Finite Instantiations in Equivalence Logic with Uninterpreted Functions

Yoav Rodeh and Ofer Shtrichman

Weizmann Institute of Science, Rehovot, Israel
IBM Haifa Research Laboratory
{yrodeh | ofers}@wisdom.weizmann.ac.il

Abstract. We introduce a decision procedure for satisfiability of equivalence logic formulas with uninterpreted functions and predicates. In a previous work ([PRSS99]) we presented a decision procedure for this problem which started by reducing the formula into a formula in equality logic. As a second step, the formula structure was analyzed in order to derive a small range of values for each variable that is sufficient for preserving the formula's satisfiability. Then, a standard BDD based tool was used in order to check the formula under the new small domain. In this paper we change the reduction method and perform a more careful analysis of the formula, which results in significantly smaller domains. Both theoretical and experimental results show that the new method is superior to the previous one and to the method suggested in [BGV99].

1 Introduction

Deciding equivalence between formulas with uninterpreted functions is of major importance due to the broad use of uninterpreted functions in abstraction. Such abstraction can be used, for example, when checking a control property of a microprocessor, and it is sufficient to specify that the operations which the ALU performs are functions, rather than specifying what these operations are. Thus, by representing the ALU as an uninterpreted function, the verification process avoids the complexity of the ALU. This is the approach taken, for example, in [BD94], where a formula with uninterpreted functions is generated, such that its validity implies the equivalence between the CPU checked and another version of it, without a pipeline. Another example is given in [PSS98], where formulas with uninterpreted functions are used for *translation validation*, a process in which the correct translation of a compiler is verified by proving the equivalence between the source and target codes after each run.

In the past few years several different BDD-based procedures for checking satisfiability of such formulas have been suggested (in contrast to earlier decision procedures that are based on computing congruence closure [BDL96] in combination with case splitting). Typically the first step of these procedures is the reduction of the original formula φ to an equality formula (a propositional formula plus the equality sign) ψ such that ψ is satisfiable iff φ is. As a second step, different procedures can be used for checking ψ.

Goel et al. suggest in [GSZAS98] to replace all comparisons in ψ with new Boolean variables, and thus create a new Boolean formula ψ'. The BDD of ψ' is calculated ignoring the transitivity constraints of comparisons. They then traverse the BDD, searching for a satisfying assignment that will also satisfy these constraints. Bryant et al. at [BV00] suggested to avoid this potentially exponential traversing algorithm by explicitly computing a small set of constraints that are sufficient for preserving the transitivity constraints of equality. By checking ψ' conjoined with these constraints using a regular BDD package they were able to verify larger designs.

In [PRSS99] we suggested a method in which the Ackermann reduction scheme [Ack54] is used to derive ψ, and then ψ's satisfiability is decided by assigning a small domain for each variable, such that ψ is satisfiable if and only if it is satisfiable under this small domain. To find this domain, the equalities in the formula are represented as a graph, where the nodes are the variables and the edges are the equalities and disequalities (*disequality* standing for $\neq$) in ψ. Given this graph, a heuristic called *range allocation* is used in order to compute a small set of values for each variable. To complete the process, a standard BDD based tool is used to check satisfiability of the formula under the computed domain.

While both [PRSS99] and [GSZAS98] methods can be applied to any equality formula, Bryant et al. suggest in [BGV99] to examine the structure of the original formula φ. They prove that if the original formula φ uses comparisons between variables and functions only in a certain syntactically restricted way (denoted *positive equality*), the domain of the reduced formula can be restricted to a unique single constant for each variable. This result can also be applied for only subsets of variables (and functions) in the formula that satisfy this condition. However, this result cannot be obtained using Ackermann's reduction. Rather they use the reduction proposed in [BV98].

The method which we propose in this paper roughly uses the framework we suggested in [PRSS99]. We will use the reduction scheme suggested in [BV98] (rather than Ackermann's scheme) in order to generalize their result in the case of positive equality formulas. We also show how this shift, together with a more careful analysis of the formula structure, allows for a construction of a different graph, which results in a provably smaller domain. The smaller implied state space is crucial, as our experiments have shown, for reducing the verification time of these formulas.

