] 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 [Müller-Olm and Seidl 2008] an analysis was proposed that determines variable-variable together with variable-constant equalities, i.e., equalities of the form $\mathbf{x}_i \doteq \mathbf{x}_j$ and $\mathbf{x}_i \doteq b$. This algorithm improves the complexity bound by reducing the exponent to 4 in the worst case and thus results in the worst-case complexity of $\mathcal{O}(n \cdot k^4)$, where n is the program size and k is the number of program variables. In this paper we extend our approach from [Müller-Olm and Seidl 2008] 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 [Horwitz et al. 1995], where no equalities between variables (let alone linear two-variable equality relations) are tracked and to the interprocedural linear constant propagation introduced by [Sagiv et al. 1996], the resulting bound is worse only by one factor k. In [Sagiv et al. 1996] the authors consider programs with instructions of the form $\mathbf{x}_i := a \cdot \mathbf{x}_j + b$ and $\mathbf{x}_i := c$. Similar to copy-constant propagation, their analysis runs in time $\mathcal{O}(n \cdot k^3)$, but only determines constant values of variables. At the expense of introducing quadratically many variables $\mathbf{x}_{ij}$ for the differences $\mathbf{x}_i - \mathbf{x}_j$, our analysis of variable differences can be reduced to this analysis. This reduction, though, would result in an algorithm of complexity $\mathcal{O}(n \cdot k^6)$, which is by a factor k^2 worse than our algorithm, and we have no clue how potential sparsity of variable dependencies could be systematically exploited in this approach. Also, this reduction does not extend to our analysis of general two-variable equalities.

Finally, we compare our domain with approaches inferring linear *inequality* relations. In the octagon approach [Miné 2001b] at most two program variables per inequality are allowed with restrictions on the values of the coefficients, permitting only inequalities of the form $\pm\mathbf{x} \pm \mathbf{y} \leq c$. Another approach [Miné 2001a] only permits inequalities of the form $\mathbf{x} - \mathbf{y} \leq c$, respectively $\pm\mathbf{x} \leq c$, which represent polyhedra as *difference-bound matrices*. In the TVPLI domain [Simon et al. 2002] at most two program variables per inequality are allowed, however, without any restrictions on the values of the coefficients. TCMs [Sankaranarayanan et al. 2005] consist of inequalities of the form $a_1\mathbf{x}_1 + \ldots a_n\mathbf{x}_n + c \geq 0$ where the coefficients $a_1, \ldots, a_n$ have to be fixed in advance.

In contrast to our domain, all of these approaches require a widening to ensure termination of fixpoint computation. Moreover, there is no general technique to lift all these approaches to a summary-based interprocedural analysis.

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

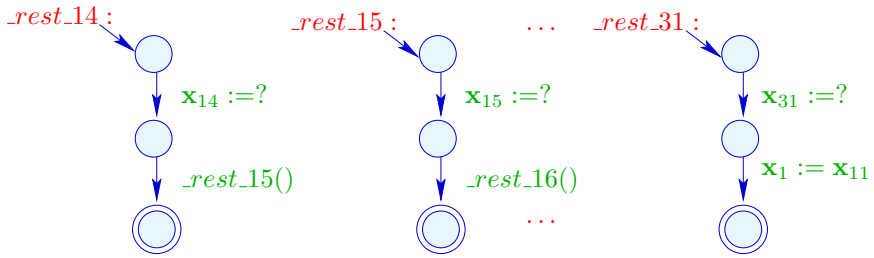


Fig. 2: Control flow representation of Fig. 1.

lattice and their complexity in Section 4. We then describe the interprocedural approach. In Section 5 we specify weakest precondition transformers and show in Section 6, how to describe procedure summaries effectively by using logical variables. We also briefly discuss a practical optimisation. Section 7 provides an application for our approach when analysing low-level code. We describe how to handle local variables in Section 8. In Section 9 we extend the variable difference analysis to general linear two-variable equalities. Finally, Section 10 contrasts the implementation of variable differences with the approach based on full linear algebra before we conclude in Section 11.

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.

Figure 2 illustrates the control flow graph for the calling sequence of system functions for restoring the local registers of a procedure from Example 1.1. Note that in our control flow representation memory access instructions are modelled via non-deterministic assignments.

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.

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

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ 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, in Section 8, we extend our approach to deal with local variables as well. For improved readability we switch from the register names rZ as used throughout our examples to the variable notation $\mathbf{x}_Z$ during the description of our framework. In the following, we assume that the program variables $\mathbf{X}$ take values from the integer domain $\mathbb{Z}$. In order to concentrate on the essentials of the analysis, we consider simplified programs. 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 $\mathbf{x}_i := a\mathbf{x}_j + b$, $\mathbf{x}_i := b$, or $\mathbf{x}_i :=?$ for $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$ and $a, b \in \mathbb{Z}$. This means that we only treat those assignments precisely where variables receive a linear term of the form $a\mathbf{x}_i + b$, while more complicated forms of assignments to a variable $\mathbf{x}_i$ are approximated by the non-deterministic assignment $\mathbf{x}_i :=?$. In this case *any* value may be assigned to the left-hand side $\mathbf{x}_i$.

A program state can be represented as a vector $x = (x_1, \ldots, x_k) \in \mathbb{Z}^k$ where $x_i \in \mathbb{Z}$ denotes the value of variable $\mathbf{x}_i$. 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 $[\![s]\!]$ of an assignment s onto a set $X \subseteq \mathbb{Z}^k$ of states:

$$\begin{array}{lcl}
[\![\mathbf{x}_i :=?]\!]\ X & = & \{x' \mid \exists\, x \in X :\ \forall\, k \neq i :\ x'_k = x_k\} \\
[\![\mathbf{x}_i := b]\!]\ X & = & \{x' \mid \exists\, x \in X :\ x'_i = b \wedge \forall\, k \neq i :\ x'_k = x_k\} \\
[\![\mathbf{x}_i := a\mathbf{x}_j + b]\!]\ X & = & \{x' \mid \exists\, x \in X :\ x'_i = ax_j + b \wedge \forall\, k \neq i :\ x'_k = x_k\}
\end{array}$$

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., [Müller-Olm and Seidl 2004; Müller-Olm and Seidl 2008], 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 $2^{\mathbb{Z}^k} \to 2^{\mathbb{Z}^k}$:

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

Here, 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 $2^{\mathbb{Z}^k}$:

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

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. In Section 9 we generalise our methods 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 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 $\bot$.

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, where node $\mathbf{0}$ represents the hard-wired value 0. An edge from $\mathbf{x}_i$ to $\mathbf{0}$ with weight b represents

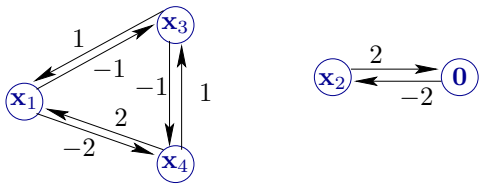


Fig. 3: Graphical representation of E.

