Finite Instantiations in Equivalence Logic with Uninterpreted Functions

Amir Pnueli[1], Yoav Rodeh[1,2], and Ofer Shtrichman[1,2]

[1] Weizmann Institute of Science, Rehovot, Israel
[2] IBM Haifa Research Laboratory
{amir | 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 ([PRSS98]) we presented a decision procedure for this problem which starts by reducing the formula to 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 other methods that were suggested in the last few years.

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 perform 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 procedures for checking satisfiability of such formulas have been suggested. Typically the first step of these procedures is the reduction of the original formula φ into a formula in equivalence logic ψ such that ψ is satisfiable iff φ is. This can be performed, for example, by using the Ackermann's reduction scheme [Ack54]. Ackermann suggested replacing each uninterpreted function instance with a new variable, and for every pair of function instances add a constraint to ψ that enforces functional consistency: if the arguments of the two function instances are equal, then the variables that replaced them are also equal. For example the formula $\varphi: \ f(x) \neq f(y)$ is reduced to the equality formula $\psi: \ (x = y \rightarrow f_1 = f_2) \wedge f_1 \neq f_2$. As a second step, different procedures can be used for checking ψ.

Goel et at. 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' conjuncted with these constraints with a regular BDD package they were able to verify larger designs.

In [PRSS98] we suggested a method in which the Ackermann reduction scheme 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. In [PRS01] we introduce new methods for range allocation given such graphs.

While both [PRSS98] 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 (called *positive equality*), the domain of the reduced formula can be restricted to a unique single constant for each variable. However, this result cannot be obtained using Ackermann's reduction. Rather they use a different reduction method as proposed in [BV98]. In section 2.2 we will describe in more detail the differences between the two reduction methods.

The method which we propose in this paper roughly uses the framework we suggested in [PRSS98]. We will use the reduction scheme suggested in [BV98] (rather than Ackermann's scheme) in order to generalize their result (in [BGV99]) in the case of positive equality formulas (we will explain this term in the next section). 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.

In the next section we give basic definitions and concepts. We then describe a simple version of the graph construction in section 3, and gradually generalize the analysis in the subsequent sections. Some experimental results are presented in section 9

2 Preliminaries and Definitions

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

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

$$
\begin{aligned}
\langle Term \rangle \quad \longleftarrow\; & \langle Term\text{-}Variable \rangle \\
& \mid \langle Function\text{-}Symbol \rangle(\langle Term \rangle, \ldots, \langle Term \rangle) \\
& \mid \mathbf{ITE}(\langle Formula \rangle, \langle Term \rangle, \langle Term \rangle)
\end{aligned}
$$

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

An equivalence logic formula, denoted as 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 will assume that formulas may contain internal variables (equivalently, term-sharing is allowed).

We will first assume that φ does not contain Boolean variables and predicates. In Section 8 we will explain how these are handled.

2.1 Deciding Satisfiability of E-formulas

We wish to check the satisfiability of an E-formula ψ with term-variable set V. In theory this implies that we need to check whether there exist some instantiation of the term-variables of V that satisfies ψ. This is an infinite set of assignments, but since ψ only queries equalities on the term-variables, it enjoys the *small model property*, which means that it is satisfiable if and only if it is satisfiable under 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 [PRSS98] 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 we will present in later sections.

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

Given an E-graph $\mathcal{G} = \langle V, Eq, Nq \rangle$, we denote $V(\mathcal{G}) = V$, $Nq(\mathcal{G}) = Nq$ and $Eq(\mathcal{G}) = Eq$. We use $<$ to denote the dub-graph relation: $\mathcal{H} < \mathcal{G}$ iff $Eq(\mathcal{H}) \subseteq Eq(\mathcal{G})$ and $Nq(\mathcal{H}) \subseteq Eq(\mathcal{G})$. We will use E-graphs to represent information we derive from the structure of a given E-formula, and they will therefore serve to represent a conservative abstraction of E-formulas.

We say that an assignment α satisfies edge (a,b) if (a,b) is an equality edge ($(a,b) \in Eq(\mathcal{G})$) and $\alpha(a) = \alpha(b)$, or if (a,b) is a disequality edge ($(a,b) \in Nq(\mathcal{G})$) and $\alpha(a) \neq \alpha(b)$. We denote $\alpha \models \mathcal{G}$ 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 [PRSS98]).

First rewrite all **ITE** terms according to the rewrite rule:

$$[t = \mathbf{ITE}(cond, t_1, t_2)] \Longrightarrow [(i \wedge t = t_1) \vee (\neg i \wedge t = t_2)]$$

Denote the resulting E-formula ψ'. Note that in ψ' all atomic propositions are equalities between term-variables.

We now construct E-graph $\mathcal{G}(\psi)$, by adding a node to V for each term-variable of ψ, and an edge for each equality or disequality of ψ'.

Obviously the distinction between the two types of edges can not be made by the plain appearance of their corresponding equalities in ψ'. The number of negations that enclose each equality term (the term's *polarity*) changes the effect of satisfying it on the satisfiability of ψ'. We thus distinguish between the two types of edges by the corresponding polarity of the equalities which they represent: the edge (a,b) belongs to $Eq(\mathcal{G}(\psi))$ if there exists a term $a = b$ in ψ' with positive polarity (i.e it is enclosed in an even number of negations). Equalities with negative polarity are put in $Nq(\mathcal{G}(\psi))$. We refer the reader to [PRSS98] for more details on this issue.

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

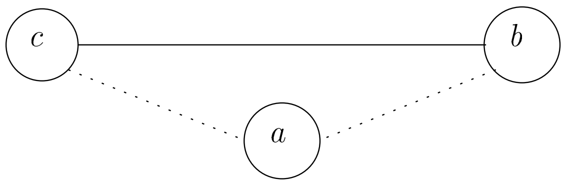

Fig. 1. The E-graph of $\psi = (a = b) \wedge (\neg(c = b) \vee (a = c))$. Dashed lines represent the edges in $Eq(\mathcal{G}(\psi))$, while solid ones represent the edges in $Nq(\mathcal{G}(\psi))$.

Definition 2. (strong adequacy of E-graphs to E-formulas): *An E-graph $\mathcal{G}$ is strongly adequate for E-formula ψ, if for every α such that $\alpha \models \psi$, any β that satisfies the same edges of $\mathcal{G}$ as α does, also satisfies ψ.*

We state the following without proof (see [PRSS98]):

Proposition 1. *$\mathcal{G}(\psi)$ is strongly adequate for ψ.*

Given such an E-graph $\mathcal{G}(\psi)$, we want to find a small set of assignments R 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 $\mathcal{H} < \mathcal{G}$, there is $\alpha \in R$ such that $\alpha \models \mathcal{H}$.*

Proposition 2. *If an E-graph $\mathcal{G}$ is strongly adequate for ψ, and assignment set R is adequate for $\mathcal{G}$, then ψ is satisfiable iff there is $\alpha \in R$ such that $\alpha \models \psi$.*

For example, the following set is adequate for the E-graph of Figure 1:

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

We say that $\mathcal{G} \models \psi$ if for every assignment α such that $\alpha \models \mathcal{G}$, $\alpha \models \psi$. Intuitively, this means that $\mathcal{G}$ contains all the information needed to satisfy ψ.

The requirement of $\mathcal{G}$ being strongly adequate for ψ can be weakened, because in order to prove proposition we only need one satisfying assignment to be in R.

Definition 4. (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 $\mathcal{H} < \mathcal{G}$ such that $\mathcal{H} \models \psi$.*

We then claim:

Proposition 3. *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 $\alpha \models \psi$.*

This weaker definition is not used in [PRSS98], but will be needed in this paper. To demonstrate the difference between the strong and weak versions of adequacy, we return to the graph appearing in Figure 1: if we remove the edge between c and b, the graph remains adequate for ψ, but it is no longer strongly adequate for it.