2 Preliminaries and Definitions

We define the logic of equality with uninterpreted functions formally. The syntax of this logic is defined as follows:

$$
\begin{aligned}
\langle Formula \rangle \longleftarrow\ & \langle Boolean\text{-}Variable \rangle \mid \\
& \langle Predicate\text{-}Symbol \rangle(\langle Term \rangle, \ldots, \langle Term \rangle) \mid \\
& \langle Term \rangle = \langle Term \rangle \mid \neg\langle Formula \rangle \mid \langle Formula \rangle \vee \langle Formula \rangle \\
\\
\langle Term \rangle \longleftarrow\ & \langle Term\text{-}Variable \rangle \mid \\
& \langle Function\text{-}Symbol \rangle(\langle Term \rangle, \ldots, \langle Term \rangle) \mid \\
& \mathbf{ITE}(\langle Formula \rangle, \langle Term \rangle, \langle Term \rangle)
\end{aligned}
$$

We refer to formulas in this logic as UF-formulas. We say that a UF-formula φ is satisfiable iff there is some interpretation $\mathcal{M}$ of the variables, functions and predicates of φ, such that $\mathcal{M} \models \varphi$.

An equivalence logic formula (denoted E-formula) is a UF-formula that does not contain any function and predicate symbols. Throughout the paper we use φ and ψ to denote UF-formulas and E-formulas, respectively.

We allow our formulas to contain *let* constructs of the form *let* $X = \psi$ in $\varphi(X)$, which allows *term sharing* or the representation of circuits.

For simplicity of presentation, we will treat UF-formulas with no Boolean variables and predicates. Also, we will assume there are no **ITE** terms, and every uninterpreted function has just one argument. All these extensions, including the full proofs and examples, are handled in the full version of the paper [PRS01].

3 Deciding Satisfiability of E-formulas

We wish to check the satisfiability of an E-formula ψ with variables V. In theory this implies that we need to check whether there exist some instantiation of V that satisfies ψ. Since ψ only queries equalities on the variables in V, it enjoys the *small model property*, which means that it is satisfiable iff it is satisfiable over a finite domain. It is not hard to see that the finite domain implied by letting each variable in V the range over $\{1 \ldots |V|\}$ is sufficient. However, this approach is not very practical, since it leads to a state space of $|V|^{|V|}$.

In [PRSS99] we suggested a more refined analysis, where rather than considering only $|V|$, we examine the actual structure of ψ, i.e. the equalities and disequalities in ψ. This analysis enables the derivation of a state space which is empirically much smaller than $|V|^{|V|}$. In this section we repeat the essential definitions from this work, except for several changes which are necessary for the new techniques that will be presented in later sections.

Definition 1. (E-graphs): *An E-graph $\mathcal{G}$ is a triplet $\mathcal{G} = \langle V, EQ, DQ \rangle$, where V is the set of vertices, and EQ (Equality edges) and DQ (Disequality edges) are sets of unordered pairs of vertices.*

Given an E-graph $\mathcal{G} = \langle V, EQ, DQ \rangle$, we let $V(\mathcal{G}) = V$, $DQ(\mathcal{G}) = DQ$ and $EQ(\mathcal{G}) = EQ$. We use $\leq$ to denote the sub-graph relation: $\mathcal{H} \leq \mathcal{G}$ iff $V(\mathcal{H}) = V(\mathcal{G})$, $EQ(\mathcal{H}) \subseteq EQ(\mathcal{G})$ and $DQ(\mathcal{H}) \subseteq DQ(\mathcal{G})$. We will use E-graphs to represent partial information derived from the structure of a given E-formula; they can be viewed as a conservative abstraction of E-formulas.

We say that an assignment α (assigning values to the variables in V) satisfies edge (a, b) if (a, b) is an equality edge and $\alpha(a) = \alpha(b)$, or if (a, b) is a disequality edge and $\alpha(a) \neq \alpha(b)$. We write $\alpha \models \mathcal{G}$ if α satisfies all edges of $\mathcal{G}$. $\mathcal{G}$ is said to be satisfiable if there exists some α such that $\alpha \models \mathcal{G}$.

Construction of E-graph $\mathcal{G}(\psi)$: For an E-formula ψ we construct the E-graph $\mathcal{G}(\psi)$ (this is a construction suggested in [PRSS99]) by placing a node in $\mathcal{G}(\psi)$ for each variable of ψ, and a (dis)equality edge for each (dis)equality term of ψ — by "equality" term we mean that the equality term appears under an even number of negations, and by "disequality", under an odd number.

