Inter-procedural Two-Variable Herbrand Equalities

Stefan Schulze Frielinghaus, Michael Petter and Helmut Seidl

Technische Universität München, Boltzmannstrasse 3, 85748 Garching, Germany,
`{schulzef,petter,seidl}@in.tum.de`,
WWW: `http://www2.in.tum.de/~{schulzef,petter,seidl}`

Abstract. We prove that all valid Herbrand equalities can be inter-procedurally inferred for programs where all assignments are taken into account whose right-hand sides depend on at most one variable. The analysis is based on procedure summaries representing the weakest pre-conditions for finitely many generic post-conditions with template variables. In order to arrive at effective representations for all occurring weakest pre-conditions, we show for almost all values possibly computed at run-time, that they can be uniquely factorized into tree patterns and a terminating ground term. Moreover, we introduce an approximate notion of subsumption which is effectively decidable and ensures that finite conjunctions of equalities may not grow infinitely. Based on these technical results, we realize an effective fixpoint iteration to infer all inter-procedurally valid Herbrand equalities for these programs.

How can we infer that an equality such as $\mathbf{x} \doteq \mathbf{y}$ holds at some program point, if the operators by which the program variables $\mathbf{x}$ and $\mathbf{y}$ are computed, do not satisfy obvious algebraic laws? This is the case, e.g., when either very high-level operations such as `sqrt`, or very low-level operations such as bit-shift are involved or, generally, for floating-point calculations. Still, the equality $\mathbf{x} \doteq \mathbf{y}$ can be inferred, if $\mathbf{x}$ and $\mathbf{y}$ are computed by means of *syntactically* identical terms of operator applications. The equality then is called *Herbrand* equality. The problem of inferring valid Herbrand equalities dates back to [1] where it was introduced as the famous *value numbering* problem. Since quite a while, algorithms are known which, in absence of procedures, infer *all* valid Herbrand equalities [11,21]. These algorithms can even be tuned to run in polynomial time, if only invariants of polynomial size are of interest [7]. Surprisingly little is known about Herbrand equalities if recursive procedure calls are allowed. In [17] it has been observed that the intra-procedural techniques can be extended to programs with local variables and *functions* – but without global variables. The ideas there are strong enough to generally infer all Herbrand *constants* in programs with procedures and both local and global variables, i.e., invariants of the form $\mathbf{x} \doteq t$ where t is ground. Another tractable case of invariants is obtained if only assignments are taken into account whose right-hand sides have at most *one occurrence* of a variable [18]. Thus, assignment $\mathbf{x} = f(\mathbf{y}, a)$; is considered while assignments such as $\mathbf{x} = f(\mathbf{y}, \mathbf{y})$; or $\mathbf{x} = f(\mathbf{y}, \mathbf{z})$; are approximated with $\mathbf{x} = ?$;, i.e., by an assignment of an unknown value to $\mathbf{x}$. The idea is to encode ground terms as numbers. Then Herbrand equalities can be represented as polynomial equalities with a fixed number of variables and of bounded degree. Accordingly, techniques from linear algebra are sufficient to infer all valid Herbrand equalities for such programs. As a special case, Petter's class of programs from [18] subsumes those programs where only *unary* operators are involved. Such programs have been considered by [8]. Interestingly, the latter paper arrives at decidability by a completely different line of argument, namely, by exploiting properties of the free monoid generated from the unary operators. Another avenue to decidability is to restrict the control structure of programs to be analyzed. In [5], the restricted class of *Sloopy* Programs is introduced where the format of loop as well as recursion is drastically restricted. For this class an algorithm is not only provided to decide arbitrary equalities between variables but also disequalities.

On the other hand, when only affine numerical expressions as well as affine program invariants are of concern, the set of valid invariants at a program point form a *vector space* which can be effectively represented. This observation is exploited in [14] to apply methods from linear algebra to infer all valid affine program invariants. These methods later have been adapted to the case where values of variables are not from a field, but where integers will overflow at some power of 2, i.e., are taken from a modular ring. Note that in the latter structure, some number different from 0 may be a zero divisor and thus does not have a multiplicative inverse [15]. For some applications, an analysis of *general* equalities is not necessary. In applications such as coalescing of registers [16] or detection of local variables in low-level code [4], it suffices to infer equalities involving two variables only. In the affine case, algorithms for inferring all two-variable equalities can be constructed which have better complexities as the corresponding algorithms for general equalities [4].

The question whether or not *all* inter-procedurally valid Herbrand equalities can be inferred, is still open. Here, we consider the case of Herbrand equalities containing two variables only. These are equalities such as $\mathbf{x} \doteq f(g(\mathbf{y}), \mathbf{y}, a)$, i.e., right-hand sides of equalities may contain only a single variable, but this multiple times. Accordingly, in programs only assignments are taken into account whose right-hand sides contain (arbitrarily many) occurrences of at most one variable. Our main result is that under this provision, *all* inter-procedurally valid two-variable Herbrand equalities can be inferred.

Our novel analysis is based on calculating weakest pre-conditions for all occurring post-conditions. Since there may be infinitely many potential post-conditions for a called procedure, we rely on *generic* post-conditions to obtain finite representations of procedure summaries. In a generic post-condition *second-order* variables are used as place-holders for yet unknown relationships between program variables. In the generic post-condition

$$A(\mathbf{x}) \doteq B(\mathbf{y})$$

the second-order variables A and B take as values terms with (possibly multiple occurrences of) *holes* (which we call *templates*). To realize our algorithm for inferring all inter-procedurally valid two-variable equalities, we thus require

– a method to finitely represent all occurring conjunctions of equalities,
– a method for proving that one conjunction subsumes another conjunction, i.e., a method to detect when the greatest fixpoint computation has terminated;
– a guarantee that the fixpoint ever will be reached.

Note here that the equalities occurring during the weakest pre-condition computation of a generic post-condition may contain occurrences of second-order variables. Thus, subsumption between conjunctions of equalities is subtly related to second-order unification [6]. Second-order unification asks whether a conjunction of equalities possibly containing second-order variables is satisfiable. Since long, it is known that generally, second-order unification is undecidable. Undecidability of second-order unification even holds if only a single unary second-order variable is involved [12]. In contrast, the problem of *context* unification, i.e., the variant of second-order unification where second-order variables range over terms with single occurrences of holes only, has recently been proven to be decidable [10]. It is worth mentioning that neither of the two cases directly applies to our application, since we consider unary second-order variables (as context unification) but let variables range over terms with one or multiple occurrences of holes (differently from context unification). To the best of our knowledge, decidability of satisfiability is still open for our case.

In this paper, we will not solve the satisfiability problem for the given unification problem. Instead, we introduce two novel ideas to circumvent this problem and still infer all inter-procedurally valid two-variable Herbrand equalities. First, we introduce a notion of *approximate* subsumption. This means that our algorithm does not allow to prove implications between all conjunctions of equalities — but at least sufficiently many so that accumulation of *infinite* conjunctions is ruled out. Second, we note that subsumption is not required for arbitrary valuations of program variables. Instead it suffices to consider values which may possibly be constructed by the program at run-time. For programs where every right-hand side of assignments contain occurrences of single variables only, we observe that the ground terms possibly occurring at run-time, have a specific structure, which allows for a *unique factorization* of these terms into irreducible templates — at least, if these ground terms are sufficiently *large*. Our factorization result applied to these kind of values, enables us to make use of the monoidal methods of [8]. This approach, which works for sufficiently large terms, then is complemented with a dedicated treatment of finitely many exceptional cases. By that, we ultimately succeed to construct an effective approximative subsumption algorithm which allows us to restrict the number of equalities in occurring conjunctions and to determine all valid two-variable Herbrand equalities.

In order to arrive at our key result, namely an algorithm to infer all valid inter-procedural two-variable Herbrand equalities, we thus build on the following two novel technical constructions:

– a method to uniquely factorize the kind of values possibly occuring at run-time (except finitely many) of a given program;

– a notion of approximative subsumption which is decidable and still guarantees that every occurring conjunction of equalities is effectively equivalent to a finite conjunction.

Subsequently, we sketch how not only all two-variable equalities, but *all* interprocedurally valid Herbrand equalities can be inferred, if only all right-hand sides in assignments each contain occurrences of at most one variable.

Our paper is organized as follows. Section 1 briefly introduces our programming model. Section 2 presents our basic **WP** based approach of inferring all valid program invariants. In Section 3, we provide general background on the cancellation and factorization properties of terms and prove a first compactness result for equalities with template variables but no occurrences of program variables. In Section 4 we then provide an algorithm for inferring all two-variable equalities — at least, for programs which are *initialization-restricted* (see Section 4 for a precise definition of this restriction). Technically, this restriction implies that all occurring terms can be uniquely factorized into irreducible terms. In order to arrive at an algorithm for programs which are not initialization-restricted, we complement this approach in Section 5 with a dedicated treatment of values where a unique factorization is not possible. Finally, Section 6 indicates how our methods can be extended to general Herbrand equalities.

1 Programs

For the purpose of this paper, we consider imperative programs which consist of a finite set P of procedures such as:

```
Herbrand x, y;
main() {
    x = a;
    y = a;
    p();
}

p() {
    if (*) {
        x = f(x, x);
        p();
        y = f(y, y);
    }
}
```

Instead of operating on the syntax of programs, we prefer to represent each procedure by a (non-deterministic) control flow graph. Figure 1 shows, e.g., the control flow graphs for the given example program. Formally, the control flow graph for a procedure p consists of:

– A finite set N_p of program points where $s_p, r_p \in N_p$ represent the start and return point of the procedure p;
– A finite set E_p of edges (u, s, v) where $u, v \in N_p$ are program points and s denotes a basic statement.