We will now rephrase the decision procedure for the satisfiability of UF-formulas as suggested in [PRSS98] according to the above definitions:

1. Reduce UF-formula φ to an 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.

We will show that the replacement of the reduction scheme in Step 1 to the reduction proposed by [BV98] enabled us to generalize their result on formulas in the restricted logic of positive equality.

2.2 Reduction Methods

Ackermann's Reduction Scheme: We will denote the Ackermann reduction of a UF-formula φ to an E-formula ψ by $T^A(\varphi)$.

Function instances of φ are numbered in some arbitrary order. For φ', a sub-formula or sub-term of φ, we define $simp(\varphi')$ to be the result of replacing in φ' every function instance F_i by a new term-variable f_i.

For function instance F_i of φ, define $ARG_l(F_i)$ to be the term of φ corresponding to the l-th argument of F_i. Denote by $\mathcal{F}$ the set of function symbols in φ. $T^A(\varphi)$ is then defined as follows:

$$\left[\bigwedge_{F\in\mathcal{F}}\bigwedge_{i,j}\left(\bigwedge_{l} simp(ARG_l(F_i)) = simp(ARG_l(F_j)) \rightarrow f_i = f_j\right)\right] \wedge simp(\varphi)$$

Clearly, φ is satisfiable iff $T^A(\varphi)$ is.

Example 2. We will use the following UF-formula in the next few sections to illustrate the different constructions:

$$\begin{aligned}\varphi_1 := {} & F(F(F(y))) \neq F(F(y)) \wedge \\ & F(F(y)) \neq F(x) \wedge \\ & x = F(y)\end{aligned}$$

We will number the function instances arbitrarily, except from the fact that function instances with syntactically equivalent arguments are given the same index number:

$$\begin{aligned}\varphi_1 := {} & F_4(F_3(F_1(y))) \neq F_3(F_1(y)) \wedge \\ & F_3(F_1(y)) \neq F_2(x) \wedge \\ & x = F_1(y)\end{aligned} \tag{1}$$

The resulting $T^A(\varphi_1)$ in this case will be:

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

Bryant et al. Reduction Scheme: We will denote this type of reduction of a UF-formula φ to an E-formula ψ by $T^B(\varphi)$.

The numbering of the function instances of φ in this scheme is important. It should respect the natural pre-order $\prec$ imposed by φ. For two sub-formulas or sub-terms φ' and φ'' of φ, $\varphi' \prec \varphi''$ iff φ' is in the input cone of φ''. For every two function instances F_i and F_j, if $F_i \prec F_j$ we must have $i < j$. The indexing of the functions in Equation 1 respect this requirement.

The reduced formula $T^B(\varphi)$ is given by replacing the function instance F_i in φ by the term $F_i^\star$ for all i:

$$
\begin{array}{ll}
F_i^\star = case & \\
\quad \bigwedge_l T^B(ARG_l(F_i)) = T^B(ARG_l(F_1)) & : f_1; \\
\quad \bigwedge_l T^B(ARG_l(F_i)) = T^B(ARG_l(F_2)) & : f_2; \\
\quad \vdots & \vdots \\
\quad \bigwedge_l T^B(ARG_l(F_i)) = T^B(ARG_l(F_{i-1})) & : f_{i-1}; \\
\quad 1 & : f_i \\
esac &
\end{array}
$$

Note that since the numbering of the function instances respects the natural pre-order of φ, it is guaranteed that no cyclic definition is possible.

Example 3. $T^B(\varphi_1)$ is given by:

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

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

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

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 instance, and all equality terms of φ appear in negative polarity[1]. In [BGV99] it was shown that given a UF-formula in positive equality and by using the above reduction, it is possible to check for satisfiability by assigning each term variable a single unique constant.

3 A Simplified Graph Construction

In this section we describe a simplified construction of E-graphs that are adequate for $T^A(\varphi)$, where φ is without **ITE** terms and all function instances have only one argument. The fact that we consider $T^A(\varphi)$ rather than $T^B(\varphi)$ implies that we will not be able to generalize the results of [BGV99] yet. In Section 5 we will give the general construction and proof.

[1] 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.

3.1 Definitions

Given an E-graph $\mathcal{G}$, we denote:

- $u \approx_{\mathcal{G}} v$ if there is a simple path in $\mathcal{G}$ between u and v consisting of equality edges.
- $u \asymp_{\mathcal{G}} v$ if there is a simple path in $\mathcal{G}$ between u and v such that one edge in the path is a disequality edge and all other edges are equality edges.

We have the following simple observations for a satisfiable E-graph $\mathcal{G}$:

- If $u \approx_{\mathcal{G}} v$ then $\mathcal{G} \models u = v$ (every $\beta \models \mathcal{G}$ satisfies $\beta(u) = \beta(v)$).
- If $u \asymp_{\mathcal{G}} v$ then $\mathcal{G} \models u \neq v$
- If $u \not\approx_{\mathcal{G}} v$ (there is no equality edge path between u and v in $\mathcal{G}$) then adding disequality edge (u, v) to $\mathcal{G}$ will leave it satisfiable.
- If $u \not\asymp_{\mathcal{G}} v$ then adding equality edge (u, v) to $\mathcal{G}$ will leave it satisfiable.

We denote by $arg(F_i)$ the variable corresponding to the argument of F_i — the i-th instance of F in φ (excluding, as before, instances with syntactically equivalent arguments). This means that in case this argument is a function instance G_j (and not simply a term-variable) then $arg(F_i)$ is the new term-variable g_j.

3.2 Graph Construction

Given a UF-formula φ we construct the following E-graph $\mathcal{G}$:

1. The vertices are the term-variables of $T^A(\varphi)$,
2. Add to $\mathcal{G}$ all the edges of $\mathcal{G}(simp(\varphi))$.
3. For every pair of F_i and F_j, where $i < j$ and $arg(F_i) \approx_{\mathcal{G}} arg(F_j)$, add the following edges:
   (a) Add (f_i, f_j) to $Eq(\mathcal{G})$.
   (b) If $f_i \asymp_{\mathcal{G}} f_j$ add $(arg(F_i), arg(F_j))$ to $Nq(\mathcal{G})$.
   Repeat this process until a fix-point is reached.
4. For every u, v, such that $u \not\approx_{\mathcal{G}} v$, add the edge (u, v) to $Nq(\mathcal{G})$. As will be explained in Section 7, these edges do not affect the resulting state-space, and we therefore refer to them as 'free edges'.

Example 4. We show the graph construction for φ_1. $simp(\varphi_1) = ((f_4 \neq f_3) \wedge (f_3 \neq f_2) \wedge (x = f_1))$ and the resulting graph after step 2 appears in Figure 2. The graph after the fix-point is reached in step 3, is depicted in Figure 3.

Notice that our construction has a sort of a cone-of-influence nature, since after the transformation, the arguments of uninterpreted functions disappear.

It is interesting to compare $\mathcal{G}$ to the graph that was constructed in [PRSS98]. In the latter, all edges in $T^A(\varphi)$ are added to the graph, resulting in the graph shown in Figure 4. If we ignore the free edges that were added in Step 4 (since they do not increase the size of the assignment set needed), then we get that $\mathcal{G} < \mathcal{G}(T^B(\varphi))$. We notice the following:

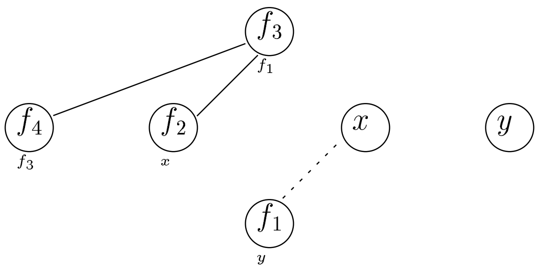

Fig. 2. The E-graph for φ_1 after step 2. Under the vertex corresponding to function variable f_i, we've added $arg(F_i)$.

