Building small equality graphs for deciding equality logic with Uninterpreted Functions

Yoav Rodeh a, Ofer Strichman b, *

a Tel-Hai College, Tel-Hai, Israel
b Information Systems Engineering, Technion, Haifa, Israel

Received 19 September 2004; revised 1 June 2005
Available online 13 September 2005

Abstract

The logic of equalities with Uninterpreted Functions is used in the formal verification community mainly for proofs of equivalence: proving that two versions of a hardware design are the same, or that input and output of a compiler are semantically equivalent are two prominent examples of such proofs. We introduce a new decision procedure for this logic that generalizes two leading decision procedures that were published in the last few years: the Positive Equality approach suggested by Bryant et al. [Exploiting positive equality in a logic of equality with uninterpreted functions, in: Proc. 11th Intl. Conference on Computer Aided Verification (CAV’99), 1999], and the Range-Allocation algorithm suggested by Pnueli et al. [The small model property: how small can it be? Information and Computation 178 (1) (2002) 279–293]. Both of these methods reduce this logic to pure Equality Logic (without Uninterpreted Functions), and then, due to the small model property that such formulas have, find a small domain to each variable that is sufficiently large to maintain the satisfiability of the formula. The state-space spanned by these domains is then traversed with a BDD-based engine. The Positive Equality approach identifies terms that have a certain characteristic in the original formula (before the reduction to pure Equality Logic), and replaces them with unique constants. The Range-Allocation algorithm analyzes the structure of the formula after the reduction to equality logic with a graph-based procedure to allocate a small set of values to each variable. The former, therefore, has an advantage when a large subset of the terms can be replaced with constants, and disadvantage in the other cases. In this paper we essentially merge the two methods, while improving both with a more careful analysis of the formula’s

* Corresponding author.
E-mail addresses: ofers@ie.technion.ac.il (O. Strichman), yoavr@telhai.ac.il (Y. Rodeh).

0890-5401/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2005.08.001
1. Introduction

The logic of Equalities with Uninterpreted Functions (EUF) has been used rather extensively in the last decade in the formal verification community, and several sophisticated decision procedures for this logic were suggested [12,3,15,4] (see [20] and Appendix A for a survey). Using Uninterpreted Functions rather than the original functions in the verification condition abstracts away information that is not necessarily needed for the proof, and hence simplifies and generalizes the proof. From a given EUF-formula $\varphi_{\text{EUF}}$, it is possible to derive an Equality formula ($E$-formula from now on) $\varphi^E$ that is satisfiable if and only if there exists a satisfying interpretation to $\varphi_{\text{EUF}}$. The reduction to Equality Logic must preserve the functional consistency property which is common to all functions, i.e., that they return the same value if instantiated with the same arguments. When functional consistency is all that is necessary for the proof, it simplifies greatly the task of performing it mechanically. It is only left, then, in these cases, to decide a pure E-formula.

Several examples of usage of this logic in the verification community are: proving equivalence between two versions of hardware designs [9,4]; Translation Validation [16], a process in which the correctness of a compiler’s translation is proven by checking the equivalence of the source and target codes; and checking a control property of a microprocessor, where it is sufficient to specify that the operations which the ALU perform are functions, rather than specifying what these operations are (thus avoiding the complexity of the ALU). This is the approach taken, for example, in [9], where a formula with Uninterpreted Functions is generated, such that its validity implies the equivalence between the two versions of the CPU, with and without a pipeline.

By now there are quite a few decision procedures for equality logic, most of which we cover in Appendix A. Here, we briefly mention two prior works that are directly related to what we suggest: the Positive Equality approach suggested by Bryant et al. [3] which was later extended to Robust Positive Equality by Lahiri et al. [13], and the Range-Allocation approach suggested by Pnueli, Rodeh, Strichman and Siegel in [15]. Both of these methods are instantiations of the following scheme:

1. Reduce $\varphi_{\text{EUF}}$ to an E-formula $\varphi^E$ such that $\varphi^E$ is satisfiable if and only if there exists an interpretation which satisfies $\varphi_{\text{EUF}}$.
2. Analyze the formula’s predicates (by examining a graph representation of the formula, as we will later explain) to calculate an adequate domain $R$ for each variable in $\varphi^E$, that is, a domain large enough so that if there exists a satisfying assignment for $\varphi^E$, then there exists such an assignment within this domain.
3. Check (symbolically) if any of the assignments in $R$ satisfy $\varphi^E$.

---

In Section 4, we will redefine this scheme more accurately after giving the necessary formal definitions. Our suggested procedure replaces Steps 1 and 2 with one improved step. The main idea is to exploit information which is only visible in the original formula $\varphi^{EUF}$ before its reduction to $\varphi^{E}$.

The Positive Equality approach identifies terms that have a certain syntactical characteristic in the original formula, and replaces them with unique constants. Briefly, these terms need to be compared to one another only under negative polarity and not be used as guards in ITE expressions (see Section 5), in order to be substituted with constants.

