Interprocedural Herbrand Equalities

Markus Müller-Olm[1], Helmut Seidl[2], and Bernhard Steffen[1]

[1] Universität Dortmund, FB 4, LS V, 44221 Dortmund, Germany
{mmo,steffen}@ls5.cs.uni-dortmund.de
[2] TU München, Lehrstuhl für Informatik II, 80333 München, Germany
seidl@in.tum.de

Abstract. We present an aggressive interprocedural analysis for inferring value equalities which are independent of the concrete interpretation of the operator symbols. These equalities, called *Herbrand equalities*, are therefore an ideal basis for truly machine-independent optimizations as they hold on every machine. Besides a general *correctness* theorem, covering arbitrary call-by-value parameters and local and global variables, we also obtain two new *completeness* results: one by constraining the analysis problem to *Herbrand constants*, and one by allowing *side-effect-free functions* only. Thus if we miss a constant/equality in these two scenarios, then there exists a separating interpretation of the operator symbols.

1 Introduction

Analyses for finding definite equalities between variables or variables and expressions in a program have been used in program optimization for a long time. Knowledge about definite equalities can be exploited for performing and enhancing powerful optimizing program transformations. Examples include constant propagation, common subexpression elimination, and branch elimination [3, 8], partial redundancy elimination and loop-invariant code motion [18, 22, 12], and strength reduction [23]. Clearly, it is undecidable whether two variables always have the same value at a program point even without interpreting conditionals [17]. Therefore, analyses are bound to detect only a subset, i.e., a safe approximation, of all equivalences. Analyses based on the *Herbrand interpretation* of operator symbols consider two values equal only if they are constructed by the same operator applications. Such analyses are said to detect *Herbrand equalities*. Herbrand equalities are precisely those equalities which hold independent of the interpretation of operators. Therefore, they are an ideal basis for truly machine-independent optimizations as they hold on every machine, under all size restrictions, and independent of the chosen evaluation strategy.

In this paper, we propose an aggressive interprocedural analysis of Herbrand equalities. Note that a straight-forward generalization of intraprocedural inference algorithms to programs with procedures using techniques along the lines of [7, 20, 13] fails since the domain of Herbrand equalities is obviously infinite. Besides a general *correctness* theorem, covering arbitrary call-by-value parameters and local and global variables, we also obtain two new *completeness* results: One by constraining the analysis problem to *Herbrand constants*, and one by allowing *side-effect-free functions* only. Thus if we miss a constant/equality in these constrained scenarios, then a separating interpretation of the operator symbols can be constructed.

For reasons of exposition, we treat the case of side-effect-free functions, which constitutes an interesting class of programs in its own, separately first. The key technical idea here is to abstract the effect of a function call $\mathbf{x}_1 := f(\mathbf{x}_1, \ldots, \mathbf{x}_k)$, $\mathbf{x}_i$ program variables, by a *conditional* assignment, i.e., a pair $(\phi, \mathbf{x}_1 := e)$ consisting of a precondition ϕ together with an assignment $\mathbf{x}_1 := e$, e some term, where ϕ is a conjunction of Herbrand equalities. If the precondition is satisfied, the function call behaves like the assignment $\mathbf{x}_1 := e$, otherwise, like an assignment of an unknown value. The interesting observation is that for functions without side-effects, this is not only *sound*, i.e., infers only valid Herbrand equalities between variables, but also *complete*, i.e., infers for every program point u *all* equalities which are valid at u. In fact, our algorithm is the first inter-procedural analysis of Herbrand equalities which is complete on this class of programs. Moreover, its running time asymptotically coincides with that of the best intraprocedural algorithms for the same problem [22, 9]. Technically, the conditional assignments for functions are determined by effective weakest precondition computations for particular postconditions. For side-effect-free functions, the postcondition takes the form $\mathbf{y} \doteq \mathbf{x}_1$ where $\mathbf{y}$ is a fresh variable and $\mathbf{x}_1$ is the variable that receives the return value of the function. In the next step, we generalize this analysis to functions with *multiple return values*. Such functions correspond to procedures accessing and modifying multiple global variables. The resulting analysis is sound; moreover, we prove that it is strong enough to find all *Herbrand constants*, i.e., determines for every program point u all equalities $\mathbf{x}_j \doteq t$ for variables $\mathbf{x}_j$ and ground terms t.

Related Work. Early work on detecting equalities without considering the meaning of the operator symbols dates back to Cocke and Schwartz [4]. Their technique, the famous *value numbering*, was developed for basic blocks and assigns hash values to computations. While value numbering can be rather straightforwardly extended to forking programs, program joins pose nontrivial problems, because the concept of value equality based on equal hash numbers is too fine granular. In his seminal paper [11], Kildall presents a generalization that extends Cocke's and Schwartz's technique to flow graphs with loops by explicitly representing the equality information on terms in form of *partitions*, which allows one to treat joins of basic blocks in terms of intersection. This gave rise to a number of algorithms focusing on efficiency improvement [17, 1, 3, 19, 8, 10].

The connection of the originally pragmatic techniques to the Herbrand interpretation has been established in [21] and Steffen et al. [22], which present provably *Herbrand complete* variants of Kildall's technique and a compact representation of the Herbrand equalities in terms of *structured partition DAGs (SPDAGs)*. Even though these DAGs provide a redundancy-free representation, they still grow exponentially in the number of program terms. This problem was recently attacked by Gulwani and Necula, who arrived at a polynomial algorithm by showing that SPDAGs can be pruned, if only equalities of bounded size are of interest [9]. This observation can also be exploited for our structurally rather different interprocedural extension.

Let us finally mention that all this work abstracts conditional branching by nondeterministic choice. In fact, if equality guards are taken into account then determining whether a specific equality holds at a program point becomes undecidable [15]. Disequality constraints, however, can be dealt with intraprocedurally [15]. Whether or not inter-procedural extensions are possible is still open.

The current paper is organized as follows. In Section 2 we introduce un-interpreted programs with side-effect-free functions as the abstract model of programs for which our Herbrand analysis is complete. In Section 3 we collect basic facts about conjunctions of Herbrand equalities. In Section 4 we present the weakest precondition computation to determine the effects of function calls. In Section 5 we use this description of effects to extend an inference algorithm for intraprocedurally inferring all valid Herbrand equalities to deal with side-effect-free functions as well. In Section 6 we generalize the approach to a sound analysis for procedures accessing global variables and indicate that it infers all Herbrand constants. Finally, in Section 7 we summarize and describe further directions of research.

2 Herbrand Programs

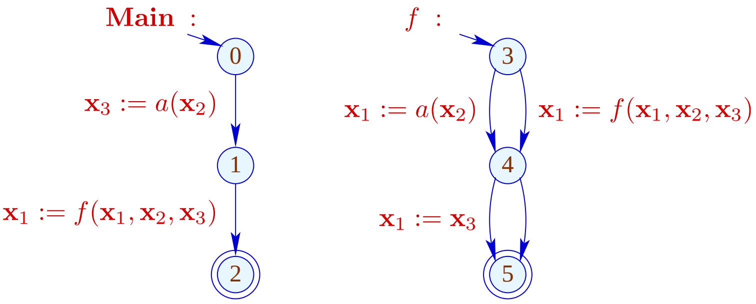
$\mathbf{Main}$:
0
$\mathbf{x}_3 := a(\mathbf{x}_2)$
1
$\mathbf{x}_1 := f(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$
2
f :
3
$\mathbf{x}_1 := a(\mathbf{x}_2)$
$\mathbf{x}_1 := f(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$
4
$\mathbf{x}_1 := \mathbf{x}_3$
5

Fig. 1. A small Herbrand program.

We model programs by systems of nondeterministic flow graphs that can recursively call each other as in Figure 1. Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ be the set of variables the program operates on. We assume that the basic statements in the program are either assignments of the form $\mathbf{x}_j := t$ for some expression t possibly involving variables from $\mathbf{X}$ or *nondeterministic* assignments $\mathbf{x}_j := ?$ and that branching in general is nondeterministic. Assignments $\mathbf{x}_j := \mathbf{x}_j$ have no effect onto the program state. They can be used as skip statements as, e.g., at the right edge from program point 4 to 5 in Figure 1 and also to abstract guards. Nondeterministic assignments $\mathbf{x}_j := ?$ safely abstract statements in a source program our analysis cannot handle, for example input statements.

A *program* comprises a finite set Funct of *function names* that contains a distinguished function $\mathbf{Main}$. First, we consider side-effect-free functions with call-by-value parameters and single return values. Without loss of generality, every call to a function f is of the form: $\mathbf{x}_1 := f(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ — meaning that the values of all variables are passed to f as actual parameters, and that the variable $\mathbf{x}_1$ always receives the return value of f which is the final value of $\mathbf{x}_1$ after execution of f.[3] In the body of f, the variables $\mathbf{x}_2, \ldots, \mathbf{x}_k$ serve as local variables. More refined calling conventions, e.g., by using designated argument variables or passing the values of expressions into formal parameters can easily be reduced to our case. Due to our standard layout of calls, each call is uniquely represented by the name f of the called function. In Section 6, we will extend our approach to procedures which read and modify *global* variables. These globals will be the variables $\mathbf{x}_1, \ldots, \mathbf{x}_m$, $m \leq k$. Procedures f are then considered as functions computing *vector assignments* $(\mathbf{x}_1, \ldots, \mathbf{x}_m) := f(\mathbf{x}_1, \ldots, \mathbf{x}_k)$.