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 3.1. 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 $\mathbf{x}_1, \ldots, \mathbf{x}_k$ 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. With respect to this array representation, two 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 3.2. *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 $E = e_1 \wedge \ldots \wedge e_r$ 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 $i = 1, \ldots, r$ 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 $\mathcal{O}(r + k^2)$. □

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 3.2, we obtain:

LEMMA 4.1. *The greatest lower bound of n normalised conjunctions of equalities $E_1 \sqcap \ldots \sqcap E_n$ can be computed in time $\mathcal{O}((n + k) \cdot k)$.*

PROOF. For computing the greatest lower bound $E = E_1 \sqcap \ldots \sqcap E_n$ we start with $E \leftarrow E_1$ and then successively compute $E \leftarrow E \wedge e$ for all $k \cdot (n-1)$ equalities e from the remaining conjunctions of equalities (applying the algorithm from Lemma 3.2). Altogether the computation of the greatest lower bound requires time $\mathcal{O}(n \cdot k)$ for array look-ups and time $\mathcal{O}(m \cdot k)$ for computing joins of connected components. As the number m of possibly occurring join operations is bounded by k, we arrive at the overall runtime $\mathcal{O}((n + k) \cdot k)$. □

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

LEMMA 4.2. *The length h of every strictly increasing chain* false $\sqsubset E_1 \sqsubset \ldots \sqsubset E_h$ *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. □

Example 4.3. Figure 4 illustrates the evolution of connected components in strictly implying normalised conjunctions of equalities $E_1 \sqsubset E_2 \sqsubset E_3$:

- $E_1 = (\mathbf{x}_2 \doteq 2) \wedge (\mathbf{x}_3 \doteq \mathbf{x}_1 + 1) \wedge (\mathbf{x}_4 \doteq \mathbf{x}_1 + 2)$.
- $E_2 = (\mathbf{x}_2 \doteq 2) \wedge (\mathbf{x}_4 \doteq \mathbf{x}_1 + 2)$.
- $E_3 = (\mathbf{x}_4 \doteq \mathbf{x}_1 + 2)$.

Figure 4 demonstrates that the number of strongly connected components is increasing with the decreasing number of relations between the variables. □

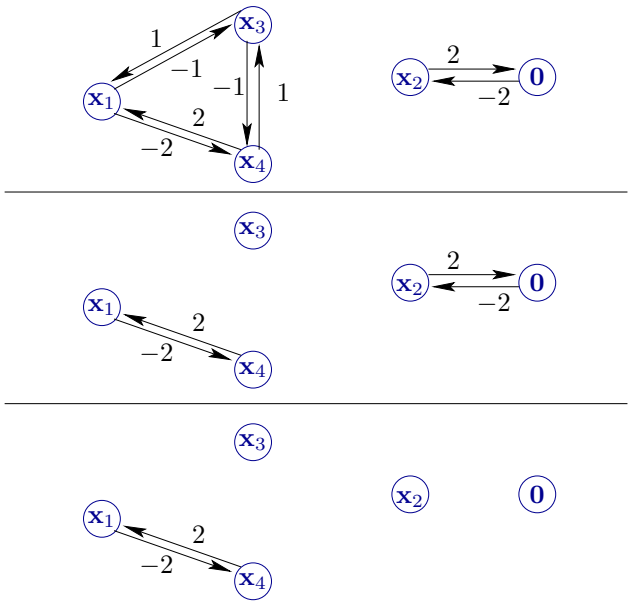


Fig. 4: Increasing chain of conjunctions of equalities.

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 4.4. *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 4.5. *Variables $\mathbf{x}_i$ and $\mathbf{x}_j$ are in the same component of $\mathcal{G}(E)$ iff the following two properties hold:*

- *$\mathbf{x}_i$ and $\mathbf{x}_j$ are in the same component of $\mathcal{G}(E_1)$ and $\mathcal{G}(E_2)$;*
- *$\delta(\mathbf{x}_i) = \delta(\mathbf{x}_j)$.* □

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 $\delta(\mathbf{x}_j)$.

Example 4.6. 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 δ yields the partitioning as specified

in the right table:

E_1 and E_2 in normal form:

$\mathbf{X}$	E_1	E_2
$\mathbf{x}_1$	$\mathbf{x}_1$	$\mathbf{x}_1$
$\mathbf{x}_2$	$\mathbf{x}_2$	$\mathbf{x}_2$
$\mathbf{x}_3$	$\mathbf{x}_1$	$\mathbf{x}_2 - 5$
$\mathbf{x}_4$	$\mathbf{x}_2 + 5$	$\mathbf{x}_2 + 5$
$\mathbf{x}_5$	$\mathbf{x}_1 + 5$	$\mathbf{x}_2$
$\mathbf{x}_6$	$\mathbf{x}_1 + 3$	$\mathbf{x}_2 + 1$
$\mathbf{x}_7$	$\mathbf{x}_1 + 2$	$\mathbf{x}_2$

... and after partitioning:

$\mathbf{X}$	E_1	E_2	δ
$\mathbf{x}_1$	$\mathbf{x}_1$	$\mathbf{x}_1$	0
$\mathbf{x}_2$	$\mathbf{x}_2$	$\mathbf{x}_2$	0
$\mathbf{x}_4$	$\mathbf{x}_2 + 5$	$\mathbf{x}_2 + 5$	0
$\mathbf{x}_6$	$\mathbf{x}_1 + 3$	$\mathbf{x}_2 + 1$	2
$\mathbf{x}_7$	$\mathbf{x}_1 + 2$	$\mathbf{x}_2$	2
$\mathbf{x}_3$	$\mathbf{x}_1$	$\mathbf{x}_2 - 5$	5
$\mathbf{x}_5$	$\mathbf{x}_1 + 5$	$\mathbf{x}_2$	5

□

Let Π 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 b_h denote the constant of the right-hand side of the equality for $\mathbf{x}_h$. 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. Let again b_h denote the constant of the right-hand side of the equality for $\mathbf{x}_h$. 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 $\mathcal{O}(k \cdot \log(k))$.

Example 4.7. *(Example 4 continued)* 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, i.e., $\mathbf{x}_3$ 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 $E = (\mathbf{x}_4 \doteq \mathbf{x}_2 + 5) \wedge (\mathbf{x}_5 \doteq \mathbf{x}_3 + 5) \wedge (\mathbf{x}_7 \doteq \mathbf{x}_6 - 1)$. □

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, α and γ 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 $[\![s]\!]^\sharp$ for every assignment s by:

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

Here, $\exists^\sharp \mathbf{x}_i.\, E$ denotes the *abstract existential quantification*, i.e., $\exists^\sharp \mathbf{x}_i.\, E = \bot$ if $E = \bot$, 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 $\bot$ 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 $[\![s]\!]^\sharp$ is distributive where $[\![s]\!]^\sharp\ E$ can be computed in time $\mathcal{O}(k)$ for a normalised conjunction E. Moreover, we find:

LEMMA 4.8. *For a set of program states $X \subseteq \mathbb{Z}^k$ and an arbitrary assignment s:*

$$\alpha([\![s]\!]X) = [\![s]\!]^\sharp(\alpha(X))$$

PROOF. The proof of Lemma 4.8 is by construction. □

5. INTERPROCEDURAL ANALYSIS

In order to construct 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 [Müller-Olm and Seidl 2008] 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 preconditions of 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. Since the constants c may take arbitrary integer values, there are still infinitely many equalities of interest. Instead of dealing with equalities for each constant c separately, we introduce a fresh variable denoted by the symbol $\bullet$ which is not accessed by the program, but may be instantiated with any constant c. Accordingly, for computing the representation of procedures we

consider post-conditions of the form:

$$\mathbf{x}_j \doteq \bullet \qquad \text{or} \qquad \mathbf{x}_i \doteq \mathbf{x}_j + \bullet$$

for global variables $\mathbf{x}_i, \mathbf{x}_j$ where, w.l.o.g., $i > j$. We call postconditions of this form *generic*. The variable $\bullet$ thus acts as a *logical variable* which allows to relate a value occurring in the postcondition with a value possibly occurring in the precondition.