The Range-Allocation algorithm, on the other hand, does not make this distinction. It analyzes the structure of the formula after the reduction to Equality Logic with a graph-based procedure to allocate a small set of values to each variable. The equalities in the formula are represented as a graph called the E-graph, where the nodes are the variables and the edges are the equalities and disequalities (disequality standing for $\neq$) in $\varphi$. This graph represents an abstraction of the E-formula because it disregards its Boolean structure. Given this graph, the Range Allocation heuristic computes, in polynomial time, a small set of values for each variable that is sufficient to preserve the satisfiability of all satisfiable equality formulas with the same underlying E-graph. Positive Equality, therefore, has an advantage when a large part of the formula can be replaced with constants, and a disadvantage in the other cases. As for empirical evidence, apparently different sets of experiments result in different conclusions: in [15] we witnessed the superiority of the Range-Allocation algorithm over Positive Equality based on a set of benchmarks taken from the Translation Validation problem [17]. In [22], on the other hand, Velev and Bryant witnessed the opposite based on hardware examples, although it was based on a naive implementation of Range-Allocation [23]. They also witnessed the superiority of Positive Equality over the method of [12]. Appendix A contains more information on these earlier works.

In this paper, however, we hope to make these conflicting conclusions obsolete, as we essentially suggest an algorithm that enjoys both worlds: it allocates a single constant to any Positive Equality term, and a small range to the others. Further, it applies a more careful analysis of the formula’s structure, which results in guaranteed smaller E-graphs and hence smaller allocated ranges compared to [15]. We can therefore claim that our new algorithm is provably dominant over both methods, theoretically as well as empirically.

2. A motivating example and a road-map

The explanation of our method and the comparison to previous methods is rather lengthy and complicated. We therefore start with an example that uses some of the basic notions that we later formally define.

Consider the following simple satisfiable Equality formula with Uninterpreted Functions:

$$f(x_1) \neq f(x_2) \land f(x_1) = f(x_3) \land ((x_1 \neq x_2) \lor (x_1 \neq x_3)).$$

To reduce this formula to Equality Logic we use Bryant et al.’s reduction [3]:

$$f_1 \neq ITE(x_1 = x_2, f_1, f_2) \land f_1 = ITE(x_1 = x_3, f_1, ITE(x_2 = x_3, f_2, f_3)) \land ((x_1 \neq x_2) \lor (x_1 \neq x_3)).$$

(1)
Note how each Uninterpreted Function instance is replaced with an If-Then-Else (ITE) expression that refers to new term variables $f_1$, $f_2$, and $f_3$. Generally, the function instances of each function are ordered arbitrarily (in this case the order is $f(x_1)$, $f(x_2)$, $f(x_3)$), except when there are nested functions: in the latter case the order should respect the natural order defined by the subexpression relation: if an instance $a$ is an argument of instance $b$, then the index associated with instance $a$ should be smaller than the index associated with instance $b$. Given this order, the first instance is replaced with a new variable ($f_1$ in this case), and each consecutive instance is replaced with an ITE expression that maintains its functional consistency with the previous instances. In case of nested functions, only the most external function instance remains. In Section 5, we will describe this process in more detail.

For the sake of clarity, rather than using nested ITE expressions, as in (1), we will use ‘place-holders’ $F_1^* \ldots F_3^*$ for each Uninterpreted Function:

$$F_1^* \neq F_2^* \land F_1^* = F_3^* \land ((x_1 \neq x_2) \lor (x_1 \neq x_3)),$$

where

$$F_1^* := f_1;$$
$$F_2^* := \begin{cases} f_1 & x_1 = x_2; \\ f_2 & x_1 = x_2 \land f_1 \neq f_1; \\ \text{TRUE}. & \end{cases}$$
$$F_3^* := \begin{cases} f_1 & x_1 = x_3; \\ f_2 & x_2 = x_3; \\ f_3 & \text{TRUE}. \end{cases}$$

As a second step we build the E-graph corresponding to this formula. An E-graph has a node for each variable in the formula, a dashed edge for each equality, and a solid edge for each disequality. We consider the polarity of each edge after all negations are pushed to the atoms and ITE expressions are ‘flattened.’ For example, the left conjunct in our formula is replaced with

$$(x_1 = x_2 \land f_1 \neq f_1) \lor (x_1 \neq x_2 \land f_1 \neq f_2).$$

This gives us the following graph:

Note how $x_1$, $x_2$, and $x_3$ are connected to each other with both type of edges. This is because of the predicates comparing them in the ITE expressions. Indeed, as can be seen in the example above (3), the conditions in the ITE expression are evaluated under both polarities ($x_1 = x_2$ and $x_1 \neq x_2$).

Next, we analyze this graph and find an adequate range of values to each variable. By adequate we mean that every satisfiable subset of edges (equality predicates) can be satisfied from these ranges (we define this concept formally in Section 3). Without going into the details of how we perform this
analysis (see [18,15] for more details, and also in later sections), the following allocation of values is adequate for this graph:

\[
\begin{align*}
  f_1 &\mapsto \{0\}, & f_2 &\mapsto \{1\}, & f_3 &\mapsto \{0\}, \\
  x_1 &\mapsto \{0\}, & x_2 &\mapsto \{0,1\}, & x_3 &\mapsto \{0,1,2\}.
\end{align*}
\]

For example, the subset of predicates

\[
\{(f_1 \neq f_2), (x_1 = x_2), (x_2 \neq x_3), (x_3 \neq x_1)\}
\]

can be satisfied from the above ranges with the assignment

\[
(f_1, f_2, x_1, x_2, x_3) \leftarrow (0,1,0,0,1).
\]

As a third step, we use either SAT or BDDs to traverse the allocated finite range to find a satisfying assignment, if one exists.