Let Stmt be the set of assignments and calls. Program execution starts with a call to $\mathbf{Main}$. Each function name $f \in \mathsf{Funct}$ is associated with a *control flow graph* $G_f = (N_f, E_f, \mathsf{st}_f, \mathsf{ret}_f)$ that consists of a set N_f of *program points*; a set of edges $E_f \subseteq N_f \times \mathsf{Stmt} \times N_f$; a special *entry (or start) point* $\mathsf{st}_f \in N_f$; and a special *return point* $\mathsf{ret}_f \in N_f$. We assume that the program points of different functions are disjoint: $N_f \cap N_g = \emptyset$ for $f \neq g$. This can always be enforced by renaming program points. Moreover, we denote the set of edges labeled with assignments by Base and the set of edges calling some function f by Call.

We consider *Herbrand interpretation* of terms, i.e., we maintain the structure of expressions but abstract from the concrete meaning of operators. Let Ω denote a signature consisting of a set Ω_0 of constant symbols and sets $\Omega_r, r > 0$, of operator symbols of rank r which possibly may occur in right-hand sides of assignment statements or values. Let $\mathcal{T}_\Omega$ the set of all formal terms built up from Ω. For simplicity, we assume that the set Ω_0 is non-empty, and there is at least one operator. Note that under this assumption, the set $\mathcal{T}_\Omega$ is infinite. Let $\mathcal{T}_\Omega(\mathbf{X})$ denote the set of all terms with constants and operators from Ω which additionally may contain occurrences of variables from $\mathbf{X}$. Since we do not interpret constants and operators, a *state* assigning values to the variables is conveniently modeled by a mapping $\sigma : \mathbf{X} \to \mathcal{T}_\Omega$. Such mappings are also called *ground substitutions*. Accordingly, the effect of one execution of a function can be represented by a term $e \in \mathcal{T}_\Omega(\mathbf{X})$ which describes how the result value for variable $\mathbf{x}_1$ is constructed from the values of the variables $\mathbf{x}_1, \ldots, \mathbf{x}_k$ before the call. Note that such effects nicely can be accumulated from the rear where every assignment $\mathbf{x}_j := t$ extends the effect by substituting t for variable $\mathbf{x}_j$.

We define the collecting semantics of a program which will be abstracted in the sequel. Every assignment $\mathbf{x}_j := t$ induces a transformation $[\![\mathbf{x}_j := t]\!] : 2^{\mathbf{X} \to \mathcal{T}_\Omega} \to 2^{\mathbf{X} \to \mathcal{T}_\Omega}$ of the set of program states *before* the assignment into the set of states after the assignment, and a transformation $[\![[\mathbf{x}_j := t]\!]] : 2^{\mathcal{T}_\Omega(\mathbf{X})} \to 2^{\mathcal{T}_\Omega(\mathbf{X})}$ of the set of function effects accumulated *after* the assignment into the effects including the assignment:

$$[\![\mathbf{x}_j := t]\!]\, S = \{\sigma[\mathbf{x}_j \mapsto \sigma(t)] \mid \sigma \in S\} \qquad [\![[\mathbf{x}_j := t]\!]]\, T = \{e[t/\mathbf{x}_j] \mid e \in T\}$$

Here $\sigma(t)$ is the term obtained from t by replacing each occurrence of a variable $\mathbf{x}_i$ by $\sigma(\mathbf{x}_i)$ and $\sigma[\mathbf{x}_j \mapsto t']$ is the substitution that maps $\mathbf{x}_j$ to $t' \in \mathcal{T}_\Omega$ and $\mathbf{x}_i \neq \mathbf{x}_j$ to $\sigma(\mathbf{x}_i)$. Moreover, $e[t/\mathbf{x}_j]$ denotes the result of substituting t in e for variable $\mathbf{x}_j$. Similarly, we have two interpretations of the non-deterministic assignment $\mathbf{x}_j := ?$:

$$\begin{aligned}
[\![\mathbf{x}_j := ?]\!]\, S &= \bigcup\{[\![\mathbf{x}_j := c]\!]\, S \mid c \in \mathcal{T}_\Omega\} = \{\sigma[\mathbf{x}_j \mapsto \sigma(c)] \mid c \in \mathcal{T}_\Omega, \sigma \in S\} \\
[\![[\mathbf{x}_j := ?]\!]]\, T &= \bigcup\{[\![[\mathbf{x}_j := c]\!]]\, T \mid c \in \mathcal{T}_\Omega\} = \{e[c/\mathbf{x}_j] \mid c \in \mathcal{T}_\Omega, e \in T\}
\end{aligned}$$

[3] Alternatively, we could view the variable $\mathbf{x}_1$ as one global variable which serves as scratch pad for passing information from a called procedure back to its caller.

Thus, $\mathbf{x}_j := ?$ is interpreted as the non-deterministic choice between *all* assignments of values to $\mathbf{x}_j$. In a similar way, we reduce the semantics of calls to the semantics of assignments, here to the variable $\mathbf{x}_1$. For determining the sets of reaching states, we introduce a binary operator $[\![\mathsf{call}]\!] : 2^{\mathcal{T}_\Omega(\mathbf{X})} \times 2^{\mathbf{X} \to \mathcal{T}_\Omega} \to 2^{\mathbf{X} \to \mathcal{T}_\Omega}$ which uses a set of effects of the called function to transform the set of states before the call into the set of states after the call. For transforming sets of effects, we rely on a binary operator $[\![\![\mathsf{call}]\!]\!] : 2^{\mathcal{T}_\Omega(\mathbf{X})} \times 2^{\mathcal{T}_\Omega(\mathbf{X})} \to 2^{\mathcal{T}_\Omega(\mathbf{X})}$ which takes the effects of a called function to extend the effects accumulated after the call. We define:

$$\begin{array}{lll} [\![\mathsf{call}]\!]\,(T, S) & = \bigcup\{[\![\mathbf{x}_1 := t]\!]\,S \mid t \in T\} & = \{\sigma[\mathbf{x}_1 \mapsto \sigma(t)] \mid t \in T, \sigma \in S\} \\ [\![\![\mathsf{call}]\!]\!]\,(T_1, T_2) & = \bigcup\{[\![\![\mathbf{x}_1 := t]\!]\!]\,T_2 \mid t \in T_1\} & = \{e[t/\mathbf{x}_1] \mid t \in T_1, e \in T_2\} \end{array}$$

Thus, a call is interpreted as the non-deterministic choice between all assignments $\mathbf{x}_1 := t$ where t is a potential effect of the called function. We use the operators $[\![\![\ldots]\!]\!]$ to characterize the sets of effects of functions, $\mathbf{S}(f) \subseteq \mathcal{T}_\Omega(\mathbf{X})$, $f \in \mathsf{Funct}$, by means of a constraint system $\mathbf{S}$:

$$\begin{array}{llll} [\mathrm{S1}] & \mathbf{S}(f) & \supseteq \mathbf{S}(\mathsf{st}_f) & \\ [\mathrm{S2}] & \mathbf{S}(\mathsf{ret}_f) & \supseteq \{\mathbf{x}_1\} & \\ [\mathrm{S3}] & \mathbf{S}(u) & \supseteq [\![\![s]\!]\!]\,(\mathbf{S}(v)) & \text{if } (u, s, v) \in \mathsf{Base} \\ [\mathrm{S4}] & \mathbf{S}(u) & \supseteq [\![\![\mathsf{call}]\!]\!]\,(\mathbf{S}(f), \mathbf{S}(v)) & \text{if } (u, f, v) \in \mathsf{Call} \end{array}$$

Note that the effects are accumulated in sets $\mathbf{S}(u) \subseteq \mathcal{T}_\Omega(\mathbf{X})$ for program points u from the *rear*, i.e., starting from the return points. Calls are dealt with by constraint [S4]. If the ingoing edge (u, f, v) is a call to a function f, we extend the terms already found for v with the potential effects of the called function f by means of the operator $[\![\![\mathsf{call}]\!]\!]$. Obviously, the operators $[\![\![\mathbf{x}_j := t]\!]\!]$ and hence also the operators $[\![\![\mathbf{x}_j := ?]\!]\!]$ and $[\![\![\mathsf{call}]\!]\!]$ are monotonic (even distributive). Therefore, by Knaster-Tarski's fixpoint fixpoint theorem, the constraint system $\mathbf{S}$ has a unique least solution whose components (for simplicity) are denoted by $\mathbf{S}(u), \mathbf{S}(f)$ as well.

We use the effects $\mathbf{S}(f)$ of functions and the operators $[\![\ldots]\!]$ to characterize the sets of *reaching program states*, $\mathbf{R}(u), \mathbf{R}(f) \subseteq (\mathbf{X} \to \mathcal{T}_\Omega)$, by a constraint system $\mathbf{R}$:

$$\begin{array}{llll} [\mathrm{R1}] & \mathbf{R}(\mathbf{Main}) & \supseteq \mathbf{X} \to \mathcal{T}_\Omega & \\ [\mathrm{R2}] & \mathbf{R}(f) & \supseteq \mathbf{R}(u)\,, & \text{if } (u, f, \_) \in \mathsf{Call} \\ [\mathrm{R3}] & \mathbf{R}(\mathsf{st}_f) & \supseteq \mathbf{R}(f) & \\ [\mathrm{R4}] & \mathbf{R}(v) & \supseteq [\![s]\!]\,(\mathbf{R}(u))\,, & \text{if } (u, s, v) \in \mathsf{Base} \\ [\mathrm{R5}] & \mathbf{R}(v) & \supseteq [\![\mathsf{call}]\!]\,(\mathbf{S}(f), \mathbf{R}(u))\,, & \text{if } (u, f, v) \in \mathsf{Call} \end{array}$$

Again, since all occurring operators are monotonic (even distributive), this constraint system has a unique least solution whose components are denoted by $\mathbf{R}(u)$ and $\mathbf{R}(f)$.

