Testing Herbrand Equalities and Beyond

Markus Müller-Olm[1]*, Oliver Rüthing[2], and Helmut Seidl[3]

[1] FernUniversit¨at Hagen, FB Informatik, LG PI 5, Universit¨atsstr. 1, 58097 Hagen, Germany
mmo@ls5.informatik.uni-dortmund.de
[2] Universit¨at Dortmund, FB 4, LS V, 44221 Dortmund, Germany
ruething@ls5.cs.uni-dortmund.de
[3] TU M¨unchen, Informatik, I2, 85748 M¨unchen, Germany
seidl@in.tum.de

Abstract. A Herbrand equality between expressions in a program is an equality which holds relative to the Herbrand interpretation of operators. We show that the problem of *testing* validity of positive Boolean combinations of Herbrand equalities at a given program point is decidable — even in presence of disequality guards. This result vastly extends classical results which cannot deal with disjunctions and are always based on an abstraction of conditional branching with non-deterministic choice. We also show that in the classic setting where all guards are ignored conjunctions of Herbrand equalities can be tested in polynomial time. As an application of our method, we show how to derive all valid Herbrand constants in programs with disequality guards.
Finally, we show that in presence of equality guards instead of disequality guards, testing of a given Herbrand equality becomes undecidable.

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 where this information can be used for performing and enhancing powerful transformations like (partial) redundancy elimination including loop invariant code motion [19, 21, 12], strength reduction [22], constant propagation and branch elimination [3, 7].

Since determining whether two variables always have the same value at a program point is an undecidable problem even without interpreting conditionals [18], analyses are usually restricted to detect only a subset, i.e., a safe approximation, of all equivalences. Analyses based on Herbrand interpretation of operators consider two values equal only if they are constructed by the same operator applications. Cocke and Schwartz [4] presented the earliest such technique for finding equalities inside basic blocks. Since their technique operates by assigning hash values to computations, the detection of (Herbrand-)equivalences is often also referred to as *value numbering*. In his seminal paper [11], Kildall presents a technique for *global value numbering* that extends Cocke's and Schwartz's technique to flow graphs with loops. In contrast to a number of algorithms focusing more on efficiency than on precision [18, 1, 3, 20, 7, 8],

* On leave from Universit¨at Dortmund.

Kildall's algorithm detects all Herbrand equalities in a program. However, the representation of equalities can be of exponential size in terms of the argument program. This deficiency is still present in the algorithm for partial redundancy elimination of Steffen et al. [21] which employs a variant of Kildall's algorithm using a compact representation of Herbrand equivalences in terms of *structured partition DAGs (SPDAGs)*. Recently, Gulwani and Necula proposed a polynomial time variant of this algorithm exploiting the fact that SPDAGs can be pruned, if only equalities of bounded size are searched for [9].

The analyses based on Herbrand interpretation mentioned above ignore guards in programs.[4] In this paper, we present an analysis that fully interprets besides the assignments in the program also all the disequality guards with respect to Herbrand interpretation. More specifically, we show that the problem of *testing* the validity of positive Boolean combinations of Herbrand equalities at a given program point is decidable — even in presence of non-equality guards. (A Herbrand equality between expressions in a program is an equality which holds relative to Herbrand interpretation of operators; a positive Boolean combination of Herbrand equalities is a formula constructed from Herbrand equalities by means of disjunction and conjunction.) We also present a PSPACE lower bound for this problem. This result vastly extends classical results which cannot deal with disjunctions and are always based on an abstraction of conditional branching with non-deterministic choice. As an application of our method, we show how to derive all valid Herbrand constants in program with non-equality guards.

In order to show the decidability result, we rely on effective weakest precondition computations using a certain lattice of assertions. While we have used the idea of effective weakest precondition computations before [13–16], the type of assertions and the kind of results exploited is quite different here. In [13–16] assertions are represented by bases of vector spaces or polynomial ideals and results from polynomial and linear algebra are exploited. Here we use equivalence classes of certain types of formulas as assertions and syntactic techniques from automatic theorem proving. In order to introduce our technique in a simpler scenario and as a second application we show that in the classic setting where all guards are ignored conjunctions of Herbrand equalities can be tested in polynomial time.

The considerations of this paper belong to a line of research in which we try to identify smoothly defined classes of (abstractions of) programs and analysis problems for which complete analyses are possible. Here, we abstract from the equality guards — and rely on Herbrand interpretation. There are two reasons why we must ignore equality guards. The first reason is that we cannot hope for a complete treatment of equality guards; c.f. sect. 5, theo. 6. The second reason is subtle but even more devastating: using Herbrand interpretation of programs with equality guards for inferring definite equalities w.r.t. another interpretation — which is what we are up to when we use Herbrand interpretation in program analysis — is unsound. The reason is that an equality might be invalid w.r.t. Herbrand interpretation but valid w.r.t. the "real" interpretation. Thus,

[4] The branch sensitive methods [3, 7, 2] based on the work of Click and Cooper [3] unify value numbering with constant propagation and elimination of dead branches. However, the value numbering component of these methods is based on the work of Alpern, Wegman and Zadeck [1] which is restricted to the detection of a small fragment of Herbrand equalities only.

it can happen that a Herbrand interpretation based execution would not pass an equality guard while executions based on the real semantics would do so. In this case, the Herbrand interpretation based analysis would consider too few executions, making it unsound. Note that this problem does not occur for disequality guards, because, whenever an equality is invalid w.r.t. the "real" interpretation it is also invalid w.r.t. Herbrand interpretation.

In Section 2 we introduce *Herbrand programs* as an abstract model of programs for which our analyses are complete. Moreover, we analyze the requirements a lattice of assertions must satisfy in order to allow weakest precondition computations. In Section 3 we present an analysis that tests conjunctions of Herbrand equalities in Herbrand programs *without* disequality guards in polynomial time. This analysis is extended in Section 4 to an analysis that tests arbitrary positive Boolean combinations of Herbrand equalities in Herbrand programs *with* disequality guards. For this analysis we can show termination but we do not have an upper bound for its running time. In Section 5 we show that there are no effective and complete analysis procedures for Herbrand programs with equality instead of disequality guards. Also we provide a PSPACE lower bound for testing of Herbrand equalities in Herbrand programs with disequality guards.

2 Herbrand Programs and Weakest Preconditions

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ be the set of variables the program operates on. We assume that the variables take values which are constructed from variables and constants by means of operator application. 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. Note that in examples, we will omit brackets around the arguments of unary operators and often write binary operators *infix*. Let $\mathcal{T}_\Omega$ be the set of all formal terms built up from Ω. For simplicity, we assume that the set Ω_0 is non-empty and that there is at least one operator. Given this, 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 $\mathbf{X}$. In the present context, we will not interpret constants and operators. Thus, a *state* assigning values to the variables is conveniently modeled by a *ground substitution* $\sigma : \mathbf{X} \to \mathcal{T}_\Omega$.