Preconditions of generic equalities then may be conjunctions of equalities of one of the following forms:

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

for global variables $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$ with $i > j$ and constants $a, b \in \mathbb{Z}$. Let us call such equalities *parametric*. Note that every generic equality is also parametric. (Choose $a = 1$ and $b = 0$ in parametric postconditions of types (2) and (3).) Note further that $0 \doteq a \cdot \bullet + b$ is only satisfiable over $\mathbb{Z}$ iff $a = b = 0$ or a divides b. In the latter case it has the unique solution $\bullet = \frac{-b}{a}$.

Example 5.1. Consider the procedure p:

$$\textbf{void } p() \ \{ \ \mathbf{x}_1 = \mathbf{x}_1 + 1; \mathbf{x}_2 = \mathbf{x}_2 - 1; \}$$

which increments the global variable $\mathbf{x}_1$ and decrements the global variable $\mathbf{x}_2$. The weakest preconditions of $\mathbf{x}_1 \doteq \bullet$, $\mathbf{x}_2 \doteq \bullet$ and $\mathbf{x}_2 \doteq \mathbf{x}_1 + \bullet$ then are given by:

$\mathbf{x}_1 \doteq \bullet$	$\mathbf{x}_1 \doteq \bullet - 1$
$\mathbf{x}_2 \doteq \bullet$	$\mathbf{x}_2 \doteq \bullet + 1$
$\mathbf{x}_2 \doteq \mathbf{x}_1 + \bullet$	$\mathbf{x}_2 \doteq \mathbf{x}_1 + \bullet + 2$

In case, the second assignment in the body of p is $\mathbf{x}_2 = 5$, we obtain the following preconditions:

$\mathbf{x}_1 \doteq \bullet$	$\mathbf{x}_1 \doteq \bullet - 1$
$\mathbf{x}_2 \doteq \bullet$	$0 \doteq \bullet - 5$
$\mathbf{x}_2 \doteq \mathbf{x}_1 + \bullet$	$\mathbf{x}_1 \doteq - \bullet + 4$

Thus, the postcondition $\mathbf{x}_2 \doteq \bullet$ can only hold if $\bullet$ equals 5. □

Satisfiability of single parametric equalities e 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 $f : \mathbb{Z}^k \to \mathbb{Z}^k$ of concrete sets of states to a completely distributive transformation $\mathsf{ext}\ f : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ of sets of extended states by defining:

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

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

$$\begin{array}{lll} [\![\mathbf{x}_i :=?]\!]^\top\ e & = \forall \mathbf{x}_i.\, e & = \begin{cases} \bot & \text{if } e \text{ contains } \mathbf{x}_i \\ e & \text{otherwise} \end{cases} \\ [\![\mathbf{x}_i := c]\!]^\top\ e & = e\,[c/\mathbf{x}_i] & \\ [\![\mathbf{x}_i := \mathbf{x}_j + c]\!]^\top\ e & = e\,[\mathbf{x}_j + c/\mathbf{x}_i] & \end{array}$$

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 $\bot$ 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 , e.g.,

$$[\![\mathbf{x}_4 := \mathbf{x}_1 + 5]\!]^\top(\mathbf{x}_4 \doteq \mathbf{x}_3 + 2) \;=\; (\mathbf{x}_1 + 5 \doteq \mathbf{x}_3 + 2) \;=\; (\mathbf{x}_3 \doteq \mathbf{x}_1 + 3)\,.$$

By construction, we have:

LEMMA 5.2. *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:* $\mathsf{ext}\,[\![s]\!]\,(Z) \models E$ *iff* $Z \models [\![s]\!]^\top E$. □

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 $S^\top$:

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

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 5.3. *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:* $\mathsf{ext}\,S[u]\ Z \models E$ *iff* $Z \models S^\top[u]\ E$.

PROOF. The proof of Theorem 5.3 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 generic postconditions, i.e., postconditions of the forms $\mathbf{x}_j \doteq \bullet$ or $\mathbf{x}_i \doteq \mathbf{x}_j + \bullet$ with $i > j$. 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 $\mathsf{ext}^\top$. The operator $\mathsf{ext}^\top$ takes a function $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X})$ and transforms it into a full WP transformer of type $\mathbb{E}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X})$. For a single equality e involving globals and $\bullet$, this transformer is defined by:

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

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:

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

Here, we assume that $f^\top$ corresponds to a computation that definitely does not terminate, 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. In fact, we could have chosen any $\mathbf{x}_i$ to perform the distinction whether f terminates or not. Finally, for arbitrary conjunctions $E = e_1 \wedge \ldots \wedge e_m$, we set

$$\mathsf{ext}^\top(f^\top)(E) = \mathsf{ext}^\top(f^\top)(e_1) \wedge \ldots \wedge \mathsf{ext}^\top(f^\top)(e_m)$$

Let $f : 2^{\mathbb{Z}^k} \to 2^{\mathbb{Z}^k}$ 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 6.1. *Let $f : 2^{\mathbb{Z}^k} \to 2^{\mathbb{Z}^k}$ denote a concrete transformer which is completely distributive and uniform. Furthermore, let $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X})$ denote a function where for all $Z \subseteq \mathbb{Z}^{k+1}$ and $e \in \mathbb{P}_\bullet(\mathbf{X})$, $\mathsf{ext}\ f(Z) \models e$ iff $Z \models f^\top(e)$. Then also*

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

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

PROOF. Since f and $\mathsf{ext}^\top(f^\top)$ 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

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

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 $(x, \_)$. Then also $\mathsf{ext}\ f(\{(x, c)\}) = \emptyset$ 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 $\mathsf{ext}^\top(f^\top)(e) = \top$, 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

$$\begin{array}{ll} \mathsf{ext}\ f\,(Z) \models e & \text{iff } Z \models e \\ & \text{iff } Z \models \mathsf{ext}^\top(f^\top)(e) \end{array}$$