3 Herbrand Equalities

A substitution $\sigma : \mathbf{X} \to \mathcal{T}_\Omega(\mathbf{X})$ (possibly containing variables in the image terms) satisfies a conjunction of equalities $\phi \equiv s_1 \doteq t_1 \wedge \ldots \wedge s_m \doteq t_m$ (where $s_i, t_i \in \mathcal{T}_\Omega(\mathbf{X})$ and "$\doteq$" a formal equality symbol) iff $\sigma(s_i) = \sigma(t_i)$ for $i = 1, \ldots, m$. Then we also write $\sigma \models \phi$. We say, ϕ is valid at a program point u iff it is valid for all states $\sigma \in \mathbf{R}(u)$.

As we rely on Herbrand interpretation here, an equality which is valid at a program point u is also called a valid *Herbrand equality* at u.

Let us briefly recall some basic facts about conjunctions of equations. A conjunction ϕ is *satisfiable* iff $\sigma \models \phi$ for at least one σ. Otherwise, i.e., if ϕ is unsatisfiable, ϕ is logically equivalent to false. This value serves as the bottom value of the lattice we use in our analysis. The greatest value is given by the *empty* conjunction which is always true and therefore also denoted by true. The ordering is by logical implication "$\Rightarrow$". Whenever the conjunction ϕ is satisfiable, then there is a *most general* satisfying substitution σ, i.e., $\sigma \models \phi$ and for every other substitution τ satisfying ϕ, $\tau = \tau_1 \circ \sigma$ for some substitution τ_1. Such a substitution is often also called a *most general unifier* of ϕ. In particular, this means that the conjunction ϕ is equivalent to $\bigwedge_{\mathbf{x}_i \neq \sigma(\mathbf{x}_i)} \mathbf{x}_i \doteq \sigma(\mathbf{x}_i)$. Thus, every satisfiable conjunction of equations is equivalent to a (possibly empty) finite conjunction of equations $\mathbf{x}_{j_i} \doteq t_i$ where the left-hand sides $\mathbf{x}_{j_i}$ are distinct variables and none of the equations is of the form $\mathbf{x}_j \doteq \mathbf{x}_j$. Let us call such conjunctions *reduced*. The following fact is crucial for proving termination of our proposed fixpoint algorithms.

Proposition 1. *For every sequence $\phi_0 \Leftarrow \ldots \Leftarrow \phi_m$ of pairwise inequivalent conjunctions ϕ_j using k variables, $m \leq k+1$.* □

Proposition 1 follows since for satisfiable reduced non-equivalent conjunctions ϕ_i, ϕ_{i+1}, $\phi_i \Leftarrow \phi_{i+1}$ implies that ϕ_{i+1} contains strictly more equations than ϕ_i.

In order to construct an abstract lattice of properties, we consider *equivalence classes* of conjunctions of equations which, however, will always be represented by one of their members. Let $\mathbb{E}(\mathbf{X}')$ denote the set of all (equivalence classes of) finite reduced conjunctions of equations with variables from $\mathbf{X}'$. This set is partially ordered w.r.t. "$\Rightarrow$" (on the representatives). The pairwise greatest lower bound always exists and is given by conjunction "$\wedge$". Since by Proposition 1, all descending chains in this lattice are ultimately stable, not only finite but also infinite subsets $X \subseteq \mathbb{E}(\mathbf{X}')$ have a greatest lower bound. Hence, $\mathbb{E}(\mathbf{X}')$ is a *complete* lattice.

4 Weakest Preconditions

For reasoning about return values of functions, we introduce a fresh variable $\mathbf{y}$ and determine for every function f the weakest precondition, $\mathbf{WP}(f)$, of the equation $\mathbf{y} \doteq \mathbf{x}_1$ w.r.t. f. Given that the set of effects of f equals $T \subseteq \mathcal{T}_\Omega(\mathbf{X})$, the weakest precondition of $\mathbf{y} \doteq \mathbf{x}_1$ is given by $\bigwedge\{\mathbf{y} \doteq e \mid e \in T\}$ – which is equivalent to a finite conjunction due to the compactness property of Proposition 1. Intuitively, true as precondition means that the function f has an empty set of effects only, whereas $\phi' \wedge \mathbf{y} \doteq e$ expresses that the single value returned for $\mathbf{x}_1$ is e — under the assumption that ϕ' holds. Thus, ϕ' implies all equalities $e \doteq e'$, $e' \in T$. In particular, if ϕ' is unsatisfiable, i.e., equivalent to false, then the function may return different values.

For computing preconditions, we will work with the subset $\mathbb{E}_\mathbf{y}$ of $\mathbb{E}(\mathbf{X} \cup \{\mathbf{y}\})$ of (equivalence classes of) conjunctions ϕ of equalities with variables from $\mathbf{X} \cup \{\mathbf{y}\}$ which are either equivalent to true or equivalent to a conjunction $\phi' \wedge \mathbf{y} \doteq e$ for some $e \in \mathcal{T}_\Omega(\mathbf{X})$. We can assume that ϕ' does not contain y, since any occurrence of $\mathbf{y}$ in ϕ' can be replaced with e. We introduce a function $\alpha_\mathbf{S} : 2^{\mathcal{T}_\Omega(\mathbf{X})} \to \mathbb{E}_\mathbf{y}$ by:

$$\alpha_{\mathbf{S}}(T) = \bigwedge_{e \in T} (\mathbf{y} \doteq e)$$

By transforming arbitrary unions into conjunctions, $\alpha_{\mathbf{S}}$ is an *abstraction* in the sense of [6]. Our goal is to define abstract operators $[\![\mathbf{x}_j := t]\!]^\sharp$, $[\![\mathbf{x}_j := ?]\!]^\sharp$ and $[\![\mathsf{call}]\!]^\sharp$.

A precondition $[\![\mathbf{x}_j := t]\!]^\sharp\,\phi$ of a conjunction of equalities ϕ for an assignment $\mathbf{x}_j := t$ can be obtained by the well-known rule:

$$[\![\mathbf{x}_j := t]\!]^\sharp \phi = \phi[t/\mathbf{x}_j]$$

where $\phi[t/\mathbf{x}_j]$ denotes the formula obtained from ϕ by substituting t for $\mathbf{x}_j$. This transformation returns the *weakest* precondition for the assignment. The transformer for nondeterministic assignments is reduced to the transformation of assignments:

$$[\![\mathbf{x}_j := ?]\!]^\sharp \phi = \bigwedge_{c \in \mathcal{T}_\Omega} [\![\mathbf{x}_j := c]\!]^\sharp \phi = \bigwedge_{c \in \mathcal{T}_\Omega} \phi[c/\mathbf{x}_j]$$

By assumption, $\mathcal{T}_\Omega$ contains at least two elements $t_1 \neq t_2$. If ϕ contains $\mathbf{x}_j$, then $\phi[t_1/\mathbf{x}_j] \wedge \phi[t_2/\mathbf{x}_j]$ implies $t_1 \doteq t_2$ (because we are working with Herbrand interpretation) which is false by the choice of t_1, t_2. Hence, the transformer can be simplified to:

$$[\![\mathbf{x}_j := ?]\!]^\sharp \phi \;=\; \phi[t_1/\mathbf{x}_j] \wedge \phi[t_2/\mathbf{x}_j] \;=\; \begin{cases} \mathsf{false} & \text{if } \mathbf{x}_j \text{ occurs in } \phi \\ \phi & \text{otherwise} \end{cases}$$

The first equation means that $\mathbf{x}_j := ?$ is semantically equivalent (w.r.t. weakest preconditions of Herbrand equalities) to the nondeterministic choice between the two assignments $\mathbf{x}_j := t_1$ and $\mathbf{x}_j := t_2$.

In order to obtain safe preconditions for calls, we introduce a binary operator $[\![\mathsf{call}]\!]^\sharp$. In the first argument, this operator takes a precondition ϕ_1 of a function body for the equation $\mathbf{y} \doteq \mathbf{x}_1$. The second argument of $[\![\mathsf{call}]\!]^\sharp$ is a postcondition ϕ_2 after the call. We define:

$$\begin{aligned} [\![\mathsf{call}]\!]^\sharp(\mathsf{true}, \phi_2) &= \mathsf{true} \\ [\![\mathsf{call}]\!]^\sharp(\phi' \wedge (\mathbf{y} \doteq e), \phi_2) &= \begin{cases} \phi' \wedge \phi_2[e/\mathbf{x}_1] & \text{if } \mathbf{x}_1 \text{ occurs in } \phi_2 \\ \phi_2 & \text{otherwise} \end{cases} \end{aligned}$$

If the weakest precondition of $\mathbf{y} \doteq \mathbf{x}_1$ is true, we return true, since a set of effects is abstracted with true only if it is empty. In order to catch the intuition of the second rule of the definition, first assume that ϕ' is true. This corresponds to the case where the abstracted set of effects consists of a single term e only. The function call then is semantically equivalent to the assignment $\mathbf{x}_1 := e$. Accordingly, our definition gives: $[\![\mathsf{call}]\!]^\sharp(\mathbf{y} \doteq e, \phi_2) \;= \phi_2[e/\mathbf{x}_1]$. In general, different execution paths may return different terms e' for $\mathbf{x}_1$. The precondition ϕ' then implies that all these e' equal e. If ϕ_2 does not contain $\mathbf{x}_1$, ϕ_2 is not affected by assignments to $\mathbf{x}_1$ anyway. Therefore in this case, $[\![\mathsf{call}]\!]^\sharp(\phi_1, \phi_2) = \phi_2$. If on the other hand, ϕ_2 contains the variable $\mathbf{x}_1$, then ϕ_2 holds after the call provided $\phi_2[e/\mathbf{x}_1]$ holds before the call as well as ϕ'.