We go in this article one step further and generate a smaller E-graph that results in drastically smaller ranges. For the example above we build the following graph:

In contrast to the previous graph, here we do not automatically add edges that correspond to the conditions in the ITE expressions. Instead, we analyze the comparisons between the function instances. Since there is a disequality edge between \(f_1\) and \(f_2\), then we need to allow their respective arguments to be different: this is why we need a disequality edge between \(x_1\) and \(x_2\). Similarly, since there is an equality edge between \(f_1\) and \(f_3\), we need to allow their respective arguments to be equal to one another. This is why we add an equality edge between \(x_1\) and \(x_3\). The generalization of this mechanism begins in Section 6. An adequate range for the smaller E-graph is

\[
\begin{align*}
  f_1 &\mapsto \{0\}, & f_2 &\mapsto \{1\}, & f_3 &\mapsto \{0\}, \\
  x_1 &\mapsto \{0\}, & x_2 &\mapsto \{1\}, & x_3 &\mapsto \{0,1\}
\end{align*}
\]

which represents a state-space of 2 (the size of the product domain). The previous construction resulted in a state-space of 6.

Empirically, most of the edges in E-graphs originate from these conditions. In particular, there is a clique between all the arguments corresponding to instantiations of the same function (in our example these are \(x_1\), \(x_2\), and \(x_3\)). Our experimental results show a reduction of tens of orders of magnitudes in state-spaces (e.g., from \(10^{20}\) to 10) and a reduction of solving time that made it possible to solve instances that could not be solved with the original E-graph construction. Such a drastic reduction occurs even with relatively small graphs that have several dozen nodes: while previously
each node was allocated several dozen values, most of them are allocated a unique constant with our method.

Since our method generalizes the Positive-Equality method, let us just mention that the latter allocates, in the above example, a constant to \( f_2 \), say \( \{0\} \), and five values, e.g. \( \{1, 2, 3, 4, 5\} \), to all other variables. This allocation results in a state-space of \( 5^5 = 3125 \).

3. Preliminaries and definitions

We define the logic of Equality with Uninterpreted Functions formally. The syntax of this logic is defined as follows:

\[
\langle \text{Formula} \rangle \leftarrow \langle \text{Boolean-Variable} \rangle \\
| \langle \text{Predicate-Symbol} \rangle(\langle \text{Term} \rangle, \ldots, \langle \text{Term} \rangle) \\
| \langle \text{Term} \rangle = \langle \text{Term} \rangle \\
| \neg \langle \text{Formula} \rangle \\
| \langle \text{Formula} \rangle \lor \langle \text{Formula} \rangle \\
| \text{ITE}(\langle \text{Formula} \rangle, \langle \text{Formula} \rangle, \langle \text{Formula} \rangle)
\]

\[
\langle \text{Term} \rangle \leftarrow \langle \text{Term-Variable} \rangle \\
| \langle \text{Function-Symbol} \rangle(\langle \text{Term} \rangle, \ldots, \langle \text{Term} \rangle) \\
| \text{ITE}(\langle \text{Formula} \rangle, \langle \text{Term} \rangle, \langle \text{Term} \rangle)
\]

We will assume that term-sharing is allowed.

We refer to formulas in this logic as EUF-formulas. We say that an EUF-formula \( \varphi_{\text{EUF}} \) is valid if and only if for every interpretation \( \mathcal{M} \) of the variables, functions and predicates of \( \varphi_{\text{EUF}} \), \( \mathcal{M} \models \varphi_{\text{EUF}} \).

An equivalence logic formula, denoted by \( \text{E-formula} \), is an EUF-formula that does not contain any function and predicate symbols. Throughout the paper we use \( \varphi_{\text{EUF}} \) and \( \varphi_{\text{E}} \) to denote EUF-formulas and E-formulas, respectively.

We will start by considering the problem of deciding the satisfiability of E-formulas that do not contain ITE terms, Boolean variables, and predicates. As our presentation proceeds, we will consider the more general problem.

4. Deciding satisfiability of simple E-formulas

We wish to check the satisfiability of an E-formula \( \varphi_{\text{E}} \) with term-variable set \( V \). In theory this implies that we need to check whether there exists some assignment \( \alpha \) to the term-variables of \( V \) that satisfies \( \varphi_{\text{E}} \) (marked \( \alpha \models \varphi_{\text{E}} \)). It is clear that it is enough to check assignments that assign only natural numbers, i.e., \( \alpha : V \rightarrow \mathbb{N} \), but this still implies checking an infinite set of assignments. However, since \( \varphi_{\text{E}} \) 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 and bounded domain.

In the case of Equality Logic, the range \( \{1 \ldots |V|\} \) for each variable is sufficient. A better range (and also the lower bound in the worst case) is \( 1 \ldots i \) to the \( i \)th variable, according to some arbitrary ordering of the variables [15].
The fact that a satisfying assignment exists from a range polynomial in the number of variables if the formula is satisfiable shows that deciding satisfiability of E-formulas is in NP, and therefore is clearly NP-complete (via a trivial reduction from deciding satisfiability of Boolean formulas). However, this approach is not very practical, as it implies going over |V|! assignments. Note, however, that this is already better, at least theoretically, than adding explicit transitivity constraints as in [5] that results in a state-space of O(2^V^2) and a larger formula.

In [15] we suggested a more refined analysis, where rather than considering only the number of variables |V|, we examine the actual structure of ϕ^E, or more specifically, the equality predicates in ϕ^E. This analysis enables the derivation of a state-space which is empirically much smaller than |V|!. In this section we repeat the essential definitions from [15], except for several changes which are necessary for the new techniques.