Example 1. The E-formula $\psi_1 = (a = b) \wedge (\neg(c = b) \vee (a = c))$, results in the E-graph:

$$\mathcal{G}(\psi_1) = \langle \{a, b, c\}, \{(a, b), (a, c)\}, \{(c, b)\} \rangle$$

Notice that every proper subgraph of $\mathcal{G}(\psi_1)$ is satisfiable.

The important property of $\mathcal{G}(\psi)$ is that any two assignments α_1 and α_2 that satisfy exactly the same edges of $\mathcal{G}(\psi)$, will give the same result for ψ; i.e., $\alpha_1 \models \psi$ iff $\alpha_2 \models \psi$. This means that if ψ is satisfiable, then there is some satisfiable $\mathcal{H} \leq \mathcal{G}(\psi)$ such that every assignment that satisfies all edges of $\mathcal{H}$ will satisfy ψ (this $\mathcal{H}$ consists of all the edges of $\mathcal{G}(\psi)$ that are satisfied by ψ's satisfying assignment). We wish to generalize this property of $\mathcal{G}(\psi)$.

Definition 2. (adequacy of E-graphs to E-formulas): *An E-graph $\mathcal{G}$ is adequate for E-formula ψ, if either ψ is not satisfiable, or there exists a satisfiable $\mathcal{H} \leq \mathcal{G}$ such that for every assignment α such that $\alpha \models \mathcal{H}$, $\alpha \models \psi$.*

For example, $\mathcal{G}(\psi)$ is adequate for ψ. We use the fact that an E-graph is adequate for ψ for finding a small set of assignments that will be sufficient for checking ψ:

Definition 3. (adequacy of assignment sets to E-graphs): *Given an E-graph $\mathcal{G}$, and R, a set of assignments to $V(\mathcal{G})$, we say that R is adequate for $\mathcal{G}$ if for every satisfiable $\mathcal{H} \leq \mathcal{G}$, there is an assignment $\alpha \in R$ such that $\alpha \models \mathcal{H}$.*

Proposition 1. *If E-graph $\mathcal{G}$ is adequate for ψ, and assignment set R is adequate for $\mathcal{G}$, then ψ is satisfiable iff there is $\alpha \in R$ such that $\alpha \models \psi$.*

Example 2. For our E-formula ψ_1 of Example 1, the following set is adequate for $\mathcal{G}(\psi_1)$:

$$R = \{(a \leftarrow 0, b \leftarrow 0, c \leftarrow 0), (a \leftarrow 0, b \leftarrow 0, c \leftarrow 1), (a \leftarrow 0, b \leftarrow 1, c \leftarrow 0)\}$$

Indeed, the assignment $(a \leftarrow 0, b \leftarrow 0, c \leftarrow 0) \in R$, satisfies ψ_1.

The range allocation procedure of [PRSS99] calculates an adequate assignment set R for a given input E-graph $\mathcal{G}$. In that procedure, the resulting R has an extra property: every $\alpha \in R$ is *diverse* w.r.t. $\mathcal{G}$. By this we mean that for every $u, v \in V(\mathcal{G})$, if u and v are not connected via equality edges in $\mathcal{G}$, then $\alpha(u) \neq \alpha(v)$. In [PRS01] we show how to alter any range allocator so that its output assignment set will be diverse w.r.t. the input E-graph (while retaining adequacy), without increasing the assignment set size. In light of this, we alter Definition 2 and Definition 3, by considering only assignments that are diverse w.r.t. to $\mathcal{G}$ (replace "assignment" by "assignment that is diverse w.r.t. $\mathcal{G}$" in both these definitions). This leaves Proposition 1 true, does not cause an increase in the size of the possible adequate assignment sets (as we just commented), and makes it easier for us to find an adequate E-graph for a given E-formula.

We will now rephrase the decision procedure for the satisfiability of UF-formulas as suggested in [PRSS99] according to the above definitions:

1. Reduce UF-formula φ to E-formula ψ using Ackermann's reduction.
2. Calculate the E-graph $\mathcal{G}(\psi)$.
3. Calculate an adequate set of assignments R for $\mathcal{G}(\psi)$.
4. Check if any of the assignments in R satisfies ψ. (This step is done symbolically, not by exhaustive search of R).