For simplicity, we proceed in the style of Sharir/Pnueli in [20] and consider parameterless procedures which operate on global variables only. In the following,

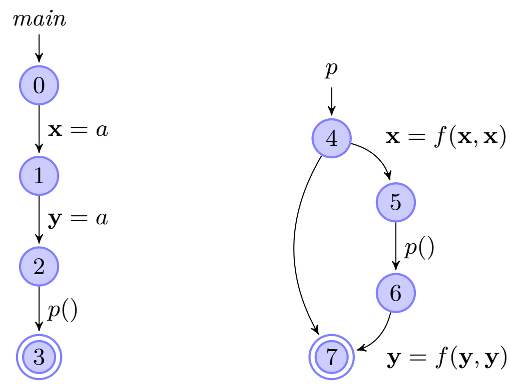

Fig. 1. The corresponding CFGs for the example program.

$\mathbf{X}$ denotes the finite set of program variables. As *values*, we consider uninterpreted operator expressions only. Thus, values are constructed from atomic values by means of (uninterpreted) operator applications. Let Ω denote a signature containing a non-empty set of atomic values Ω_0 and sets $\Omega_k, k > 0$, of constructors of rank k. Then $\mathcal{T}_\Omega$ denotes the set of all possible (ground) terms over Ω, and $\mathcal{T}_\Omega(\mathbf{X})$ the set of all possible terms over Ω and (possibly) occurrences of program variables from $\mathbf{X}$. In general, we will omit brackets around the argument of unary symbols. Thus, we may, e.g., write $h\mathbf{x}$ instead of $h(\mathbf{x})$.

As basic statements, we only consider assignments and procedure calls. An assignment $\mathbf{x} = ?$ non-deterministically assigns *any* value to the program variable $\mathbf{x}$, whereas an assignment $\mathbf{x} = t$ assigns the value constructed according to the right-hand side term $t \in \mathcal{T}_\Omega(\mathbf{X})$. A procedure call is of the form $p()$ for a procedure name p.

In this paper, we only consider assignments whose right-hand sides contain occurrences of at most one variable. The assignments occurring in the example program from Figure 1 have this property. Note that this program does not fall into Petter's class, since the right-hand sides of assignments contain more than one occurrence of a variable. In *general* programs with arbitrary assignments, the assignments with right-hand sides not conforming to the given restriction may, e.g., be abstracted by the non-deterministic assignment of *any* value.

2 Computing Weakest Pre-conditions

In order to prove a given assertion or infer all valid invariants, we would like to calculate weakest pre-conditions, to determine for every program point the assumptions to be met for the queried assertion to hold at the given program point. Since the program model makes use of non-deterministic branching, we may assume w.l.o.g. that every program point is *reachable*. In particular, this implies that no procedure is definitely non-terminating, i.e., that for every procedure p, there is at least one execution path from the start point of p reaching the end point of p.

Example 1. Consider the program from Figure 1. At program exit, the invariant $\mathbf{x} \doteq \mathbf{y}$ holds. In a proof of this fact by means of a **WP** computation, weakest pre-conditions must be provided for procedure p and all assertions $\mathbf{x} \doteq t_k$, $k \geq 0$, where $t_0 = \mathbf{y}$ and for $k > 0$, $t_k = f(t_{k-1}, t_{k-1})$. This set of post-conditions is not only infinite, but also makes use of an ever increasing number of variable occurrences. Thus, an immediate encoding, e.g., into bounded degree polynomials as in [18] is not obvious. □

In order to summarize the effect of a procedure for multiple but similar post-conditions, we tabulate the weakest pre-conditions for *generic* post-conditions only. Generic post-conditions are assertions which contain *template variables* which later may be instantiated differently in different contexts for arriving post-conditions. This idea has been applied, e.g., for affine equalities [14,16,4], for polynomial equalities [13,18], or for Herbrand equalities with unary operators [8]. The generic post-conditions which are of interest here, are of the forms

$$A\mathbf{x} \doteq C \quad \text{or} \quad A\mathbf{x} \doteq B\mathbf{y}$$

where $\mathbf{x}, \mathbf{y}$ are program variables, the *ground* template variable C is meant to receive a constant value, and the template variables A, B take *templates* as values, i.e., terms over the ranked alphabet Ω and having at least one occurrence of the (fresh) place holder variable $\bullet$. Computing weakest pre-conditions operates on assertions where an assertion is a (possibly infinite) conjunction of equalities. The equalities occurring during weakest pre-condition calculations are of the forms:

$$As \doteq C \quad \text{or} \quad As \doteq Bt$$

where s, t are terms possibly containing a program variable, i.e., $s, t \in \mathcal{T}_\Omega(\mathbf{X})$.

Consider a mapping σ which assigns *appropriate values* to the program variables from $\mathbf{X}$ as well as to the (non-ground or ground) template variables A, B, C. This means that σ assigns ground terms to the variables in $\mathbf{X} \cup \{C\}$ and templates to A, B. Such a mapping is called *variable assignment*. The variable assignment σ *satisfies* the equality $s \doteq t$ ($\sigma \models (s \doteq t)$ for short) iff $\sigma^*(s) = \sigma^*(t)$ where σ^* is the natural tree homomorphism corresponding to σ, which is the identity on all operators in Ω. The homomorphism σ^* maps, e.g., the application At of the template variable A to the term t into $\sigma(A)[\sigma^*(t)/\bullet]$, i.e., the *substitution* of the term $\sigma^*(t)$ into the occurrences of the dedicated variable $\bullet$ in the template $\sigma(A)$. Substitution into the dedicated variable $\bullet$ is an associative binary operation where the neutral element is the template consisting of $\bullet$ alone. In the following, we denote this operation by juxtaposition.

Consider, e.g., an assignment σ with $\sigma(A) = h(\bullet, \bullet)$, and $\sigma(B) = \bullet$, and $\sigma(\mathbf{x}) = a$. Then

$$\sigma^*(A\mathbf{x}) = h(\bullet, \bullet)\, a = h(a, a) = \bullet\, h(a, a) = \sigma^*(Bh(\mathbf{x}, a))$$

holds. Therefore, σ satisfies the equality $A\mathbf{x} \doteq Bh(\mathbf{x}, a)$. In the following, we will no longer distinguish between σ and σ^*.

The variable assignment σ satisfies the conjunction ϕ of equalities ($\sigma \models \phi$ for short), iff $\sigma \models e$ for all equalities $e \in \phi$.

In our application, it will be convenient not to consider arbitrary variable assignments, but only those which map program variables to *reasonable* values as shown in the following. For a subset $T \subseteq \mathcal{T}_\Omega$ of ground terms, we call a variable assignment σ a T-assignment, if σ maps program variables $\mathbf{x}$ to values $\sigma(\mathbf{x}) \in T$ only.

The conjunction ϕ then is called *T-satisfiable* if there is some T-assignment σ with $\sigma \models \phi$. Otherwise, it is *T-unsatisfiable*. Conjunctions ϕ, ϕ' are *T-equivalent* if for every T-assignment σ, $\sigma \models \phi$ iff $\sigma \models \phi'$. Obviously, an empty conjunction is satisfied by every variable assignment and therefore equal to $\top$ (true), while all T-unsatisfiable conjunctions are T-equivalent. As usual, these are denoted by $\bot$ (false). Finally, a conjunction ϕ' is *T-subsumed* by a conjunction ϕ, if ϕ is T-equivalent to $\phi \wedge \phi'$.

If the set T by which we have relativized the notions of satisfiability, equivalence and subsumption equals the full set $\mathcal{T}_\Omega$, we may also drop the prefixing with T. In particular, we have for any T that satisfiability, equivalence and subsumption imply T-satisfiability, T-equivalence and T-subsumption, while the reverse implication may not necessarily hold.

In the following, we recall the ingredients of weakest pre-condition computation for assignments as well as for procedure calls as provided, e.g. in [9] or [2]. The weakest pre-condition of ϕ w.r.t. assignments are given by:

$$[\![\mathbf{x} = t]\!]^{\mathsf{T}}\ \phi = \phi[t/\mathbf{x}]$$
$$[\![\mathbf{x} = ?]\!]^{\mathsf{T}}\ \phi = \forall\,\mathbf{x}.\ \phi$$

Thus, the weakest pre-condition for an assignment $\mathbf{x} = t$ is given by substitution of the term t into all occurrences of the variable $\mathbf{x}$ in the post-conditions, while the weakest pre-condition for a non-deterministic assignment $\mathbf{x} = ?$ of any value is given by universal quantification. For Herbrand equalities, universal quantification can be computed as follows. Recall that universal quantification commutes with conjunction. Therefore, it suffices to consider single equalities e. If $\mathbf{x}$ does not occur in e, then $\forall\,\mathbf{x}.\,e$ is equivalent to e. If $\mathbf{x}$ occurs only on one side of e, then $\forall\,\mathbf{x}.\,e = \bot$. Now assume that $\mathbf{x}$ occurs on both sides of e. If e is of the form $s\mathbf{x} \doteq t\mathbf{x}$ for templates s, t (no template variables), then either $s = t$ and hence e as well as $\forall\,\mathbf{x}.\,e$ is equivalent to $\top$, or $s \neq t$, in which case $\forall\,\mathbf{x}.\,e$ equals $\bot$. If e is of the form $As\mathbf{x} \doteq Bt\mathbf{x}$ for templates s, t, then $\forall\,\mathbf{x}.\,e$ is equivalent to $As \doteq Bt$.