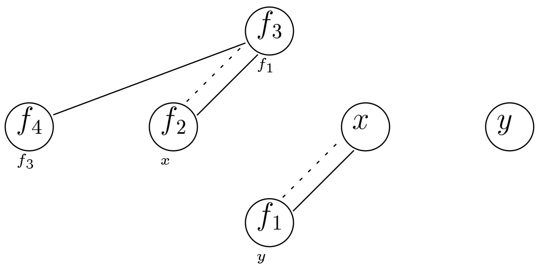

Fig. 3. The E-graph for φ_1 after the fix point of step 3 is reached.

Claim. If assignment set R is adequate for E-graph H, then it is adequate for any E-graph H' such that $H' < H$.

And therefore the assignment set size that resulting from our procedure should generally be smaller. We cannot guarantee this because the procedure for finding an adequate set of assignments for an E-graph may actually give a larger assignment set for a smaller graph, but this should happen only rarely. An adequate set of assignments for $\mathcal{G}$ (Figure 3 plus all free edges) is given, for example, by the following assignment set:

$$\begin{array}{lll} \alpha(x) = 0 & \alpha(y) = 1 & \alpha(f_1) \in \{0, 2\} \\ \alpha(f_2) = 3 & \alpha(f_4) = 5 & \alpha(f_3) \in \{3, 4\} \end{array}$$

Thus, we only need to check 4 assignments. On the other hand, we claim (the proof is not trivial) that any set of assignments that is adequate for the graph of Figure 4 must be of size at least 16. For example, the following assignment set is adequate:

$$\begin{array}{lll} \alpha(f_1) \in \{0, 1\} & \alpha(f_2) \in \{0, 1\} & \alpha(f_3) = 0 \\ \alpha(f_4) \in \{0, 1\} & \alpha(x) \in \{f_1, 2\} & \alpha(y) = 3 \end{array}$$

Notice that one of the values x can get is the value of f_1. Therefore, x can range over $\{0, 1, 2\}$, but this does not mean that the size of the assignment set is 24 (it

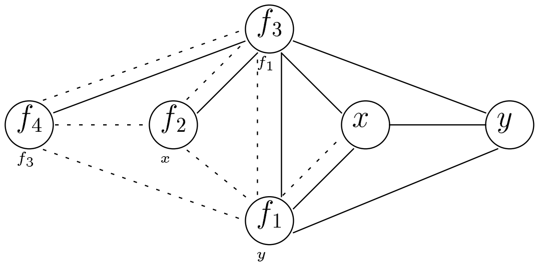

Fig. 4. The E-graph for φ_1, as constructed in , where all equalities are represented as edges in the graph (note the clique of equality edges between all function variables (f_1, f_2, f_4, f_3), and the clique of disequality edges between all the function arguments (x, y, f_1, f_3)). The larger number of edges eventually results in larger domains.

is really 16), because not all combinations of values are possible. For example, there are no assignments in the set with $x = 1$ and $f_1 = 0$. See [PRS01] for more details on procedures that produce this type of assignment sets.

3.3 Proof of Construction

Theorem 1. *The Graph $\mathcal{G}$ that is constructed by the above procedure is adequate for $T^A(\varphi)$.*

Proof. Assume there is a satisfying assignment α to $T^A(\varphi)$ (otherwise $\mathcal{G}$ is adequate for $T^A(\varphi)$ trivially). We will construct a satisfiable E-graph $\mathcal{H} < \mathcal{G}$, such that $\mathcal{H} \models T^A(\varphi)$. In other words, we will show that if $\beta \models \mathcal{H}$ then:

- $\beta \models simp(\varphi)$
- For all F_i, F_j, if $\beta(arg(F_i)) = \beta(arg(F_j))$ then $\beta(f_i) = \beta(f_j)$.

If these two claims are true then $\beta \models T^A(\varphi)$, implying $\mathcal{H} \models T^A(\varphi)$.

- Denote by $\mathcal{H}_0 < \mathcal{G}$ the graph of all edges of $\mathcal{G}$ that α satisfies.
- Create $\mathcal{H}_1$ from $\mathcal{H}_0$ by successively adding disequality edges from $\mathcal{G}$ to $\mathcal{H}_1$ while it remains satisfiable.
- Create $\mathcal{H}_2$ from $\mathcal{H}_1$ by successively adding equality edges from $\mathcal{G}$ to $\mathcal{H}_2$ while it remains satisfiable.

We define $\mathcal{H}$ to be $\mathcal{H}_2$. Clearly $\mathcal{H}$ is satisfiable and $\mathcal{H} < \mathcal{G}$.

If $\beta \models \mathcal{H}$, then $\beta \models simp(\varphi)$ since $\mathcal{G}(simp(\varphi)) < \mathcal{G}$, $\alpha \models simp(\varphi)$, and β satisfies all edges of $\mathcal{G}$ that α does.

It remains to prove that if $\beta(arg(F_i)) = \beta(arg(F_j))$, then $\beta(f_i) = \beta(f_j)$. We will prove this by proving that $(f_i, f_j) \in Eq(\mathcal{H})$.

Lemma 1. *For $\beta \models \mathcal{H}$:*

1. *If $\beta(u) = \beta(v)$ then $u \approx_{\mathcal{G}} v$.*

2. If $u \not\asymp_{\mathcal{G}} v$ and $(u,v) \in Eq(\mathcal{G})$ then $\beta(u) = \beta(v)$.

Proof. If $\beta \models \mathcal{H}$:

1. If $u \not\approx_{\mathcal{G}} v$ then the disequality edge (u,v) is in $\mathcal{G}$ as a free edge, and will always be added to $\mathcal{H}_1$. Since $u \not\approx_{\mathcal{H}_1} v$, then $\beta(u) \neq \beta(v)$.
2. The equality edge (u,v) of $\mathcal{G}$ will always be added to $\mathcal{H}_2$, implying $\beta(u) = \beta(v)$.

□

To conclude we prove:

Claim. If $\beta(arg(F_i)) = \beta(arg(F_j))$ then $(f_i, f_j) \in Eq(\mathcal{H})$.

Proof. If $\beta(arg(F_i)) = \beta(arg(F_j))$ then by Lemma 1 (Part 1), we have that $arg(F_i) \approx_{\mathcal{G}} arg(F_j)$ and therefore $(f_i, f_j) \in Eq(\mathcal{G})$. We split to two cases:

- If $f_i \not\asymp_{\mathcal{G}} f_j$ then by Lemma 1 (Part 2), we have $\beta(f_i) = \beta(f_j)$.
- If $f_i \asymp_{\mathcal{G}} f_j$ then $(arg(F_i), arg(F_j)) \in Nq(\mathcal{G})$ by Step 3b of the graph construction procedure.
  Since $\beta(arg(F_i)) = \beta(arg(F_j))$, this edge was not taken in $\mathcal{H}_1$, meaning $arg(F_i) \approx_{\mathcal{H}_0} arg(F_j)$, but then $\alpha(arg(F_i)) = \alpha(arg(F_j))$ meaning $\alpha(f_i) = \alpha(f_j)$ and then $(f_i, f_j) \in Eq(\mathcal{H}_0)$.

□

4 Assignment Graphs

We now return to work our way towards our main result, a graph construction for general UF-formulas that will generalize the results of [BGV99] and those of section 3. For this we must use the reduction method of [BGV99]. We will also need a new kind of graph — an Assignment graph (A-graph).

Definition 5. (A-graph): *An A-graph $\mathfrak{G}$ is a quadruple $\mathfrak{G} = \langle V, Eq, Nq, As \rangle$, where V is the set of vertices, Eq and Nq are sets of unordered pairs of vertices and As (Assignment edges) is a set of ordered pairs of vertices.*

Given an A-graph $\mathfrak{G} = \langle V, Eq, Nq, As \rangle$, we denote $V(\mathfrak{G}) = V$, $Nq(\mathfrak{G}) = Nq$, $Eq(\mathfrak{G}) = Eq$ and $As(\mathfrak{G}) = As$.