In this paper we alter Steps 1 and 2 of this procedure by replacing the reduction scheme, and by calculating a different adequate E-graph for ψ. We will later show that these changes guarantee smaller state spaces and thus a more efficient procedure.

4 Bryant et al. Reduction Method

We will denote this type of reduction of a UF-formula φ to an E-formula ψ by $T^{BV}(\varphi)$. The main property of $T^{BV}(\varphi)$ is that it is satisfiable iff φ is satisfiable. The formula $T^{BV}(\varphi)$ is given by replacing for all i, the function application F_i in φ by a new term $F_i^\star$. We explain the reduction using an example (see [PRS01] or [BV98] for details):

Example 3. Consider the following formula:

$$\varphi_1 := [F(F(F(y))) \neq F(F(y))] \wedge [F(F(y)) \neq F(x)] \wedge [x = F(y)]$$

We number the function applications such that applications with syntactically equivalent arguments are given the same index number:

$$\varphi_1 := [F_4(F_3(F_1(y))) \neq F_3(F_1(y))] \wedge [F_3(F_1(y)) \neq F_2(x)] \wedge [x = F_1(y)]$$

$T^{BV}(\varphi_1)$ is given by:

$$T^{BV}(\varphi_1) := (F_4^\star \neq F_3^\star) \wedge (F_3^\star \neq F_2^\star) \wedge (x = F_1^\star)$$

$$F_1^\star := f_1 \qquad\qquad F_2^\star := \begin{cases} f_1 & x = y; \\ f_2 & \text{Otherwise}; \end{cases}$$

$$F_4^\star := \begin{cases} f_1 & F_3^\star = y; \\ f_2 & F_3^\star = x; \\ f_3 & F_3^\star = F_1^\star; \\ f_4 & \text{Otherwise}; \end{cases} \qquad F_3^\star := \begin{cases} f_1 & F_1^\star = y; \\ f_2 & F_1^\star = x; \\ f_3 & \text{Otherwise}; \end{cases}$$

The general idea is that for every function application F_j of φ we define a new variable f_j which is the "basic" value of F_j. This means that $F_j^\star = f_j$ if no smaller (index wise) function application "overrides" f_j. This can happen, when there is some $i < j$ such that the argument of F_i and F_j are equal. In this case, for the minimal such i, we have $F_j^\star = f_i$.

In comparison, Ackermann's reduction for φ_1 is given by $T^A(\varphi_1)$:

$$\begin{bmatrix} (y = x \ \rightarrow f_1 = f_2) \wedge (y = f_1 \ \rightarrow f_1 = f_3) \wedge \\ (y = f_3 \rightarrow f_1 = f_4) \wedge (x = f_1 \ \rightarrow f_2 = f_3) \wedge \\ (x = f_3 \rightarrow f_2 = f_4) \wedge (f_1 = f_3 \rightarrow f_3 = f_4) \end{bmatrix} \wedge (f_4 \neq f_3) \wedge (f_3 \neq f_2) \wedge (x = f_1)$$

A hint to why Bryant's reduction is better for our purposes is the following claim:

Claim. For every UF-formula φ, if $\alpha \models T^A(\varphi)$ then $\alpha \models T^{BV}(\varphi)$.

While the converse does not hold. Thus, $T^{BV}(\varphi)$ has more satisfying assignments and therefore it should be easier to satisfy.

5 New E-Graph Construction

Given a UF-formula φ, we wish to construct a minimal E-graph that will be adequate for $T^{BV}(\varphi)$. We will first try to disregard all function arguments. Denote by $simp(\varphi)$ the E-formula received by replacing every function application F_i by its corresponding variable f_i. For example, for φ_1 of Section 4, $simp(\varphi_1) = ((f_4 \neq f_3) \wedge (f_3 \neq f_2) \wedge (x = f_1))$. Our initial E-graph will therefore be $\mathcal{G}(simp(\varphi))$.