The definition of $[\![\mathsf{call}]\!]^\sharp(\phi_1, \phi_2)$ is independent of the chosen representation of ϕ_1. To see this, assume that ϕ_1 is also equivalent to $\phi_1' \wedge (\mathbf{y} \doteq t_1)$ for some ϕ_1', t_1 not containing $\mathbf{y}$. Then in particular, $\phi' \wedge (\mathbf{y} \doteq t)$ implies $\mathbf{y} \doteq t_1$ as well as ϕ_1' from which we deduce that ϕ' also implies $t \doteq t_1$. Therefore, $\phi' \wedge \phi_2[t/\mathbf{x}_1]$ implies $\phi_1' \wedge \phi_2[t_1/\mathbf{x}_1]$. By exchanging the roles of ϕ', t and ϕ_1', t_1 we find the reverse implication and the equivalence follows. We establish the following distributivity properties:

Proposition 2. *1.* $[\![\mathbf{x}_j := t]\!]^\sharp$ *preserves* true *and distributes over* "$\wedge$".
2. $[\![\mathbf{x}_j := ?]\!]^\sharp$ *preserves* true *and distributes over* "$\wedge$".
3. In each argument, the operation $[\![\mathsf{call}]\!]^\sharp$ *preserves* true *and distributes over* "$\wedge$".

Proof: Assertion 1 holds since substitutions preserve true and commute with "$\wedge$". Assertion 2 follows from 1, since $[\![\mathbf{x}_j := ?]\!]^\sharp\,\phi = ([\![\mathbf{x}_j := t_1]\!]^\sharp\,\phi) \wedge ([\![\mathbf{x}_j := t_2]\!]^\sharp\,\phi)$ for ground terms $t_1 \neq t_2$. For the third assertion, the statement concerning the second argument of $[\![\mathsf{call}]\!]^\sharp$ is straightforwardly verified from the definition. The same is true for the preservation of true in the first argument. It remains to verify that

$$[\![\mathsf{call}]\!]^\sharp(\phi_1 \wedge \phi_2, \phi) = [\![\mathsf{call}]\!]^\sharp(\phi_1, \phi) \wedge [\![\mathsf{call}]\!]^\sharp(\phi_2, \phi)$$

If either ϕ_1 or ϕ_2 equal false, the assertion is obviously true. The same holds if either ϕ_1 or ϕ_2 equal true. Otherwise, we can assume that for $i = 1, 2$, ϕ_i is satisfiable, reduced and of the form: $\phi'_i \wedge (\mathbf{y} \doteq e_i)$ for some ϕ'_i not containing $\mathbf{y}$. If ϕ does not contain $\mathbf{x}_1$, the assertion is again trivially true. Therefore, we additionally may assume that ϕ contains at least one occurrence of $\mathbf{x}_1$. Then by definition, $[\![\mathsf{call}]\!]^\sharp(\phi_i, \phi) = \phi'_i \wedge \phi[e_i/\mathbf{x}_1]$, and we obtain:

$$\begin{aligned}[\![\mathsf{call}]\!]^\sharp(\phi_1, \phi) \wedge [\![\mathsf{call}]\!]^\sharp(\phi_2, \phi) &= \phi'_1 \wedge \phi[e_1/\mathbf{x}_1] \wedge \phi'_2 \wedge \phi[e_2/\mathbf{x}_1] \\ &= \phi'_1 \wedge \phi'_2 \wedge (e_1 \doteq e_2) \wedge \phi[e_1/\mathbf{x}_1]\end{aligned}$$

since ϕ contains an occurrence of $\mathbf{x}_1$. On the other hand, we may also rewrite $\phi_1 \wedge \phi_2$ to: $\phi'_1 \wedge \phi'_2 \wedge (e_1 \doteq e_2) \wedge (\mathbf{y} \doteq e_1)$ where only the last equation contains $\mathbf{y}$. Therefore:

$$[\![\mathsf{call}]\!]^\sharp(\phi_1 \wedge \phi_2, \phi) = \phi'_1 \wedge \phi'_2 \wedge (e_1 \doteq e_2) \wedge \phi[e_1/\mathbf{x}_1]$$

which completes the proof. □

We construct a constraint system $\mathbf{WP}$ for preconditions of functions by applying the abstraction function $\alpha_{\mathbf{S}}$ to the constraint system $\mathbf{S}$ for collecting effects of functions. Thus, we replace $\{\mathbf{x}_1\}$ with $(\mathbf{y} \doteq \mathbf{x}_1)$ and the operators $[\![\ldots]\!]$ with $[\![\ldots]\!]^\sharp$. We obtain:

$$\begin{array}{llll}[\mathbf{WP}1] & \mathbf{WP}(f) & \Rightarrow \mathbf{WP}(\mathsf{st}_f) & \\ [\mathbf{WP}2] & \mathbf{WP}(\mathsf{ret}_f) & \Rightarrow (\mathbf{y} \doteq \mathbf{x}_1) & \\ [\mathbf{WP}3] & \mathbf{WP}(u) & \Rightarrow [\![s]\!]^\sharp(\mathbf{WP}(v))\,, & \text{if } (u,s,v) \in \mathsf{Base} \\ [\mathbf{WP}4] & \mathbf{WP}(u) & \Rightarrow [\![\mathsf{call}]\!]^\sharp(\mathbf{WP}(f), \mathbf{WP}(v))\,, & \text{if } (u,f,v) \in \mathsf{Call}\end{array}$$

By Knaster-Tarski fixpoint theorem, the constraint system $\mathbf{WP}$ has a greatest solution w.r.t. "$\Rightarrow$" which we denote with $\mathbf{WP}(f), \mathbf{WP}(u)$, $f \in \mathsf{Funct}, u \in N$. With Proposition 2, we verify that $\alpha_{\mathbf{S}}$ has the following properties:

1. $\alpha_{\mathbf{S}}(\{\mathbf{x}_1\}) = (\mathbf{y} \doteq \mathbf{x}_1)$;
2. $\alpha_{\mathbf{S}}([\![\mathbf{x}_j := t]\!]\,T) = [\![\mathbf{x}_j := t]\!]^\sharp(\alpha_{\mathbf{S}}(T))$;
3. $\alpha_{\mathbf{S}}([\![\mathbf{x}_j := ?]\!]\,T) = [\![\mathbf{x}_j := ?]\!]^\sharp(\alpha_{\mathbf{S}}(T))$;
4. $\alpha_{\mathbf{S}}([\![\mathsf{call}]\!](T_1, T_2)) = [\![\mathsf{call}]\!]^\sharp(\alpha_{\mathbf{S}}(T_1), \alpha_{\mathbf{S}}(T_2))$.

By the Transfer Lemma from fixpoint theory (c.f., e.g., [2, 5]), we therefore find:

Theorem 1 (Weakest Preconditions). *Let p be a program of size n with k variables.*

1. *For every function f of p, $\mathbf{WP}(f) = \bigwedge\{(\mathbf{y} \doteq t) \mid t \in \mathbf{S}(f)\}$; and for every program point u of p, $\mathbf{WP}(u) = \bigwedge\{(\mathbf{y} \doteq t) \mid t \in \mathbf{S}(u)\}$.*
2. *The greatest solution of constraint system $\mathbf{WP}$ can be computed in time $\mathcal{O}(n \cdot k \cdot \Delta)$ where Δ is the maximal size of a DAG representation of a conjunction occurring during the fixpoint computation.* □

Thus, the greatest solution of the constraint system $\mathbf{WP}$ precisely characterizes the weakest preconditions of the equality $\mathbf{x}_1 \doteq \mathbf{y}$. Evaluation of "$\wedge$" as well as of a right-hand side in the constraint system $\mathbf{WP}$ at most doubles the sizes of DAG representations of occurring conjunctions. Therefore, the value Δ is bounded by $2^{\mathcal{O}(n \cdot k)}$.