As usual, we assume that programs are given as control-flow graphs. An example of such a graph is shown in fig. 1. We assume that the basic statements in the program are either assignments of the form $\mathbf{x}_j := t$ where $t \in \mathcal{T}_\Omega(\mathbf{X})$ or nondeterministic assignments $\mathbf{x}_j$:=? modeling, e.g., input statements which return unknown values, and that branching in general is non-deterministic. The only control statements that we handle are *disequality guards* of the form $t_1 \neq t_2$. Note that positive Boolean combinations of disequality guards can be coded by small flow graphs as shown in fig. 2 for $(t_1 \neq t_1' \wedge t_2 \neq t_2') \vee t_3 \neq t_3'$. Assignments $\mathbf{x}_j := \mathbf{x}_j$ have no effect onto the program state. They can be used as skip statements and for abstraction of guards that are not disequality guards. Let Stmt be the set of assignments and guards. Each statement s induces a transformation, $[\![s]\!]$, on sets of program states given by

$$\begin{aligned}
[\![\mathbf{x}_j := t]\!]\, S &= \{\sigma[\mathbf{x}_j \mapsto \sigma(t)] \mid \sigma \in S\}\,, \\
[\![\mathbf{x}_j :=?]\!]\, S &= \{\sigma[\mathbf{x}_j \mapsto t'] \mid \sigma \in S, t' \in \mathcal{T}_\Omega\}\,, \text{ and} \\
[\![t_1 \neq t_2]\!]\, S &= \{\sigma \in S \mid \sigma(t_1) \neq \sigma(t_2)\}\,.
\end{aligned}$$

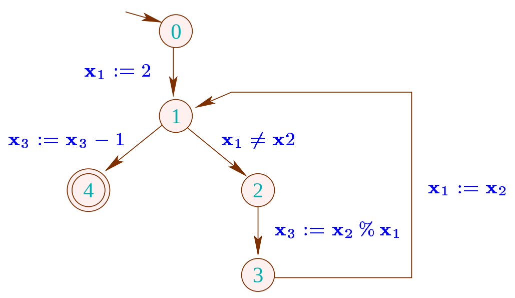

0
$\mathbf{x}_1 := 2$
1
$\mathbf{x}_3 := \mathbf{x}_3 - 1$
$\mathbf{x}_1 \neq \mathbf{x}2$
4
2
$\mathbf{x}_1 := \mathbf{x}_2$
$\mathbf{x}_3 := \mathbf{x}_2 \,\%\, \mathbf{x}_1$
3

Fig. 1. An example program.

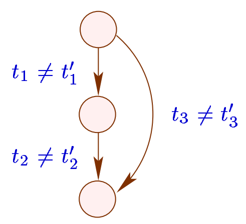


Fig. 2. Boolean combinations of guards.

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 ground substitution that maps $\mathbf{x}_j$ to $t' \in \mathcal{T}_\Omega$ and variables $\mathbf{x}_i \neq \mathbf{x}_j$ to $\sigma(\mathbf{x}_i)$. Note that for $s \equiv \mathbf{x}_j :=?$, the variable $\mathbf{x}_j$ may receive *any* value.

A *Herbrand program* is given by a *control flow graph* $G = (N, E, \mathsf{st})$ that consists of a set N of *program points*; a set of edges $E \subseteq N \times \mathsf{Stmt} \times N$; and a special *entry (or start) point* $\mathsf{st} \in N$. As common in flow analysis, we use the program's collecting semantics as primary semantic reference point. For a given set of initial states S, the collecting semantics assigns to each program point $u \in N$ the set of all those states that occur at u in some execution of the program from a state in S. It can be characterized as the least solution of the following constraint system, $\mathbf{V}_S$, on sets of states, i.e., sets of ground substitutions:

$$\begin{array}{ll} [\mathrm{V1}] & \mathbf{V}_S[\mathsf{st}] \supseteq S \\ [\mathrm{V2}] & \mathbf{V}_S[v] \supseteq [\![s]\!](\mathbf{V}_S[u])\,, \text{ for each } (u,s,v) \in E \end{array}$$

By abuse of notation we denote the components of the least solution of the constraint system $\mathbf{V}_S$ (which exists by Knaster-Tarski fixpoint theorem) by $\mathbf{V}_S[v]$, $v \in N$. Often if we have no knowledge about possible initial states we choose $S = (\mathbf{X} \to \mathcal{T}_\Omega)$.

An equation $t_1 = t_2$ is *valid* for a *substitution* $\sigma : \mathbf{X} \to \mathcal{T}_\Omega(\mathbf{X})$ iff $\sigma(t_1) = \sigma(t_2)$. Accordingly, an equation $t_1 = t_2$ is valid at a program point v from a set S of initial states iff it is valid for all $\sigma \in \mathbf{V}_S[v]$. It is called valid at a program point v if it is valid at v from $(\mathbf{X} \to \mathcal{T}_\Omega)$. These definitions are straightforwardly extended to predicate-logical formulas over equations as atomic formulas. We write $\sigma \models \phi$ if ϕ is valid for a substitution σ. We call two formulas ϕ_1, ϕ_2 *equivalent* (and write $\phi_1 \Leftrightarrow \phi_2$) if they are valid for the same substitutions. We write $\phi_1 \Rightarrow \phi_2$ if $\sigma \models \phi_1$ implies $\sigma \models \phi_2$.

For every assignment or disequality guard s, we consider the corresponding *weakest precondition transformer* $[\![s]\!]^{\mathsf{t}}$ which takes a formula ϕ and returns the weakest precondition of ϕ which must hold before execution of s such that ϕ holds after s. This transformation is given by the well-known rules:

$$[\![\mathbf{x}_j := t]\!]^{\mathsf{t}}\phi = \phi[t/\mathbf{x}_j]\,,\; [\![\mathbf{x}_j :=?]\!]^{\mathsf{t}}\phi = \forall\,\mathbf{x}_j.\,\phi\,,\; \text{and } [\![t_1 \neq t_2]\!]^{\mathsf{t}}\,\phi \;=\; (t_1 = t_2) \vee \phi\,.$$

Here $\phi[t/\mathbf{x}_j]$ denotes the formula obtained from ϕ by substituting t for $\mathbf{x}_j$. The key property which summarizes the relationship between the transformation $[\![s]\!]$ and the weakest precondition transformation $[\![s]\!]^{\mathsf{t}}$ is given in the following lemma.

Lemma 1. *For every set $S \subseteq \mathbf{X} \to \mathcal{T}_\Omega$ of ground substitutions and any formula ϕ, σ satisfies ϕ for all $\sigma \in [\![s]\!]\, S$ iff τ satisfies $[\![s]\!]^{\mathsf{t}}\phi$ for all $\tau \in S$.* $\square$

We identify the following desirable properties of a language L of formulas to be used for weakest precondition computations. First, it must be (semantically) closed under $[\![s]\!]^{\mathsf{t}}$, i.e., under substitution, universal quantification, and, if we want to handle disequality guards, disjunction. More precisely, this means that L must contain formulas equivalent to $\phi[t/\mathbf{x}_i]$, $\forall\,\mathbf{x}_i.\phi$, and $\phi \vee \phi'$, respectively, for all $\phi, \phi' \in L$. Moreover, we want the fixpoint computation for characterizing the weakest pre-conditions at every program point to terminate. Therefore, we secondly demand that L is closed under finite conjunctions, i.e., that it contains a formula equivalent to true as well as a formula equivalent to $\phi \wedge \phi'$ for all $\phi, \phi' \in L$, and that L is *compact*, i.e., for every sequence $\phi_0, \phi_1, \ldots$ of formulas, $\bigwedge_{i \geq 0} \phi_i \;\Leftrightarrow\; \bigwedge_{i=0}^{m} \phi_i$ for some $m \geq 0$.