Every transformation f which is specified for generic post-conditions to conjunctions of pre-conditions, can be uniquely extended to a transformation $\bar{f}$ of *arbitrary* post-conditions by

$$\bar{f}(\bigwedge E) \quad = \quad \bigwedge_{e \in E} \bar{f}(e)$$

where the transformation $\bar{f}$ for an arbitrary equality e is defined as follows:

$$\bar{f}(s \doteq t) = \begin{cases} f(A\mathbf{x} \doteq B\mathbf{y})[s'/A, t'/B] & \text{if } s = s'\mathbf{x}, t = t'\mathbf{y} \\ f(A\mathbf{x} \doteq C)[s'/A, t/C] & \text{if } s = s'\mathbf{x}, t \text{ ground} \\ f(A\mathbf{x} \doteq C)[s/C, t'/A] & \text{if } t = t'\mathbf{x}, s \text{ ground} \\ s \doteq t & \text{otherwise} \end{cases}$$

Subsequently, the extended function $\bar{f}$ is denoted by f as well. The procedure summaries are then characterized by the constraint system **S**:

$$\begin{array}{lll} [\![r_p]\!]^\top & \Longrightarrow \mathsf{Id} & \text{for each procedure } p \\ [\![u]\!]^\top & \Longrightarrow [\![s_p]\!]^\top \circ [\![v]\!]^\top & \text{for each } (u, p(), v) \in E \\ [\![u]\!]^\top & \Longrightarrow [\![s]\!]^\top \circ [\![v]\!]^\top & \text{for each } (u, s, v) \in E, \\ & & s \text{ assignment} \end{array}$$

where $\circ$ means the composition of the weakest pre-condition transformers and Id is the identity transformer. Thus, accumulation of weakest pre-conditions for a generic post-condition e at procedure exit r_p with e and then propagates its pre-conditions backward to the start point of p by applying the transformations corresponding to the traversed edges. Here, the subsumption relation $\Longrightarrow$ as defined for conjunction of equalities, has silently been raised to the function level. Thus, $f \Longrightarrow g$ if $f(e)$ subsumes $g(e)$ for all generic post-conditions e.

W.r.t. the ordering $\sqsubseteq$ given by $\Longrightarrow$, the **WP** transformer of procedure p then is obtained as the value for the variable corresponding to the start point s_p in the *greatest* solution to the constraint system **S**.

The **WP** transformers for all program points are characterized by the greatest solution of the constraint system **R**:

$$\begin{array}{lll} [s_{main}]^\top & \Longrightarrow \mathsf{Id} & \\ [s_p]^\top & \Longrightarrow [u]^\top & \text{for each } (u, p(), \_) \in E \\ [v]^\top & \Longrightarrow [u]^\top \circ [\![s_p]\!]^\top & \text{for each } (u, p(), v) \in E \\ [v]^\top & \Longrightarrow [u]^\top \circ [\![s]\!]^\top & \text{for each } (u, s, v) \in E, \\ & & s \text{ assignment} \end{array}$$

The value for $[v]^\top$ for program point v is meant to transform every assertion at program point v, into the corresponding weakest pre-condition at the start point of the program. Note that the constraint system for characterizing these functions makes use of the weakest pre-condition transformers of procedures as characterized by the constraint system **S**.

Assume that we are somehow given the greatest solution of the constraint system **R** where $[v]^\top$ is the corresponding transformation for program point v. In order to determine all one- or two-variable equalities which are valid when reaching the program point v, we conceptually proceed as follows:

One-variable Equality. For a program variable $\mathbf{x}$, let ψ denote the *universal closure* of $[v]^\top(A\mathbf{x} \doteq C)$. If $\psi = \bot$, then program variable $\mathbf{x}$ does not receive a constant value at program point v. Otherwise ψ is equivalent to an equality $As \doteq C$ where s is ground, i.e., $\mathbf{x} \doteq s$ is an invariant at v.

Two-variable Equality. For distinct program variables $\mathbf{x}$ and $\mathbf{y}$, let ψ denote the universal closure of $[v]^\top(A\mathbf{x} \doteq B\mathbf{y})$. If $\psi = \bot$, then no equality between $\mathbf{x}$ and $\mathbf{y}$ holds. Otherwise, ψ equals a conjunction of equalities $As_i \doteq Bt_i$, $i \in I$, for some index set I where for each $i \in I$, s_i, t_i are both ground. Then $r_1\mathbf{x} \doteq r_2\mathbf{y}$ is an invariant at v iff $r_1 s_i \doteq r_2 t_i$ for all i, i.e., any assignment σ with $\sigma(A) = r_1, \sigma(B) = r_2$ satisfies the conjunction.

Here, the *universal closure* of a conjunction ϕ is given by $\forall\,\mathbf{x}_1 \ldots \forall\,\mathbf{x}_n.\phi$, if the set of program variables equals $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.

Example 2. Consider the main procedure of the program in Section 1, as defined by the control flow graph in Figure 1. The **WP** transformer $[3]^\top$ for the endpoint 3 of the main program is given by:

$$[3]^\top = [\![\mathbf{x} = a]\!]^\top \circ [\![\mathbf{y} = a]\!]^\top \circ [\![4]\!]^\top$$

where 4 is the entry point of the procedure p. Assume that

$$[\![4]\!]^\top(A\mathbf{x} \doteq B\mathbf{y}) = (A\mathbf{x} \doteq B\mathbf{y}) \wedge (Af(\mathbf{x}, \mathbf{x}) \doteq Bf(\mathbf{y}, \mathbf{y}))$$

holds. For the program variables $\mathbf{x}, \mathbf{y}$, we therefore obtain:

$$\begin{aligned}[3]^\top(A\mathbf{x} \doteq B\mathbf{y}) &= (A\mathbf{x} \doteq B\mathbf{y})[a/\mathbf{y}][a/\mathbf{x}] \wedge (Af(\mathbf{x}, \mathbf{x}) \doteq Bf(\mathbf{y}, \mathbf{y}))[a/\mathbf{y}][a/\mathbf{x}] \\ &= (Aa \doteq Ba) \wedge (Af(a, a) \doteq Bf(a, a))\end{aligned}$$

This assertion does not contain occurrences of the program variables $\mathbf{x}, \mathbf{y}$. Therefore, it is preserved by universal quantification over program variables. Since $A = B = \bullet$ is a solution, $\mathbf{x} \doteq \mathbf{y}$ holds whenever program point 3 is reached. $\square$

In order to turn these definitions into an effective analysis algorithm, several obstacles must be overcome. So, it is not clear how general subsumption, as required in our characterization of the **WP** transformers, can be decided in presence of template variables. We observe, however, that instead of general subsumption, it suffices to rely on T-subsumption only — for a well-chosen subset $T \subseteq \mathcal{T}_\Omega$. Note that the smaller the set T is, the coarser is the subsumption relation. In particular for $T = \emptyset$, all conjunctions are T-equivalent. Since every assertion expresses a property of reaching program states, it suffices for our application to choose T as a superset of all run-time values of program variables.

The following wish list collects properties which enable us to construct an effective inter-procedural analysis of all two-variable Herbrand equalities:

T-Compactness. Every occurring conjunction ϕ is T-subsumed by a conjunction of a *finite* subset of equalities in ϕ.

Effectiveness of subsumption. T-subsumption for *finite* conjunctions can be effectively decided.

Solvability of ground equalities. The set of solutions of finite systems of equalities with template variables only, i.e., *without* occurrences of program variables can be explicitly computed.

By the first assumption, a standard fixpoint iteration for the constraint systems **S** and **R** will terminate after finitely many iterations (up to T-equivalence). By the second assumption, termination can effectively be detected, while the third assumption guarantees that for every program point and every program variable (pair of program variables) the set of all valid invariants can be extracted out of the greatest solution of **R**. In total, we arrive at an effective algorithm for inferring all valid two-variable equalities.

The assumption on decidability of T-subsumption can be further relaxed. Instead, we provide an *approximate* notion of T-subsumption which is decidable. Our approximate T-subsumption implies T-subsumption. Moreover, it is still strong enough to guarantee that every occurring conjunction of equalities is approximately T-subsumed by a finite subset of the equalities. Notions for approximate T-subsumption will be introduced in Sections 4 and 5.

In the upcoming section, we recall basic properties of the set of terms, possibly containing the variable $\bullet$. These properties will allow us to deal with conjunctions of equalities where template variables are applied to ground terms only, i.e., the case of ground equalities.

3 Factorization of Terms

Let $\mathcal{T}_\Omega(\bullet)$ denote the set of terms constructed from the symbols in Ω, possibly together with the dedicated variable $\bullet$. In [3], Engelfriet presents the following cancellation and factorization properties for terms in $\mathcal{T}_\Omega(\bullet)$:

Bottom Cancellation:
Assume that $t_1 \neq t_1'$. Then $s_1 t_1 = s_2 t_1$ and $s_1 t_1' = s_2 t_1'$ implies $s_1 = s_2$.

Top Cancellation:
Assume $\bullet$ occurs in s. Then $st_1 = st_2$ implies $t_1 = t_2$.

Factorization:
Assume $t_i \neq t_i'$ for $i = 1, 2$. Then $s_1 t_1 = s_2 t_2$ and $s_1 t_1' = s_2 t_2'$ implies that $s_1 r_1 = s_2 r_2$ for some r_1, r_2 each containing $\bullet$ where at least one of the r_i equals $\bullet$. In that case (by top cancellation), we furthermore have that both $r_2 t_1 = r_1 t_2$ and $r_2 t_1' = r_1 t_2'$.