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 $\mathsf{ext}\ f\,(\{(x,c)\}) = \emptyset$ for all (x,c), i.e., $\mathsf{ext}\ f(\mathbb{Z}^{k+1}) = \emptyset$. Since then $\mathsf{ext}\ f(\mathbb{Z}^{k+1}) \models (\mathbf{x}_1 \doteq \bullet)$, 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 $f^\top$, $\mathsf{ext}\ f\,(Z) \models (\mathbf{x}_1 \doteq \bullet)$. Since $Z' \subseteq \mathsf{ext}\ f\,(Z)$, 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. □

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

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

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 $\mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X})$ only. This means for the constraints from the second line that we must extend the transformer for the called procedure by means of $\mathsf{ext}^\top$ before the composition can be performed. In general, consider a composition $h = \mathsf{ext}^\top(f^\top) \circ g^\top$ for completely distributive functions $f^\top, g^\top$. Let e denote a generic postcondition, and assume that $e_1[t_1/\bullet] \wedge \ldots \wedge e_r[t_r/\bullet]$ is a normalised conjunction for $g^\top(e)$ where $e_i \in \mathbb{P}_\bullet(\mathbf{X})$. Then the value $(\mathsf{ext}^\top(f^\top) \circ g^\top)(e)$ is the normalised conjunction for:

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

i.e., amounts to normalising a conjunction of $\mathcal{O}(k^2)$ equalities. According to Lemma 3.2, 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 6.2.

(*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 $S^\top[u] = \mathsf{ext}^\top(S^\bullet[u])$.* □

For the first statement of the lemma, we rely on Theorem 5.3 and Lemma 6.1. Since both restriction and extension of transformers are monotonic operations, we conclude:

THEOREM 6.3. *Assume that $Z \subseteq \mathbb{Z}^{k+1}$ and $e \in \mathbb{P}_\bullet(\mathbf{X})$. Then, for every program point u:* $\mathsf{ext}\ S[u]\,(Z) \models e$ *iff* $Z \models S^\bullet[u](e)$.

PROOF. 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 global variables only (no $\bullet$ is needed here). For that, we require a transformation $\mathsf{ext}^\sharp$ which takes a weakest precondition transformer $f^\top$ for the body of a procedure and returns the corresponding *forward* transformation $\mathsf{ext}^\sharp(f^\top)$ of valid equalities. For a weakest precondition transformer $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X})$, we define $\mathsf{ext}^\sharp(f^\top) : \mathbb{E}(\mathbf{X}) \to \mathbb{E}(\mathbf{X})$ as follows. Let $E \in \mathbb{E}(\mathbf{X})$. Then $\mathsf{ext}^\sharp(f^\top)(E)$ is the conjunction of all equalities $e' = e[c/\bullet]$ for which $E \implies (\mathsf{ext}^\top(f^\top)(e'))$ or, equivalently, $E \implies (f^\top(e)[c/\bullet])$. The following lemma states that the operator $\mathsf{ext}^\sharp$ 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 6.4. *Let $f : 2^{\mathbb{Z}^k} \to 2^{\mathbb{Z}^k}$ be completely distributive and uniform and let $\mathsf{ext}\ f : 2^{\mathbb{Z}^{k+1}} \to 2^{\mathbb{Z}^{k+1}}$ be the corresponding extended transformer. Let $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X})$ be a weakest precondition transformer. Assume as in Lemma 6.1 that $\mathsf{ext}\ f\,(Z) \models e$ 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 $\mathsf{ext}^\sharp(f^\top)(E) \sqsubseteq E'$.*

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

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

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

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

by Lemma 6.1. Then we deduce:

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

and the statement of the lemma follows. □

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

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

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 6.5. *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 [Fecht and Seidl 1998] 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. 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 actually 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 4.2), each right-hand side of the constraint system $C^\sharp$ may be evaluated at most $\mathcal{O}(k)$ times. In system $C^\sharp$ 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 $C^\sharp$ 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. □

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 unmodified when applying the transformer:

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

Assume $\mathbf{x}_j \in B(f)$. Then, $f(\mathbf{x}_i \doteq \mathbf{x}_j + \bullet)$ can be recovered from $f(\mathbf{x}_i \doteq \bullet)$ 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 $\mathbf{x}_i \doteq \bullet$. From these values we can determine the set $B(f)$. Then it suffices to tabulate in addition only the preconditions for $\mathbf{x}_i \doteq \mathbf{x}_j + \bullet$ 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 $\mathbf{x}_i \doteq \bullet$, 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 $\mathbf{x}_i \doteq \bullet$ only for variables $\mathbf{x}_i \notin B'(f)$.

7. 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 $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$. The resulting control flow graph is given in Figure 5. For the analysis, memory writes are ignored and reads from memory into a variable are abstracted with corresponding non-deterministic assignments. The global variable $\mathbf{x}_1$ represents the stack pointer while the global $\mathbf{x}_2$ 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 $\mathbf{x}_1$.

Within the prologue of procedure q, 8 bytes of memory are reserved for the callee-save registers, i.e., $\mathbf{x}_3$ and $\mathbf{x}_4$, which are saved via a call to $\_save\_3$. Additionally, the stack pointer $\mathbf{x}_1$ is modified at program point 11 in order to account for the stack growth for saving $\mathbf{x}_3$ and $\mathbf{x}_4$. 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 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 $S^\top[7](\mathbf{x}_2 \doteq \mathbf{x}_1 + \bullet) = (0 \doteq \bullet)$ as the weakest precondition of the

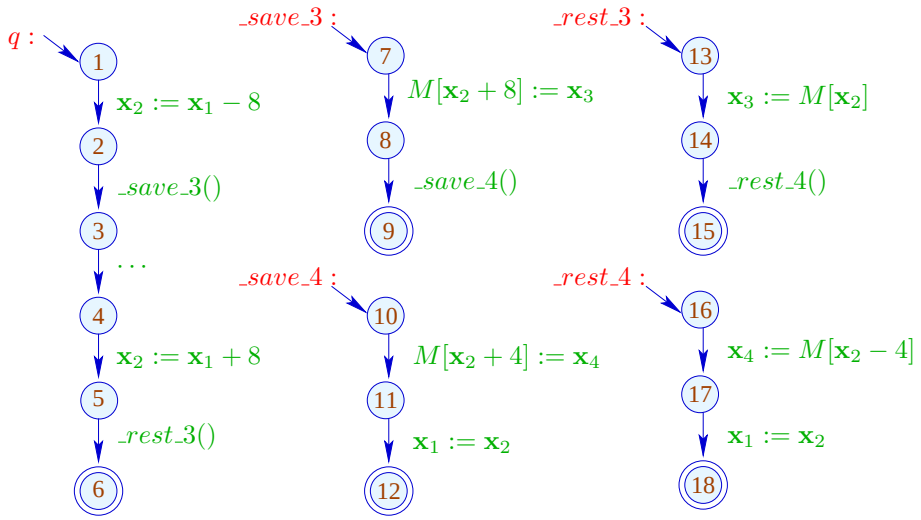


Fig. 5: Coding convention for saving and restoring registers $\mathbf{x}_3, \mathbf{x}_4$.

above generic postcondition at program point 7. For procedure $\_rest\_3$, i.e., at program point 13, the same precondition is computed, namely $S^{\top}[13](\mathbf{x}_2 \doteq \mathbf{x}_1 + \bullet) = (0 \doteq \bullet)$. 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:

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.