**Definition 1 [E-graphs].** An E-graph G is a triplet G = ⟨V, E =, E≠⟩, where V is the set of vertices, and E = (Equality edges) and E≠ (Disequality edges) are sets of unordered pairs of vertices.

From a given E-formula ϕ^E in Negation Normal Form (all negations are pushed to the atoms), we construct the E-graph G(ϕ^E): G(ϕ^E) contains a vertex for each term-variable of ϕ^E, and an edge for each equality or disequality of ϕ^E.

Given an E-graph G = ⟨V, E =, E≠⟩, we denote V(G) = V, E±(G) = E≠ and E±(G) = E±. We use ≤ to denote the subgraph relation: H ≤ G if and only if E±(H) ⊆ E±(G) and E≠(H) ⊆ E≠(G).

We say that an assignment α satisfies an edge e = (a, b) (marked α |= e) if e is an equality edge (e ∈ E±(G)) and α(a) = α(b), or if e is a disequality edge (e ∈ E≠(G)) and α(a) ≠ α(b). We denote α |= G if α satisfies all edges of G. G is said to be satisfiable if there exists some α such that α |= G.

**Example 1.** The E-formula ϕ^E_1: (a = b) ∧ ((c ≠ b) ∨ (a = c)) results in the E-graph depicted in Fig. 1.

One may view G(ϕ^E) as a conservative abstraction of ϕ^E, as it contains all the atomic equalities appearing in ϕ^E and their polarities, yet has no representation for the Boolean relation between them.

The property of G(ϕ^E) that interests us the most is the fact that satisfaction of ϕ^E depends only on the satisfaction of the sub-formulas represented by the edges of G(ϕ^E). More formally,

**Proposition 1.** Given assignments α and β over V(G(ϕ^E)), if for every edge e of G(ϕ^E), α |= e ↔ β |= e, then α |= ϕ^E ↔ β |= ϕ^E.

This implies that if we want to check whether ϕ^E is satisfiable, it is sufficient to check one satisfying assignment per satisfiable subgraph of G(ϕ^E). More generally,
**Definition 2** [adequacy of assignment sets to E-graphs]. Given an E-graph $G$, and $R$, a set of assignments to $V(G)$, we say that $R$ is adequate for $G$ if for every satisfiable $H \leq G$, there is $\alpha \in R$ such that $\alpha \models H$.

**Example 2.** The following set is adequate for the E-graph of Fig. 1:

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

In [18,15] we presented, together with Pnueli and Siegel, the Range-Allocation algorithm that analyzes the E-graph in polynomial time and finds adequate domains, i.e., a set of values to each variable, from which it is possible to derive an adequate assignment set. Enumerating the possible assignments symbolically is left for the BDD package in the last stage. For example, adequate domains for Example 2 are

$$a \mapsto \{0\}, b \mapsto \{0, 1\}, c \mapsto \{0, 1\}.$$

Note how the assignment sets of Example 2 can be derived from these domains. We will not repeat the details of the Range-Allocation algorithm here. However, we should mention that the size of the allocated domains, which reflect the overall search-space in the last stage, grows monotonically with the E-graph that it reads as input. That is, given the E-graphs $H$ and $G$ such that $H \leq G$, the Range-Allocation algorithm guarantees that the domain allocated for $H$ is smaller or equal to the domain allocated for $G$. This clearly justifies the motivation behind the current work: we will build far smaller graphs that consequently result in drastically smaller search-spaces.

The property of $G(\varphi^e)$ stated in Proposition 1 is what makes this technique correct. However, we can use a weaker property of E-graphs.

**Definition 3** [an E-graph satisfies an E-formula]. For a satisfiable E-graph $G$, we say that $G \models \varphi^e$ if $G$ has the same edges (equality predicates) as $\varphi^e$, and for every assignment $\alpha$ such that $\alpha \models G$, $\alpha \models \varphi^e$.

Less formally, this definition relates an Equality subgraph to the set of predicates that it represents: if the satisfaction of this set guarantees the satisfaction of $\varphi^e$, then we say that this subgraph satisfies $\varphi^e$.

**Definition 4** [adequacy of E-graphs to E-formulas]. An E-graph $G$ is adequate for E-formula $\varphi^e$, if either $\varphi^e$ is not satisfiable, or there exists a satisfiable $H \leq G$ such that $H \models \varphi^e$.

Clearly, $G(\varphi^e)$ is adequate for $\varphi^e$.

**Example 3.** If we remove the edge between $c$ and $b$ in the E-graph of Fig. 1, the remaining E-graph is still adequate for $\varphi^e_1$.

We claim:

**Proposition 2.** If E-graph $G$ is adequate for $\varphi^e$, and assignment set $R$ is adequate for $G$, then $\varphi^e$ is satisfiable iff there is $\alpha \in R$ such that $\alpha \models \varphi^e$.

Let us now rephrase the decision procedure for the satisfiability of an input EUF-formulas $\varphi^{EUF}$ as suggested in [15] according to the above definitions:
(1) Reduce $\varphi^{EUF}$ to an E-formula $\varphi^E$ such that $\varphi^E$ is satisfiable if and only if there exists an interpretation which satisfies $\varphi^{EUF}$.
(2) Construct the E-graph $G(\varphi^E)$.
(3) Calculate an adequate domain $R$ for $G(\varphi^E)$.
(4) Check (symbolically) if any of the assignments in $R$ satisfy $\varphi^E$.

Our suggested procedure replaces Steps 1 and 2 with a single step. The main idea is to exploit information which is only visible in the original formula $\varphi^{EUF}$ before its reduction to $\varphi^E$.