If we take for example $\varphi_2 = F_1(x) \neq F_2(y)$, then $simp(\varphi_2) = f_1 \neq f_2$. $\mathcal{G}(simp(\varphi_2))$ then contains just one disequality edge between f_1 and f_2. An adequate assignment set for $\mathcal{G}(simp(\varphi_2))$, must contain an assignment α that assigns a different value for every variable in the E-graph, since α should be diverse w.r.t. to $\mathcal{G}(simp(\varphi_2))$. For example: $\alpha(f_1) = 0, \alpha(f_2) = 1, \alpha(x) = 2, \alpha(y) = 3$. Since $T^{BV}(\varphi_2) = f_1 \neq \mathbf{ITE}(x = y, f_1, f_2)$, we get that $\alpha \models T^{BV}(\varphi_2)$. And so we found an assignment that satisfies the formula.

Assume however, that our formula is slightly different: $\varphi_3 = F_1(x) \neq F_2(y) \wedge ((x = y) \vee True)$[1]. In this case $simp(\varphi_3) = f_1 \neq f_2 \wedge ((x = y) \vee True)$. Now, $\mathcal{G}(simp(\varphi_3))$ will also contain an equality edge between x and y. In this case, a possible adequate assignment set for this E-graph contains just one assignment

[1] Of course, any decent procedure will remove the right clause, but this $True$ can be hidden as a more complex valid formula.

α: $\alpha(f_1) = 0, \alpha(f_2) = 1, \alpha(x) = \alpha(y) = 2$. In this case however, $\alpha \not\models T^{BV}(\varphi_3)$. This is because the equality edge we added, indirectly caused the disequality edge between f_1 and f_2 to be disregarded. We will therefore add a rule to augment our E-graph with more edges in this case:

Tentative Rule 1. *If there is a disequality edge between f_i and f_j, add a disequality edge between their corresponding arguments.*

But this rule is not enough. We consider the following formula:

$$\varphi_4 = (F_1(x) = z) \wedge (F_2(y) \neq z) \wedge ((x = y) \vee True)$$

$\mathcal{G}(simp(\varphi_4))$ appears in Figure 1 as $\mathcal{G}_1$. In this case, the above Tentative Rule 1 does not apply, and we are left with the same problem, since a possible adequate assignment set for this E-graph contains just one assignment α: $\alpha(f_1) = \alpha(z) = 0, \alpha(f_2) = 1, \alpha(x) = \alpha(y) = 2$, and α does not satisfy $T^{BV}(\varphi_4)$. This is because a disequality edge between f_1 and f_2 is only implied in this E-graph, and so we wish to change Tentative Rule 1 so that it identifies implied disequality requirements.

We write $u \asymp_{\mathcal{G}} v$ if there exists a simple path between u and v in $\mathcal{G}$ consisting of equality edges except for exactly one disequality edge. This is what we mean by "implied" disequality edge. What this means is that an assignment where u and v differ may be needed to satisfy the formula. We alter Tentative Rule 1:

Rule 1. *If for f_i and f_j, $f_i \asymp_{\mathcal{G}} f_j$ then add a disequality edge between their corresponding arguments.*

We now consider a similar UF-formula:

$$\varphi_5 = (True \vee (F_1(x) = z)) \wedge (F_2(y) \neq z) \wedge (x = y)$$

$\mathcal{G}(simp(\varphi_5))$ is exactly the same as before, and Rule 1 adds the disequality edge (x, y) to give $\mathcal{G}_2$ in Figure 1. The problem here is that a satisfying assignment α must satisfy $\alpha(x) = \alpha(y)$, and therefore $\alpha(F_2^\star) = \alpha(f_1)$. Since we also must have $\alpha(F_2^\star) \neq \alpha(z)$ to satisfy the formula, it implies $\alpha(f_1) \neq \alpha(z)$. This may not necessarily happen in any assignment given by the range allocator for our E-graph. This is because in our E-graph there is no representation for the fact that f_1 may "override" f_2. If we add an equality edge between f_1 and f_2 it will solve the problem. $\mathcal{G}_3$ of Figure 1 is the result of adding this edge.

We denote by $u \approx_{\mathcal{G}} v$ the case where there is an equality path between u and v in $\mathcal{G}$.

Tentative Rule 2. *For f_i and f_j, with x_i and x_j their corresponding arguments, if $x_i \approx_{\mathcal{G}} x_j$ then add the equality edge (f_i, f_j).*

This indeed solves our problem, but is not the best we can do. We have added an equality edge between f_1 and f_2 in our example, but it was not really necessary. We could have instead copied all edges involving f_2 to f_1. This is because there is no need for f_1 to be equal to f_2 if their arguments are equal. All that is needed is that the value f_1 gets respects all the requirements of f_2. Notice that this case is asymmetric: since f_1 may override f_2, only f_1 is required to answer to f_2's requirements.