In order to construct a lattice of properties from L we consider *equivalence classes of formulas*, which, however, will always be represented by one of their members. Let $\mathbb{L}$ denote the set of all equivalence classes of formulas. Then this set is partially ordered w.r.t. "$\Rightarrow$" (on the representatives) and the pairwise lower bound always exists and is given by "$\wedge$". By compactness, all descending chains in this lattice are ultimately stable. Therefore, not only finite but also infinite subsets $X \subseteq \mathbb{E}$ have a greatest lower bound. This implies that $\mathbb{L}$ is a complete lattice.

Assume that we want to check whether a formula ϕ holds at a specific program point v_t. Then we put up the following constraint system over $\mathbb{L}$:

$$\begin{array}{ll} [\mathrm{E1}] & \mathbf{WP}[v_t] \Rightarrow \phi \\ [\mathrm{E2}] & \mathbf{WP}[u] \;\Rightarrow [\![s]\!]^{\mathsf{t}}(\mathbf{WP}[v])\,,\ \text{ for each } (u,s,v) \in E \end{array}$$

Since $\mathbb{L}$ is a complete lattice, a greatest solution of the constraint system exists again by Knaster-Tarski fixpoint theorem. This solution is denoted by $\mathbf{WP}[v]$, $v \in N$, as well. We have (c.f. App. A):

Lemma 2. *Suppose ϕ_0 is a pre-condition, i.e., a formula describing initial states, and let $S_0 = \{\sigma : \mathbf{X} \to \mathcal{T}_\Omega \mid \sigma \models \phi_0\}$ be the corresponding set of initial states. Then, formula $\phi \in L$ is valid at program point v_t from S_0 iff $\phi_0 \Rightarrow \mathbf{WP}[\mathsf{st}]$.* $\square$

3 Conjunctions

We first consider conjunctions of equalities. Clearly, conjunctions of equalities are not closed under "$\vee$". Therefore, this assertion language is not able to handle disjunctions and disequality guards precisely. For closure under universal quantification, we find the following equivalence for a single equality $\mathbf{x}_i = s$

$$\forall\,\mathbf{x}_j\,.\,\mathbf{x}_i = s \;\;\Leftrightarrow\;\; \begin{cases} \mathbf{x}_i = s & \text{if } i \neq j \text{ and } \mathbf{x}_j \text{ does not occur in } s \\ \mathsf{true} & \text{if } i = j \text{ and } s \equiv \mathbf{x}_j \\ \mathsf{false} & \text{otherwise} \end{cases}$$

Thus, since $\forall \mathbf{x}_i .\ (e_1 \wedge \ldots \wedge e_m) \quad \Leftrightarrow \quad (\forall \mathbf{x}_i .\ e_1) \wedge \ldots \wedge (\forall \mathbf{x}_i .\ e_m)$, conjunctions are closed under universal quantification. Also, in absence of disequality guards, the weakest precondition of a conjunction w.r.t. a statement always is again a conjunction — or false. We collect some basic facts about conjunctions of equalities.

A *substitution* $\sigma : \mathbf{X} \to \mathcal{T}_\Omega(\mathbf{X})$ (possibly containing variables in the image terms) satisfies a conjunction of equalities $c \equiv s_1 = t_1 \wedge \ldots \wedge s_m = t_m$ iff $\sigma(s_i) = \sigma(t_i)$ for $i = 1, \ldots, m$. We then also write $\sigma \models c$. We call c *satisfiable* iff $\sigma \models c$ for at least one σ. Otherwise, i.e., if c is unsatisfiable, c is equivalent to false (the Boolean value 'false'). This value serves as the bottom value of our lattice. The greatest value is given by the *empty* conjunction which is always true and therefore also denoted by true. Whenever the conjunction c is satisfiable, then there is a *most general* satisfying substitution σ, i.e., $\sigma \models c$ and for every other substitution τ satisfying c, $\tau = \tau_1 \circ \sigma$ for some substitution τ_1. Such a substitution σ is also called *most general unifier* of the equations in c [5]. Recall that most general unifiers σ can be chosen *idempotent* meaning that $\sigma = \sigma \circ \sigma$ or, equivalently, no variable $\mathbf{x}_i$ with $\sigma(\mathbf{x}_i) \not\equiv \mathbf{x}_i$ may occur in the image $\sigma(\mathbf{x}_j)$ of any variable $\mathbf{x}_j$.

We consider compact representations of trees. In particular, we assume that identical subterms are represented only once. Therefore, we define the *size* of a term t as the number of distinct subtrees of t. Thus, e.g., the size of $t = a(b\,\mathbf{x}_1, b\,c)$ equals 5 whereas the size of $t' = a(b\,c, b\,c)$ equals 3. The size of a term t is also denoted by $|t|$. According to this definition, the size of $t[s/\mathbf{x}_i]$ is always less than $|t| + |s|$. A conjunction c is *reduced* iff c equals $\mathbf{x}_{i_1} = t_1 \wedge \ldots \wedge \mathbf{x}_{i_m} = t_m$ for distinct variables $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_m}$ such that $t_j \not\equiv \mathbf{x}_{i_j}$ for all j. Let the size $|c|$ of a finite conjunction c be the maximum of 1 and the maximal size of a term occurring in c. We show that every finite conjunction of equalities is equivalent to a reduced conjunction of at most the same size:

Lemma 3. *Every satisfiable conjunction c is equivalent to a reduced conjunction c' with $|c'| \leq |c|$. The conjunction c' can be constructed in polynomial time.*

Proof. It is not hard to show that a reduced conjunction equivalent to c is obtained by taking a most general unifier σ of c and returning the conjunction of equalities $\mathbf{x}_i = \sigma(\mathbf{x}_i)$ for the variables $\mathbf{x}_i$ with $\mathbf{x}_i \neq \sigma(\mathbf{x}_i)$. This reduced conjunction, however, may not satisfy the condition on sizes. The equation $a(\mathbf{x}_1, b\,b\,b\,\mathbf{x}_1) = a(b\,b\,c, \mathbf{x}_2)$, for example, has size 5. The most general unifier is the substitution $\sigma = \{\mathbf{x}_1 \mapsto b\,b\,c, \mathbf{x}_2 \mapsto b\,b\,b\,b\,b\,c\}$. The corresponding reduced equation system therefore would have size 6 — which does not conform to the assertion of the lemma. The reason is that most general unifiers typically are *idempotent*. If we drop this assumption, we may instead consider the substitution $\tau = \{\mathbf{x}_1 \mapsto b\,b\,c, \mathbf{x}_2 \mapsto b\,b\,b\,\mathbf{x}_1\}$ — which is neither idempotent nor a most general unifier, but yields the most general unifier after two iterations, namely, $\sigma = \tau \circ \tau$. The reduced system corresponding to τ has size 4 and therefore is small enough. Our construction of the reduced system thus is based on the construction of a substitution τ such that k-fold composition of k results in the most general unifier of c. Let σ denote an idempotent most general unifier of c. We introduce an equivalence relation $\equiv_\sigma$ on the set of variables $\mathbf{X}$ and subterms of c by $s_1 \equiv_\sigma s_2$ iff $\sigma(s_1) = \sigma(s_2)$. Then there is a partial ordering "$\leq$" on the variables $\mathbf{X}$ such that whenever $\mathbf{x}_j \equiv_\sigma t$ for some subterm $t \notin \mathbf{X}$ of c, then $\mathbf{x}_i < \mathbf{x}_j$ for all variables $\mathbf{x}_i$ occurring in t. Moreover:

– if $\sigma(\mathbf{x}_j) \in \mathbf{X}$ then $t \in \mathbf{X}$ for every t with $\mathbf{x}_j \equiv_\sigma t$.
– if $\sigma(\mathbf{x}_j) \notin \mathbf{X}$, then $\mathbf{x}_j \equiv_\sigma t$ for some subterm $t \notin \mathbf{X}$ of c.

for every variable $\mathbf{x}_j$. Let us w.l.o.g. assume that $i < j$ implies $\mathbf{x}_i < \mathbf{x}_j$. Then we define substitutions $\tau_1, \ldots, \tau_k$ by $\tau_1 = \sigma$, and for $i > 1$,

$$\tau_i(\mathbf{x}_j) \;=\; \begin{cases} t_i & \text{if} \quad i = j \\ \tau_{i-1}(\mathbf{x}_j) & \text{if} \quad i \neq j \end{cases}$$

where $t_i = \sigma(\mathbf{x}_i)$ if $\sigma(\mathbf{x}_i) \in \mathbf{X}$. Otherwise, we choose $t_i = t$ for any $t \notin \mathbf{X}$ with $\mathbf{x}_i \equiv_\sigma t$. By induction on i, we then verify that $\tau_i^i = \sigma$. We conclude that $c' \equiv \bigwedge\{\mathbf{x}_i = \tau_k(\mathbf{x}_i) \mid \tau_k(\mathbf{x}_i) \neq \mathbf{x}_i\}$ is a conjunction which is equivalent to c whose non-variable right-hand sides all are sub-terms of right-hand sides of c. Since a most general unifier can be constructed in polynomial (even linear) time, the assertion follows. □

Lemma 4. *If $c \Rightarrow c_1$ where c is satisfiable and c_1 is reduced, then c is equivalent to a reduced conjunction $c_1 \wedge c'$. In particular, c' can be computed in polynomial time.* □

Proof. Let σ, σ_1 denote idempotent most general unifiers of c and c_1, respectively. Since $c \Rightarrow c_1$, $\sigma = \sigma' \circ \sigma_1$ for some σ', which can be chosen idempotent as well, where the domains of σ_1 and σ' are disjoint. Then we simply choose c' as the reduced conjunction constructed from σ' along the same lines as in lemma 3. □

As a corollary, we obtain:

Corollary 1. *For every sequence $c_0 \Leftarrow \ldots \Leftarrow c_m$ of pairwise inequivalent conjunctions c_j, $m \leq k + 1$.* □

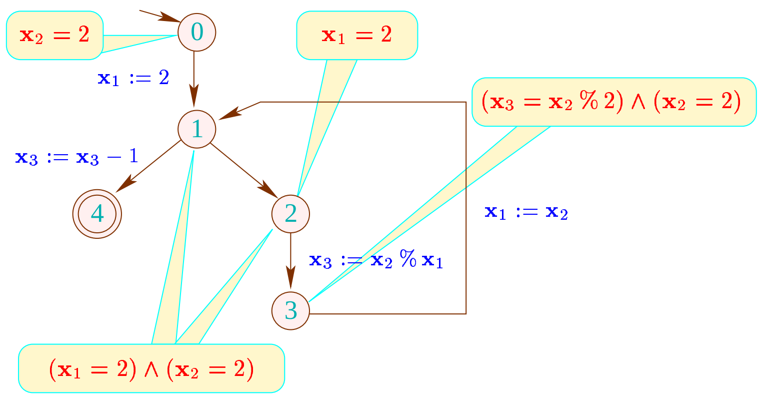
Corollary 1 implies compactness of the language of conjunctions of equalities. Let $\mathbb{E}$ denote the set of all equivalence classes of finite conjunctions of equalities $s = t$, $s, t \in \mathcal{T}_\Omega(\mathbf{X})$. In order to check validity of a conjunction c at a program point v_t, we choose $\mathbb{L} = \mathbb{E}$, compute the weakest solution of the constraint system for $\mathbf{WP}$ by fixpoint iteration, and check, if $\mathbf{WP}[\mathsf{st}]$ is equivalent to true. The latter is equivalent to validity of c at v_t by lemma 2. Let us estimate the running time of the fixpoint computation. By corollary 1, each variable in the constraint system may be updated at most $k+1$ times. The application of a transformer $[\![s]\!]^{\mathsf{t}}$ as well as conjunction can be executed in time polynomial in their inputs. In order to obtain a polynomial time algorithm for computing the values $\mathbf{WP}[v]$, it therefore remains to prove that all conjunctions which are intermediately constructed during fixpoint iteration have polynomial sizes. For this, we recall the following two facts. First, a standard worklist algorithm for computing the least fixpoint will perform $\mathcal{O}(n \cdot k)$ evaluations of right-hand sides of constraints. Assuming that w.l.o.g. all right-hand sides in the program have constant size, each evaluation of a right-hand side may increase the maximal size of an equation at most by a constant. Since the greatest lower bound operation does not increase the maximal size, we conclude that all equalities occurring during fixpoint iteration, are bounded in size by $\mathcal{O}(n \cdot k + m)$ if m is the size of the initial equation c.

Summarizing, we obtain:

Theorem 1. *Assume p is a Herbrand program without disequality guards, v_t is a program point and c is a conjunction of equalities. Then it can be decided in polynomial time whether or not c is valid in p at v_t.* □

In practice, we can stop with the fixpoint iteration for $\mathbf{WP}$ as soon as we find the value false at some reachable program point or change the value stored for the start point st because this implies that $\mathbf{WP}[\mathsf{st}]$ cannot be true. A worklist algorithm that integrates this test can be seen as a demand-driven search for a reason why c fails at v_t.

As an example, consider the program from section 2. Since we use conjunctions of equalities only, we must ignore the disequality guard. The weakest pre-conditions computed for the equality $\mathbf{x}_3 = \mathbf{x}_2 \,\%\, 2$ at program point 3 then are shown in figure 3. Since the weakest pre-condition for the start node 0 is different from true, we cannot



$\mathbf{x}_2 = 2$
0
$\mathbf{x}_1 = 2$
$\mathbf{x}_1 := 2$
$(\mathbf{x}_3 = \mathbf{x}_2 \,\%\, 2) \wedge (\mathbf{x}_2 = 2)$
1
$\mathbf{x}_3 := \mathbf{x}_3 - 1$
4
2
$\mathbf{x}_1 := \mathbf{x}_2$
$\mathbf{x}_3 := \mathbf{x}_2 \,\%\, \mathbf{x}_1$
3
$(\mathbf{x}_1 = 2) \wedge (\mathbf{x}_2 = 2)$

Fig. 3. The pre-conditions computed for $\mathbf{x}_3 = \mathbf{x}_2 \,\%\, 2$ at program point 3.

conclude that the tested equality holds.