Example 1. Consider the function f from Figure 1. First, f and every program point is initialized with the top element true of the lattice $\mathbb{E}_{\mathbf{y}}$. The first approximation of the weakest precondition at program point 4 for $\mathbf{y} \doteq \mathbf{x}_1$ at 5, then is:

$$\mathbf{WP}(4) \;=\; (\mathbf{y} \doteq \mathbf{x}_1) \wedge ([\![\mathbf{x}_1 := \mathbf{x}_3]\!]^\sharp\; (\mathbf{y} \doteq \mathbf{x}_1) \;=\; (\mathbf{y} \doteq \mathbf{x}_1) \wedge (\mathbf{y} \doteq \mathbf{x}_3)$$

Accordingly, we obtain for the start point 3,

$$\begin{aligned}\mathbf{WP}(3) &= [\![\mathsf{call}]\!]^\sharp(\mathsf{true}, \mathbf{WP}(4)) \wedge ([\![\mathbf{x}_1 := a(\mathbf{x}_2)]\!]^\sharp\; (\mathbf{WP}(4))) \\ &= \mathsf{true} \wedge (\mathbf{y} \doteq a(\mathbf{x}_2)) \wedge (\mathbf{y} \doteq \mathbf{x}_3) \\ &= (\mathbf{x}_3 \doteq a(\mathbf{x}_2)) \wedge (\mathbf{y} \doteq \mathbf{x}_3)\end{aligned}$$

Thus, we obtain $(\mathbf{x}_3 \doteq a(\mathbf{x}_2)) \wedge (\mathbf{y} \doteq \mathbf{x}_3)$ as a first approximation for the weakest precondition of $\mathbf{y} \doteq \mathbf{x}_1$ w.r.t. f. Since the fixpoint computation already stabilizes here, we have found that $\mathbf{WP}(f) \;=\; (\mathbf{x}_3 \doteq a(\mathbf{x}_2)) \wedge (\mathbf{y} \doteq \mathbf{x}_3)$. □

5 Inferring Herbrand Equalities

For computing weakest preconditions, we have relied on conjunctions of equalities, (pre-) ordered by "$\Rightarrow$" where the greatest lower bound was implemented by the logical "$\wedge$". For *inferring* Herbrand equalities, we again use conjunctions of equalities, now over the set of variables $\mathbf{X}$ alone, i.e., we use $\mathbb{E} = \mathbb{E}(\mathbf{X})$ — but now we resort to least upper bounds (instead of greatest lower bounds). Conceptually, the least upper bound $\phi_1 \sqcup \phi_2$ of two elements in $\mathbb{E}$ corresponds to the best approximation of the disjunction $\phi_1 \vee \phi_2$. Thus, it is the conjunction of all equalities implied both by ϕ_1 and ϕ_2. We can restrict ourselves to equalities of the form $\mathbf{x}_i \doteq t$ ($\mathbf{x}_i \in \mathbf{X}, t \in \mathcal{T}_\Omega(\mathbf{X})$). Accordingly,

$$\begin{aligned}\phi_1 \sqcup \phi_2 &= \bigwedge\{\mathbf{x}_j \doteq t \mid (\phi_1 \vee \phi_2) \Rightarrow (\mathbf{x}_j \doteq t)\} \\ &= \bigwedge\{\mathbf{x}_j \doteq t \mid (\phi_1 \Rightarrow (\mathbf{x}_j \doteq t)) \wedge (\phi_2 \Rightarrow (\mathbf{x}_j \doteq t))\}\end{aligned}$$

Consider, e.g., $\phi_1 \equiv (\mathbf{x}_1 \doteq g(a(\mathbf{x}_3))) \wedge (\mathbf{x}_2 \doteq a(\mathbf{x}_3))$ and $\phi_2 \equiv (\mathbf{x}_1 \doteq g(b)) \wedge (\mathbf{x}_2 \doteq b)$. Then $\phi_1 \sqcup \phi_2$ is equivalent to $\mathbf{x}_1 \doteq g(\mathbf{x}_2)$.

Conjunctions of equalities are not closed under existential quantification. Therefore, we introduce the operators $\exists^\sharp \mathbf{x}_j$ as the best approximations to $\exists\, \mathbf{x}_j$ in $\mathbb{E}$:

$$\begin{aligned}\exists^\sharp \mathbf{x}_j.\phi &= \bigwedge\{\mathbf{x}_i \doteq t \mid i \neq j, (\exists\, \mathbf{x}_j.\phi) \Rightarrow (\mathbf{x}_i \doteq t), t \text{ does not contain } \mathbf{x}_j\} \\ &= \bigwedge\{\mathbf{x}_i \doteq t \mid i \neq j, \phi \Rightarrow (\mathbf{x}_i \doteq t), t \text{ does not contain } \mathbf{x}_j\}\end{aligned}$$

So, for example, $\exists^\sharp \mathbf{x}_2.\, (\mathbf{x}_1 \doteq a(\mathbf{x}_2)) \wedge (\mathbf{x}_3 \doteq b(a(\mathbf{x}_2), c)) \;=\; \mathbf{x}_3 \doteq b(\mathbf{x}_1, c)$

We readily verify that "$\exists^\sharp \mathbf{x}_j$" preserves false and commutes with "$\sqcup$". The operations "$\sqcup$" and "$\exists^\sharp\, \mathbf{x}_j$" can be efficiently implemented on *partition DAG* representations [22] . More specifically, '$\exists^\sharp\, \mathbf{x}_j$" is linear-time whereas the least upper bound of two conjunctions with DAG representations of sizes n_1, n_2 can be performed in time $\mathcal{O}(n_1 + n_2)$ resulting in (a DAG representation of) a conjunction of size $\mathcal{O}(n_1 + n_2)$.

We define the abstraction $\alpha_{\mathbf{R}} : 2^{\mathbf{X} \to \mathcal{T}_\Omega} \to \mathbb{E}$ that maps a set of states to the conjunction of all equalities valid for all states in the set:

$$\alpha_{\mathbf{R}}(S) = \bigwedge\{x_j \doteq t \mid \forall \sigma \in S : \sigma \models x_j \doteq t\}$$

As an equality holds for a state $\sigma : \mathbf{X} \to \mathcal{T}_\Omega$ iff it is implied by the conjunction $\mathbf{x}_1 \doteq \sigma(\mathbf{x}_1) \wedge \ldots \wedge \mathbf{x}_k \doteq \sigma(\mathbf{x}_k)$ we have $\alpha_{\mathbf{R}}(S) = \bigsqcup\{\bigwedge_{i=1}^k \mathbf{x}_i \doteq \sigma(\mathbf{x}_i) \mid \sigma \in S\}$. In particular, this implies that $\alpha_{\mathbf{R}}$ commutes over unions.

We must provide abstractions of the operators $[\![\ldots]\!]$. We define:

$$
\begin{aligned}
[\![\mathbf{x}_j := t]\!]^\sharp \phi &= \exists^\sharp \mathbf{y}.\phi[\mathbf{y}/\mathbf{x}_j] \wedge (\mathbf{x}_j \doteq t[\mathbf{y}/\mathbf{x}_j]) \\
[\![\mathbf{x}_j := ?]\!]^\sharp \phi &= \bigsqcup\{[\![\mathbf{x}_j := c]\!]^\sharp \phi \mid c \in \mathcal{T}_\Omega\} \\
&= \bigsqcup\{\exists^\sharp \mathbf{y}.\phi[\mathbf{y}/\mathbf{x}_j] \wedge (\mathbf{x}_j \doteq c]) \mid c \in \mathcal{T}_\Omega\} \\
&= \exists^\sharp \mathbf{y}.\bigsqcup\{\phi[\mathbf{y}/\mathbf{x}_j] \wedge (\mathbf{x}_j \doteq c]) \mid c \in \mathcal{T}_\Omega\} \\
&= \exists^\sharp \mathbf{y}.\phi[\mathbf{y}/\mathbf{x}_j] \\
&= \exists^\sharp \mathbf{x}_j.\phi
\end{aligned}
$$

Thus, $[\![\mathbf{x}_j := t]\!]^\sharp \phi$ is the best abstraction of the strongest postcondition of ϕ w.r.t. $\mathbf{x}_j := t$ and the abstract transformer $[\![\mathbf{x}_j := ?]\!]^\sharp$ is given by abstract existential quantification. For instance, we have: $[\![\mathbf{x}_1 := \mathbf{x}_3]\!]^\sharp\, (\mathbf{x}_3 \doteq a(\mathbf{x}_2)) \;=\; (\mathbf{x}_1 \doteq a(\mathbf{x}_2)) \wedge (\mathbf{x}_3 \doteq a(\mathbf{x}_2))$ and $[\![\mathbf{x}_3 := ?]\!]^\sharp\, (\mathbf{x}_3 \doteq a(\mathbf{x}_2)) \wedge (\mathbf{x}_1 \doteq a(\mathbf{x}_2)) \;=\; (\mathbf{x}_1 \doteq a(\mathbf{x}_2))$. These definitions provide obvious implementations using partition DAGs. In particular, the abstract transformer $[\![\mathbf{x}_j := t]\!]^\sharp$ can be computed in time linear in the size n_1 of the argument and the size n_2 of (a DAG representation of) t. Moreover, the DAG representation of the result is again of size $\mathcal{O}(n_1 + n_2)$. A similar estimation also holds for nondeterministic assignments.

The crucial point in constructing an analysis is the abstract operator $[\![\mathsf{call}]\!]^\sharp$ for function calls. The first argument of this operator takes the weakest precondition ϕ_1 of $(\mathbf{y} \doteq \mathbf{x}_1)$ for a (possibly empty) set of effects of some function. The second argument ϕ_2 takes a conjunction of equalities which is valid before the call. We define:

$$
\begin{aligned}
[\![\mathsf{call}]\!]^\sharp(\mathsf{true}, \phi_2) &= \mathsf{false} \\
[\![\mathsf{call}]\!]^\sharp(\phi' \wedge (\mathbf{y} \doteq e), \phi_2) &= \begin{cases} [\![\mathbf{x}_1 := e]\!]^\sharp \phi_2 & \text{if} \quad \phi_2 \Rightarrow \phi' \\ [\![\mathbf{x}_1 := ?]\!]^\sharp \phi_2 & \text{otherwise} \end{cases}
\end{aligned}
$$

The first rule states that everything is true at an unreachable program point. Otherwise, we can write ϕ_1 as $\phi' \wedge (\mathbf{y} \doteq e)$ where ϕ' and e do not contain $\mathbf{y}$. If ϕ' is implied by the precondition ϕ_2, we are guaranteed that all return values for $\mathbf{x}_1$ are equivalent to e. In this case, the call behaves like an assignment $\mathbf{x}_1 := e$. Otherwise, at least two different return values are possible. Then we treat the function call like a non-deterministic assignment $\mathbf{x}_1 := ?$.