8. 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 [Zucker and Karhi 1995]. For simplicity, we assume that every procedure q has l local variables $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_l\}$, while the set of global variables is still $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$. Thus, a state is now described by a vector $(x_1, \ldots, x_k, y_1, \ldots, y_l) \in \mathbb{Z}^{k+l}$, which we identify with the pair (x, y) of vectors $x = (x_1, \ldots, x_k) \in \mathbb{Z}^k$ and $y = (y_1, \ldots, y_l) \in \mathbb{Z}^l$ of values for the global and local variables, respectively. Accordingly, the transformations $S[u]$ are completely distributive functions from the set $\mathbb{T} = 2^{\mathbb{Z}^{k+l}} \to 2^{\mathbb{Z}^{k+l}}$. 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 $\mathsf{enter} \in \mathbb{T}$ captures how a set of states propagates from the call to the start edge of the called procedure:

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

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 $\mathsf{H} : \mathbb{T} \to \mathbb{T}$ adjusts the transformation computed for a called procedure to the caller:

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

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:

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

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

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 [Müller-Olm and Seidl 2008], we use an operator $\mathsf{H}^\top$ which takes the representation of a weakest precondition transformer $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$ 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 enter from the concrete semantics. We define $\mathsf{enter}^\top : \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y}) \to \mathbb{E}_\bullet(\mathbf{X})$ by

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

For a weakest precondition transformer for the body of a called procedure $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$, the composition $\mathsf{enter}^\top \circ f^\top$ thus is a transformation from $\mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X})$ 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 $\mathsf{ext}^\top$ which now extends a transformer $g^\top : \mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X})$ to a transformer from $\mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y}) \to \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$. This operator is defined as follows.

For a single equality e involving globals, we define

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

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:

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

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 $E = e_1 \wedge \ldots \wedge e_r$, we define:

$$\mathsf{ext}^\top(g^\top)(E) = \mathsf{ext}^\top(g^\top)(e_1) \wedge \ldots \wedge \mathsf{ext}^\top(g^\top)(e_r)$$

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

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

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 $f : 2^{\mathbb{Z}^{k+l}} \to 2^{\mathbb{Z}^{k+l}}$ 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 $\mathsf{ext}(f) : 2^{\mathbb{Z}^{k+l+1}} \to 2^{\mathbb{Z}^{k+l+1}}$ for a completely distributive uniform transformer is defined by $\mathsf{ext}(f)(\{(x, y, c)\}) = \{(x', y', c) \mid (x', y') \in f(\{(x, y)\})\}$. Analogously to Lemma 6.1 we obtain:

LEMMA 8.1. *Let $f : 2^{\mathbb{Z}^{k+l}} \to 2^{\mathbb{Z}^{k+l}}$ denote a concrete transformer which is completely distributive and uniform. Furthermore, let $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$ denote a weakest precondition transformer where for all $Z \subseteq \mathbb{Z}^{k+l+1}$ and $e \in \mathbb{P}_\bullet(\mathbf{X})$, $\mathsf{ext}(f)(Z) \models e$ iff $Z \models f^\top(e)$. Then also*

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

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 $\mathsf{H}^\top$ to deal with function calls.

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

Note that the composition $\mathsf{H}^\top(f^\top) \circ g^\top$ 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 6.3 to programs involving local variables, we obtain:

THEOREM 8.2. *Assume that $Z \subseteq \mathbb{Z}^{k+l+1}$ and $e \in \mathbb{P}_\bullet(\mathbf{X})$. Then, for every program point u:* $\mathsf{ext}\ S[u](Z) \models e$ *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) $\mathsf{ext}\ S_i[u]\,(Z) \models e$ iff $Z \models S_i^\top[u]\,(e)$.

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

Now assume $i > 0$. Then, in the concrete semantics, $\mathsf{ext}\ S_i[u]\,(Z)$ is the union of sets $S_{i-1}[v](\mathsf{ext}\ [\![s]\!]^\sharp_{i-1}\, Z)$ for edges (u, s, v) where $[\![s]\!]^\sharp_{i-1}$ 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 $[\![s]\!]^\top_{i-1}(S_{i-1}^\top[v]\,(e))$ for every edge (u, s, v) where $[\![s]\!]^\top_{i-1}$ 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 $\mathsf{ext}\ S_{i-1}[v](\mathsf{ext}\ [\![s]\!]^\sharp_{i-1}\, Z) \models e$ iff $Z \models [\![s]\!]^\top_i(S_{i-1}^\top[v]\,(e))$. Consider the case of a procedure call $s \equiv q()$. Then

$$\begin{array}{lcl} S_{i-1}[v]([\![s]\!]^\sharp_{i-1}\, Z) & = & S_{i-1}[v](\mathsf{H}(S_{i-1}[s_q])\, Z) \quad \text{and} \\ [\![s]\!]^\top_i(S_{i-1}^\top[v]\,(e)) & = & \mathsf{H}^\top(S_{i-1}^\top[s_q])\,(S_{i-1}^\top[v]\,(e))\,. \end{array}$$

We have:

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

by induction hypothesis. Furthermore,

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

by induction hypothesis for the transformers for s_q and Lemma 8.1. 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

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

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 $\mathsf{enter}^\sharp$ 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 $\mathsf{enter}^\sharp$ is defined by:

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

For a weakest precondition transformer $f^\top : \mathbb{P}_\bullet(\mathbf{X}) \to \mathbb{E}_\bullet(\mathbf{X} \cup \mathbf{Y})$ for the body of a procedure, we define $\mathsf{H}^\sharp(f^\top) : \mathbb{E}(\mathbf{X} \cup \mathbf{Y}) \to \mathbb{E}(\mathbf{X} \cup \mathbf{Y})$ as follows. Let $E \in \mathbb{E}(\mathbf{X} \cup \mathbf{Y})$. First assume that $f^\top(\mathbf{x}_1 \doteq \bullet) = \top$. Then $\mathsf{H}^\sharp(E) = \bot$. Otherwise, $\mathsf{H}^\sharp(f^\top)(E)$ 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 $E \implies (\mathsf{H}^\top(f^\top)(e))[t/\bullet]$. In particular for $E = \bot$, we have $\mathsf{H}^\sharp(f^\top)(E) = \bot$ as well. The following lemma states that the operator $\mathsf{H}^\sharp$ allows determining 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 8.3. *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 $f : 2^{\mathbb{Z}^{k+l}} \to 2^{\mathbb{Z}^{k+l}}$ be completely distributive and uniform, and let $f^\top$ be a weakest precondition transformer. Assume as in Lemma 8.1 that for all $e \in \mathbb{P}_\bullet(\mathbf{X})$, $\mathsf{ext}(f)(Z) \models e$ 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 $E' \in \mathbb{E}(\mathbf{X} \cup \mathbf{Y})$, $\mathsf{H}(f)(Z) \models E'$ iff $\mathsf{H}^\sharp(f^\top)(E) \sqsubseteq E'$.*

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

$$\begin{array}{lll}
C^\sharp[s_{main}] \supseteq \mathsf{enter}^\sharp(\top) & & \\
C^\sharp[s_q] \supseteq \mathsf{enter}^\sharp(C^\sharp[u]) & & (u, q(), _) \text{ a call edge} \\
C^\sharp[v] \supseteq \mathsf{H}^\sharp(S^\top[s_q])\,(C^\sharp[u]) & & (u, q(), v) \text{ a call edge} \\
C^\sharp[v] \supseteq [\![s]\!]^\sharp\,(C^\sharp[u]) & & (u, s, v) \text{ an assignment edge}
\end{array}$$

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

THEOREM 8.4. *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 $\mathcal{O}(n \cdot k^2 \cdot (k+l)^2)$.*

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 8.