As a second application of wp-computations with the lattice $\mathbb{E}$ we obtain:

Theorem 2. *Assume p is a Herbrand program without disequality guards and v_t is a program point of p. Then it can be determined in polynomial time whether or not a variable $\mathbf{x}_i$ is constant at v_t, i.e., has always the same value c when program execution reaches v_t.*

Proof. We introduce the equality $\mathbf{x}_i = \mathbf{y}$ for some fresh variable $\mathbf{y}$. Then $\mathbf{x}_i$ is constant at program point v_t iff the weakest precondition $\mathbf{WP}[\mathsf{st}]$ of this equality at program entry is implied by $\mathbf{y} = c$ for some ground term $c \in \mathcal{T}_\Omega$. In this case $\mathbf{WP}[\mathsf{st}]$ either is equivalent to true — implying that v_t is not reachable, or equivalent to $\mathbf{y} = c$. In the latter case, the value c constitutes the constant value of $\mathbf{x}_i$ at program point v_t. Since $\mathbf{WP}[\mathsf{st}]$ for the given equality can be computed in polynomial time, we conclude that all program constants can be computed in polynomial time as well. □

4 Disjunctions

In this section, we consider disjunctions of conjunctions of equalities which we call *DC-formulas*. Note that every positive Boolean combination of equalities, i.e. each formula which is built up from equalities by means of conjunctions and disjunctions can be written as a DC-formula by the usual distributivity laws. Clearly, the language of DC-formulas is closed under substitution and disjunction and, again by distributivity, also under conjunction. First, we convince ourselves that it is indeed also closed under universal quantification.

Lemma 5. *Assume that $\mathcal{T}_\Omega$ is infinite. Then we have:*

1. *For every conjunction c of equalities, $\forall\, \mathbf{x}_j.\ c \;\;\Leftrightarrow\;\; c[t_1/\mathbf{x}_j] \wedge c[t_2/\mathbf{x}_j]$ for any $t_1, t_2 \in \mathcal{T}_\Omega$ with $t_1 \neq t_2$.*
2. *For every disjunction $\phi \;\equiv\; c_1 \vee \ldots \vee c_m$ of conjunctions c_i of equalities,*

$$\forall\, \mathbf{x}_j.\ \phi \;\Leftrightarrow\; (\forall\, \mathbf{x}_j.\ c_1) \vee \ldots \vee (\forall\, \mathbf{x}_j.\ c_m) \qquad \square$$

A DC-formula d need no longer have a single most general unifier. The disjunction $a\,\mathbf{x}_1 = a\,b \vee a\,c = a\,\mathbf{x}_1$, for example, has two maximally general unifiers $\{\mathbf{x}_1 \mapsto b\}$ and $\{\mathbf{x}_1 \mapsto c\}$. By lemma 3, however, each conjunction in a DC-formula d can be brought into reduced form. Let us call the resulting formula a *reduced DC-formula*. Our further considerations are based on the following fundamental theorem.

Theorem 3. *Let $d_j, j \geq 0$, be a sequence of DC-formulas such that $d_j \Leftarrow d_{j+1}$ for all $j \geq 0$. Then this sequence is ultimately stable, i.e., there is some $m \in \mathbb{N}$ such that for all $m' \geq m$, $d_m \Leftrightarrow d_{m'}$.*

Proof. If any of the d_j is unsatisfiable, i.e., equivalent to false, then all positive Boolean combinations of greater index also must be unsatisfiable, and the assertion of the theorem follows. Therefore let us assume that all d_j are satisfiable. W.l.o.g. all d_j are reduced. We successively construct a sequence Γ_j, $j \geq 0$, where $\Gamma_0 = d_0$ and Γ_{j+1} is a reduced DC-formula equivalent to $\Gamma_j \wedge d_{j+1}$ for $j \geq 0$. Since $d_j \Leftarrow d_{j+1}$ for all j, Γ_j is equivalent to d_j. For a reduced DC-formula Γ, we maintain a vector $v[\Gamma] \in \mathbb{N}^k$ where the i-th component of $v[\Gamma]$ counts the number of conjunctions in Γ with exactly i equalities. On $\mathbb{N}^k$ we consider the lexicographical ordering "$\leq$" which is given by: $(n_1, \ldots, n_k) \leq (n'_1, \ldots, n'_k)$ iff either $n_l = n'_l$ for all l, or there is some $1 \leq i \leq k$ such that $n_l = n'_l$ for all $l < i$, and $n_i < n'_i$. Recall that this ordering is a *well-ordering*, i.e., it does not admit infinite strictly decreasing sequences.

Now assume that Γ_j equals $c_1 \vee \ldots \vee c_m$ for reduced conjunctions c_i. Assume that d_{j+1} equals $c'_1 \vee \ldots \vee c'_n$ for reduced conjunctions c'_l. Then by distributivity, $\Gamma_j \wedge d_{j+1}$ is equivalent to $\bigvee_{i=1}^m c_i \wedge (c'_1 \vee \ldots \vee c'_n)$. First, assume that for a given i, $c_i \wedge c'_l$ is equivalent to c_i for some l. Then also $c_i \wedge (c'_1 \vee \ldots \vee c'_n)$ is equivalent to c_i. Let V denote the subset of all i with this property. Thus for all $i \not\in V$, c_i is *not* equivalent to any of the conjunctions $c_i \wedge c'_l$. Let $J[i]$ denote the set of all l such that $c_i \wedge c'_l$ is satisfiable. Then by lemma 3, we can construct for every $l \in J[i]$, a non-empty conjunction c_{il} such that $c_i \wedge c_{il}$ is reduced and equivalent to $c_i \wedge c'_l$. Summarizing, we construct the reduced DC-formula Γ_{j+1} equivalent to $\Gamma_j \wedge d_{j+1}$ as:

$$\left(\bigvee_{i \in V} c_i\right) \vee \left(\bigvee_{i \notin V} \bigvee_{l \in J[i]} c_i \wedge c_{il}\right)$$

According to this construction, $v[\Gamma_j] = v[\Gamma_{j+1}]$ implies that $V = \{1, \ldots, k\}$ and therefore that Γ_j is equivalent to Γ_{j+1}. Moreover, if Γ_j is not equivalent to Γ_{j+1}, then $v[\Gamma_j] > v[\Gamma_{j+1}]$. Accordingly, if the sequence $\Gamma_j, j \geq 0$, is not ultimately stable, we obtain an infinite sequence of strictly decreasing vectors — contradiction. $\square$

In particular, theorem 3 implies that compactness holds for DC-formulas as well. Note that if we consider not just positive Boolean combinations but additionally allow negation, then the compactness property is immediately lost. To see this, consider an infinite sequence $t_1, t_2, \ldots$ of pairwise distinct ground terms. Then obviously, all conjunctions $\bigwedge_{i=1}^{m} (\mathbf{x}_1 \neq t_i)$, $m \geq 0$, are pairwise inequivalent.

In order to perform effective fixpoint computations, we need an effective test for stability.