Assignment edges (edges in $As(\mathfrak{G})$) serve as a weak form of equality. We denote $a \rightarrow_{\mathfrak{G}} b$, if there is a directed path (possibly of length 0) of assignment edges from a to b in $\mathfrak{G}$. For assignment α and an A-graph $\mathfrak{G}$, we denote $\alpha \models \mathfrak{G}$ if for every $a \rightarrow_{\mathfrak{G}} b$ and $c \rightarrow_{\mathfrak{G}} d$:

1. If $(b,d) \in Eq(\mathfrak{G})$, $\alpha(a) = \alpha(c)$.
2. If $(b,d) \in Nq(\mathfrak{G})$, $\alpha(a) \neq \alpha(c)$.

Note that $\alpha \models \mathfrak{G}$ implies that α satisfies all equality and disequality edges of $\mathfrak{G}$ (by setting the paths to be of length 0). For A-graph $\mathfrak{G}$ denote by $flat(\mathfrak{G})$ the E-graph received by replacing all assignment edges of $\mathfrak{G}$ by equality edges. We say that an A-graph $\mathfrak{G}$ is satisfiable if $flat(\mathfrak{G})$ is satisfiable. This is not the most natural definition, but we will need it for our translation from A-graphs to E-graphs (Section 2.2)

For an E-formula ψ and a satisfiable A-graph $\mathfrak{G}$, we denote $\mathfrak{G} \models \psi$ if for every assignment α such that $\alpha \models \mathfrak{G}$ we have $\alpha \models \psi$. The following definitions are the exact analog of the definitions made for E-graphs:

Definition 6. (adequacy of A-graphs to E-formulas): *An A-graph $\mathfrak{G}$ is adequate for E-formula ψ, if either ψ is not satisfiable, or there exists a satisfiable $\mathfrak{H} < \mathfrak{G}$ such that $\mathfrak{H} \models \psi$.*

Definition 7. (adequacy of assignment sets to A-graphs): *Given an A-graph $\mathfrak{G}$, and R, a set of assignments to $V(\mathfrak{G})$, we say that R is adequate for $\mathfrak{G}$ if for every satisfiable $\mathfrak{H} < \mathfrak{G}$ there is $\alpha \in R$ such that $\alpha \models \mathfrak{H}$.*

The analogous proposition follows:

Proposition 4. *If A-graph $\mathfrak{G}$ is adequate for ψ, and assignment set R is adequate for $\mathfrak{G}$, then ψ is satisfiable iff there is $\alpha \in R$ such that $\alpha \models \psi$.*

In the following section, given a UF-formula φ, we will construct an A-graph $\mathfrak{G}$ such that $\mathfrak{G}$ is adequate for $T^B(\varphi)$. In section 6 we will show how to create an E-graph $\mathcal{G}$ from $\mathfrak{G}$, such that if R is adequate for $\mathcal{G}$ it is also adequate for $\mathfrak{G}$. These two procedures combined with a procedure for finding an adequate assignment set for an E-graph as described in [PRSS98] and [PRS01], give us a more efficient decision procedure for satisfiability of UF-formulas.

5 Graph Construction

In this section we describe and prove the A-graph construction for general UF-formulas. Note that we still do not handle predicates and Boolean variables which will be discussed in Section 8. The intuition behind the construction and proof is similar to those in section 3, but there are many new details and complications.

In this section we use a different UF-formula as our running example, which we will denote by φ_2. We use shorthand notation for this formula, by writing $F_i(\ldots)$ whenever F_i's arguments were already specified:

$$\begin{aligned}\varphi_2 := {} & F_2(\mathbf{ITE}(F_1(b) = u, a, b)) = u \wedge \\ & F_3(F_2(\ldots)) \neq F_4(\mathbf{ITE}(a = b, u, a))\end{aligned}$$

This example will assist us later in clarifying the definitions and the construction procedure.

In the following discussion the distinction between F_i, f_i, and $F_i^\star$ is crucial. F_i is the i-th function instance of a UF-formula φ (according to our predetermined numbering of the function instances of φ). f_i is the term-variable of $T^B(\varphi)$ that was introduced by the reduction, and $F_i^\star$ is the term of $T^B(\varphi)$ which replaces F_i in φ.

5.1 Definitions

We start with some notations for a UF-formula φ, assignment α to the variables of $T^B(\varphi)$, and an A-graph $\mathfrak{G}$. For this purpose we will use the following example assignment γ to φ_2's variables:

$$\gamma(a)=0 \quad \gamma(b)=0 \quad \gamma(u)=1$$
$$\gamma(f_1)=1\ \gamma(f_2)=2\ \gamma(f_3)=3\ \gamma(f_4)=4$$

1. For function instance F_i of φ, define $ARG_l(F_i)$ to be the term of φ corresponding to the l-th argument of F_i.

   Example 5. $ARG_1(F_1)=b$ and $ARG_1(F_4)=\mathbf{ITE}(a=b,u,a)$.
2. For a term t of $T^B(\varphi)$, we define $\alpha(t)$ to be the evaluation of t under α.
3. For term-variable v of $T^B(\varphi)$, define $s_\alpha(v)$ ('s' standing for *source*):
   - If v is an original term-variable of φ then $s_\alpha(v)=v$.
   - If $v=f_j$, then take the minimal i such that for all l, $\alpha(T^B(ARG_l(F_i)))=\alpha(T^B(ARG_l(F_j)))$, and set $s_\alpha(v)=f_i$.

   Example 6. In φ_2:

   $$s_\gamma(a)=a \quad s_\gamma(b)=b \quad s_\gamma(u)=u$$
   $$s_\gamma(f_1)=f_1\ s_\gamma(f_2)=f_1\ s_\gamma(f_3)=f_3\ s_\gamma(f_4)=f_3$$
4. Define assignment $\hat{\alpha}$ to be:
   - For v, a term-variable of φ, set $\hat{\alpha}(v)=\alpha(v)$.
   - For $v=f_i$, $\hat{\alpha}(f_i)=\alpha(F_i^\star)$.

   Notice that $\hat{\alpha}(v)=\alpha(s_\alpha(v))$. $\hat{\alpha}$ can be seen as the real assignment emanating from α, and for φ' a sub-term or sub-formula of φ we have $\alpha(T^B(\chi))=\hat{\alpha}(simp(\chi))$. In particular $\alpha\models T^B(\varphi)$ iff $\hat{\alpha}\models simp(\varphi)$.

   Example 7.

   $$\hat{\gamma}(a)=0 \quad \hat{\gamma}(b)=0 \quad \hat{\gamma}(u)=1$$
   $$\hat{\gamma}(f_1)=1\ \hat{\gamma}(f_2)=1\ \hat{\gamma}(f_3)=3\ \hat{\gamma}(f_4)=3$$

   This is because
   - $\gamma(T^B(ARG(F_2)))=\gamma(T^B(ARG(F_1)))=0$
   - $\gamma(T^B(ARG(F_4)))=\gamma(T^B(ARG(F_3)))=1$
5. For term t of φ, define $vals(t)$ to be:
   - If t is a term-variable then $vals(t)=\{t\}$.
   - If $t=F_i(\ldots)$ then $vals(t)=\{f_i\}$.
   - If $t=\mathbf{ITE}(cond,t_1,t_2)$ then $vals(t)=vals(t_1)\cup vals(t_2)$.

   Notice that $\alpha(T^B(t))=\hat{\alpha}(v)$ for some $v\in vals(t)$, depending on the evaluation of the Boolean conditions in the relevant **ITE** terms.

   Example 8. In our example φ_1:
   - $vals(F_3(\ldots))=\{f_3\}$
   - $vals(\mathbf{ITE}(a=b,u,a))=\{u,a\}$
   - $vals(ARG_1(F_2))=\{a,b\}$
6. For $u,v\in V(\mathfrak{G})$, we mark $u\approx_{\mathfrak{G}} v$ iff $u\approx_{flat(\mathfrak{G})} v$, and $u\asymp_{\mathfrak{G}} v$ iff $u\asymp_{flat(\mathfrak{G})} v$. Recall that for an A-graph $\mathfrak{G}$, $flat(\mathfrak{G})$ is $\mathfrak{G}$ where all assignment edges are replaced by equalities.