Using these cancellation properties, we obtain a complete method for dealing with equalities *without* occurrences of program variables.

For one-variable equalities alone, we have the following results concerning subsumption and compactness:

Theorem 1.

1. *A single equality $As \doteq C$ for some ground term s has exactly one solution where $A = \bullet$.*
2. *Consider the conjunction $As_1 \doteq C \wedge As_2 \doteq C$ for terms $s_1 \neq s_2$ containing the same variable $\mathbf{x}$. If the conjunction is satisfiable, then the value of $\mathbf{x}$ is uniquely determined.*

Proof. We only prove the second assertion. The conjunction $As_1 \doteq C \wedge As_2 \doteq C$ is equivalent to the conjunction $As_1 \doteq C \wedge s_1 \doteq s_2$. The most general unifier of s_1, s_2 maps $\mathbf{x}$ to a ground subterm of s_1, s_2 if the conjunction is satisfiable. □

As a consequence, we obtain:

Corollary 1. *Consider finite conjunctions of equalities of the form $As \doteq C$.*

1. *Subsumption for these is decidable.*
2. *Every satisfiable conjunction is equivalent to a conjunction of at most $n + 1$ equalities where n is the number of program variables.*

Since the weakest pre-condition of a generic one-variable equality consists of equalities of the form $As \doteq C$ only, Corollary 1 suffices to infer all inter-procedurally valid one-variable equalities. In the following, we therefore concentrate on the two-variable case where the weakest pre-condition consists of conjunctions of equalities of the form $As \doteq Bt$. First, we observe:

Theorem 2.

1. *A single equality $As \doteq Bt$ for ground terms s, t has only finitely many solutions $A = r_1, B = r_2$ with templates r_1, r_2 of which at least one equals $\bullet$.*
2. *Consider the conjunction $As_1 \doteq Bt_1 \wedge As_2 \doteq Bt_2$ for ground terms $s_1 \neq s_2$ and $t_1 \neq t_2$. Then it has either no solution or there are templates r_1, r_2 of which at least one equals $\bullet$ such that the conjunction is equivalent to $Ar_1 \doteq Br_2$. In the latter case, $A = r_2$, $B = r_1$ is the single solution where at least one of the templates equals $\bullet$.*
3. *Consider the (finite) conjunction $\bigwedge_{i=1}^{k}(As_i \doteq Bt_i)$ for ground terms s_i, t_i. Then the set of all solutions where either the template for A or for B equals $\bullet$, can be effectively computed.*

Proof. For a proof of the first statement, w.l.o.g. assume that s is at least as large as t. Then for size reasons, $r_1 = \bullet$. This means that $s = r_2 t$ must hold. If t is not a subterm of s, there is no solution at all. Otherwise, i.e., if s contains occurrences of t, then every solution r_2 is obtained from s by replacing a non-empty set of occurrences of t with $\bullet$.

Now consider the second statement. If the pair of equalities is satisfiable then by factorization, there are templates r_1, r_2 of which at least one equals $\bullet$ such that $Ar_1 \doteq Br_2$ holds. Since at the same time $r_2 s_i \doteq r_1 t_i$ holds, the equality $Ar_1 \doteq Br_2$ is equivalent to the conjunction. Moreover, there is exactly one solution $A = r_1', B = r_2'$ where at least one of the templates r_i' equals $\bullet$, namely, $r_1' = r_2$, $r_2' = r_1$.

Finally, consider the third statement. If $k = 1$, the assertion follows from statement 1. Therefore now let $k > 1$. First assume that for some i, j, $s_i \neq s_j$ and $t_i \neq t_j$. Then by statement 2, the conjunction is unsatisfiable or there is exactly one pair r_1, r_2 of templates one of which equals $\bullet$, such that $A = r_1, B = r_2$ is a solution of the conjunction $As_i \doteq Bt_i \wedge As_j \doteq Bt_j$. If in the latter case, $r_1 s_l \doteq r_2 t_l$ for all l, we have obtained a single solution. Otherwise, the conjunction is unsatisfiable. Now assume that no such i, j exists. Then either the conjunction is unsatisfiable or all equalities are syntactically equal. □

Example 3. Consider the two equalities:

$$Af(a,gb,gb) \doteq Bgb \qquad Af(a,gc,gb) \doteq Bgc$$

Then $A = \bullet$ and $B = f(a, \bullet, gb)$ is the only solution for A, B where at least one of the templates equals $\bullet$. □

Applying the arguments which we used to prove Theorem 2, we obtain:

Corollary 2. *Consider a conjunction* $\bigwedge_{i=1}^{n} As_i \doteq Bt_i$ *with ground terms* s_i, t_i.

1. *If it is satisfiable, it is equivalent to the conjunction of at most two conjuncts.*
2. *If it is unsatisfiable, there are at most three conjuncts whose conjunction is unsatisfiable.*

By Theorem 2, the assumption **solvability of ground equalities** from Section 2 is met. Thus, it remains to solve the constraint systems **S** and **R**, i.e., to construct an approximate T-subsumption relation which is both effective and guarantees that every conjunction is approximately T-subsumed by the conjunction of a finite subset of equalities. In order to construct such a relation, we require stronger insights into the structure of templates and their compositions. Let $\mathcal{C}_\Omega$ denote the subset of all terms in $\mathcal{T}_\Omega(\bullet)$ which contain at least one occurrence of $\bullet$, i.e., $\mathcal{C}_\Omega = \mathcal{T}_\Omega(\bullet) \setminus \mathcal{T}_\Omega$. The terms in $\mathcal{C}_\Omega$ have also been called *templates*. The set $\mathcal{C}_\Omega$, equipped with substitution, is a *free monoid* with neutral element $\bullet$. This monoid is *infinitely* generated from the irreducible elements in $\mathcal{C}_\Omega$. As usual, we call an element t *irreducible* if t cannot be non-trivially decomposed into a product, i.e., $t = uv$ implies that $t = u$ with $v = \bullet$ or $t = v$ with $u = \bullet$.

While templates can be uniquely factored, this is no longer the case for ground terms, i.e., terms without variable occurrences.

Example 4. Consider the ground term $t = h(f(h(1), h(1)))$, together with the templates $s_1 = h(f(\bullet, h(1)))$, $s_2 = h(f(h(1), \bullet))$ and $s_3 = h(f(\bullet, \bullet))$. All these three templates are distinct. Still,

$$t = s_1\ h(\bullet)\ 1 = s_2\ h(\bullet)\ 1 = s_3\ h(\bullet)\ 1$$ □

Thus, unique factorization of arbitrary ground terms cannot be hoped for. Still, we observe that unique factorization can be obtained — at least up to any fixed finite set of ground terms. Let G denote a finite set of ground terms which is closed by subterms.

Let M_G denote the sub-monoid of all templates $m \in \mathcal{C}_\Omega$ whose ground subterms all are contained in G. Then we have:

Theorem 3. *Assume that* $S \subseteq \mathcal{T}_\Omega$ *which is closed by subterms. If* $G \subseteq S$, *then every ground term* $t \in \mathcal{T}_\Omega \setminus S$, *can be uniquely factored into* $t = mx$ *such that*

(A) $m \in M_G$ *and* $x \notin S$;
(B) x *is minimal with property (A).*

Example 5. Consider the term

$$t = f(h(f(2, h(1))), h(f(2, h(1))))$$

and assume that the set G of forbidden ground subterms is given by $G = \{h(1), 1\}$ and $S = G$. Then t can be decomposed into:

$$f(\bullet, \bullet)\ h(\bullet)\ f(\bullet, h(1))\ 2$$

If on the other hand, $S = G = \{2\}$, we obtain the decomposition:

$$f(\bullet, \bullet)\ h(\bullet)\ f(2, \bullet)\ h(\bullet)\ 1$$

If finally, S and G are empty, the term x of Theorem 3 is the minimal subterm such that the occurrences of x contains all ground leaves of t. This means that $x = f(2, h(1))$, and we obtain the decomposition:

$$f(\bullet, \bullet)\ h(\bullet)\ f(2, h(1))$$ □

The unique decomposition of the ground term t claimed by Theorem 3, is constructed as follows. Let X denote the set of minimal subterms x' of t such that $x' \notin G$. Then we construct the least subterm $x \notin S$ of t such that all occurrences of subterms $x' \in X$ in t are contained in some occurrence of x. This subterm is uniquely determined. Then define m as the term obtained from t by replacing all occurrences of x with $\bullet$. This term m is also uniquely determined with $t = mx$. Moreover by construction, all ground subterms of m are contained in G.

Example 6. Consider the program from example 1. In this program, no non-ground right-hand side contains ground subterms. Accordingly, the set G is empty. Since the only ground right-hand side equals the atom a, the decomposition Theorem 3 allows to uniquely decompose all run-time values of this program into right-hand sides of assignments. □

Theorem 3 allows to extend the monoidal techniques of Gulwani et al. [8] for unary operators to programs where all run-time values can be uniquely factorized into right-hand sides. This extension is given in Section 4. For completeness reasons, we also present simplified versions of the algorithms for monoidal equalities from [8] in Appendix A. The general case where unique factorization of all run-time values can no longer be guaranteed, subsequently is presented in Section 5.