Lemma 6. *It is decidable for DC formulas d, d' whether or not $d \Rightarrow d'$.*

Proof. Assume $d \equiv c_1 \vee \ldots \vee c_r$ and $d' \equiv c'_1 \vee \ldots \vee c'_s$ for conjunctions c_i, c'_j. W.l.o.g. we assume that all conjunctions c_i are satisfiable and thus have a most general unifier σ_i. Then $d \Rightarrow d'$ iff $\sigma \models d$ implies $\sigma \models d'$ for all substitutions σ. The latter is the case iff for every i we can find some j such that $\sigma_i \models c'_j$. Since it is decidable whether or not a substitution satisfies a conjunction of equalities, the assertion follows. Note that this decision procedure for implications requires polynomial time. $\square$

We now extend the lattice $\mathbb{E}$ to a lattice $\mathbb{D}$ of equivalence classes of DC-formulas. Again, the ordering is given by implication "$\Rightarrow$" where the binary greatest lower bound operation is "$\wedge$". By theorem 3, all descending chains in $\mathbb{D}$ are ultimately stable. Similar to $\mathbb{E}$, we deduce that $\mathbb{D}$ is in fact a *complete* lattice and therefore amenable to fixpoint computations. Note however that, opposed to the complete lattice $\mathbb{E}$, the new lattice does not have finite height, i.e., there exist strictly descending chains of arbitrary lengths. This more general lattice also allows us to treat disjunctions and hence also Herbrand programs which, besides assignments, contain disequality guards $t_1 \neq t_2$. In particular by lemma 6, we can detect when stability has been reached. We obtain the main result of this section:

Theorem 4. *Assume p is a Herbrand program, possibly with disequality guards. For every program point v_t of p and every positive Boolean combination of equalities d, it is decidable whether or not d is valid at v_t.* $\square$

Consider again the example program from section 2. Assuming that we want to test whether $\mathbf{x}_3 = \mathbf{x}_2 \% 2$ holds at program point 3, we compute the weakest pre-conditions for the program points $0, \ldots, 3$ as shown in figure 4. Now, we indeed find the pre-condition true for the start node of the program implying that the tested equality is valid at program point 3.

Generalizing the idea from Section 3 for constant propagation, we obtain:

Theorem 5. *For a Herbrand program p possibly with disequality guards and a program point v_t of p, it can be decided whether a variable $\mathbf{x}_i$ is constant at v_t.* $\square$

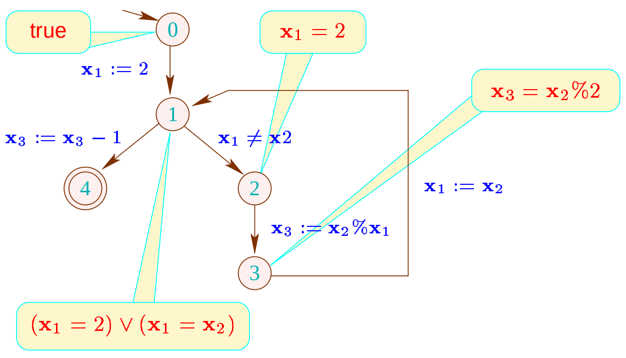

true
$\mathbf{x}_1 = 2$
$\mathbf{x}_3 = \mathbf{x}_2 \% 2$
$(\mathbf{x}_1 = 2) \vee (\mathbf{x}_1 = \mathbf{x}_2)$
0
1
2
3
4
$\mathbf{x}_1 := 2$
$\mathbf{x}_3 := \mathbf{x}_3 - 1$
$\mathbf{x}_1 \neq \mathbf{x}2$
$\mathbf{x}_3 := \mathbf{x}_2 \% \mathbf{x}_1$
$\mathbf{x}_1 := \mathbf{x}_2$

Fig. 4. The pre-conditions computed for $\mathbf{x}_3 = \mathbf{x}_2 \,\%\, 2$ at program point 3.

5 Limitations and Lower Bounds

In [17], we showed for *affine* programs, i.e., programs where the standard arithmetic operators except division are treated precisely, that equality guards allow us to encode Post's correspondence problem. In fact, multiplication with powers of 2 and addition of constants was used to simulate the concatenation with a given string. For Herbrand programs, we simply may encode letters by unary operators. Thus, we obtain:

Theorem 6. *Testing whether a given equality holds at some program point in a Herbrand program with equality guards of the form* $\mathbf{x}_i = \mathbf{x}_j$ *is undecidable.* □

We conclude that precision cannot be achieved if we do not ignore equality guards. Turning to our algorithm for testing disjunctions, we recall that termination of the fix-point algorithm is based on an argument on the well-foundedness of the lexicographical ordering. Sadly enough, this argument does not provide any clue to derive an explicit complexity bound for the algorithm. We at least can show, however, that an algorithm with polynomial worst case runtime cannot be hoped for.

Theorem 7. *It is at least PSPACE-hard to decide in a Herbrand program with disequality guards whether a given Herbrand relation is true or not.*

We prove Theorem 7 by means of a reduction from the language-universality problem of non-deterministic finite automata (NFA), a well-known PSPACE-complete problem. Due to lack of space this proof is deferred to App B.

6 Conclusion

We presented an algorithm for testing equalities in Herbrand programs. In absence of disequality guards, our algorithm runs in polynomial time. By using positive Boolean combinations of equalities, we generalized this base algorithm to deal with programs containing disequality guards and to a test of positive Boolean combinations of equalities. We then showed that the algorithm is sufficient to find all Herbrand constants in such programs. Many challenging problems remain. First, termination of the generalized algorithm is based on well-founded orderings. We provided only a first step to a complexity analysis by establishing a PSPACE lower bound. This proof, however, did not exploit the full strength of Herbrand programs. Therefore it still leaves room for, perhaps, larger lower bounds. On the other hand, a more constructive termination proof could help to derive explicit upper complexity bounds.

Finally, we note that any validity test can be used to *infer* all valid assertions up to a given size. It is still unclear, though, how to decide whether or not there is *any* finite disjunction of equalities that holds at a given program point of a Herbrand program.

References

1. B. Alpern, M. Wegman, and F. K. Zadeck. Detecting equality of variables in programs. In *Conf. Record of the 15th ACM POPL*, Jan. 1988.
2. P. Briggs, K. D. Cooper, and L. T. Simpson. Value numbering. *Software- Practice and Experience*, 27(6):701–724, June 1997.
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. D. Duffy. *Principles of Automated Theorem Proving*. Wiley, 1991.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1978.
7. K. Gargi. A sparse algorithm for predicated global value numbering. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, pages 45–56. ACM Press, 2002.
8. S. Gulwani and G. C. Necula. Global value numbering using random interpretation. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 342–352. ACM Press, 2004.
9. S. Gulwani and G. C. Necula. A polynomial-time algorithm for global value numbering. In *Proc. Int. Static Analysis Symposium (SAS'2004),*. Springer Verlag, 2004. To appear.
10. J. B. Kam and J. D. Ullman. Monotone data fbw analysis frameworks. Technical Report 169, Department of Electrical Engineering, Princeton University, Princeton, NJ, 1975.
11. G. A. Kildall. A unifi ed approach to global program optimization. In *Conf. Record of the fi rst ACM POPL*, pages 194 – 206, Boston, MA, 1973.
12. J. Knoop, O. R¨uthing, and B. Steffen. Code motion and code placement: Just synonyms? In *Proc. 6th ESOP*, Lecture Notes in Computer Science 1381, pages 154 – 196, Lisbon, Portugal, 1998. Springer-Verlag.
13. M. M¨uller-Olm and O. R¨uthing. The complexity of constant propagation. In *10th European Symposium on Programming (ESOP)*, pages 190–205. LNCS 2028, Springer-Verlag, 2001.
14. M. M¨uller-Olm and H. Seidl. Polynomial constants are decidable. In *9th Static Analysis Symposium (SAS)*, pages 4–19. LNCS 2477, Springer-Verlag, 2002.
15. M. M¨uller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. In *Proceedings 31st POPL*, pages 330–341, 2004.
16. M. M¨uller-Olm and H. Seidl. Computing polynomial program invariants. *Information Processing Letters*, to appear.