5. Reducing Uninterpreted Functions to Equality Logic

Given an EUF-formula $\varphi^{EUF}$, we wish to generate an E-formula $\varphi^E$ such that $\varphi^{EUF}$ is satisfiable iff $\varphi^E$ is satisfiable.

We will use the following EUF-formula throughout this section to illustrate the reduction:

$$\varphi^{EUF}_1 := F(F(F(x_1))) \neq F(F(x_1)) \land$$
$$F(F(x_1)) \neq F(x_2) \land$$
$$x_2 = F(x_1).$$

For each function symbol (only $F$ in this case) we number the function instances from the inside-out, and give equal indices to instances with syntactically equivalent arguments. Thus, the numbering respects the sub-term ordering of $\varphi^{EUF}$. In other words, for every two function instances $F_i$ and $F_j$, if $F_i$ appears as a sub-term of the term $F_j(\ldots)$ in $\varphi^{EUF}$, then we must have $i < j$. This results in

$$\varphi^{EUF}_1 := F_4(F_3(F_1(x_1))) \neq F_3(F_1(x_1)) \land$$
$$F_3(F_1(x_1)) \neq F_2(x_2) \land$$
$$x_2 = F_1(x_1).$$

For function instance $F_i$ of $\varphi^{EUF}$, define $\text{arg}_l(F_i)$ to be the term of $\varphi^{EUF}$ corresponding to the $l$-th argument of $F_i$.

Example 4.

$$\text{arg}_1(F_1) = x_1$$
$$\text{arg}_1(F_4) = F_3(F_1(x_1)).$$

The following translation is due to Bryant et al. [6,3]. We denote the resulting formula from this translation with $T(\varphi^{EUF})$. $T(\varphi^{EUF})$ is given by replacing the function instance $F_i$ in $\varphi^{EUF}$ with the term $F_i^*$ for all $i$,

$$F_i^* = \begin{cases} 
 f_1 & T(\text{arg}_1(F_i)) = T(\text{arg}_1(F_i)), \\
 f_2 & T(\text{arg}_1(F_i)) = T(\text{arg}_1(F_2)), \\
 \vdots & \vdots \\
 f_{i-1} & T(\text{arg}_1(F_i)) = T(\text{arg}_1(F_{i-1})), \\
 f_i & \text{TRUE.}
\end{cases}$$
Note that since the numbering of the function instances respects the sub-term ordering of $\phi^{EUF}$, it is guaranteed that no cyclic definition is possible. The $F^*$ symbols should be thought of as ‘place holders’ only necessary for convenient notation. Practically only the expressions that they represent are present in the translated formula, in the form of ITE expressions as was shown before.

**Example 5.** Given $\phi^{EUF}_1$ from Eq. 5, $T(\phi^{EUF}_1)$ is given by:

$$T(\phi^{EUF}_1) := (F^*_4 \neq F^*_3) \land (F^*_3 \neq F^*_2) \land (x_2 = F^*_1),$$

where

$$F^*_1 := f_1; \quad F^*_2 := \left\{ \begin{array}{ll}
  f_1 & x_1 = x_2; \\
  f_2 & \text{TRUE};
\end{array} \right.$$

$$F^*_3 := \left\{ \begin{array}{ll}
  f_1 & F^*_1 = x_1; \\
  f_2 & F^*_2 = x_2; \\
  f_3 & \text{TRUE};
\end{array} \right.$$

$$F^*_4 := \left\{ \begin{array}{ll}
  f_1 & F^*_3 = x_1; \\
  f_2 & F^*_3 = x_2; \\
  f_3 & F^*_3 = F^*_1; \\
  f_4 & \text{TRUE}.
\end{array} \right.$$

6. E-graph construction: an informal discussion

Given an EUF-formula $\phi^{EUF}$, we wish to construct a minimal adequate E-graph for $T(\phi^{EUF})$. In this section we try to explain the intuition behind our suggested construction of E-graphs, which we term *Minimal-E*. In this section we will ignore ITE expressions, predicates, and Boolean variables for simplicity. These will be handled in later sections.

For a $\phi^{EUF'}$, a sub-formula or sub-term of $\phi^{EUF}$, we define $\text{simp}(\phi^{EUF'})$ to be the result of replacing in $\phi^{EUF'}$ every function instance $F_i$ by a new term-variable $f_i$. For example:

**Example 6.**

$$\text{simp}(F_4(F_3(F_1(y)))) = f_4,$$

$$\text{simp}(\text{arg}_1(F_3)) = f_1,$$

$$\text{simp}(\phi^{EUF}_1) = ((f_4 \neq f_3) \land (f_3 \neq f_2) \land (x = f_1)).$$

We will explain the intuition behind the construction with a series of attempts, each one improving upon the previous attempt. We begin with a naive approach in which we build a graph only according to $\text{simp}(\phi^{EUF})$. Consider $\phi^{EUF}_5$:

$$\phi^{EUF}_5 := F_1(x_1) \neq F_2(x_2) \land ((x_1 = x_2) \lor \text{TRUE})$$

for which

$$\text{simp}(\phi^{EUF}_5) := f_1 \neq f_2 \land ((x_1 = x_2) \lor \text{TRUE})$$
Fig. 2. An E-graph based on $\text{simp}(\varphi_{EUF}^5)$.

\[
T(\varphi_{EUF}^5) := F_1^* \neq F_2^* \land ((x_1 = x_2) \lor \text{TRUE})
\]  

(13)

together with:

\[
\begin{align*}
F_1^* & := f_1; \\
F_2^* & := \begin{cases} f_1 & x_1 = x_2; \\
                    f_2 & \text{TRUE}. \end{cases}
\end{align*}
\]

(14)

Obviously any decent procedure will remove the right clause in $T(\varphi_{EUF}^5)$, but this TRUE can be hidden as a more complex valid formula. Suppose that we build a graph only according to the top-level formula, $\text{simp}(\varphi_{EUF}^5)$. The corresponding E-graph contains one disequality edge between $f_1$ and $f_2$, and one equality edge between $x_1$ and $x_2$, as appears in Fig. 2.