We change Tentative Rule 2 to the following rule:

Rule 2. *For f_i and f_j, where $i < j$, with x_i and x_j their corresponding arguments, if $x_i \approx_{\mathcal{G}} x_j$ then do one of the following:*

1. *add equality edge (f_i, f_j), or*
2. *for every (dis)equality edge (f_j, w) add a (dis)equality edge (f_i, w).*

And so, in our example, instead of adding an equality edge (f_1, f_2), we add a disequality edge (f_1, z) — see $\mathcal{G}_4$ of Figure 1.

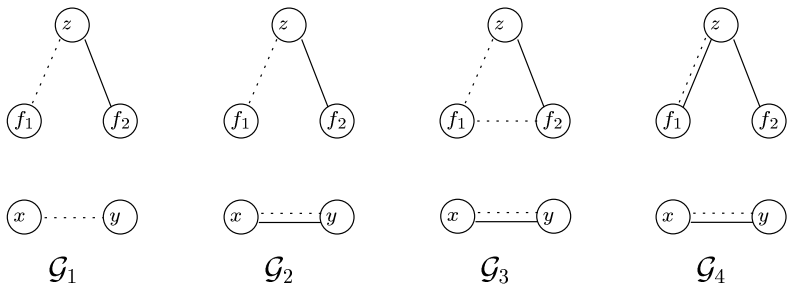

Fig. 1. The iterative E-graph construction process. Dashed lines represent equality edges, solid lines represent disequality edges.

The general idea of our new construction is therefore to start with $\mathcal{G}(simp(\varphi))$, and then apply Rule 1 and Rule 2 until no new edges are added. There are some missing details, specifically, the second option of Rule 2 needs to be postponed until the whole E-graph is constructed. We show the exact E-graph construction in the next section. Notice that this construction has a cone-of-influence flavor, since in $simp(\varphi)$ the arguments of uninterpreted functions disappear, and then only edges emanating from edges already in the E-graph are added.

6 Formal Description of E-graph Construction

We define an A-graph (marked by $\mathfrak{G}$) to be an E-graph with the addition of *assignment edges*, which are directed. For an A-graph $\mathfrak{G}$ denote by $flat(\mathfrak{G})$ the E-graph resulting from replacing every assignment edge of $\mathfrak{G}$ by an equality edge.

For function application F_i of φ, define $arg(F_i)$ to be the variable of $T^{BV}(\varphi)$ corresponding to the argument of F_i. This means that if the argument of F_i is a variable v, then $arg(F_i) = v$, and if it is a function application G_j, then $arg(F_i) = g_j$.

The E-graph construction procedure is divided to two parts:

1. **A-graph construction:** Given a UF-formula φ we construct an A-graph $\mathfrak{G}$:
   (a) Let the vertices of $\mathfrak{G}$ be the variables of $T^{BV}(\varphi)$.
   (b) Add all edges of $\mathcal{G}(simp(\varphi))$ to $\mathfrak{G}$.

(c) For every F_i and F_j such that $i < j$ and $arg(F_i) \approx_{flat(\mathfrak{G})} arg(F_j)$, add the following edges:
   i. Add assignment edge (f_i, f_j) to $\mathfrak{G}$.
   ii. If $f_i \asymp_{flat(\mathfrak{G})} f_j$ then add disequality edge $(arg(f_i), arg(f_j))$ to $\mathfrak{G}$.

(d) Repeat step 1c until a no new edges are added.

Example 4. For the UF-formula φ_1 of Example 3, the algorithm constructs the A-graph $\mathfrak{G}$ of Figure 2, while $\mathcal{G}$ is the E-graph constructed by the procedure suggested in [PRSS99].