17. M. M¨uller-Olm and H. Seidl. A note on Karr's algorithm. In *ICALP 2004*, to appear.
18. J. H. Reif and R. Lewis. Symbolic evaluation and the gobal value graph. In *Conf. Record of the 4th ACM POPL*, pages 104 – 118, Los Angeles, CA, 1977.
19. B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Global value numbers and redundant computations. In *Conf. Record of the 15th ACM POPL*, pages 12 – 27, San Diego, CA, 1988.
20. O. R¨uthing, J. Knoop, and B. Steffen. Detecting equalities of variables: Combining effi ciency with precision. In *Proc. 6th Int. Static Analysis Symposium* (*SAS'99*), Lecture Notes in Computer Science 1694, pages 232–247, Venice, Italy, 1999. Springer-Verlag.
21. B. Steffen, J. Knoop, and O. R¨uthing. The value fbw graph: A program representation for optimal program transformations. In *Proc. Third ESOP*, Lecture Notes in Computer Science 432, pages 389 – 405, Copenhagen, Denmark, 1990. Springer-Verlag.
22. B. Steffen, J. Knoop, and O. R¨uthing. Effi cient code motion and an adaption to strength reduction. In *Proc. 4th International Joint Conference on the Theory and Practice of Software Development* (*TAPSOFT*), Lecture Notes in Computer Science 494, pages 394 – 415, Brighton, UK, 1991. Springer-Verlag.

A Proof of Lemma 2

Consider a single program execution path $\pi \in \mathsf{Stmt}^*$. Define the collecting semantics $[\![\pi]\!]\, S$ of π relative to S by: $[\![\epsilon]\!]\, S = S$ and $[\![\pi' s]\!]\, S = [\![s]\!]\, ([\![\pi']\!]\, S)$. Accordingly, define the weakest precondition $[\![\pi]\!]^{\mathsf{t}}$ of ϕ along π by: $[\![\epsilon]\!]^{\mathsf{t}}\, \phi = \phi$ and $[\![\pi' s]\!]^{\mathsf{t}}\, \phi = [\![\pi']\!]^{\mathsf{t}}\, ([\![s]\!]^{\mathsf{t}}\, \phi)$.
Claim 1: For every path π, set of states S and formula ϕ, $\sigma \models \phi$ for all $\sigma \in [\![\pi]\!]\, S$ iff $\tau \models [\![\pi]\!]^{\mathsf{t}}\, \phi$ for all $\tau \in S$.
For a proof of claim 1, we proceed by induction on the length of π. Obviously, the claim is true for $\pi = \epsilon$. Otherwise, $\pi = \pi' s$ for some shorter path π' and a statement s. Define $S' = [\![\pi']\!]\, S$ and $\phi' = [\![s]\!]^{\mathsf{t}}\, \phi$. By lemma 1, $\sigma \models \phi$ for all $\sigma \in [\![s]\!]\, S'$ iff $\sigma' \models \phi'$ for all $\sigma' \in S'$. By inductive hypothesis for π' and ϕ', however, the latter statement is equivalent to $\tau \models [\![\pi']\!]^{\mathsf{t}}\, \phi'$ for all $\tau \in S$, Since by definition, $[\![s]\!]\, S' = [\![\pi]\!]\, S$ and $[\![\pi']\!]^{\mathsf{t}}\, \phi' = [\![\pi]\!]\, \phi$, the assertion follows. □
Claim 2: Let Π denote the set of paths from st to v_t. Then

1. $\mathbf{V}_S[v_t] = \bigcup\{[\![\pi]\!]\, S \mid \pi \in \Pi\}$;
2. $\mathbf{WP}[\mathsf{st}] = \bigwedge\{[\![\pi]\!]^{\mathsf{t}}\, \phi \mid \pi \in \Pi\}$.

Note that the second statement of claim 2 is in fact well-defined as $\mathbb{L}$ is a complete lattice. Claim 2 follows from Kam and Ullman's classic MOP=MFP theorem [10] since both the transfer functions $[\![s]\!]$ of the constraint system for the collecting semantics as well as the transfer functions $[\![s]\!]^{\mathsf{t}}$ of the constraint system for the weakest precondition distribute over union and conjunction, respectively. □

By the first part of claim 2, ϕ is valid at v_t from S_0 iff $\sigma \models \phi$ for all $\pi \in \Pi$, $\sigma \in [\![\pi]\!]\, S_0$. By claim 1, this is the case iff $\tau \models [\![\pi]\!]^{\mathsf{t}}\, \phi$ for all $\pi \in \Pi$, $\tau \in S_0$. By the second part of claim 2, this is true iff $\tau \models \mathbf{WP}[\mathsf{st}]$ for all $\tau \in S_0$. The latter is true iff $\phi \Rightarrow \mathbf{WP}[\mathsf{st}]$. □

B Proof of Theorem 7

As mentioned, we prove Theorem 7 by means of a polynomial-time reduction from the language-universality problem of non-deterministic finite automata (NFA). This is known to be a PSPACE-complete problem (cf. the remark to Problem AL1 in [6]). An instance of the problem is given by an NFA $\mathcal{A}$ over an alphabet Σ. The problem is to decide whether $\mathcal{A}$ accepts the universal language, i.e., whether $L(\mathcal{A}) = \Sigma^*$.