Example 2. Consider, e.g., the call of function f in $\mathbf{Main}$ in Fig. 1. By Example 1, $\mathbf{WP}(f)$ equals $\phi_1 = (\mathbf{x}_3 \doteq a(\mathbf{x}_2)) \wedge (\mathbf{y} \doteq \mathbf{x}_3)$. Before the call, $\phi_2 = (\mathbf{x}_3 \doteq a(\mathbf{x}_2))$ holds. Accordingly, we obtain:

$$
\begin{aligned}
[\![\mathsf{call}]\!]^\sharp((\mathbf{x}_3 \doteq a(\mathbf{x}_2)) \wedge (\mathbf{y} \doteq \mathbf{x}_3), \mathbf{x}_3 \doteq a(\mathbf{x}_2)) &= [\![\mathbf{x}_1 := \mathbf{x}_3]\!]^\sharp(\mathbf{x}_3 \doteq a(\mathbf{x}_2)) \\
&= (\mathbf{x}_1 \doteq a(\mathbf{x}_2)) \wedge (\mathbf{x}_3 \doteq a(\mathbf{x}_2)) . \qquad \square
\end{aligned}
$$

In order to precisely infer *all* valid Herbrand equalities, we observe:

Proposition 3. *1. $[\![\mathbf{x}_j := t]\!]^\sharp$ and $[\![\mathbf{x}_j := ?]\!]^\sharp$ preserve* false *and commute with "$\sqcup$".*

2. In the first argument, $[\![\mathsf{call}]\!]^\sharp$ maps true *to* false *and translates "$\wedge$"into "$\sqcup$", i.e.,*

$$[\![\mathsf{call}]\!]^\sharp(\mathsf{true}, \phi) = \mathsf{false} \quad \textit{and} \quad [\![\mathsf{call}]\!]^\sharp(\phi_1 \wedge \phi_2, \phi) = [\![\mathsf{call}]\!]^\sharp(\phi_1, \phi) \sqcup [\![\mathsf{call}]\!]^\sharp(\phi_2, \phi) .$$

In the second argument, $[\![\mathsf{call}]\!]^\sharp$ preserves false *and commutes with "$\sqcup$", i.e.,*

$$[\![\mathsf{call}]\!]^\sharp(\phi, \mathsf{false}) = \mathsf{false} \quad \textit{and} \quad [\![\mathsf{call}]\!]^\sharp(\phi, \phi_1 \sqcup \phi_2) = [\![\mathsf{call}]\!]^\sharp(\phi, \phi_1) \sqcup [\![\mathsf{call}]\!]^\sharp(\phi, \phi_2) .$$

Proof: Statement 1 easily follows from the definitions. Therefore we only prove the second statement about the properties of $[\![\mathsf{call}]\!]^\sharp$. The assertion concerning the second argument easily follows from assertion 1. The assertion about the transformation of true in the first argument follows from the definition. Therefore, it remains to consider a conjunction $\phi_1 \wedge \phi_2$ in the first argument of $[\![\mathsf{call}]\!]^\sharp$. We distinguish two cases.

Case 1: $\phi_1 \wedge \phi_2$ is not satisfiable, i.e., equivalent to false. Then $[\![\mathsf{call}]\!]^\sharp(\phi_1 \wedge \phi_2, \phi) = [\![\mathbf{x}_1 := ?]\!]^\sharp\, \phi$. If any of the ϕ_i is also not satisfiable, then $[\![\mathsf{call}]\!]^\sharp(\phi_i, \phi) = [\![\mathbf{x}_1 := ?]\!]^\sharp\, \phi$ which subsumes the effect of any assignment $\mathbf{x}_1 := e$ onto ϕ, and the assertion follows. Therefore assume that both ϕ_1 and ϕ_2 are satisfiable. Each of them then can be written as $\phi_i' \wedge (\mathbf{y} \doteq e_i)$. If any of the ϕ_i' is not implied by ϕ, then again $[\![\mathsf{call}]\!]^\sharp(\phi_i, \phi) = [\![\mathbf{x}_1 := ?]\!]^\sharp\, \phi$ which subsumes the effect of the assignment $\mathbf{x}_1 := e_{3-i}$ onto ϕ. Thus,

$$[\![\mathsf{call}]\!]^\sharp(\phi_1, \phi) \sqcup [\![\mathsf{call}]\!]^\sharp(\phi_2, \phi) \;=\; [\![\mathsf{call}]\!]^\sharp(\phi_i, \phi) \;=\; [\![\mathbf{x}_1 := ?]\!]\; \phi \;=\; [\![\mathsf{call}]\!]^\sharp(\phi_1 \wedge \phi_2, \phi)\,.$$

If on the other hand, both ϕ_i' are implied by ϕ, then $\phi_1' \wedge \phi_2'$ is satisfiable. Thus, $\sigma(e_1) \neq \sigma(e_2)$ for any $\sigma \models \phi_1' \wedge \phi_2'$. In particular, $e_1 \doteq e_2$ cannot be implied by ϕ. Since ϕ_i' is implied by ϕ, $[\![\mathsf{call}]\!]^\sharp(\phi_i, \phi) = [\![\mathbf{x}_1 := e_i]\!]^\sharp\, \phi$. On the other hand, for every ψ containing $\mathbf{x}_1$, it is impossible that both $\phi \Rightarrow \psi[e_1/\mathbf{x}_1]$ and $\phi \Rightarrow \psi[e_2/\mathbf{x}_1]$ hold. Therefore, the least upper bound of $[\![\mathsf{call}]\!]^\sharp(\phi_1, \phi)$ and $[\![\mathsf{call}]\!]^\sharp(\phi_2, \phi)$ is given by the conjunction of all ψ implied by ϕ which do not contain $\mathbf{x}_1$. This conjunction precisely equals $[\![\mathbf{x}_1 := ?]\!]^\sharp\; \phi = [\![\mathsf{call}]\!]^\sharp(\mathsf{false}, \phi)$, and the assertion follows.

Case 2: $\phi_1 \wedge \phi_2$ is satisfiable. Then also both of the ϕ_i are satisfiable and can be written as conjunctions $\phi_i' \wedge (\mathbf{y} \doteq e_i)$ for some ϕ_i' and e_i not containing $\mathbf{y}$. If ϕ does not imply $\phi_1' \wedge \phi_2'$, then both sides of the equation are equal to $[\![\mathbf{x}_1 := ?]\!]\;\phi$ and nothing is to prove. Therefore, assume that $\phi \Rightarrow \phi_1' \wedge \phi_2'$. If ϕ also implies $e_1 \doteq e_2$, then for every ψ, $\phi \Rightarrow \psi[e_1/\mathbf{x}_1]$ iff $\phi \Rightarrow \psi[e_2/\mathbf{x}_1]$. Therefore in this case,

$$[\![\mathsf{call}]\!]^\sharp(\phi_i, \phi) \;=\; [\![\mathbf{x}_1 := e_1]\!]^\sharp\; \phi \;=\; [\![\mathbf{x}_1 := e_2]\!]^\sharp\; \phi \;=\; [\![\mathsf{call}]\!]^\sharp(\phi_1 \wedge \phi_2, \phi)$$

and the assertion follows. If ϕ does not imply $e_1 \doteq e_2$, the least upper bound of $[\![\mathbf{x}_1 := e_i]\!]^\sharp\; \phi$ is the conjunction of all ψ not containing $\mathbf{x}_1$ which are implied by ϕ — which equals:

$$[\![\mathbf{x}_1 := ?]\!]^\sharp\; \phi \;=\; [\![\mathsf{call}]\!]^\sharp(\phi' \wedge (\mathbf{y} \doteq e_1), \phi) \;=\; [\![\mathsf{call}]\!]^\sharp(\phi_1 \wedge \phi_2, \phi)$$

for $\phi' \equiv \phi_1' \wedge \phi_2' \wedge (e_1 \doteq e_2)$, and the assertion follows. □

Applying the abstraction $\alpha_{\mathbf{R}}$ to the constraint system $\mathbf{R}$ of reaching states, we obtain the constraint system $\mathbf{H}$:

$$\begin{array}{lll}
[\mathbf{H}1] & \mathbf{H}(\mathbf{Main}) \Leftarrow \mathsf{true} & \\
[\mathbf{H}2] & \mathbf{H}(f) \quad\;\; \Leftarrow \mathbf{H}(u)\,, & \text{if } (u, f, \_) \in \mathsf{Call} \\
[\mathbf{H}3] & \mathbf{H}(\mathsf{st}_f) \quad \Leftarrow \mathbf{H}(f) & \\
[\mathbf{H}4] & \mathbf{H}(v) \quad\;\; \Leftarrow [\![s]\!]^\sharp(\mathbf{H}(u))\,, & \text{if } (u, s, v) \in \mathsf{Base} \\
[\mathbf{H}5] & \mathbf{H}(v) \quad\;\; \Leftarrow [\![\mathsf{call}]\!]^\sharp(\mathbf{WP}(f), \mathbf{H}(u))\,, & \text{if } (u, f, v) \in \mathsf{Call}
\end{array}$$

Note that $\mathbf{WP}(f)$ is used in constraint $\mathbf{H}5$ as a summary information for function f. Note also that $\mathbf{H}$ specifies a forwards analysis while $\mathbf{WP}$ accumulates information in a backwards manner. Again by Knaster-Tarski fixpoint theorem, the constraint system $\mathbf{H}$ has a least solution which we denote with $\mathbf{H}(f), \mathbf{H}(u)$, $f \in \mathsf{Funct}, u \in N$. By Proposition 3, we have:

1. $\alpha_{\mathbf{R}}(\mathbf{X} \to \mathcal{T}_\Omega) = \mathsf{true}$;
2. $\alpha_{\mathbf{R}}([\![\mathbf{x}_j := t]\!]\; S) = [\![\mathbf{x}_j := t]\!]^\sharp(\alpha_{\mathbf{R}}(S))$;
3. $\alpha_{\mathbf{R}}([\![\mathbf{x}_j := ?]\!]\; S) = [\![\mathbf{x}_j := ?]\!]^\sharp(\alpha_{\mathbf{R}}(S))$;
4. $\alpha_{\mathbf{R}}([\![\mathsf{call}]\!](T, S)) = [\![\mathsf{call}]\!]^\sharp(\alpha_{\mathbf{S}}(T), \alpha_{\mathbf{R}}(S))$.