   We extend this definition to sets of vertices U_1 and U_2, and mark $U_1\approx_{\mathfrak{G}} U_2$ if there are some $u_1\in U_1$ and $u_2\in U_2$ such that $u_1\approx_{\mathfrak{G}} u_2$.

5.2 Graph Construction

Given a UF-formula φ we construct an A-graph $\mathfrak{G}$:

1. The vertices are the set of term-variables of $T^B(\varphi)$.
2. Add at least all edges of $\mathcal{G}(simp(\varphi))$ to $\mathfrak{G}$ (Naturally one would take exactly $\mathcal{G}(simp(\varphi))$, but we need this weaker statement for the proof).
3. For every F_i and F_j such that $i < j$ and for all l $vals(ARG_l(F_i)) \approx_{\mathfrak{G}} vals(ARG_l(F_j))$, add the following edges:
   - Add (f_i, f_j) to $As(\mathfrak{G})$.
   - If $f_i \asymp_{\mathfrak{G}} f_j$ then for all l, for all $v_i \in vals(ARG_l(F_i))$ and $v_j \in vals(ARG_l(F_j))$ add edge (v_i, v_j) to $Nq(\mathfrak{G})$. Also mark F_i and F_j as *important*.
4. For every important F_i, if for some l the term $t = \mathbf{ITE}(cond, t_1, t_2)$ appears in $simp(ARG_l(F_i))$, then add all edges of $\mathcal{G}(cond)$ and $\mathcal{G}(\neg cond)$ to $\mathfrak{G}$. Also add $cond$ to set C (which is initially empty).
5. Repeat steps 3 and 4 until a fix-point is reached.
6. For every u, v, such that $u \not\approx_{\mathfrak{G}} v$, add edge (u, v) to $Nq(\mathfrak{G})$. Denote all these edges *free* (see Section 7).

Example 9. For φ_2, the edges are added as follows:

1. In Step 2, we add (f_3, f_4) to $Nq(\mathfrak{G})$ and (u, f_2) to $Eq(\mathfrak{G})$. The state of $\mathfrak{G}$ at this point is described in Figure 5(A).
2. In Step 3, we add the following edges:
   - (f_1, f_2) to $As(\mathfrak{G})$ since $b \in vals(ARG_1(f_1))$ and $b \in vals(ARG_1(f_2))$.
   - (f_2, f_4) to $As(\mathfrak{G})$ since $a \in vals(ARG_1(f_2))$ and $a \in vals(ARG_1(f_4))$.
   - (f_3, f_4) is added to $As(\mathfrak{G})$ since $f_2 \approx_{\mathfrak{G}} u$. Also, since $f_3 \asymp_{\mathfrak{G}} f_4$, F_3 and F_4 are marked as important, and (f_2, u) and (f_2, a) are added to $Nq(\mathfrak{G})$
3. Since F_4 was marked as important, $(a = b) \in C$. Therefore, in Step 4, we add (a, b) to both $Eq(\mathfrak{G})$ and $Nq(\mathfrak{G})$[2].
4. Now (this wasn't the case before), since $a \approx_{\mathfrak{G}} b$, we add (f_1, f_4) to $As(\mathfrak{G})$. See part B of Figure 5.
5. We now add all free edges. For every $x \in \{a, b\}$ and $y \in \{f_1, f_2, f_3, f_4, u\}$ we add (x, y) to $Nq(\mathfrak{G})$.

Notice that throughout the construction we did not examine the condition of the **ITE** term appearing in the argument of F_2, since F_2 was not marked as important. This is the "cone of influence" effect we were aiming for.

We now prove the soundness of the suggested construction.

[2] Using a more careful analysis of the values that the arguments that F_3 and F_4 can be assigned, it can be observed that it is not necessary to add (a, b) to $Nq(\mathfrak{G})$. This and other optimizations were left out for simplicity.

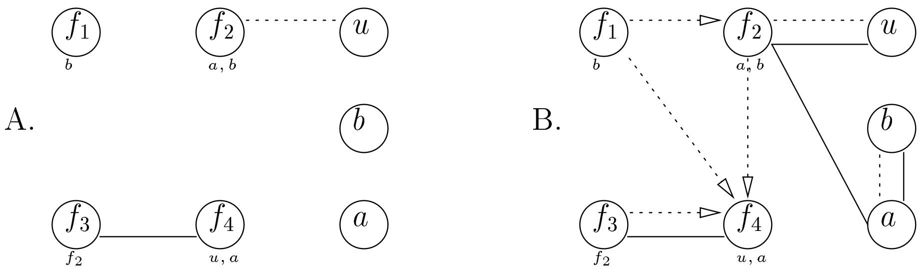

Fig. 5. Figure A is φ_2's A-graph after Step 2. Figure B is φ_2's A-graph after the fix-point is reached (before free edges are added). Under a vertex corresponding to function variable f_i, we've added the list of vertices in $vals(ARG_1(F_i))$.

5.3 Proof of Soundness

Theorem 2. *The A-graph $\mathcal{G}(\psi)$ as constructed in the previous section, is adequate for $T^B(\varphi)$.*

The suggested construction gives rise to many different A-graphs, depending on what A-graph we start with in Step 2 (recall that we only have a minimum requirement in this step). We prove by induction on $|C|$ that all of them are adequate for $T^B(\varphi)$. The base case where $C = \emptyset$ is the difficult part, so we start with the induction step.

If $cond \in C$ then there is F_i and l such that a term $t = \mathbf{ITE}(cond, t_1, t_2)$ appears in $simp(ARG_l(F_i))$, where f_i is marked important. This term also appears in φ as $t' = \mathbf{ITE}(cond', t_1', t_2')$ where $t = simp(t')$ (we use $'$ to mark the corresponding un-simplified term appearing in φ).

Define:

$$\rho_1 = cond' \wedge \varphi[t' \leftarrow t_1']$$
$$\rho_2 = \neg cond' \wedge \varphi[t' \leftarrow t_2']$$

Clearly, $\varphi = \rho_1 \vee \rho_2$. It is now sufficent to prove that $\mathfrak{G}$ is adequate for $T^B(\rho_1)$ and for $T^B(\rho_2)$, since then $\mathfrak{G}$ is adequate for $T^B(\varphi)$.

We first show that any $\mathfrak{G}$ constructed for φ by our procedure can also be constructed for ρ_i ($i \in \{1, 2\}$) if one starts in Step 2 of the construction with our $\mathfrak{G}$ (the one constructed for φ) instead of $\mathcal{G}(simp(\rho_i))$.

For this we need to show that $\mathcal{G}(simp(\rho_i)) < \mathfrak{G}$. For example, if we consider ρ_1, $\mathcal{G}(simp(\rho_1))$ is $\mathcal{G}(simp(\varphi)) \cup \mathcal{G}(simp(cond'))$. Clearly $\mathcal{G}(simp(\rho_1)) < \mathcal{G}(simp(\varphi)) < \mathfrak{G}$. Also, since $cond \in C$ then $\mathcal{G}(cond) < \mathfrak{G}$ (Step 4). Since $simp(cond') = cond$, we have that $\mathcal{G}(simp(\rho_1)) < \mathfrak{G}$.

We also need to show that no new edges are added in the next steps. But this is true because nothing in the procedure has changed except possibly removed some elements from $vals(ARG_l(F_i))$ for some F, i and l. Therefore $\mathfrak{G}$ is a graph that can be constructed by our procedure for ρ_1 and ρ_2.

In both constructions (for ρ_1 and ρ_2), the size of the set C will always be smaller than in the construction for φ, since $cond$ will no longer be part of the formula, and no new conditions will be added to C. Therefore, by the induction hypothesis, $\mathfrak{G}$ is adequate for $T^B(\rho_1)$ and $T^B(\rho_2)$.