Without loss of generality, we may assume that $\Sigma = \{0, 1\}$. So suppose given an NFA $\mathcal{A} = (\Sigma, S, \delta, s_1, F)$, where $\Sigma = \{0, 1\}$ is the underlying alphabet, $S = \{s_1, \ldots, s_k\}$ is the set of states, $\delta \subseteq S \times \Sigma \times S$ is the transition relation, s_1 is the start state, and $F \subseteq S$ is the set of accepting states. From this NFA, $\mathcal{A}$, we construct a Herbrand program π which uses k variables $\mathbf{x}_1, \ldots, \mathbf{x}_k$ that correspond to the states of the automaton and another set $\mathbf{y}_1, \ldots, \mathbf{y}_k$ of auxiliary variables. These variables hold the values 0 or 1 only in executions of π. Consider first the programs π^i_σ for $\sigma \in \Sigma$, $i \in \{1, \ldots, k\}$ pictured in Fig. 5 that are used as building blocks in the construction of π. As mentioned in sect. 2, the finite disjunctions and conjunctions of disequality guards used in π^i_σ (and later in π) can be coded by simple disequality guards. It is not hard to see that the following is valid:

Lemma 7. *For each initial state, in which the variables $\mathbf{x}_1 \ldots, \mathbf{x}_k$ hold only the values* 0 *and* 1, π^i_σ *has a unique execution. This execution sets $\mathbf{y}_i$ to* 1 *if and only if $\mathbf{x}_j$ holds* 1 *for some σ-predecessor s_j of s_i. Otherwise, it sets $\mathbf{y}_i$ to* 0. □

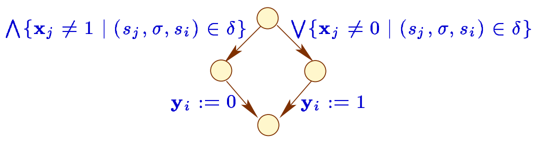

$\bigwedge\{\mathbf{x}_j \neq 1 \mid (s_j, \sigma, s_i) \in \delta\}$
$\bigvee\{\mathbf{x}_j \neq 0 \mid (s_j, \sigma, s_i) \in \delta\}$
$\mathbf{y}_i := 0$
$\mathbf{y}_i := 1$

Fig. 5. The program π^i_σ.

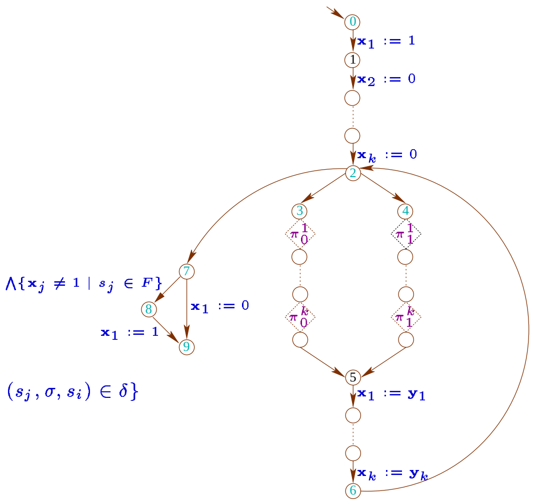


Fig. 6. The program π.

Consider now the program π shown in Fig. 6. Intuitively, each path from the initial program point 0, to the program point 2 corresponds to a word $w \in \Sigma^*$ and vice versa. Execution of the initializing assignments on the direct path from 0 to 2 corresponds to the empty word, ε. Each execution of the loop body amounts to a prolongation of the corresponding word by one letter. If the left branch is taken in the loop body (the one via program point 3) then the word is extended by the letter 0; if the right branch is taken (the one via program point 4), the word is extended by the letter 1. Let p_w be the path from program node 0 to node 2 that corresponds to the word w. We prove:

Lemma 8. *After execution of p_w variable $\mathbf{x}_i$ (for $i = 1, \ldots, k$) holds the value* 1 *if state s_i is reachable in the automaton under the word w. Otherwise, $\mathbf{x}_i$ holds* 0. □

Proof. We prove Lemma 8 by induction on the length of w.

Base Case: Under the empty word, just the initial state s_1 is reachable in $\mathcal{A}$. As the initialization sets $\mathbf{x}_1$ to 1 and the variables $\mathbf{x}_2, \ldots, \mathbf{x}_k$ to 0, the property claimed in the lemma is valid for the empty word.

Induction Step: Suppose $w = w'0$ with $w' \in \Sigma^*$; the case $w = w'1$ is similar. Let p be the cycle-free path from 2 to itself via 3. Then $p_w = p_{w'}p$.
Assume s_i is reachable under the word w in $\mathcal{A}$. Then, clearly, there is a 0-predecessor s_j of s_i in $\mathcal{A}$ that is reachable under w'. Thus, by the induction hypothesis, $\mathbf{x}_j$

holds 1 after execution of $p_{w'}$. Consider executing p. The programs $\pi_0^1, \dots, \pi_0^{i-1}$ do not change $\mathbf{x}_j$. Thus, by Lemma 7, the program π_0^i sets $\mathbf{y}_i$ to 1 and this value is copied to $\mathbf{x}_i$ in the i-th assignment after program point 5 because the programs $\pi_0^{i+1}, \dots, \pi_0^k$ do not change $\mathbf{y}_i$.
Finally, assume that s_i is not reachable under the word w in $\mathcal{A}$. Then, clearly, no σ-predecessor s_j of s_i in $\mathcal{A}$ is reachable under w'. Thus, by the induction hypothesis, for all 0-predecessors s_j of s_i, $\mathbf{x}_j$ holds 0 after execution of $p_{w'}$. The programs $\pi_0^1, \dots, \pi_0^{i-1}$ do not change these values. Thus, by Lemma 7, the program π_0^i sets $\mathbf{y}_i$ to 0 and this value is copied to $\mathbf{x}_i$ in the i-th assignment after program point 5 because the programs $\pi_0^{i+1}, \dots, \pi_0^k$ do not change $\mathbf{y}_i$. □

It is not hard to see from this property that there is an execution of π that passes the guard at the edge between the nodes 7 and 8 if and only if $L(\mathcal{A}) \neq \Sigma^*$. This implies:

Lemma 9. *The relation* $\mathbf{x}_1 = 0$ *is valid at node* 9 *of program* π *iff* $L(\mathcal{A}) = \Sigma^*$. □

Proof. We prove both directions of the equivalence claimed in Lemma 9 separately:

"⇒": The proof is by contraposition. Assume $L(\mathcal{A}) \neq \Sigma^*$. Let $w \in \Sigma^*$ such that $w \notin L(\mathcal{A})$. This implies that no state $s_j \in F$ is reachable in $\mathcal{A}$ under w. Therefore, after executing p_w all variables $\mathbf{x}_j$ with $s_j \in F$ hold 0 by Lemma 8 such that the condition $\bigwedge\{\mathbf{x}_j \neq 1 \mid s_j \in F\}$ is satisfied. Hence, we can proceed this execution via the nodes 7, 8, and 9. After this execution, however, $\mathbf{x}_1$ holds 1 such that the relation $\mathbf{x}_1 = 0$ is invalidated.
"⇐": Assume $L(\mathcal{A}) = \Sigma^*$. Then after any execution from the initial program node 0 to node 2 one of the variables $\mathbf{x}_j$ with $s_j \in F$ holds the value 1 because the word corresponding to this execution is accepted by $\mathcal{A}$. Therefore, the path $2, 7, 8, 9$ is not executable, such that $\mathbf{x}_1$ is set of 0 whenever 9 is reached. Therefore, the relation $\mathbf{x}_1 = 0$ is valid at program point 9. □

Note that our PSPACE-hardness proof does not use the full power of Herbrand programs and Herbrand relations. We just use constant assignments of the form $\mathbf{x} := 0$ and $\mathbf{x} := 1$, copying assignments of the form $\mathbf{x} := \mathbf{y}$, and disequality guards of the form $\mathbf{x} \neq 0$ and $\mathbf{x} \neq 1$, where 0 and 1 are two different constants. Moreover, we just need to check whether a relation of the form $\mathbf{x} = 0$ is valid at a given program point.