We finally obtain:

Theorem 2 (Soundness and Completeness for Side-effect-free Functions). *Assume p is a Herbrand program of size n with k variables.*

1. *For every function f, $\mathbf{H}(f) = \bigsqcup\{\bigwedge_{i=1}^{k} \mathbf{x}_i \doteq \sigma(\mathbf{x}_i) \mid \sigma \in \mathbf{R}(f)\}$; and for every program point u, $\mathbf{H}(u) = \bigsqcup\{\bigwedge_{i=1}^{k} \mathbf{x}_i \doteq \sigma(\mathbf{x}_i) \mid \sigma \in \mathbf{R}(u)\}$.*
2. *Given the values $\mathbf{WP}(f)$, $f \in \mathsf{Funct}$, the least solution of the constraint system $\mathbf{H}$ can be computed in time $\mathcal{O}(n \cdot k \cdot \Delta)$ where Δ is the maximal size of a DAG representation of an occurring conjunction.*

By statement 1 of the theorem, our analysis of side-effect-free functions is not only sound, i.e., never returns a wrong result, but *complete*, i.e., we compute for every program point u and for every function f, the conjunction of *all* equalities which are valid when reaching u and a call of f, respectively. Each application of "$\sqcup$" as well as of any right-hand side in the constraint system $\mathbf{H}$ may at most double the sizes of DAG representations of occurring conjunctions. Together with the corresponding upper bound for the greatest solution of the constraint system $\mathbf{WP}$, the value Δ therefore can be bounded by $2^{\mathcal{O}(n \cdot k)}$. Indeed, this upper bound is *tight* in that it matches the corresponding lower bound for the intra-procedural case [9].

Example 3. Consider again the program from Figure 1. At the start point 0 of $\mathbf{Main}$, no non-trivial equation holds. Therefore, $\mathbf{H}(0) = \mathsf{true}$. For program point 1, we have:

$$\mathbf{H}(1) = [\![\mathbf{x}_3 := a(\mathbf{x}_2)]\!]^\sharp \mathsf{true} = \mathbf{x}_3 \doteq a(\mathbf{x}_2)$$

In Section 4, we have computed the weakest precondition of $\mathbf{y} \doteq \mathbf{x}_1$ for the function f as $(\mathbf{x}_3 \doteq a(\mathbf{x}_2)) \wedge (\mathbf{y} \doteq \mathbf{x}_3)$. Since $\mathbf{H}(1)$ implies the equation $\mathbf{x}_3 \doteq a(\mathbf{x}_2)$, we obtain a representation of all equalities valid at program exit 2 by:

$$\begin{aligned} \mathbf{H}(2) &= [\![\mathsf{call}]\!]^\sharp(\mathbf{WP}(f), \mathbf{H}(1)) = [\![\mathbf{x}_1 := \mathbf{x}_3]\!]^\sharp(\mathbf{x}_3 \doteq a(\mathbf{x}_2)) \\ &= (\mathbf{x}_3 \doteq a(\mathbf{x}_2) \wedge (\mathbf{x}_1 \doteq a(\mathbf{x}_2)) \end{aligned}$$

Thus at the return point of $\mathbf{Main}$ both $\mathbf{x}_3 \doteq a(\mathbf{x}_2)$ and $\mathbf{x}_1 \doteq a(\mathbf{x}_2)$ holds. □

6 Programs with Global Variables

In this section, we indicate how our inference algorithm for side-effect-free functions can be extended to an inference algorithm for functions with multiple return values. For the following, we assume that the first m variables are global or, equivalently, that a run of a function f simultaneously computes new values for all variables $\mathbf{x}_1, \ldots, \mathbf{x}_m$. Thus, a function call is now denoted by the vector assignment: $(\mathbf{x}_1, \ldots, \mathbf{x}_m) := f(\mathbf{x}_1, \ldots, \mathbf{x}_k)$. One execution of a function is modeled by a tuple $\tau = (e_1, \ldots, e_m)$ where $e_j \in \mathcal{T}_\Omega(\mathbf{X})$ expresses how the value of variable $\mathbf{x}_j$ after the call depends on the values of the variables before the call. This tuple can also be viewed as a substitution $\tau : \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \to \mathcal{T}_\Omega(\mathbf{X})$. Accordingly, we change the concrete semantics of a call to:

$$\begin{aligned} [\![\mathsf{call}]\!](T, S) &= \{\sigma[\mathbf{x}_1 \mapsto \sigma(e_1), \ldots, \mathbf{x}_m \mapsto \sigma(e_m)] \mid (e_1, \ldots, e_m) \in T, \sigma \in S\} \\ [\![\mathsf{call}]\!](T_1, T_2) &= \{\tau_1 \circ \tau_2 \mid \tau_i \in T_i\} \end{aligned}$$

In order to obtain effective approximations of the set of effects of function calls, we conceptually abstract one function call computing the values of m variables, by m function calls each of which computes the value of one global variable independently of the others. Technically, we abstract sets of k-tuples with k-tuples of sets. This means that we perform for each variable $x_j \in \{x_1, \ldots, x_m\}$ a separate analysis $\mathbf{P}_j$ of the function body. Accordingly, we generalize the system $\mathbf{WP}$ to a constraint system $\mathbf{P}$:

$$\begin{array}{lll}
[\mathbf{P}_{\mathrm{j}}1] & \mathbf{P}_j(f) \quad\;\; \Rightarrow \mathbf{P}_j(\mathsf{st}_f) & \\
[\mathbf{P}_{\mathrm{j}}2] & \mathbf{P}_j(\mathsf{ret}_f) \Rightarrow (\mathbf{y}_j \doteq \mathbf{x}_j) & \\
[\mathbf{P}_{\mathrm{j}}3] & \mathbf{P}_j(u) \quad\;\; \Rightarrow [\![s]\!]^\sharp(\mathbf{P}_j(v))\,, & \text{if } (u,s,v) \in \mathsf{Base} \\
[\mathbf{P}_{\mathrm{j}}4] & \mathbf{P}_j(u) \quad\;\; \Rightarrow [\![\mathsf{call}_m]\!]^\sharp(\mathbf{P}_1(f),\ldots,\mathbf{P}_m(f),\mathbf{P}_j(v))\,, & \text{if } (u,f,v) \in \mathsf{Call}
\end{array}$$

Here for $j = 1,\ldots,m$, $\mathbf{y}_j$ is a distinct fresh variable meant to receive the return value for the global variable $\mathbf{x}_j$. The key difference to the constraint system $\mathbf{WP}$ is the treatment of calls by means of the new operator $[\![\mathsf{call}_m]\!]^\sharp$. This operator takes $m+1$ arguments $\phi_1,\ldots,\phi_m,\psi$ (instead of 2 in Section 4). For $j = 1,\ldots,m$, the formula $\phi_j \in \mathbb{E}_{\mathbf{y}_j}$ represents a precondition of the call for the equality $\mathbf{x}_j \doteq \mathbf{y}_j$. The formula ψ on the other hand represents a postcondition for the call. We define:

$$\begin{array}{ll}
[\![\mathsf{call}_m]\!]^\sharp(\ldots,\mathsf{true},\ldots,\psi) & = \mathsf{true} \\
[\![\mathsf{call}_m]\!]^\sharp(\phi'_1 \wedge (\mathbf{y}_1 \doteq e_1),\ldots,\phi'_m \wedge (\mathbf{y}_m \doteq e_m),\psi) & = \bigwedge_{i\in I}\phi'_i \wedge \psi[e_1/\mathbf{x}_1,\ldots,e_m/\mathbf{x}_m]
\end{array}$$

where $I = \{i \in \{1,\ldots,m\} \mid \mathbf{x}_i$ occurs in $\psi\}$. As in Section 4, $\phi_j \Leftrightarrow \mathsf{true}$ implies that the set of effects is empty. In this case, the operator returns true. Therefore, now assume that for every j, ϕ_j is equivalent to $\phi'_j \wedge \mathbf{y}_j \doteq e_j$ where ϕ'_j and e_j contain only variables from $\mathbf{X}$. If for all j, ϕ'_j equals true, i.e., the return value for $\mathbf{x}_j$ equals e_j, then the call behaves like the substitution $\psi[e_1/\mathbf{x}_1,\ldots,e_m/\mathbf{x}_m]$, i.e., the multiple assignment $(x_1,\ldots,x_m) := (e_1,\ldots,e_m)$. Otherwise, we add the preconditions ϕ'_i for every $\mathbf{x}_i$ occurring in ψ to guarantee that all return values for $\mathbf{x}_i$ are equal to e_i.