We proceed to prove the case where $C = \emptyset$. By the fact that C is empty, we know that for every important F_i, for every l, $vals(ARG_l(F_i))$ contains exactly one term-variable. In this case we mark this variable by $arg_l(F_i)$.

Assume there is a satisfying assignment α to $T^B(\varphi)$. W.l.o.g we can assume that $\alpha = \hat{\alpha}$ because $\hat{\alpha}$ is also a satisfying assignment to φ.

We will construct a satisfiable A-graph $\mathfrak{H} < \mathfrak{G}$, such that if $\beta \models \mathfrak{H}$ then $\hat{\beta}$ satisfies all equality and disequality edges of $\mathfrak{H}$. $\mathfrak{H}$ will also contain all the edges of $\mathfrak{G}$ that α satisfies, therefore: $\hat{\beta} \models simp(\varphi)$, because $\alpha \models simp(\varphi)$ and $\mathcal{G}(simp(\varphi)) < \mathfrak{G}$. This means that $\beta \models T^B(\varphi)$.

- Denote by $\mathfrak{H}_0 < \mathfrak{G}$ the graph of all equality and disequality edges of $\mathfrak{G}$ that α satisfies.
- Create $\mathfrak{H}_1$ starting from $\mathfrak{H}_0$ and using the following closure rule: If for all l, $vals(ARG_l(F_i)) \approx_{\mathfrak{H}_1} vals(ARG_l(F_j))$ then add the edge (f_i, f_j) to $As(\mathfrak{H}_1)$.
- Create $\mathfrak{H}_2$ from $\mathfrak{H}_1$. For every pair F_i and F_j where $i < j$, Proceed if the following conditions are true:
  - $(f_i, f_j) \notin As(\mathfrak{H}_1)$.
  - $f_i \asymp_{\mathfrak{G}} f_j$.
  - $(f_i, f_j) \in As(\mathfrak{G})$.

  The last two conditions imply that F_i and F_j are important. therefore $arg_l(F_i)$ and $arg_l(F_j)$ are defined. We also know that $(arg_l(F_i), arg_l(F_j)) \in Nq(\mathfrak{G})$. Since $(f_i, f_j) \notin As(\mathfrak{H}_1)$, there is some l such that $arg_l(F_i) \not\approx_{\mathfrak{H}_1} arg_l(F_j)$. Add $(arg_l(F_i), arg_l(F_j))$ to $Nq(\mathfrak{H}_2)$.
- Create $\mathfrak{H}_3$ from $\mathfrak{H}_2$ by successively adding assignment edges from $\mathfrak{G}$ to $\mathfrak{H}_3$ that do not render it unsatisfiable.
- Add all free edges of $\mathfrak{G}$ to $\mathfrak{H}_3$ to create $\mathfrak{H}_4$.

We define $\mathfrak{H}$ to be $\mathfrak{H}_4$. By the construction, $\mathfrak{H}$ is satisfiable and $\mathfrak{H} < \mathfrak{G}$.

Lemma 2. *For $\beta \models \mathfrak{H}$:*

1. *If $\hat{\beta}(v) = \hat{\beta}(u)$ then $v \approx_{\mathfrak{G}} u$.*
2. *Let $u = s_\beta(v)$. If $u \neq v$, then $(u, v) \in As(\mathfrak{G})$.*

Proof. The proof proceeds by a bit complex induction on the pairs (u, v). For two variables x, y we denote $x \prec y$, if $y = f_i$, and $x \neq f_i$ is in the fan-in cone of F_i in $T^B(\varphi)$. This is the standard pre-order on terms in a formula. Note that if $s_\beta(a) \neq a$, then $s_\beta(a) \prec a$.

1. We prove 1 on an unordered pair (u, v), by assuming 2 on all unordered pairs (x, y) such that either $x = u$ or $x = v$, and $y \prec x$.
2. We prove 2 on unordered pair (u, v), by assuming 1 on all unordered pairs (u', v') such that $u' \prec u$ and $v' \prec v$.

This is a valid induction if $\prec$ is any non-reflexive partial-order on a finite set (which is the case here).

These are the two proofs:

1. We want to prove that if $\hat{\beta}(u) = \hat{\beta}(v)$ then $u \approx_{\mathfrak{G}} v$.
   If $\beta(u) = \beta(v)$ then $u \approx_{\mathfrak{G}} v$, since if $u \not\approx_{\mathfrak{G}} v$ then by Step 6 of the A-graph construction, disequality edge (u, v) will be in $Nq(\mathfrak{G})$ and it will always be added to $\mathfrak{H}_4$.
   If $\hat{\beta}(u) = \hat{\beta}(v)$ then $\beta(s_\beta(u)) = \beta(s_\beta(v))$. By the same argument as above we have that $s_\beta(u) \approx_{\mathfrak{G}} s_\beta(v)$.
   If $s_\beta(u) \neq u$, then according to our induction hypothesis (2), $(s_\beta(u), u) \in As(\mathfrak{G})$, meaning $s_\beta(u) \approx_{\mathfrak{G}} u$. Clearly, if $s_\beta(u) = u$ then also $s_\beta(u) \approx_{\mathfrak{G}} u$. For the same reason, $s_\beta(v) \approx_{\mathfrak{G}} v$.
   Since $\approx_{\mathfrak{G}}$ is an equivalence relation, $s_\beta(u) \approx_{\mathfrak{G}} s_\beta(v)$ (as we've shown above), $s_\beta(u) \approx_{\mathfrak{G}} u$ and $s_\beta(v) \approx_{\mathfrak{G}} v$ we get that $u \approx_{\mathfrak{G}} v$.
2. Let $u = s_\beta(v)$. If $u \neq v$ then for some function F, $v = f_j$ and $u = f_i$, where $i < j$. Also, for all l, $\beta(T^B(ARG_l(F_i))) = \beta(T^B(ARG_l(F_j)))$. Equivalently, $\hat{\beta}(simp(ARG_l(F_i))) = \hat{\beta}(simp(ARG_l(F_j)))$. This means that for every l, there is some $v_l \in vals(ARG_l(F_i))$ and $u_l \in vals(ARG_l(F_j))$ such that $\hat{\beta}(u_l) = \hat{\beta}(v_l)$. Using our induction hypothesis (1), for all l, $u_l \approx_{\mathfrak{G}} v_l$, implying $vals(ARG_l(F_i)) \approx_{\mathfrak{G}} vals(ARG_l(F_j))$, This means that $(f_i, f_j) \in As(\mathfrak{G})$.

$\square$

To conclude we prove:

Claim. For $\beta \models \mathfrak{H}$:

1. $\hat{\beta}$ satisfies all equality and disequality edges of $\mathfrak{H}$.
2. Let $u = s_\beta(v)$. If $u \neq v$. then $(u, v) \in As(\mathfrak{H})$.