4 Initialization-restricted Programs

Assume that R is the set of ground right-hand sides of assignments, and G is the set of ground subterms of non-ground right-hand sides of assignments of our program. Then generally, each value x possibly constructed at run-time by the program is of the form $x = x'r$ where $r \in R$ and $x' \in M_G$. This means that for pre-conditions ϕ possibly occurring in a **WP** calculation for a program invariant, we are only interested in variable assignments σ which map each program variable $\mathbf{x}$ to a possible run-time value for $\mathbf{x}$, i.e., to a value from the set $M_G R$. Henceforth, we therefore no longer consider general satisfiability, equivalence and subsumption, but only T-satisfiability, T-equivalence and T-subsumption for $T = M_G R$. This restriction is crucial for the generalization of the monoidal techniques from [8]. In the following, we first consider the sub-class of programs p where set R of ground right-hand sides of p satisfies the two properties:

1. $R \cap G = \emptyset$.
2. The elements in R are mutually incomparable ground terms, i.e., for $r_1, r_2 \in R$, r_1 is a subterm of r_2 iff $r_1 = r_2$.

The program p then is called *initialization-restricted* (or IR for short).

Example 7. Assume that the non-ground right-hand sides of assignments of the program are $f(\mathbf{x}, h(1))$ and $f(2, h(\mathbf{y}))$. Then the set G is given by $G = \{1, h(1), 2\}$. A suitable set R of ground right-hand sides might be, e.g., $R = \{0, a\}$. □

Our condition here is not as restrictive as it might seem. Programs where each variable is initialized by a non-deterministic assignment, are all IR. The same holds true for programs where all non-ground right-hand sides of assignments do not contain ground terms, and variables are initialized with atoms only. The latter property is met by our example 1. By suitably massaging variable initializations, it also comprises all programs using monadic operators only (as in [8]).

We distinguish between two-variable equalities of the following formats:

$$
\begin{array}{lll}
[F_{\mathbf{x},\mathbf{y}}] & As\mathbf{x} \doteq Bt\mathbf{y} & \text{where } s, t \in M_G \\
[F_{\cdot,\mathbf{x}}] & As \doteq Bt\mathbf{x} & \text{where } s \in T \text{ and } t \in M_G \\
[F_{\mathbf{x},\cdot}] & At\mathbf{x} \doteq Bs & \text{where } s \in T \text{ and } t \in M_G
\end{array}
$$

For each format separately, we observe:

Theorem 4.

T-subsumption. *For finite sets E, E' of two-variable equalities of the same format it is decidable whether $\bigwedge E$ T-subsumes $\bigwedge E'$ or not.*

T-compactness. *Every T-satisfiable conjunction of a set E of two-variable equalities of the same format is T-subsumed by a conjunction of a subset of at most three equalities in E.*

For a proof see Appendix B. It relies on the unique factorization property together with the monoidal techniques from Section A. Since T-subsumption is decidable, at least for equalities of the same format, we define an approximate T-subsumption relation $\bigwedge E \implies^\sharp \bigwedge E'$ for conjunctions of equalities as follows. Let E_F and E'_F denote the subsets of equalities of the same format F in E and E', respectively. Then $\bigwedge E \implies^\sharp \bigwedge E'$ holds iff $\bigwedge E_F$ T-subsumes $\bigwedge E'_F$ for all formats F. Hence, by Theorem 4, we obtain:

Corollary 3. *Assume that n is the number of program variables.*

Approximate T-subsumption. *For finite sets E, E' of two-variable equalities, it is decidable whether $\bigwedge E$ approximately T-subsumes $\bigwedge E'$ or not.*

Approximate T-compactness. *Every T-satisfiable conjunction of a set E of two-variable equalities is approximately T-subsumed by a conjunction of a subset of at most $\mathcal{O}(n^2)$ equalities in E.*

Overall, we therefore conclude for IR programs:

Theorem 5. *Assume that p is an IR program. Then for every program point u, the set of all two-variable equalities can be determined that are valid when reaching program point u.*

Proof. By Corollary 3, the greatest solutions of the constraint systems **S** and **R** can be effectively computed. Let $[u]^\top$, u program point, denote the greatest solution of the system **R**. Then the set of valid equalities $s\mathbf{x} \doteq t\mathbf{y}$ between program variables $\mathbf{x}$, $\mathbf{y}$ is given by the set of solutions to a system of ground equalities which are obtained by universal quantification over all program variables of the conjunction of equalities $[u]^\top(A\mathbf{x} \doteq B\mathbf{y})$. By Theorem 2, a representation of the set of solutions for the template variables A, B in this conjunction can be explicitly computed. Likewise, the set of valid equalities $x \doteq t$ for program variable $\mathbf{x}$ and ground term t can be extracted from the universal quantification over all program variables of the conjunction of equalities $[u]^\top(A\mathbf{x} \doteq C)$. The resulting conjunction may either equal $\bot$ (no constant value for $\mathbf{x}$) or contain only the variable C. Consequently, the possible constant value for $\mathbf{x}$ and program point u can also be effectively computed. This completes the proof. ◻

Example 8. According to our constructions in Section 2 and Theorem 2, the set of all inter-procedurally valid assertions can be obtained from the greatest solutions to the constraint systems **S** and **R**. Consider, e.g., the constraint system **R** for the recursive procedure p from Section 1, as defined by the control flow graph of Figure 1. If Round-Robin iteration is applied to calculate the transformers $[\![u]\!]^\top$ for the program points $u = 4, 5, 6, 7$, we obtain for the generic post-condition $A\mathbf{x} \doteq B\mathbf{y}$ the result depicted by Table 1 where in the ith column, we have

Table 1. Round-Robin iteration for the procedure p from Figure 1

	1	2	3
7	$A\mathbf{x} \doteq B\mathbf{y}$		
6	$A\mathbf{x} \doteq Bf(\bullet,\bullet)\mathbf{y}$		
5	$\top$	$A\mathbf{x} \doteq Bf(\bullet,\bullet)\mathbf{y}$	$Af(\bullet,\bullet)\mathbf{x} \doteq Bf(\bullet,\bullet)f(\bullet,\bullet)\mathbf{y}$
4	$A\mathbf{x} \doteq B\mathbf{y}$	$Af(\bullet,\bullet)\mathbf{x} \doteq Bf(\bullet,\bullet)\mathbf{y}$	$Af(\bullet,\bullet)f(\bullet,\bullet)\mathbf{x} \doteq Bf(\bullet,\bullet)f(\bullet,\bullet)\mathbf{y}$

only displayed pre-conditions which have additionally been attained in the ith iteration for the program points $7, 6, 5$ and 4, respectively. For convenience, we have displayed the terms in equalities according to their unique factorizations.

For program point 4, the two equalities after the second iteration, imply:

$$Af(\bullet,\bullet)A^- \doteq Bf(\bullet,\bullet)B^-$$

The second equality for program point 4 together with this identity imply that

$$Af(\bullet,\bullet)A^- Af(\bullet,\bullet)\mathbf{x} \doteq Bf(\bullet,\bullet)B^- Bf(\bullet,\bullet)\mathbf{y}$$

from which the third equality for program point 4 as provided by the third iteration follows. Thus, Round-Robin fixpoint iteration reaches the greatest fixpoint after the third iteration. □

5 Unrestricted Programs

Our analysis of IR programs relied on the fact that all run-time values of program variables can be uniquely factorized. This was made possible since in IR programs the "bottom end" of values can be uniquely identified by means of the ground right-hand sides from R. In general, though, ground right-hand sides could very well also occur as subterms of other right-hand sides in the program. In this case, we can no longer assume that R serves as such a handy set of end marker terms. At first sight, therefore, the monoidal method seems no longer applicable. A second look, however, reveals that the monoidal method essentially fails only, where program variables take *small* values. Again, let R and G denote the set of all ground right-hand sides and the set of all ground subterms of non-ground right-hand sides of assignments in the program, respectively. We call a term $t \in M_G R$ *small* if it is a ground subterm of a right-hand side of an assignment. Let us denote the (finite) set of all small terms by S. The terms in $M_G R$ which are not small, are called *large*. Let $\bar{R}$ be the set of *minimal* elements in $M_G R$ which are large, i.e., not contained in S. Then by Theorem 3, every large term t, i.e., every term $t \in L$ can be uniquely factored such that $t = mr$ where $m \in M_G$ and $r \in \bar{R}$. For small terms, i.e., for terms in S, on the other hand, we cannot hope for unique factorizations. Since there are finitely many small terms only, we take care of small terms by two means:

- We restrict the formats $[F_{\mathbf{x},\cdot}]$ and $[F_{\cdot,\mathbf{x}}]$ from the last section to the case where the occurring ground terms are large and introduce dedicated sub-formats $[F_{\mathbf{x},s}]$ and $[F_{s,\mathbf{x}}]$ for each small term s in the equalities.
- For T-subsumption, we single out the case of subsumption w.r.t. assignments of large terms only and treat subsumption w.r.t. assignments assigning small terms separately.

Thus, we now consider the following formats of two-variable equalities:

$[F_{\mathbf{x},\mathbf{y}}]$	$As\mathbf{x} \doteq Bt\mathbf{y}$	where $s, t \in M_G$
$[F_{\cdot,\mathbf{x}}]$	$As \doteq Bt\mathbf{x}$	where $s \in L$ and $t \in M_G$
$[F_{s,\mathbf{x}}]$	$As \doteq Bt\mathbf{x}$	where $s \in S$ and $t \in M_G$
$[F_{\mathbf{x},\cdot}]$	$At\mathbf{x} \doteq Bs$	where $s \in L$ and $t \in M_G$
$[F_{\mathbf{x},s}]$	$At\mathbf{x} \doteq Bs$	where $s \in S$ and $t \in M_G$

In the following, let us call a substitution σ of program variables *small*, if for every program variable $\mathbf{x}$, $\sigma(\mathbf{x})$ either equals $\mathbf{x}$ or is a small ground term. The notions of satisfiability, equivalence and subsumption restricted to the set T can be inferred by means of the corresponding notions restricted to the set L of large terms only. We have:

- A conjunction ϕ of equalities is T-satisfiable iff there is a small substitution σ such that $\sigma(\phi)$ is L-satisfiable.
- A conjunction ϕ T-subsumes an equality e, iff for every small substitution σ, $\sigma(\phi)$ L-subsumes $\sigma(e)$.