2. **Transforming the A-graph to an E-graph:** The second step of the procedure is to transform the A-graph $\mathfrak{G}$ to an E-graph $\mathcal{G}$. For two vertices u and v, we denote $v \sqsubseteq_{\mathcal{G}} u$, if:
   (a) for every $(v, w) \in EQ(\mathcal{G})$, $(u, w) \in EQ(\mathcal{G})$.
   (b) for every $(v, w) \in DQ(\mathcal{G})$, $(u, w) \in DQ(\mathcal{G})$.
   We proceed:
   (a) Initially, $\mathcal{G} = \langle V(\mathfrak{G}), EQ(\mathfrak{G}), DQ(\mathfrak{G})\rangle$
   (b) While there are vertices u, v, such that (u, v) is an assignment edge of $\mathfrak{G}$, and either $(u, v) \notin EQ(\mathcal{G})$ or $v \sqsubseteq_{\mathcal{G}} u$, choose one of the following options:
      i. add edge (u, v) to $EQ(\mathcal{G})$.
      ii. A. for every $(v, w) \in EQ(\mathcal{G})$ add (u, w) to $EQ(\mathcal{G})$.
          B. for every $(v, w) \in DQ(\mathcal{G})$ add (u, w) to $DQ(\mathcal{G})$.

Theorem 1. *If E-graph $\mathcal{G}$ is constructed by the above procedure run on UF-formula φ, then $\mathcal{G}$ is adequate for φ.*

Note that the Part 2 of the procedure requires a choice between two options. In our implementation we choose greedily between the two options, choosing the one which minimizes the number of equality edges added to $\mathcal{G}$.

Example 5. $\mathcal{G}_1$ and $\mathcal{G}_2$ in Figure 2 are the two possible E-graphs resulting from applying this Part 2 to $\mathfrak{G}$. As we can see both $\mathcal{G}_1$ and $\mathcal{G}_2$ are much smaller than $\mathcal{G}$ (the E-graph constructed by [PRSS99]). In fact, we can show that any adequate assignment set for $\mathcal{G}$ is of size at least 16, and on the other hand, there is an assignment set of size 4 for $\mathcal{G}_1$, and of size 2 for $\mathcal{G}_2$.

7 Comparison with Previous Methods

If we examine the E-graph construction of [PRSS99], we see that it is basically the same as this new construction, except there is no conditioning on when to add new edges, instead, they are always added. In other words, remove all conditions of Step 1c in Part 1 of the procedure, and for every F_i and F_j add a disequality edge between their arguments, and an equality edge between f_i and f_j. Therefore, our E-graph will always be smaller than in [PRSS99], resulting in a smaller state space.

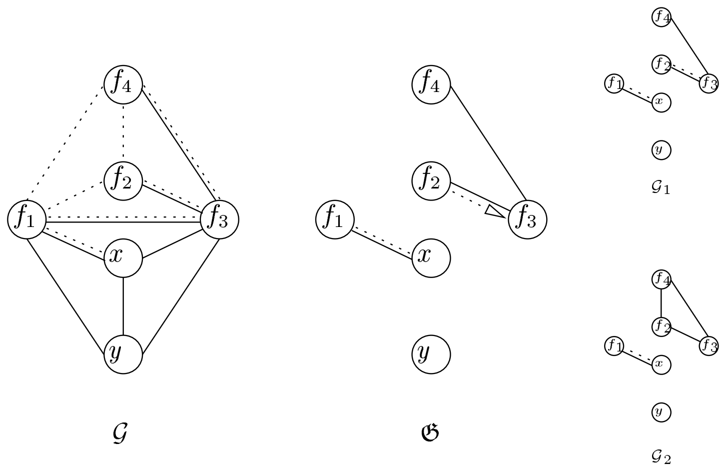

Fig. 2. Dashed lines represent equality edges, solid lines represent disequality edges, and dashed directed lines represent assignment edges.

In [BGV99], it is proved that for a UF-formula φ in *positive equality*, every variable of $T^{BV}(\varphi)$ can be instantiated to a single constant. A UF-formula φ is said to be of *positive equality* if no equality terms of φ are in the input cone of a function application, and all equality terms of φ appear under an odd number of negations — they are in negative polarity[2]. It is easy to see that our A-graph construction for such formulas will result in an A-graph with no equality edges. Then, if we use our greedy heuristic for the Part 2 of the procedure, it will result in an E-graph consisting of only disequality edges. An adequate range for such an E-graph contains just one assignment, assigning each variable a distinct constant. We therefore achieve this optimal result for the positive equality segment of the formula, while improving on the other variables (since they give a range of $1 \ldots i$ to the i-th variable, resulting in a state space of $n!$, which we will always improve upon — see [PRSS99]).