A possible assignment set for this graph can contain the single assignment:

\[
(x_1, x_2, f_1, f_2) \leftarrow (0, 0, 2, 3)
\]

(14)

which does not satisfy $T(\varphi_{EUF}^5)$. This is because the graph fails to represent the fact that $f_1 \neq f_2$ implies $x_1 \neq x_2$. Adding a disequality edge between $x_1$ and $x_2$ can solve this problem, since it forces the allocated domain to include at least one assignment such as

\[
(x_1, x_2, f_1, f_2) \leftarrow (0, 1, 2, 3)
\]

which satisfies $T(\varphi_{EUF}^5)$ (an adequate domain can be $R : x_1 \mapsto \{0\}, x_2 \mapsto \{0, 1\}, f_1 \mapsto \{2\}, f_2 \mapsto \{3\}$).

But how do we generalize this case? Suppose we say that we need to add a disequality edge between the arguments $x_i, x_j$ of $f_i$ and $f_j$ if $(f_i, f_j) \in E_\neq$ and $(x_i, x_j) \in E_\neq$. This indeed solves the case of $\varphi_{EUF}^5$, but consider now $\varphi_{EUF}^6$:

\[
\begin{align*}
\varphi_{EUF}^6 & := (F_1(x_1) = z) \land (F_2(x_2) \neq z) \land ((x_1 = x_2) \lor \text{TRUE}), \\
\text{simp}(\varphi_{EUF}^6) & := f_1 = z \land (f_2 \neq z) \land ((x_1 = x_2) \lor \text{TRUE}).
\end{align*}
\]  

(15)

The E-graph for $\varphi_{EUF}^6$ appears as $G_1$ in Fig. 3. As before dashed lines represent equality edges and solid lines represent disequality edges. In the case of $\varphi_{EUF}^6$, the above stated rule does not apply, and
we are left with the same problem, since a possible adequate domain for this E-graph can contain the single assignment:

\[(x_1, x_2, z, f_1, f_2) \leftarrow (0, 0, 1, 1, 2)\]

which does not satisfy \(T(\phi_{EUF}^6)\).

This is because there is no disequality edge between \(f_1\) and \(f_2\), but nevertheless the disequality between them is implied by the path through \(z\). So we need to generalize our rule in a way that it refers to disequality paths instead of disequality edges, and equality paths instead of equality edges. This will enable us to identify implied equality and disequality requirements.

**Definition 5 [Equality Path].** There is an Equality Path between \(u\) and \(v\) in \(G\), denoted \(u =^G_v\), if there is a simple path in \(G\) between \(u\) and \(v\) in \(E\).

**Definition 6 [Disequality Path].** There is a Disequality Path between \(u\) and \(v\) in \(G\), denoted \(u \neq^G_v\), if there is a simple path in \(G\) between \(u\) and \(v\) such that one edge in the path is from \(E\) and all other edges are from \(E\).

We can now define our first rule:

**Rule 1.** For \(f_i\) and \(f_j\) with arguments \(x_i\) and \(x_j\), respectively, if \(f_i \neq^G f_j\) and \(x_i =^G x_j\), then add a disequality edge between \(x_i\) and \(x_j\).

Applying this rule to \(\phi_{EUF}^6\), we get \(G_2\) of Fig. 3, which solves this case. We proceed by considering a similar EUF-formula,

\[
\phi_{EUF}^7 = (\text{true} \lor (F_1(x_1) = z)) \land (F_2(x_2) \neq z) \land (x_1 = x_2) \tag{17}
\]

\(G(\text{simp} (\phi_{EUF}^7))\) is exactly the same as before (\(G_1\) in Fig. 3, and Rule 1 adds the disequality edge \((x_1, x_2)\) to give \(G_2\) in Fig. 3. The problem here is that a satisfying assignment \(\alpha\) must satisfy \(\alpha(x_1) = \alpha(x_2)\), and therefore \(\alpha(F_2^\bullet) = \alpha(F_1^\bullet)\). Since we also must have \(\alpha(F_2^\bullet) \neq \alpha(z)\) to satisfy the formula, it implies \(\alpha(F_1^\bullet) \neq \alpha(z)\). This may not necessarily happen in any assignment in an adequate assignment set for our E-graph. This is because in our E-graph there is no representation for the fact that \(f_1\) may “override” \(f_2\), i.e., if \(x_1 = x_2\) then \(F_2^\bullet\) is evaluated to \(f_1\). If we add an equality edge between \(f_1\) and \(f_2\) it will solve the problem. \(G_3\) of Fig. 3 is the result of adding this edge.
(Suggested) Rule 2. For $f_i$ and $f_j$, with $x_i$ and $x_j$ their corresponding arguments, if $x_i =^*_G x_j$ then add the equality edge $(f_i, f_j)$.