As in Section 5, we can use the greatest solution of $\mathbf{P}$ to construct a constraint system $\mathbf{H}'$ from $\mathbf{H}$ by replacing the constraints $\mathbf{H}5$ for calls with the new constraints:

$$[\mathbf{H}5'] \quad \mathbf{H}(v) \Leftarrow [\![\mathsf{call}_m]\!]^\sharp(\mathbf{P}_1(f),\ldots,\mathbf{P}_m(f),\mathbf{H}(u))\,, \ \text{if } (u,f,v) \in \mathsf{Call}$$

Here, the necessary new abstract operator $[\![\mathsf{call}_m]\!]^\sharp$ for calls is defined by:

$$\begin{array}{ll}
[\![\mathsf{call}_m]\!]^\sharp(\ldots,\mathsf{true},\ldots,\psi) & = \mathsf{false} \\
[\![\mathsf{call}_m]\!]^\sharp(\phi'_1 \wedge (\mathbf{y}_1 \doteq e_1),\ldots,\phi'_m \wedge (\mathbf{y}_m \doteq e_m),\psi) & = \\
\qquad\qquad \exists^\sharp \mathbf{y}_1,\ldots,\mathbf{y}_m.\psi[\mathbf{y}/\mathbf{x}] \wedge \bigwedge_{j\in I}(\mathbf{x}_j \doteq e_j[\mathbf{y}/\mathbf{x}]\,) &
\end{array}$$

where $[\mathbf{y}/\mathbf{x}]$ is an abbreviation for the replacement $[\mathbf{y}_1/\mathbf{x}_1,\ldots,\mathbf{y}_m/\mathbf{x}_m]$ and I denotes the set $\{i \mid \psi \Rightarrow \phi'_i\}$. We find:

Theorem 3 (Soundness). *Assume we are given a Herbrand program p with m globals.*

1. *The greatest solution of the constraint system $\mathbf{P}$ for p yields for every function f of p, safe preconditions for the postconditions $\mathbf{x}_i \doteq \mathbf{y}_i$, $i = 1,\ldots,m$.*
2. *The least solution of the constraint system $\mathbf{H}'$ for p yields for every program point u of p, a conjunction of Herbrand equalities which are valid at u.*

The analysis has running-time $\mathcal{O}(n \cdot m^2 \cdot k \cdot \Delta)$ where n is the size of the program and Δ is the maximal size of a conjunction occurring during the analysis. □

At each evaluation of a constraint during the fixpoint computation for $\mathbf{P}$ the maximal size of a conjunction is at most multiplied by a factor of $(m+1)$. Since the number of such evaluations is bounded by $\mathcal{O}(n \cdot m \cdot k)$, we conclude that Δ is bounded by $(m+1)^{\mathcal{O}(n\cdot m\cdot k)}$. Beyond mere soundness, we can say more about the quality of our analysis.

In fact, it is strong enough to determine all interprocedural *Herbrand constants*, i.e., to infer for all program points, all equalities of the form $\mathbf{x}_j \doteq t$, t a ground term.

Theorem 4 (Completeness for Constants). *Assume p is a Herbrand program of size n with m globals. Then the following holds:*

1. *For every program point u of p, every variable $\mathbf{x}_j \in \mathbf{X}$ and ground term t, the equality $\mathbf{x}_j \doteq t$ holds at u iff it is implied by $\mathbf{H}_m(u)$.*
2. *All Herbrand constants up to size d can be determined in time $\mathcal{O}(n \cdot m^2 \cdot k^2 \cdot d)$.*

Thus, our algorithm allows for maximally precise interprocedural propagation of Herbrand constants. Moreover, if we are interested in constants up to a given size only, the algorithm can be tuned to run in polynomial time.

Proof: [Sketch] The idea for a proof of the first assertion of Theorem 4 is to introduce a new liberal notion of effect of a function which describes the effect by means of a tuple of sets (instead of a set of tuples). Similar to Sections 4 and 5 one then proves that the constraint systems $\mathbf{P}$ together with $\mathbf{H}_m$ precisely compute all Herbrand equalities valid relative to the liberal notion of effect. This implies that our analysis is *sound*. In order to prove that it is *complete* for equalities $\mathbf{x}_j \doteq t$, t a ground term, we show that if two states at a program point u computed with the liberal effect result in different values for $\mathbf{x}_j$ then there are also two states at u computed according to the strict notion of effects which differ in their values for $\mathbf{x}_j$. □

7 Conclusion

We have presented an interprocedural algorithm for inferring valid Herbrand equalities. Our analysis is *complete* for side-effect-free functions in that it allows us to infer *all* valid Herbrand equalities. We also indicated that our analysis for procedures with more than one global still allows us to determine *all* Herbrand constants. Constant propagation can even be tuned to run in polynomial time if we are interested in constants of bounded size only. Our key idea for the case of side-effect-free functions is to describe the effect of a function by its weakest precondition of the equality $\mathbf{y} \doteq \mathbf{x}_1$.

It remains for future work to investigate the practical usability of the proposed analysis. It also might be interesting to see whether other interprocedural analyses can take advantage of a related approach. In [16], for instance, we discuss an application for determining affine relations. In [15] we have presented an analysis of Herbrand equalities which takes disequality guards into account. It is completely open in how far this intra-procedural analysis can be generalized to some inter-procedural setting.

References

1. B. Alpern, M. Wegman, and F. K. Zadeck. Detecting Equality of Variables in Programs. In *15th ACM Symp. on Principles of Programming Languages (POPL)*, pages 1–11, 1988.
2. K. R. Apt and G. D. Plotkin. Countable Nondeterminism and Random Assignment. *Journal of the ACM*, 33(4):724–767, 1986.
3. C. Click and K. D. Cooper. Combining Analyses, Combining Optimizations. *ACM Transactions on Programming Languages and Systems*, 17(2):181–196, 1995.

4. J. Cocke and J. T. Schwartz. Programming languages and their compilers. Courant Institute of Mathematical Sciences, NY, 1970.
5. P. Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Electronic Notes in Theoretical Computer Science*, 6, 1997. URL: `www.elsevier.nl/locate/entcs/volume6.html`.
6. P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *4th ACM Symp. on Principles of Programming Languages (POPL)*, 1977.
7. P. Cousot and R. Cousot. Static Determination of Dynamic Properties of Recursive Procedures. In E. Neuhold, editor, *IFIP Conf. on Formal Description of Programming Concepts*, pages 237–277. North-Holland, 1977.
8. K. Gargi. A Sparse Algorithm for Predicated Global Value Numbering. In *ACM Conf. on Programming Language Design and Implementation (PLDI)*, pages 45–56, 2002.
9. S. Gulwani and G. C. Necula. A Polynomial-time Algorithm for Global Value Numbering. In *11th Int. Static Analysis Symposium (SAS),*. Springer Verlag, 2004.
10. S. Gulwani and G. C. Necula. Global Value Numbering Using Random Interpretation. In *31st ACM Symp. on Principles of Programming Languages (POPL)*, pages 342–352, 2004.
11. G. A. Kildall. A Unified Approach to Global Program Optimization. In *First ACM Symp. on Principles of Programming Languages (POPL)*, pages 194–206, 1973.
12. J. Knoop, O. Rüthing, and B. Steffen. Code Motion and Code Placement: Just Synonyms? In *6th ESOP*, LNCS 1381, pages 154–196. Springer-Verlag, 1998.
13. J. Knoop and B. Steffen. The Interprocedural Coincidence Theorem. In *Compiler Construction (CC)*, pages 125–140. LNCS 541, Springer-Verlag, 1992.
14. S. S. Muchnick and N. D. Jones, editors. *Program Flow Analysis: Theory and Applications*. Prentice Hall, Engelwood Cliffs, New Jersey, 1981.
15. M. Müller-Olm, O. Rüthing, and H. Seidl. Checking Herbrand Equalities and Beyond. In *Proceedings of VMCAI 2005*. to appear, Springer-Verlag, 2005.
16. M. Müller-Olm, H. Seidl, and B. Steffen. Interprocedural Analysis for Free. Technical Report 790, Fachbereich Informatik, Universität Dortmund, 2004.
17. J. H. Reif and R. Lewis. Symbolic Evaluation and the Gobal Value Graph. In *4th ACM Symp. on Principles of Programming Languages (POPL)*, pages 104–118, 1977.
18. B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Global Value Numbers and Redundant Computations. In *15th ACM Symp. on Principles of Programming Languages (POPL)*, pages 12–27, 1988.
19. O. Rüthing, J. Knoop, and B. Steffen. Detecting Equalities of Variables: Combining Efficiency with Precision. In *6th Int. Static Analysis Symposium (SAS)*, LNCS 1694, pages 232–247. Springer-Verlag, 1999.
20. M. Sharir and A. Pnueli. Two Approaches to Interprocedural Data Flow Analysis. In [14], chapter 7, pages 189–233.
21. B. Steffen. Optimal Run Time Optimization—Proved by a New Look at Abstract Interpretations. In *Proc. 2nd International Joint Conference on Theory and Practice of Software Development (TAPSOFT'87)*, LNCS 249, pages 52–68. Springer Verlag, 1987.
22. B. Steffen, J. Knoop, and O. Rüthing. The Value Flow Graph: A Program Representation for Optimal Program Transformations. In *3rd European Symp. on Programming (ESOP)*, LNCS 432, pages 389–405. Springer-Verlag, 1990.
23. B. Steffen, J. Knoop, and O. Rüthing. Efficient Code Motion and an Adaption to Strength Reduction. In *4th International Joint Conference on the Theory and Practice of Software Development* (*TAPSOFT*), LNCS 494, pages 394–415. Springer-Verlag, 1991.