8 Experimental Results and Conclusions

We implemented our new graph construction procedure, and then used the range allocator of [PRSS99] to construct a new procedure for checking satisfiability of UF-formulas. We compared our decision procedure with that of [PRSS99] on many example formulas that were generated by a tool for compiler translation validation [PSS98]. The results appear in Table 1, where the prefix *New* denotes the results of this paper, and the prefix *Old* the results of [PRSS99]. *space* denotes

[2] The confusion between 'positive' equality and 'negative' polarity is due to the fact that in [BGV99], where this term was introduced, the analysis referred to validity checking, rather than satisfiability as in this paper.

the resulting assignment set size. Since in all cases encountered the verification procedure either proved that the formula valid in less than 1 sec, or ran out of memory, we do not write the exact running time. Instead we write $\surd$ if the run completed, and $\times$ if it didn't. *Num. vars* denotes the number of variables in the example. There were many examples were both methods resulted in a very small state space (and running time), and therefore we mention only those were there was a significant difference between the two methods.

Table 1. New vs. Old E-graph construction.

Example	New-finished	Old-finished	New-space	Old-space	Num. vars
15	$\surd$	$\times$	121	121	13
22	$\surd$	$\times$	2	$9.8 \cdot 10^{46}$	114
25	$\surd$	$\times$	1	$5.9 \cdot 10^{47}$	114
27	$\surd$	$\surd$	2	11520	26
43	$\surd$	$\times$	4	$3.4 \cdot 10^{108}$	160
44	$\surd$	$\surd$	4	$2.5 \cdot 10^{11}$	46
46	$\surd$	$\surd$	2	$1.6 \cdot 10^{22}$	67
47	$\surd$	$\surd$	1	$4.9 \cdot 10^{9}$	52

As can be seen from the table, the new graph construction has an extreme effect on the state space size. Indeed, by using the new graph construction we were able to verify formulas which we could not with the previous method.

To conclude, we showed that the combination of Bryant et al. reduction method, Pnueli et al. range allocation, and a more careful analysis of the formula structure are very effective for verifying equality formulas with uninterpreted functions.

References

[Ack54] W. Ackermann, "Solvable Cases of the Decision Problem", Studies in logic and the foundations of mathematics, North-Holland, Amsterdam, 1954.

[BD94] J.R. Burch and D.L. Dill, "Automatic Verification of Microprocessor Control", In *Computer-Aided Verification CAV '94* .

[BDL96] Clark W. Barrett, David L. Dill and Jeremy R. Levitt, "Validity Checking for Combinations of Theories with Equality", In *Formal Methods in Computer Aided Design FMCAD '96* .

[GSZAS98] A. Goel, K. Sajid, H. Zhou, A. Aziz and V. Singhal, "BDD Based Procedures for a Theory of Equality with Uninterpreted Functions", In *Computer-Aided Verification CAV '98* .

[HIKB96] R. Hojati, A. Isles, D. Kirkpatrick and R. K. Brayton, "Verification Using Finite Instantiations and Uninterpreted Functions", In *Formal Methods in Computer Aided Design FMCAD '96* .

[PRSS99] A. Pnueli, Y. Rodeh, M. Siegel and O. Shtrichman, "Deciding Equality Formulas by Small Domain Instantiations", In *Computer-Aided Verification CAV '99* .

[PSS98] A. Pnueli, M. Siegel and O. Shtrichman, "Translation Validation for Synchronous Languages", In *International Colloquium on Automata, Languages and Programming ICALP '98* .

[PRS01] A. Pnueli, Y. Rodeh and O. Shtrichman, "Finite Instantiations in Equivalence Logic with Uninterpreted Functions", Technical report, Weizmann Institute of Science, 2001. `www.wisdom.weizmann.ac.il/~verify/publication/2001/yrodeh_tr2001.ps.gz`

[BV98] R. E. Bryant and M. Velev, "Bit-level Abstraction in the Verification of Pipelined Microprocessors by Correspondence Checking", In *Formal Methods in Computer Aided Design FMCAD '98* .

[BGV99] R. E. Bryant, S. German and M. N. Velev, "Exploiting Positive Equality in a Logic of Equality with Uninterpreted Functions", In *Computer-Aided Verification CAV '99* .

[BV00] R. E. Bryant and M. N. Velev, "Boolean satisfiability with transitivity constraints", In *Computer-Aided Verification CAV 2000* .