Proof. We use the same induction strategy as in the proof of Lemma 2:

1. Recall we marked $a \to_{\mathfrak{G}} b$ if there is a directed assignment edge path (possibly of length 0) from a to b in $\mathfrak{G}$. Using our induction hypothesis (2), $s_\beta(u) \to_{\mathfrak{H}} u$ and $s_\beta(v) \to_{\mathfrak{H}} v$. Since $\beta \models \mathfrak{H}$, we get that if $(u, v) \in Eq(\mathfrak{H})$ then $\beta(s_\beta(u)) = \beta(s_\beta(v))$ and if $(u, v) \in Nq(\mathfrak{H})$ then $\beta(s_\beta(u)) \neq \beta(s_\beta(v))$. Since always $\hat{\beta}(a) = \beta(s_\beta(a))$, we conclude.
2. Let $u = s_\beta(v)$. From Lemma 2, we know that $(u, v) \in As(\mathfrak{G})$. We split to two cases:
   - If $u \not\asymp_{\mathfrak{G}} v$ then the edge (u, v) of $As(\mathfrak{G})$ will always be added to $\mathfrak{H}_3$ since it will never make $\mathfrak{H}$ unsatisfiable.
   - Assume $u \asymp_{\mathfrak{G}} v$. We know that $v = f_j$ and $u = f_i$ where $i < j$, and that f_i and f_j are important. Therefore, for every l, $arg_l(F_i)$ and $arg_l(F_j)$ are defined. Since $(f_i, f_j) \in As(\mathfrak{G})$ and $f_i \asymp_{\mathfrak{G}} f_j$ then $(arg_l(F_i), arg_l(F_j)) \in Nq(\mathfrak{G})$.
     If for some l, $(arg_l(F_i), arg_l(F_j)) \in Nq(\mathfrak{H})$ then using our induction hypothesis (1), we get $\hat{\beta}(arg_l(F_i)) \neq \hat{\beta}(arg_l(F_j))$, but this contradicts the fact that $s_\beta(f_j) = f_i$.
     Therefore for all l, $(arg_l(F_i), arg_l(F_j)) \notin Nq(\mathfrak{H})$, but then already in $\mathfrak{H}_1$ for all l, $arg_l(F_i) \approx_{\mathfrak{H}_1} arg_l(F_j)$. If this is not true then for some l the disequality edge $(arg_l(F_i), arg_l(F_j))$ would be taken in $\mathfrak{H}_2$.
     Now, since for all l, $arg_l(F_i) \approx_{\mathfrak{H}_1} arg_l(F_j)$, then $(f_i, f_j) \in As(\mathfrak{H}_1)$. Therefore $(u, v) \in As(\mathfrak{H})$.

$\square$

6 Transforming A-graphs to E-graphs

Given an A-graph $\mathfrak{G}$, we wish to construct an E-graph $\mathcal{G}$ such that if assignment set R is adequate for $\mathcal{G}$ it will also be adequate for $\mathfrak{G}$.

For two vertices u and v, we denote $v \subseteq_{\mathcal{G}} u$, if

- for every $(v, w) \in Eq(\mathcal{G})$, $(u, w) \in Eq(\mathcal{G})$.
- for every $(v, w) \in Nq(\mathcal{G})$, $(u, w) \in Nq(\mathcal{G})$.

The following procedure transforms A-graphs to E-graphs:

1. Initially, $\mathcal{G} = \langle V(\mathfrak{G}), Eq(\mathfrak{G}), Nq(\mathfrak{G})\rangle$
2. While there are vertices u, v, such that $(u, v) \in As(\mathfrak{G})$ and either $(u, v) \notin Eq(\mathcal{G})$ or $v \subseteq_{\mathcal{G}} u$, choose one of the following options:
   (a) add edge (u, v) to $Eq(\mathcal{G})$.
   (b) – for every $(v, w) \in Eq(\mathcal{G})$ add (u, w) to $Eq(\mathcal{G})$.
   – for every $(v, w) \in Nq(\mathcal{G})$ add (u, w) to $Nq(\mathcal{G})$.

Theorem 3. *If R is adequate for $\mathcal{G}$ then it is also adequate for $\mathfrak{G}$.*

Proof. Take some satisfiable $\mathfrak{H} < \mathfrak{G}$. We construct a satisfiable $\mathcal{H} < \mathcal{G}$ such that if $\beta \models \mathcal{H}$ then $\beta \models \mathfrak{H}$.

Since $\mathfrak{H}$ is satisfiable, there is some α that satisfies $\mathfrak{H}$ where all of $\mathfrak{H}$'s assignment edges are replaced by equality edges (recall the definition of the satisfiability of an A-graph). We will prove that $\mathcal{H}$ is satisfiable by showing that $\alpha \models \mathcal{H}$.

Start with $Eq(\mathcal{H}) = Eq(\mathfrak{H})$, and $Nq(\mathcal{H}) = Nq(\mathfrak{H})$. Clearly, at this stage $\alpha \models \mathcal{H}$. For every edge $(u, v) \in As(\mathfrak{H})$ (Notice that $\alpha(u) = \alpha(v)$):

- If $(u, v) \in Eq(\mathcal{G})$ add (u, v) to $Eq(\mathcal{H})$. Obviously $\alpha \models \mathcal{H}$,
- Otherwise, we have that $v \subseteq_{\mathcal{G}} u$. Add the following edges to $\mathcal{H}$:
  1. for every $(v, w) \in Eq(\mathcal{H})$ add (u, w) to $Eq(\mathcal{H})$.
  2. for every $(v, w) \in Nq(\mathcal{H})$ add (u, w) to $Nq(\mathcal{H})$.

  Since $\alpha \models \mathcal{H}$ before this step, and $\alpha(v) = \alpha(u)$, it is easy to see that also after this step $\alpha \models \mathcal{H}$.

We now prove that if $\beta \models \mathcal{H}$ then $\beta \models \mathfrak{H}$. Take some $a \rightarrow_{\mathfrak{H}} b$ and $c \rightarrow_{\mathfrak{H}} d$. We need to prove that:

- If $(b, d) \in Eq(\mathfrak{H})$ then $\beta(a) = \beta(c)$.
- If $(b, d) \in Nq(\mathfrak{H})$ then $\beta(a) \neq \beta(c)$.

To simplify the proof we weaken the definition of $\approx$ and $\asymp$, so that the paths need not be simple. Notice that under this definition we still have:

- If $x \approx_{\mathcal{H}} y$ then $\beta(x) = \beta(y)$
- If $x \asymp_{\mathcal{H}} y$ then $\beta(x) \neq \beta(y)$

We will therefore prove:

– If $(b,d) \in Eq(\mathfrak{H})$ then $a \approx_{\mathcal{H}} c$.
– If $(b,d) \in Nq(\mathfrak{H})$ then $a \asymp_{\mathcal{H}} c$.

We will prove this by induction on the sum of the lengths of the assignment edge paths from a to b and from c to d. If this sum is 0, then $a = b$ and $c = d$, and since we copied all equality and disequality edges of $\mathfrak{H}$ to $\mathcal{H}$, the claim follows.

If this sum is greater than 0, then one of these paths is of length greater than 0. W.l.o.g, assume this is the path between a and b. Mark by a' the next vertex after a in the path to b (this may be b itself). According to our induction hypothesis:

– If $(b,d) \in Eq(\mathfrak{H})$ then $a' \approx_{\mathcal{H}} c$.
– If $(b,d) \in Nq(\mathfrak{H})$ then $a' \asymp_{\mathcal{H}} c$.

Since $(a,a') \in As(\mathfrak{H})$, we have one of the two cases:

1. $(a,a') \in Eq(\mathcal{H})$. But then clearly a satisfies the two claims, since we simply prolong the path $a' \approx_{\mathcal{H}} c$ (in the first case) or $a' \asymp_{\mathcal{H}} c$ (in the second) by the equality edge (a,a').
2. Otherwise, we know that $a' \subseteq_{\mathcal{H}} a$. This time instead of prolonging these paths, we replace their first edge. For example, if $(b,d) \in Nq(\mathcal{H})$, then $a' \asymp_{\mathcal{H}} c$. This means there is a path from a' to c in $\mathcal{H}$ consisting of equality edges except one edge which is a disequality edge. Take the first edge in this path (a',x), and replace it by the same type (equality or disequality) of edge (a,x). We know this edge is in $\mathcal{H}$, since $a' \subseteq_{\mathcal{H}} a$.

□

We showed a general method for translating of A-graphs to E-graphs. When using this method one has to choose between the two options for every assignment edge: either replace it by an equality edge, or copy all edges of the end vertex to the start vertex.

Note that if the original UF-formula φ is in positive equality then the A-graph constructed in section 5.2 contains no equality edges, and therefore we can translate it to an E-graph with no equality edges. An adequate set of assignments for such an E-graph contains just one assignment in which every variable is assigned a distinct constant. This is exactly the result of [BGV99].

In our implementation, we use a greedy approach for choosing between the options, where we try to minimize the number of equality edges of the resulting E-graph. In the case of positive equality formulas, we therefore get the optimal result of [BGV99].