According to this observation, it seems plausible to consider the analogue of Theorem 4 for L-subsumption and L-compactness only. We obtain:

Theorem 6.

L-subsumption. *For finite sets E, E' of two-variable equalities of the same format it is decidable whether $\bigwedge E$ L-subsumes $\bigwedge E'$ or not.*

L-compactness. *Every L-satisfiable conjunction of a set E of two-variable equalities of the same format is L-subsumed by a conjunction of a subset of at most three equalities in E.*

Proof. For equalities of the formats $[F_{\mathbf{x},\mathbf{y}}], [F_{\mathbf{x},\cdot}], [F_{\cdot,\mathbf{x}}]$ the proofs are analogous to the corresponding proofs for Theorem 4 where the set T is replaced with the set $L = M_G\bar{R}$, i.e., instead of the set R we rely on the set $\bar{R}$. Therefore now consider equalities of the format $[F_{s,\mathbf{x}}]$ for a small term $s \in S$.

W.l.o.g., let $As \doteq Bt\mathbf{x}$ and $As \doteq Bt'\mathbf{x}$ be two equalities of this format. If $t \neq t'$, then their conjunction is either contradictory, or $t\mathbf{x}, t'\mathbf{x}$ have a ground unifier which maps $\mathbf{x}$ to a value from G — in contradiction to the assumption that $\mathbf{x}$ takes values from L only.

Therefore, each conjunction of a set E of equalities of the format $[F_{s,\mathbf{x}}]$ either is L-equivalent to $\bot$ or to a single equality in E, and the assertion of the theorem follows. The same argument also applies for the format $[F_{\mathbf{x},s}]$. $\square$

Given that L-subsumption is decidable, at least for equalities of the same format, and that also L-compactness holds, we define an approximate T-subsumption relation $\bigwedge E \implies^\sharp \bigwedge E'$ as follows. Let E_F and E'_F denote the subsets of equalities of format F, in E and E', respectively. Then $\bigwedge E \implies^\sharp \bigwedge E'$ holds iff for all small substitutions σ, $\bigwedge \sigma(E_F)$ L-subsumes $\bigwedge \sigma(E'_F)$ for all formats F. As a consequence of Theorem 6, we obtain:

Theorem 7. *Assume that n is the number of program variables and m is the cardinality of the set S of small terms.*

Approximate T-subsumption. *For finite sets E, E' of two-variable equalities, it is decidable whether $\bigwedge E$ approximately T-subsumes $\bigwedge E'$ or not.*

Approximate T-compactness. *Every T-satisfiable conjunction of a set E of two-variable equalities is approximately T-subsumed by a conjunction of a subset of at most $\mathcal{O}(n^2 \cdot m^2)$ equalities in E.*

A proof is provided in the long version of this paper [19]. Due to Theorem 7, representations of the greatest solutions of the constraint systems **S** and **R** can be effectively computed. By that, we arrive at our main result:

Theorem 8. *Assume that all right-hand sides of assignments in an arbitrary program contain at most one variable. Then all valid inter-procedurally two-variable Herbrand equalities can be inferred.*

The proof is analogous to the proof of Theorem 5 — only that Theorem 7 is used instead of Corollary 3.

Example 9. Consider a variant of the program from Section 1 where the non-ground assignments are given by:

$$\mathbf{x} = f(\mathbf{x}, a, \mathbf{x}) \qquad \text{and} \qquad \mathbf{y} = f(\mathbf{y}, a, \mathbf{y})$$

The set of small terms then is given by $S = \{a\}$, while the set of smallest large terms is given by $\bar{R} = \{f(a, a, a)\}$.

Now consider the constraint system **R** for the recursive procedure p as defined by the control flow graph of Figure 1 with the modified assignments. Let us concentrate on the start point 4 of p. Round-Robin iteration for the transformer $[\![4]\!]^{\top}$ for the generic post-condition $A\mathbf{x} \doteq B\mathbf{y}$, successively will produce the equalities depicted by Table 2, where in the ith column, we again only have

Table 2. Round-Robin iteration of Example 9

	1	2	3
7	$A\mathbf{x} \doteq B\mathbf{y}$		
6	$A\mathbf{x} \doteq Bf(\mathbf{y}, a, \mathbf{y})$		
5	$\top$	$A\mathbf{x} \doteq Bf(\mathbf{y}, a, \mathbf{y})$	$Af(\mathbf{x}, a, \mathbf{x}) \doteq Bf(f(\mathbf{y}, a, \mathbf{y}), a, f(\mathbf{y}, a, \mathbf{y}))$
4	$A\mathbf{x} \doteq B\mathbf{y}$	$Af(\mathbf{x}, a, \mathbf{x}) \doteq Bf(\mathbf{y}, a, \mathbf{y})$	$Af(f(\mathbf{x}, a, \mathbf{x}), a, f(\mathbf{x}, a, \mathbf{x})) \doteq Bf(f(\mathbf{y}, a, \mathbf{y}), a, f(\mathbf{y}, a, \mathbf{y}))$

displayed pre-conditions which have additionally been attained in the ith iteration for the program points $7, 6, 5$ and 4, respectively. For program point 4, we can argue as in Example 8 in order to verify that the first two equalities L-subsume the third one. Therefore, it remains to consider the given iteration for any small assignment to the program variables $\mathbf{x}, \mathbf{y}$.

If $\mathbf{x} = \mathbf{y} = a$, then $A = B$ must hold and the third equality is implied. If $\mathbf{x} = a$, but $\mathbf{y}$ is bound to large terms, then the first equality is of the format $[F_{a,\mathbf{y}}]$ while the subsequent equalities are of the format $[F_{\cdot,\mathbf{y}}]$. Accordingly, the first equality must be kept separately. For the second and third equalities the techniques from Theorem 6 again allow to derive the monoidal equality:

$$Af(\bullet, a, \bullet)A^{-} \doteq Bf(\bullet, a, \bullet)B^{-}$$

implying that the equality provided in the fourth iteration will be subsumed. A similar argument applies to the case where $\mathbf{y} = a$ while $\mathbf{x}$ is bound to large values only. Thus, Round-Robin fixpoint iteration reaches the greatest fixpoint after the fourth iteration. □

6 Multi-variable Equalities

In this section, we extend our methods to arbitrary equalities such as

$$\mathbf{x} \doteq f(g\mathbf{y}, \mathbf{z})$$

where, w.l.o.g., the left-hand side is a plain program variable while the right-hand side is a term possibly containing occurrences of more than one variable. Still, we consider programs where each right-hand side contains occurrences of at most one variable only. Here, we indicate how for any program point v and any given candidate Herbrand equality $\mathbf{x} \doteq s$, we verify whether or not the equality is valid whenever v is reached. There are only constantly many candidate equalities of this form, namely, all equalities which hold for a variable assignment σ_v computed by a single run of the program reaching v. Since such a single run can be effectively computed before-hand, we conclude:

Theorem 9. *Assume that all right-hand sides of assignments in an arbitrary program contain at most one variable. Then all inter-procedurally valid Herbrand equalities can be inferred.*

Now consider the single Herbrand equality $\mathbf{x} \doteq s$, where s contains occurrences of the program variables $\mathbf{y}_1, \ldots, \mathbf{y}_r$. Then we construct new generic post-conditions as follows. First, we consider all substitutions σ which map each variable $\mathbf{y}_i$ in s either to a fresh template variable C_i or an expression $A_i\mathbf{y}'_i$ for a fresh template variable A_i and any program variable $\mathbf{y}'_i$. Then the new generic post-conditions are of the form $\mathbf{x}' \doteq s'$ where $\mathbf{x}'$ is any program variable, and s' is a subterm of $s\sigma$. Note that this set may be large but is still finite. In a practical implementation, we may, however, tabulate for each procedure the weakest pre-conditions only for those post-conditions which are really required. Since we envision that for realistic programs, only few of these equalities for each procedure will be necessary to prove the queried assertion e_t at target point u_t, the potential exponential blow-up will still be not an obstacle.

Example 10. Assume the equality we are interested in is $\mathbf{x} \doteq f(g\mathbf{y}, \mathbf{z})$, then, e.g.,

$$\mathbf{x} \doteq f(gA_1\mathbf{y}, A_2\mathbf{z}) \qquad \mathbf{y} \doteq f(gA_1\mathbf{x}, A_2\mathbf{z})$$

are new generic post-conditions to be considered, as well as

$$\mathbf{z} \doteq f(gC, A\mathbf{y}) \qquad \mathbf{y} \doteq f(gA\mathbf{z}, C)$$ □

Starting from a new generic post-condition $\mathbf{x} \doteq p$, repeatedly computing weakest pre-conditions w.r.t. assignments may result in conjunctions of equalities which can be simplified to one of the following forms:

- $s \doteq C_i$ or $s \doteq A_i t_i$ where s and t_i contain occurrences of at most one program variable each;

– $\mathbf{y} \doteq p'$, i.e., the left-hand side is a plain program variable, and the right-hand side p' is obtained from a subterm of p by substituting each occurrence of a program variable $\mathbf{y}_i$ with some term t_i containing occurrences of at most one program variable each.