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, 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

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

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:

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

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 9.1.
Consider the following conjunction E of parametric equalities in normal form:

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

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:

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

□

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 from 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 $\mathbf{x}_j$ 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 neither of this is 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 6.5.

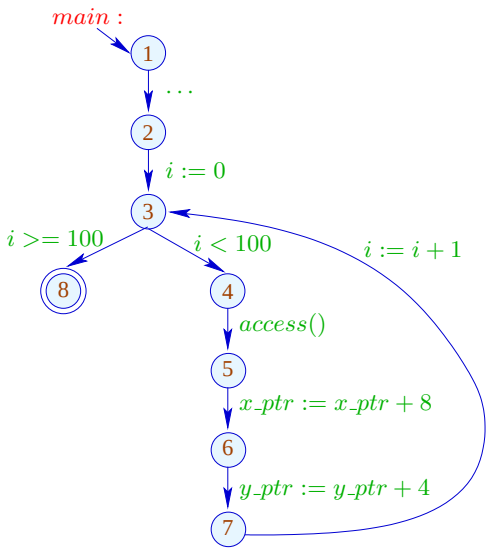
THEOREM 9.2. *The set of all equalities of the form $\mathbf{x}_i \doteq a\mathbf{x}_j + b$ with $a, b \in \mathbb{Q}$ which are valid for an interprocedural program of size n with k global variables can be computed in time $\mathcal{O}(n \cdot k^4)$.*

Example 9.3. 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 next to it.

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

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. □

10. EXPERIMENTAL RESULTS

We have implemented the interprocedural analysis of variable differences in our assembly analyser VoTUM [Votum 2010] where the 40 hardware registers of the *PowerPC* are modelled by means of global variables. For a comparison, we have also implemented the full interprocedural analysis of linear equalities from [Müller-Olm and Seidl 2007] based on vector spaces of matrices. Our benchmark suite consists of *PowerPC* assemblies of publicly available programs such as the cryptography library **`openSSL`**, and the HTTP server **`thttpd`**. Moreover, we experimented with the five larger programs **`ls`**, **`basename`**, **`vdir`**, **`chmod`**, **`chgrp`** from the Unix *GNU Coreutils* package, and also ran the prototype on **`gzip`** from *SPECint*. The binaries of these programs have sizes between 0.6 and 2.9 MB. We conducted our experiments on a 2.2 GHz quad-core machine equipped with 16 GB of physical memory where the algorithm occupies just a single core. Since our main contribution consists in an *interprocedural* equality analysis, we compare the time and memory consumption of the procedure effect computation of the two approaches. In order to reduce memory consumption the implementation of the linear algebra-based analysis from [Müller-Olm and Seidl 2007] relies on sparse matrices. Similarly, the implementation of the variable differences analysis represents the

Table I: Benchmark Results

Program	Size	Instr	Procs	T(LA)	M(LA)	T(VD)	M(VD)	T(oVD)	M(oVD)
openSSL	2.9MB	613882	6232	—	—	1239sec.	3865MB	563sec.	2678MB
thttpd	0.8MB	189034	1197	297sec.	3348 MB	323sec.	1512MB	229sec.	1365MB
basename	0.6MB	139271	907	356sec.	922MB	223sec.	894MB	123sec.	622MB
chgrp	0.7MB	164420	1082	324sec.	1032MB	265sec.	1027MB	147sec.	645MB
chmod	0.6MB	148702	990	408sec.	913MB	247sec.	1091MB	169sec.	651MB
ls	0.8MB	177022	1203	339sec.	1060MB	298sec.	1321MB	277sec.	953MB
vdir	0.8MB	177022	1202	411sec.	1048MB	299sec.	1335MB	216sec.	925MB
gzip	0.7MB	162380	1026	363sec.	2825 MB	284sec.	1472MB	187sec.	955MB

Size: the binary file size in MB; **Instr**: the number of assembler instructions; **Procs**: the number of procedures; **T**: the absolute running time in seconds; **M**: the peak memory consumption in MB.

precondition of a generic equality not by a matrix but a (possibly sparse) map. Table I summarises our results.

For each benchmark from our suite, we compared the interprocedural analysis of linear equalities (**LA**) with the interprocedural analysis of variable differences (**VD**) as well as with the interprocedural analysis of variable differences where the optimisation from Section 6 is taken into account (**oVD**). For each of these analyses we track the following parameters:

T:. the absolute running time in seconds;
M:. the peak memory consumption in MB.

Evaluating our experimental results, we have:

- The linear algebra analysis consumes definitely more memory than the (optimised) variable difference analysis. Moreover, our biggest example program, **`openSSL`**, could not be analysed with the linear algebra approach (**LA**) as the memory consumption exceeded the heap limit of 16GB. For (almost) all the example programs the variable difference analysis terminated faster. Interestingly, for benchmark **`thttpd`** the linear algebra approach still beats the unoptimised version of variable differences. The reason is that linear algebra has been implemented with sparse matrices, while the implementation based on variable differences explores quadratically many generic postconditions. Compared to the variable difference analysis (**VD**), the running time of the optimised variable difference analysis (**oVD**) has improved between 17% up to 48%, whereas the memory consumption decreased by around 20%. Summarising, the optimised variable difference analysis (**oVD**) outperforms the linear algebra approach (**LA**) for the given application by a factor of 2 while using only 75% memory.
- The standard code generation scheme for *PowerPC* code (which consists in three-address code) does not heavily rely on the addition of two registers. In practise, the variable difference analysis yields precise results for identifying local variables on the stack—approximately 85% of all the stack accesses could be identified—as well as for checking the stack pointer invariant—for approximately 95% of all the procedures the stack pointer invariant could be verified and for the remaining 5% the analysis indicates where the stack pointer invariant may be violated. While in general, the linear algebra approach clearly may detect more equalities than just variable differences, this difference in precision, though, seems not to matter for this particular application.

11. CONCLUSION

In this paper we 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 [Müller-Olm and Seidl 2008]. 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 [Müller-Olm and Seidl 2008]. 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.

Acknowledgement. The ideas for this work have been developed during the cooperation within the OpIAT project (MU 1508/1-1 and SE 551/13-1) funded by the Deutsche Forschungsgemeinschaft (DFG).

References

ALPERN, B., WEGMAN, M., AND ZADECK, F. K. 1988. Detecting equality of variables in programs. In *15th ACM Symp. on Principles of Programming Languages (POPL)*. 1–11.