This indeed solves our problem, but is not the best that we can do. We added an equality edge between $f_1$ and $f_2$ in our example, but it is not really necessary. Instead, we can copy 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 to satisfy $\varphi^E$, when using Bryant et al.’s reduction (because in this case both $F_1^*$ and $F_2^*$ are assigned the value of $f_1$). But if $F_2^*$ is assigned the value of $f_1$, then we need to make sure that $f_1$ can satisfy all the constraints over $F_2^*$. Allowing $f_1$ to be equal to $f_2$ in the E-graph is only one way to do this (the Range Allocation algorithm guarantees that every two variables with an Equality Path between them can satisfy the same constraints), but there is another way as well: simply copy all constraints over $f_2$ to $f_1$. Notice that this case is asymmetric: since $f_1$ may override $f_2$, only $f_1$ is required to respect $f_2$’s requirements. The additional option can help us add less equality edges, which in general impose larger allocated domains.

We therefore change the suggested rule above 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 $x_i =^*_G x_j$ then do one of the following:

1. add equality edge $(f_i, f_j)$, or
2. (a) for every equality edge $(f_j, w)$ add an equality edge $(f_i, w)$, and
   (b) for every disequality edge $(f_j, w)$ add a disequality edge $(f_i, w)$.

And so, in our example, instead of adding an equality edge $(f_1, f_2)$, we can add a disequality edge $(f_1, z)$, which results in $G_4$ of Fig. 3. The formalization of applying Rule 2 requires a special graph called *assignment graph* that we will show in the next section.

Note that the asymmetry between $f_1$ and $f_2$ suggests that there is another optimization problem here: the function elimination order (the indices that we give to function instances) is not unique, as there are many orders that respect the sub-term ordering between function applications. We can do better than just selecting between them arbitrarily if we select an order that minimizes the resulting allocated domain (minimizing the number of added dashed edges by Rule 2 is a good strategy to achieve this goal). This type of optimal ordering is the main idea behind the Robust Positive Equality method of [13], although there the goal was somewhat different. We adopt their strategy for MINIMAL-E nevertheless to generalize their result. More details about this optimization problem appear in Appendix C.

Finally, consider the simple formula:

$$\varphi^E_{8} = F_1(x_1) \neq F_2(x_2). \quad (18)$$

The E-graph now only contains $x_1, x_2, f_1,$ and $f_2$ as variables and only one disequality edge between $f_1$ and $f_2$. Rule 1 does not apply here because there is no equality path between $x_1$ and $x_2$. We need to check whether an adequate range for this graph satisfies $T(\varphi^E_{8})$:

$$T(\varphi^E_{8}) : F_1^* \neq F_2^* \quad (19)$$
and

\[ F_1^* := f_1 \quad F_2^* := \begin{cases} f_1 & x_1 = x_2; \\ f_2 & \text{TRUE}. \end{cases} \]

A possible adequate domain for the corresponding graph contains just one assignment, for example:

\[ (x_1, x_2, f_1, f_2) \leftarrow (0, 0, 0, 1). \quad (20) \]

This assignment does not satisfy \( T(\phi_{\text{EUF}}) \), and so our attempt fails. What we need to do is to ensure that the allocated domain guarantees a certain diversity property, i.e., that unless the graph requires otherwise, it gives different values to different variables. The good news about this requirement is that it does not increase the size of the necessary domain. Let us denote this rule as Rule 3:

**Rule 3.** If \( u \not= v \) does not hold then add a disequality edge between \( u \) and \( v \).

This rule should be applied last, after we know all the requirements over the edges.

Now, applying Rule 3 results in an E-graph which is a solid clique between the nodes \( \{x_1, x_2, f_1, f_2\} \). A possible adequate assignment set for this E-graph contains one assignment,

\[ (x_1, x_2, f_1, f_2) \leftarrow (1, 2, 3, 4) \]

which satisfies \( T(\phi_{\text{EUF}}) \).

**Summary.** MINIMAL-E constructs an E-graph from an EUF-formula \( \varphi_{\text{EUF}} \) as appears in Algorithm 1. Notice that this construction somewhat reminds a cone-of-influence reduction, since in \( \text{simp}(\varphi_{\text{EUF}}) \) the arguments of Uninterpreted Functions disappear, and then only edges emanating from edges already in the E-graph are added.

**Algorithm 1** The MINIMAL-E algorithm for an EUF-formula \( \varphi_{\text{EUF}} \) without ITE expressions, predicates, and Boolean variables.

1. Construct \( \mathcal{G}(\text{simp}(\varphi_{\text{EUF}})) \).
2. Apply Rules 1 and 2 until no new edges are added.
3. Apply Rule 3.

The construction so far did not consider the case in which we have ITE expressions in the original formula (not those that result from the reduction of Uninterpreted Functions), which complicates things. We delay the treatment of these expressions to Section 8. In that section we will also give the full construction and proof, which involves new kind of graphs called Assignment Graphs. This is the topic of the next section.
7. Assignment Graphs

We now return to work our way towards our main result, a graph construction for general EUF-formulas that will generalize the results of [3]. We will also need a new kind of graph, called an Assignment Graph, or A-graph for short.

**Definition 7 [A-graph].** An A-graph is a quadruple $G \uparrow = \langle V, E=, E\neq, E\uparrow \rangle$, where $V$ is the set of vertices, $E=, E\neq$ are sets of unordered pairs of vertices and $E\uparrow$ (Assignment edges) is a set of ordered pairs of vertices.