Example 11. Consider, e.g., the generic post-condition $\mathbf{x} \doteq f(gA_1\mathbf{y}, A_2\mathbf{z})$. Then

$$\begin{array}{lll} [\![\mathbf{x} = f(\mathbf{x}, h\mathbf{x})]\!]^\top(\mathbf{x} \doteq f(gA_1\mathbf{y}, A_2\mathbf{z})) & = & f(\mathbf{x}, h\mathbf{x}) \doteq f(gA_1\mathbf{y}, A_2\mathbf{z}) \\ & = & (\mathbf{x} \doteq gA_1\mathbf{y}) \wedge (h\mathbf{x} \doteq A_2\mathbf{z}) \end{array}$$

which means that we equivalently obtain two two-variable equalities. Likewise, for an assignment to one of the program variables on the right, we have:

$$[\![\mathbf{y} = f(b, \mathbf{y})]\!]^\top(\mathbf{x} \doteq f(gA_1\mathbf{y}, A_2\mathbf{z})) \quad = \quad \mathbf{x} \doteq f(gA_1 f(b, \mathbf{y}), A_2\mathbf{z})$$

which is an equality of the form described in the second item. □

The equalities from the first item contain at most program variable on each side. They can be dealt with in the same way as we did for plain two-variable equalities. They are even somewhat simpler, in that only one template variable occurs (instead of two). The equalities of the second item, on the other hand, we may group into equalities which agree in the variable on the left as well as in the constructor applications outside the template variables A_i. Of each such group it suffices to keep exactly one equality. Any conjunction with another equality from the same group will allow us to simplify the second equality to a conjunction of equalities with at most one program variable on each side.

Example 12. Assume that we are given the conjunction of the two equalities:

$$\mathbf{x} \doteq f(gA_1\mathbf{y}, A_2\mathbf{z}) \qquad \mathbf{x} \doteq f(gA_3h\mathbf{y}, A_4g\mathbf{z})$$

This conjunction is equivalent to the first equality together with:

$$f(gA_1\mathbf{y}, A_2\mathbf{z}) \doteq f(gA_3h\mathbf{y}, A_4g\mathbf{z})$$

The latter equality, now, is equivalent to the conjunction of:

$$A_1\mathbf{y} \doteq A_3h\mathbf{y} \qquad A_2\mathbf{z} \doteq A_4g\mathbf{z}$$

which is a finite conjunction of two-variable equalities. □

Thus, in the course of **WP** computation for any of the new generic post-conditions, we obtain conjunctions which (up to finitely many exceptions) consists of two-variable equalities only, to which we can apply our methods from Section 5. In summary, we thus find that it can be effectively checked whether or not a general Herbrand equality is inter-procedurally valid at a given program point v.

7 Conclusion

We provided an analysis which infers all inter-procedurally valid Herbrand equalities for programs where all assignments are taken into account whose right-hand sides depend on at most one variable. The novel analysis is based on three main ideas. First, we restricted general satisfiability, subsumption and equivalence to satisfiability, subsumption and equivalence w.r.t. a set of values subsuming all possible run-time values of a given program. Together with our factorization theorem, this allowed us to apply the monoidal methods from [8] to effectively infer all inter-procedurally valid two-variable Herbrand equalities, at least for programs, which we called *initialization-restricted*. In the second step, we abandoned this restriction by introducing the extra distinction between *large* values (which can be uniquely factored) and *small* ones (of which there are only finitely many). Finally, we showed how general Herbrand equalities could be handled. For convenience, we presented the construction for programs with global variables only. The techniques, however, can be extended to programs with both local and global variables as provided in the long version of this paper [19]. In addition we show in the long version of this paper that an implementation of the analysis can be provided which runs in polynomial time.

References

1. J. Cocke and J. T. Schwartz. *Programming Languages and Their Compilers: Preliminary Notes.* Courant Institute of Mathematical Sciences, New York University, 1970.
2. P. Cousot. Methods and logics for proving programs. In J. van Leeuwen, editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, chapter 15, pages 843–993. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990.
3. J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 241–286. Academic Press, 1980.
4. A. Flexeder, M. Müller-Olm, M. Petter, and H. Seidl. Fast interprocedural linear two-variable equalities. *ACM Trans. Program. Lang. Syst.*, 33(6):21:1–21:33, 2011.
5. G. Godoy and A. Tiwari. Invariant checking for programs with procedure calls. In J. Palsberg and Z. Su, editors, *Static Analysis, 16th International Symposium (SAS)*, pages 326–342. Springer, LNCS 5673, 2009.
6. W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.
7. S. Gulwani and G. C. Necula. A polynomial-time algorithm for global value numbering. In R. Giacobazzi, editor, *Static Analysis, 11th International Symposium (SAS)*, pages 212–227. Springer, LNCS 3148, 2004.
8. S. Gulwani and A. Tiwari. Computing procedure summaries for interprocedural analysis. In R. Nicola, editor, *Programming Languages and Systems, 16th European Symposium on Programming (ESOP)*, pages 253–267. Springer, LNCS 4421, 2007.
9. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

10. A. Jeż. Context unification is in PSPACE. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 244–255. Springer, LNCS 8573, 2014.
11. G. A. Kildall. A unified approach to global program optimization. In *1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 194–206. ACM, 1973.
12. J. Levy and M. Veanes. On the undecidability of second-order unification. *Information and Computation*, 159(1-2):125–150, 2000.
13. M. Müller-Olm, M. Petter, and H. Seidl. Interprocedurally analyzing polynomial identities. In B. Durand and W. Thomas, editors, *23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 50–67. Springer, LNCS 3884, 2006.
14. M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. In N. D. Jones and X. Leroy, editors, *31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 330–341. ACM, January 2004.
15. M. Müller-Olm and H. Seidl. Analysis of modular arithmetic. *ACM Trans. Program. Lang. Syst.*, 29(5):29:1–29:27, 2007.
16. M. Müller-Olm and H. Seidl. Upper adjoints for fast inter-procedural variable equalities. In *Programming Languages and Systems, 17th European Symposium on Programming (ESOP)*, pages 178–192. Springer, LNCS 4960, 2008.
17. M. Müller-Olm, H. Seidl, and B. Steffen. Interprocedural herbrand equalities. In S. Sagiv, editor, *Programming Languages and Systems, 14th European Symposium on Programming (ESOP)*, pages 31–45. Springer, LNCS 3444, 2005.
18. M. Petter. *Interprocedural Polynomial Invariants*. PhD thesis, Institut für Informatik, Technische Universität München, September 2010.
19. S. Schulze Frielinghaus, M. Petter, and H. Seidl. Inter-procedural two-variable herbrand equalities. *arXiv e-prints*, 2014. http://arxiv.org/abs/1410.4416.
20. M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. In S. S. Muchnick and N. D. Jones, editors, *Program Flow Analysis: Theory and Application*, pages 189–233. Prentice-Hall, 1981.
21. B. Steffen, J. Knoop, and O. Rüthing. The value flow graph: A program representation for optimal program transformations. In *Programming Languages and Systems, 3rd European Symposium on Programming (ESOP)*, pages 389–405. Springer, LNCS 432, 1990.

A Equalities over a Free Monoid

Consider a free monoid M_Σ with set of generators Σ. As usual, the neutral element of M_Σ is denoted by ϵ. Let F_Σ be the corresponding free group. F_Σ can be considered as the free monoid generated from $\Sigma \cup \Sigma^-$ (where $\Sigma^- = \{a^- \mid a \in \Sigma\}$ is the set of formal inverses of elements in Σ with $\Sigma \cap \Sigma^- = \emptyset$) modulo exhaustive application of the cancellation rules $a \cdot a^- = a^- \cdot a = \epsilon$ for all $a \in \Sigma$. In particular, the neutral element of F_Σ is given by ϵ, and the inverse g^{-1} of an element $g = a_1 \ldots a_k, a_i \in \Sigma \cup \Sigma^-$, is given by $g^{-1} = a_k^{-1} \ldots a_1^{-1}$ where $x^{-1} = x^-$ and $(x^-)^{-1} = x$ for $x \in \Sigma$.

For every $w \in M_{\Sigma \cup \Sigma^-}$, the *balance* $|w|$ is the difference between the number of occurrences of positive and negative letters in w, respectively. Thus, $|aba^-b^-c| = 1$

and $|a^- b| = 0$. Note that the balance stays invariant under application of the cancellation rules. Also, $|uv| = |u|+|v|$ and $|u^{-1}| = -|u|$. Accordingly, the balance $|\cdot| : F_\Sigma \to \mathbb{Z}$ is a group homomorphism. Furthermore, we call w *non-negative* if $|w'| \geq 0$ for all prefixes w' of w. This property is also preserved by cancellation and concatenation but not by inverses. Instead, we have:

Lemma 1. *If both u and v are non-negative, and $|u| \geq |v|$ then also uv^{-1} is non-negative.*

Proof. Consider a prefix x of uv^{-1}. If x is a prefix of u, $|x| \geq 0$ since u is non-negative. Otherwise, $x = uv'^{-1}$ for some suffix v' of v. Then $|v'| \leq |v|$, since v is non-negative. Therefore, $|uv'^{-1}| = |u| - |v'| \geq |u| - |v| \geq 0$. □

We consider equations of the form:

$$AuA^{-1} = Bu'B^{-1} \tag{1}$$

where A, B are variables which take values in M_Σ and u, u' are maximally canceled. If the equation is satisfiable, then necessarily $|u| = |u'|$ holds. Assume from now on that u, u' are maximally canceled, and $|u| = |u'|$. Furthermore, we assume that $|u| \geq 0$ and u, u' are both non-negative. We then have:

Lemma 2. *If $|u| = |u'| = 0$, then the equation* (1) *either is trivial, is equivalent to an equation $As = B$ or an equation $A = Bs$ for some $s \in M_\Sigma$ or is contradictory.*

Proof. Assume $u = \epsilon$. Then $B = Bu'$. Thus either $u' = \epsilon$ and the equation is trivial, or $u' \neq \epsilon$ and the equation is contradictory.