Example 10. We demonstrate the transformation with the help of φ_1, the UF formula that was first presented in Section 2.2. The resulting A-graph of φ_1 from the construction is described in Figure 6, and the result of transforming it to an E-graph (using the greedy approach) is presented in Figure 7. This graph has an adequate set of assignments that consists of 2 assignments, as opposed to 4 in our simplified construction in Section 3 and 16 in the procedure we presented in [PRSS98] .

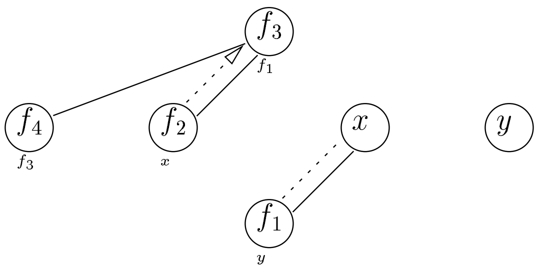

Fig. 6. The A-graph of φ_1.

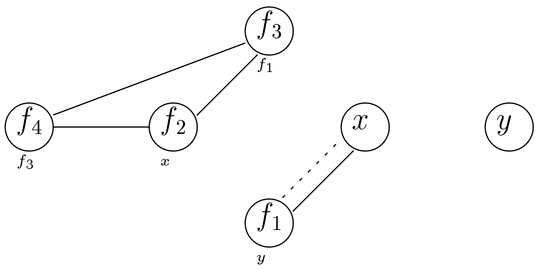

Fig. 7. The A-graph of φ_1 transformed to an E-graph.

7 Free Edges

Recall the last step of our graph construction: If $u \not\approx_{\mathfrak{G}} v$ then the disequality edge (u, v) is added to $\mathfrak{G}$. We denoted these edges as free edges. The reason for this is that they do not impose any increase in the size of the assignment set R that is adequate for $\mathfrak{G}$.

Given an E-graph $\mathcal{G}$ (we will handle A-graphs later) and a set R that is adequate for $\mathcal{G}$, we construct assignment set R' that is adequate for $\mathcal{G}'$, where $\mathcal{G}'$ is $\mathcal{G}$ plus all free edges. We will show that $|R'| = |R|$.

Define $\mathcal{C}(\mathcal{G}) = \{C_1, C_2, \ldots\}$ to be the set of connected components of the graph $\langle V(\mathcal{G}), Eq(\mathcal{G})\rangle$. Notice that $u \approx_{\mathcal{G}} v$ iff there is some i such that $u, v \in C_i$. Take some number N such that for every $\alpha \in R$ and every $v \in V(\mathcal{G})$, $N > 2 \cdot |\alpha(v)|$. For $\alpha \in R$ we define α' to be the assignment that satisfies:

$$\forall i\ \forall v \in C_i,\ \alpha'(v) = \alpha(v) + i \cdot N$$

Define $R' = \{\alpha' \mid \alpha \in R\}$. We claim that R' is adequate for $\mathcal{G}'$.

Proof. Take $\mathcal{H}' < \mathcal{G}'$. $\mathcal{H}'$ is composed of an $\mathcal{H} < \mathcal{G}$ plus some free edges of $\mathcal{G}$. Since R is adequate for $\mathcal{H}$, there exists $\alpha \in R$ such that $\alpha \models \mathcal{H}$. We claim that $\alpha' \models \mathcal{H}'$. Take some edge (u, v) of $\mathcal{H}'$. There are $C_i, C_j \in \mathcal{C}(\mathcal{G})$ such that $u \in C_i$ and $v \in C_j$. We have that $\alpha'(u) = i \cdot N + \alpha(u)$ and $\alpha'(v) = j \cdot N + \alpha(v)$. Subtracting one from another we get:

$$\alpha'(v) - \alpha'(u) = (i - j) \cdot N + \alpha(v) - \alpha(u)$$

– If $i \neq j$ then (u, v) must be a disequality edge (either free or originally in $\mathcal{G}$). Since $N > |\alpha(u) - \alpha(v)|$ we get that $|\alpha'(u) - \alpha'(v)| > |(i - j)| \cdot N - N$. Since $i \neq j$ we have that $|\alpha'(u) - \alpha'(v)| > 0$, implying $\alpha'(u) \neq \alpha'(v)$.
– If $i = j$, then α satisfies edge (u, v) (equality or disequality). Also $\alpha'(u) - \alpha'(v) = \alpha(u) - \alpha(v)$, and therefore α' also satisfies this edge.

$\square$

Thus, adding free edges to E-graphs does not increase the size of the assignment set. In the case of A-graphs, if we examine the transformation to E-graphs that we presented in section 6, we see that if $u \not\approx_{\mathfrak{G}} v$ in the original A-graph, then also $u \not\approx_{\mathcal{G}} v$ in the resulting E-graph. Therefore, the free edges that we add to the original A-graph appear as free edges in the resulting E-graph, and as we have just shown, it will not increase the size of the resulting assignment set.

8 Uninterpreted Predicates

Up to now we assumed that there are no predicate symbols or Boolean variables in the formula. We will justify this by showing how we simulate predicates using uninterpreted functions (Boolean variables are merely predicates with 0 arguments). Given φ, a UF-formula with uninterpreted predicates, we transform φ to φ' without uninterpreted predicates.

For every predicate symbol P in φ, take a new function symbol F^P and a new term-variable $true^P$. Now replace in φ every occurrence of $P(\ldots)$, by $(F^P(\ldots) = true^P)$ to get φ'. It is not hard to see that φ' is satisfiable if and only if φ is.

The problem we are facing is that instead of adding a Boolean variable for each predicate instance (as happens in [BGV99] reduction), we added a term-variable and possibly increased the state space for checking φ. But a more careful examination proves that this is not the case.

Note that in the A-graph of φ', all the new variables introduced for predicate P are isolated (there are no edges between them and any of the other variables). Therefore we can find an adequate assignment set for each such predicate separately from each other and from the assignment set for the original graph (which is unaffected by this addition). We first transform the component's A-graph to an E-graph by replacing all assignment edges by equality edges (this is a valid transformation by section 6). In our case the graph of the component of some predicate P cannot contain disequality edges other than $(f_i^P, true^P)$. An adequate set of assignments for this kind of E-graph is given by letting each variable range over $\{0, 1\}$, and setting $true^P$ to be the constant 1.

The state-space is therefore not increased. In fact this method gives us the ability to treat predicates with the new method we used for functions, achieving a similar cone-of-influence effect.

9 Experimental Results

As was mentioned before, the graphs that are generated by the method which was presented in this paper are always smaller or equal to those that were generated in [PRSS98], and therefore will result in a smaller state-space. This is not only a theoretical result, since we were able to check in seconds UF-formulas that we could not solve at all by using the previous method.

As for [BGV99], we generalize their result for the restricted logic of positive equality, and therefore on these kind of formulas are supposed to get more or less the same results, depending on implementation details. However, according to our experience most formulas are not purely positive. For those parts that are not in positive equality, our method is provably better: given the variables $\{v_0, \ldots, v_n\}$ belonging to the non-positive part, they assign variable v_i the range $\{1, \ldots, i\}$, resulting in a state space of $n!$, while we use the range allocation algorithm, which analyzes the formula's structure and allocates significantly smaller domains.

References

[Ack54] W. Ackerman, "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. Kirkpatrik and R. K. Brayton, "Verification Using Finite Instantiations and Uninterpreted Functions", In *Formal Methods in Computer Aided Design FMCAD 96* .

[PRSS98] A. Pnueli, Y. Rodeh, M. Seigel and O. Shtrichman, "Deciding Equality Formulas by Small Domain Instantiations", In *Computer-Aided Verification CAV 99* . 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] Y. Rodeh and O. Shtrichman, "Dynamic Range Allocation", Submitted to *Computer Aided Verification CAV 2001.*

[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* .