FECHT, C. AND SEIDL, H. 1998. Propagating differences: An efficient new fixpoint algorithm for distributive constraint systems. *Nord. J. Comput. 5,* 4, 304–329.

GEORGE, L. AND APPEL, A. W. 1996. Iterated register coalescing. *ACM Transactions on Programming Languages and Systems (TOPLAS) 18,* 3, 300–324.

GULWANI, S. AND NECULA, G. C. 2004. A polynomial-time algorithm for global value numbering. In *11th Int. Static Analysis Symposium (SAS)*. Springer-Verlag, LNCS 3148, 212–227.

HORWITZ, S., REPS, T. W., AND SAGIV, M. 1995. Precise interprocedural dataflow analysis via graph reachability. In *22nd ACM Symp. on Principles of Programming Languages (POPL)*. 49–61.

KNOOP, J. AND STEFFEN, B. 1992. The interprocedural coincidence theorem. In *Compiler Construction (CC)*. LNCS 541, Springer-Verlag, 125–140.

MINÉ, A. 2001a. A new numerical abstract domain based on difference-bound matrices. In *Proc. of the 2d Symp. on Programs as Data Objects (PADO II)*. Springer-Verlag, LNCS 2053, 155–172.

MINÉ, A. 2001b. The octagon abstract domain. In *Proc. of the Workshop on Analysis, Slicing, and Transformation (AST'01)*. IEEE. IEEE CS Press, 310–319.

MÜLLER-OLM, M., RÜTHING, O., AND SEIDL, H. 2005. Checking Herbrand Equalities and Beyond. In *Verification Meets Model-Checking and Abstract Interpretation (VMCAI)*. Springer-Verlag, LNCS 3385, 79–96.

MÜLLER-OLM, M. AND SEIDL, H. 2004. Precise interprocedural analysis through linear algebra. In *31st ACM Symp. on Principles of Programming Languages (POPL)*. 330–341.

MÜLLER-OLM, M., SEIDL, H., AND STEFFEN, B. 2005. Interprocedural Herbrand Equalities. In *14th European Symposium on Programming (ESOP)*. Springer-Verlag, LNCS 3444, 31–45.

MÜLLER-OLM, M. AND SEIDL, H. 2007. Analysis of modular arithmetic. *ACM Transactions on Programming Languages and Systems (TOPLAS) 29,* 5.

MÜLLER-OLM, M. AND SEIDL, H. 2008. Upper adjoints for fast inter-procedural variable equalities. In *17th European Symposium on Programming (ESOP)*. Springer-Verlag, LNCS 4960.

SAGIV, S., REPS, T. W., AND HORWITZ, S. 1996. Precise interprocedural dataflow analysis with applications to constant propagation. *Theor. Comput. Sci. 167,* 1&2, 131–170.

SANKARANARAYANAN, S., SIPMA, H., AND MANNA, Z. 2005. Scalable analysis of linear systems using mathematical programming. In *6th International Conference, Verification, Model Checking and Abstract Interpretation (VMCAI)*. Springer-Verlag, LNCS 3385, 25–41.

SHARIR, M. AND PNUELI, A. 1981. Two approaches to interprocedural data flow analysis. In *Program Flow Analysis: Theory and Application*. Prentice Hall, 189–234.

SIMON, A., KING, A., AND HOWE, J. M. 2002. Two variables per linear inequality as an abstract domain. In *Logic Based Program Development and Transformation (LOPSTR)*. Springer-Verlag, LNCS 2664, 71–89.

STEFFEN, B., KNOOP, J., AND RÜTHING, O. 1990. The value flow graph: A program representation for optimal program transformations. In *3rd European Symp. on Programming (ESOP)*. Springer-Verlag, LNCS 432, 389–405.

Votum 2010. *VoTUM*. `http://www2.in.tum.de/votum`.

ZUCKER, S. AND KARHI, K. 1995. *System V Application Binary Interface: PowerPC Processor Supplement*. `http://refspecs.freestandards.org/elf/elfspec_ppc.pdf`.

APPENDIX

A. AN ALGORITHM FOR THE LEAST UPPER BOUND

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:

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

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 Π_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 Π_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 $X_0, \ldots, X_4$ 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 A.1.
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 $X_0, \ldots, X_4$ 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:

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

□

B. ALGORITHMS FOR ABSTRACT TRANSFORMERS

In the following, we define the abstract effect $[\![s]\!]^\sharp$ for every assignment s, by providing an implementation for the abstract transformers for equivalence relations $E \neq \bot$. If the abstract transformer $[\![]\!]^\sharp : E \to E$ is applied to the least element $\bot$ of our lattice, we obtain $\bot$, denoting that all possible equalities are valid.

```
1:  [[x_i :=?]]^♯ E =
2:  if (E(x_i) ≠ (x_i, 0)) then
3:     E(x_i) ← (x_i, 0);
4:     return E;
5:  else
6:     X ← {x_j | E(x_j) = (x_i, c_k), j ≠ i }
7:     if (X ≠ ∅) then
8:        choose smallest x_k ∈ X
9:        for all x_l ∈ X with E(x_l) = (x_i, c_l), l ≠ k do
10:          E(x_l) ← (x_k, c_l − c_k);
11:       E(x_k) ← (x_k, 0);
12:    E(x_i) ← (x_i, 0);
```

13: **return** E;

When a variable $\mathbf{x}_i$ is set unknown, we have to clear all right-hand sides of equalities from $\mathbf{x}_i$, having $\mathbf{x}_i$ as reference variable.

1: $[\![\mathbf{x}_i := \mathbf{x}_j + c]\!]^\sharp\ E\ =$
2: **let** $(\mathbf{x}_{j'}, c') = E(\mathbf{x}_j)$ **in**
3: **if** $((i == j) \wedge (i == j'))$ **then**
4: **for all** $\mathbf{x}_k$ **with** $E(\mathbf{x}_k) = (\mathbf{x}_i, c_k),\ k \neq i$ **do**
5: $E(\mathbf{x}_k) \leftarrow (\mathbf{x}_i, c_k - c)$;
6: **return** E;
7: **else if** $((i == j) \wedge (i \neq j'))$ **then**
8: **let** $(\mathbf{x}_{i'}, c_i) = E(\mathbf{x}_i)$ **in**
9: $E(\mathbf{x}_i) \leftarrow (\mathbf{x}_{i'}, c_i + c)$;
10: **return** E;
11: **else**
12: $E \leftarrow [\![\mathbf{x}_i :=?]\!]^\sharp\ E$
13: **if** $(i > j')$ **then**
14: $E(\mathbf{x}_i) \leftarrow (\mathbf{x}_{j'}, c' + c)$;
15: **return** E;
16: **else**
17: **for all** $\mathbf{x}_k$ **with** $E(\mathbf{x}_k) = (\mathbf{x}_{j'}, c_k),\ k \neq j'$ **do**
18: $E(\mathbf{x}_k) \leftarrow (\mathbf{x}_i, c_k - c - c')$;
19: $E(\mathbf{x}_{j'}) \leftarrow (\mathbf{x}_i, -c - c')$;
20: **return** E;