Therefore, assume that $u \neq \epsilon \neq u'$. Then u and u' must be of the form $u = xyz^{-1}$, $u' = x'y'z'^{-1}$ for maximal $x, x', z, z' \in M_\Sigma$, i.e., y, y' each are either equal to ϵ or of the form $a^- wb$ for some $a, b \in \Sigma$. Then all x, x', z, z' are different from ϵ. Then equation (1) is equivalent to:

$$Ax = Bx' \qquad \wedge \qquad y = y' \qquad \wedge \qquad Az = Bz'$$

By bottom cancellation, these three equations either are equivalent to one fixed relation between $As = B$ or $A = Bs$ for some $s \in M_\Sigma$, or to a contradiction. □

Example 13. Consider the equation

$$Affg^{-1}f^{-1}A^{-1} \doteq Bfg^{-1}B^{-1}$$

which is, according to Lemma 2, equivalent to

$$Aff \doteq Bf \qquad \wedge \qquad \epsilon \doteq \epsilon \qquad \wedge \qquad Afg \doteq Bg$$

By bottom cancellation, we conclude that the conjunction is equivalent to a solved equation $Af \doteq B$. □

Now assume that there is another equation:

$$AvA^{-1} = Bv'B^{-1} \tag{2}$$

with non-negative v, v' where $|v| = |v'|$.

Theorem 10. *The two equations* (1) *and* (2) *are effectively equivalent either to one solved equation, or to a single equation of the form* (1) *or are contradictory.*

Proof. We perform an induction on the sum of balances $|u|+|v|$. W.l.o.g., assume that $|u| \geq |v|$. If $|v| = 0$, then the assertion follows from Lemma 2. Therefore, assume that $|v| > 0$, and $r \geq 1$ is the maximal number such that $|v^r| = r \cdot |v| \leq |u|$. Then we construct the elements uv^{-r} and $u'v'^{-r}$, which are both non-negative by Lemma 1. Let w, w' be obtained from uv^{-r} and $u'v'^{-r}$ by exhaustively applying the cancellation rules. By construction, these are non-negative as well. Then we consider the equation:

$$AwA^{-1} = Bw'B^{-1} \tag{3}$$

which is implied by the two equations (1) and (2).

If $w = \epsilon$, then either $w' = \epsilon$ holds and the equation (3) is trivial, or $w' \neq \epsilon$ and equation (3) is contradictory. In the first case, the equation (2) is implied by equation (1), while in the second case the two given equations (1) and (2) are contradictory. The same argument applies when $w' = \epsilon$ with the roles of A, B exchanged. Therefore now assume that $w \neq \epsilon \neq w'$. Otherwise, the pair of equations (1) and (2) is equivalent to the pair of equations (2) and (3), where the sum of balances $|w| + |v| \leq |w| + r \cdot |v| = |u| < |u| + |v|$ has decreased. For these, the claim follows by inductive hypothesis. □

In [8] a similar argument is presented. The argument there together with the resulting algorithm has been significantly simplified by introducing the extra notion of *non-negativity*.

B Proof of Theorem 4

In order to prove the theorem we show that every T-satisfiable conjunction of equalities of the same format is effectively T-subsumed by a conjunction of at most three equalities. Furthermore, the proof indicates that, given three equalities, it can be effectively decided whether or not a fourth equality is T-subsumed or not. We consider one case of the assertion of the theorem after the other.

Same variable on both sides. Consider the two distinct equalities

$$As_1\mathbf{x} \doteq Bt_1\mathbf{x} \qquad As_2\mathbf{x} \doteq Bt_2\mathbf{x}$$

where $s_i, t_i \in M_G$, and assume that the conjunction of them is T-satisfiable. We claim that then $s_1\mathbf{x} \neq s_2\mathbf{x}$ and $t_1\mathbf{x} \neq t_2\mathbf{x}$. For that, we convince ourselves first that $s_1 \neq s_2$ and $t_1 \neq t_2$ must hold. Then for a contradiction, assume that $s_1\mathbf{x} \doteq s_2\mathbf{x}$. Since $s_1 \neq s_2$, their unifier must map $\mathbf{x}$ to a ground term of s_1 and s_2. These ground terms are all contained in G, whereas we only consider values for $\mathbf{x}$ in $M_G R$, which is disjoint from G. A similar argument also shows that $t_1\mathbf{x} \neq t_2\mathbf{x}$ holds. Thus by factorization, $Ar_1 \doteq Br_2$ must hold for some $r_1, r_2 \in M_G$ of which at least one equals $\bullet$. Due to unique factorization, we then may cancel $\mathbf{x}$ on both sides, resulting in the equalities $As_1 \doteq Bt_1$ and $As_2 \doteq Bt_2$. These can be simplified to one equality $Ar_1 \doteq Br_2$ for some $r_1, r_2 \in M_G$ where $r_i = \bullet$ for at least one i. Hence, the second equality is T-subsumed by the first one.

One-sided single variable. Consider the three distinct equalities

$$As_1 \doteq Bt_1\mathbf{x} \qquad As_2 \doteq Bt_2\mathbf{x} \qquad As_3 \doteq Bt_3\mathbf{x}$$

where $s_i \in M_G R$ and $t_i \in M_G$, and assume that the conjunction of them is T-satisfiable. Again, we argue that all s_i must be distinct as well as all $t_i\mathbf{x}$. Then again by factorization, $Ar_1 \doteq Br_2$ for some templates r_1, r_2 of which at least one equals $\bullet$. By unique factorization, $s_1 = s_1'r$ for some $s_1' \in M_G$ and $r \in R$. Therefore, again by unique factorization, the value for $\mathbf{x}$ also must terminate in the term r, i.e., is of the form $\mathbf{x} = x'r$ for some $x' \in M_G$. Accordingly, also s_2, s_3 can be factored as $s_i = s_i'r$ for suitable $s_i' \in M_G$. Canceling out the ground terms r, we obtain the monoid equalities:

$$As_1' \doteq Bt_1x' \qquad As_2' \doteq Bt_2x' \qquad As_3' \doteq Bt_3x'$$

Assume w.l.o.g., that the balance of s_1 is less or equal to the balances of s_2 and s_3. Then the conjunction of the three equalities is T-equivalent to:

$$As_1' \doteq Bt_1x' \qquad As_2's_1'^{-1}A^{-1} \doteq Bt_2t_1^{-1}B^{-1} \qquad As_3's_1'^{-1}A^{-1} \doteq Bt_3t_1^{-1}B^{-1}$$

where $s_2's_1'^{-1}, t_2t_1^{-1}, s_3's_1'^{-1}, t_3t_1^{-1}$ all are non-negative. According to Theorem 10, the two last equalities are either T-equivalent to each other, which means that the initial conjunction is T-equivalent to the conjunction of the two equalities

$$As_1 \doteq Bt_1\mathbf{x} \qquad As_2 \doteq Bt_2\mathbf{x}$$

and the assertion follows. Otherwise, they are T-equivalent to an equality $Ar_1 \doteq Br_2$ for templates r_1, r_2 of which at least one equals $\bullet$. A fourth equality is then either T-subsumed or falsifies the conjunction of equalities. A similar argument applies to equalities of the form $At_i\mathbf{x} \doteq Bs_i$.

Different variables on both sides. Consider the three distinct equalities

$$As_1\mathbf{x} \doteq Bt_1\mathbf{y} \qquad As_2\mathbf{x} \doteq Bt_2\mathbf{y} \qquad As_3\mathbf{x} \doteq Bt_3\mathbf{y}$$

for distinct program variables $\mathbf{x}, \mathbf{y}$ where $s_i, t_i \in M_G$, and assume that the conjunction of them is T-satisfiable. As before, we argue that $s_i\mathbf{x} \neq s_j\mathbf{x}$, $t_i\mathbf{y} \neq t_j\mathbf{y}$ for all $i \neq j$ must hold. Then by factorization, A is a prefix of B or vice versa. But then, due to unique factorization, also As_1 is a prefix of Bt_1 or vice versa. This means that there are $\mathbf{u}, \mathbf{v} \in M_G$ of which one equals $\bullet$ such that $As_1\mathbf{u} \doteq Bt_1\mathbf{v}$, which (by top cancellation) implies that $\mathbf{vx} = \mathbf{uy}$ holds. From that, we conclude that $As_i\mathbf{u} \doteq Bt_i\mathbf{v}$ for all i. Assume again w.l.o.g. that the balance of s_1 is less or equal to the balances of s_2 and s_3. We then proceed as in the last case to obtain the T-equivalent three equalities:

$$As_1\mathbf{u} \doteq Bt_1\mathbf{v} \qquad As_2s_1^{-1}A^{-1} \doteq Bt_2t_1^{-1}B^{-1} \qquad As_3s_1^{-1}A^{-1} \doteq Bt_3t_1^{-1}B^{-1}$$

where $s_2s_1^{-1}, t_2t_1^{-1}, s_3s_1^{-1}, t_3t_1^{-1}$ all are non-negative. According to Theorem 10, the latter two equalities again are T-equivalent to an equality $Ar_1 \doteq Br_2$ for templates r_1, r_2 of which at least one equals $\bullet$, or are T-equivalent to each other, and the assertion of the theorem follows. This completes the proof. $\square$