The effect of $\mathbf{x}_i := \mathbf{x}_j + c$ is implemented by simply assigning the term $\mathbf{x}_j + c$ to variable $\mathbf{x}_j$. Subsequently, the variable order has to be restored such that the reference variables are of a smaller number than the variables on the left-hand side. In the worst case all of the maximally k equalities have to be adjusted. For the special case of describing the effect of $\mathbf{x}_i := \mathbf{x}_i + c$ in our domain of normalised equality relations, all those right-hand sides with reference variable $\mathbf{x}_i$ have to be adjusted by c.

$$
\begin{aligned}
[\![\mathbf{x}_i := c]\!]^\sharp\ E &= [\![\mathbf{x}_i := \mathbf{x}_0 + c]\!]^\sharp\ E \\
[\![\mathbf{x}_i := \mathbf{x}_j]\!]^\sharp\ E &= [\![\mathbf{x}_i := \mathbf{x}_j + 0]\!]^\sharp\ E \\
[\![\mathbf{x}_i := \mathbf{x}_j + \mathbf{x}_k]\!]^\sharp\ E\ &= \begin{cases} [\![\mathbf{x}_i := \mathbf{x}_j + c_k\,]\!]^\sharp\ E & \textbf{if}\ \ E(\mathbf{x}_k) = (\mathbf{x}_0, c_k) \\ [\![\mathbf{x}_i := \mathbf{x}_k + c_j\,]\!]^\sharp\ E & \textbf{if}\ \ E(\mathbf{x}_j) = (\mathbf{x}_0, c_j) \\ [\![\mathbf{x}_i :=?\,]\!]^\sharp\ E & \textbf{otherwise} \end{cases}
\end{aligned}
$$

The algorithm for normalising a given conjunction $E = \bigwedge e_i$ of equalities e_i is obtained via the following algorithm:

1: given: $E = \bigwedge e_i$
2: $E' = \emptyset$
3: **for all** $e \in E$ **do**
4: **if** $(e \equiv \mathbf{x}_i \doteq b)$ **then**
5: **if** $(E'(\mathbf{x}_i) == (\mathbf{x}_0, b'))$ **then**
6: **if** $(b \neq b')$ **then**
7: **return** false;
8: **else**
9: **let** $E'(\mathbf{x}_i) = (\mathbf{x}_h, b')$ **in**
10: $E'(\mathbf{x}_h) \leftarrow (\mathbf{x}_0, b - b')$; $E'[\mathbf{x}_h/b - b']$;

```
11:   else
12:     let  e = x_i ≐ x_j + b in
13:     if (E'(x_i) == (x_0, b_1)) then
14:       E'(x_j) ← (x_0, b_1 − b);  E'[x_j/b_1 − b];
15:     else if (E'(x_j) == (x_0, b_2)) then
16:       E'(x_i) ← (x_0, b_2 + b);  E'[x_i/b_2 + b];
17:     else
18:       let  E'(x_i) = x_{h_1} + b_1  ∧  E'(x_j) = x_{h_2} + b_2 in
19:       if ((h_1 == h_2)  ∧  (b_1 ≠ b_2 + b)) then
20:         return  false;
21:       else if (h_1 ≠ h_2) then
22:         if (h_1 < h_2) then
23:           E'(x_{h_1}) ← (x_{h_2}, b + (b_2 − b_1));  E'[x_{h_2}/x_{h_1} + b_1 − (b + b_2)];
24:         else
25:           E'(x_{h_2}) ← (x_{h_1}, b_1 − (b + b_2));  E'[x_{h_1}/x_{h_2} + b + (b_2 − b_1)];
26: return  E';
```

Here, $E[\mathbf{x}_i/t]$ denotes that in the conjunction of equalities E, all occurrences of variable $\mathbf{x}_i$ in right-hand sides of equalities are substituted by the term t.

Next, we present the algorithm for computing the least upper bound for two given normalised conjunctions of equalities E_1, E_2, i.e., $E_1 \sqcup E_2$. The function $\mathbf{partition}$ partitions the variables of tow conjunctions of equalities such that the properties from the proof of Lemma 4.4 hold. The $\mathbf{partition}$ is of type $2^{2^{\mathbf{X}} \times \mathbb{Z}}$, computing a set of tuples. Each tuple consist of a set of registers and a number d from $\mathbb{Z}$ denoting the difference between two equalities.
In particular:

$$\begin{aligned}\mathbf{partition} = \bigcup (X, d) \mid\ & \forall \mathbf{x}_i \in X. E_1(\mathbf{x}_i) = (\mathbf{x}_{h_1}, c_{i_1}); E_2(\mathbf{x}_i) = (\mathbf{x}_{h_2}, c_{i_2}) : \\ & \qquad d = c_{i_1} - c_{i_2} \\ & \forall \mathbf{x}_i, \mathbf{x}_j \in X \mid |X| > 1. E_1(\mathbf{x}_i) = (\mathbf{x}_{h_1}, c_{i_1}); E_1(\mathbf{x}_j) = (\mathbf{x}_{h_1}, c_{j_1}); \\ & E_2(\mathbf{x}_i) = (\mathbf{x}_{h_2}, c_{i_2}); E_2(\mathbf{x}_j) = (\mathbf{x}_{h_2}, c_{j_2}) : \\ & \qquad d = c_{i_1} - c_{i_2} = c_{j_1} - c_{j_2}\end{aligned}$$

Accordingly, the function $\delta : \mathbf{X} \mapsto \mathbb{Z}$ computes the difference of two variables w.r.t. two conjunctions E_1, E_2.

```
1:  given: E_1, E_2
2:  E = ∅
3:  Π = partition(E_1, E_2)
4:  for all Q ∈ Π | Q = (X, d) do
5:    if (|Q| == 1 | Q = ({x_i}, _)) then
6:      if ((E_1(x_i) == E_2(x_i))  ∧  E_1(x_i) == (x_0, b)) then
7:        E(x_i) ← (x_0, b)
8:      else
9:        E(x_i) = (x_i, 0)
10:   else
11:     if (∀x_j ∈ X : (E_1(x_j) == (x_0, b))  ∧  (E_2(x_j) == (x_0, b'))) then
12:       if (d == 0) then
13:         ∀x_j ∈ X : E(x_j) ← E_1(x_j)
14:       else
15:         choose smallest  x_h ∈ X
16:         ∀x_j ∈ X; j ≠ h :
17:         let  E_1(x_j) = (_, b_j);  E_1(x_h) = (_, b_h) in
18:         E(x_j) ← (x_h, b_j − b_h)
19:     else
```

20: **let** $E_1(\mathbf{x}_j) = (\mathbf{x}_{h_1}, b_j)$; $E_2(\mathbf{x}_j) = (\mathbf{x}_{h_2}, b_j)$;
21: **choose smallest** $\mathbf{x}_h \in X$
22: $E(\mathbf{x}_j) \leftarrow (\mathbf{x}_h, b_j - b_h)$
23: **return** E;

Received Month Year; revised Month Year; accepted Month Year