Given an A-graph $G \uparrow = \langle V, E=, E\neq, E\uparrow \rangle$, we denote $V(G \uparrow) = V$, $E\neq(G \uparrow) = E\neq$, $E=(G \uparrow) = E= and E\uparrow(G \uparrow) = E\uparrow$. We denote $a \rightarrow_{G \uparrow} b, if there is a directed path (possibly of length 0) of assignment edges from $a$ to $b$ in $G \uparrow$.

Assignment edges (edges in $E\uparrow$) serve as a weak form of equality. Intuitively, the meaning of an assignment edge from $a$ to $b$ is that the value of $b$ is ‘overridden’ by $a$’s value, i.e., all edges adjacent to $b$ are checked against $a$’s value in addition to being checked against $b$’s value. The relevance of this term to Rule 2 is clear.

Formally, for an assignment $\alpha$ and an A-graph $G \uparrow$, we denote $\alpha \models G \uparrow$ if for every $a \rightarrow_{G \uparrow} b$ and $c \rightarrow_{G \uparrow} d$:

1. If $(b, d) \in E=(G \uparrow)$, $\alpha(a) = \alpha(c)$.
2. If $(b, d) \in E\neq(G \uparrow)$, $\alpha(a) \neq \alpha(c)$.

Note that $\alpha \models G \uparrow$ implies that $\alpha$ satisfies all equality and disequality edges of $G \uparrow$ (by setting the paths to be of length 0). For an A-graph $G \uparrow$ denoted by $flatEq \ G \uparrow$ the E-graph was obtained by replacing all assignment edges of $G \uparrow$ by equality edges. We say that an A-graph $G \uparrow$ is satisfiable if $flatEq \ G \uparrow$ is satisfiable. This is not the most natural definition, but we will need it for our translation from A-graphs to E-graphs (Section 9). Note that if $\alpha \models flatEq \ G \uparrow$ then $\alpha \models G \uparrow$.

However, the fact that $\alpha \models G \uparrow$ does not necessarily imply that $\alpha \models flatEq \ G \uparrow$. For example, the assignment $(x, y, z) \leftarrow (1, 2, 3)$ satisfies the A-graph $G \uparrow$ that has one disequality edge between $y$ and $z$ and one assignment edge from $x$ to $y$. On the other hand, this assignment does not satisfy $flatEq \ G \uparrow$.

For an E-formula $\varphi^e$ and a satisfiable A-graph $G \uparrow$, we denote $G \uparrow \models \varphi^e$ if for every assignment $\alpha$ such that $\alpha \models G \uparrow$ we have $\alpha \models \varphi^e$. The following definitions are the exact analog of the definitions associated with E-graphs:

**Definition 8 [adequacy of A-graphs to E-formulas].** An A-graph $G \uparrow$ is adequate for E-formula $\varphi^e$, if either $\varphi^e$ is not satisfiable, or there exists a satisfiable $H \uparrow \leq G \uparrow$ such that $H \uparrow \models \varphi^e$.

**Definition 9 [adequacy of assignment sets to A-graphs].** Given an A-graph $G \uparrow$, and $R$, a set of assignments to $V(G \uparrow)$, we say that $R$ is adequate for $G \uparrow$ if for every satisfiable $H \uparrow \leq G \uparrow$ there is $\alpha \in R$ such that $\alpha \models H \uparrow$.

The analogous proposition follows:

**Proposition 3.** If A-graph $G \uparrow$ is adequate for $\varphi^e$, and assignment set $R$ is adequate for $G \uparrow$, then $\varphi^e$ is satisfiable iff there is $\alpha \in R$ such that $\alpha \models \varphi^e$. 
In the following section, given an EUF-formula \( \varphi^{EUF} \), we will construct an A-graph \( G \uparrow \) such that \( G \uparrow \) is adequate for \( T(\varphi^{EUF}) \). In Section 9 we will show how to create an E-graph \( G \) from \( G \uparrow \), such that if \( R \) is adequate for \( G \) it is also adequate for \( G \uparrow \). These two procedures combined with a procedure for finding an adequate assignment set for an E-graph give us a more efficient decision procedure for the satisfiability of EUF-formulas.

8. The minimal A-graph construction

In this section, we describe and prove the A-graph construction for general EUF-formulas (including ITE expressions). Note that we still do not handle predicates and Boolean variables. They will be discussed in Section 11.

Example 7. In the following example we write \( F_i(\cdot) \) whenever \( F_i \)'s arguments were already specified for better readability:

\[
\varphi^E_{9} := F_2(ITE(F_1(b) = u, a, b)) = u \land F_3(F_2(\cdot)) \neq F_4(ITE(a = b, u, a)).
\]

(21)

The resulting E-formula is:

\[
T(\varphi^E_{9}) := (F_2^* = u) \land (F_3^* \neq F_4^*)
\]

(22)

\[
A_1 := b \quad F_1^* := f_1;
\]

\[
A_2 := ITE(F_1^*(b) = u, a, b) \quad F_2^* := \begin{cases} f_1 & A_2 = A_1; \\ f_2 & \text{TRUE}; \end{cases}
\]

\[
A_3 := F_2^* \quad F_3^* := \begin{cases} f_1 & A_3 = A_1; \\ f_2 & A_3 = A_2; \\ f_3 & \text{TRUE}; \end{cases}
\]

\[
A_4 := ITE(a = b, u, a) \quad F_4^* := \begin{cases} f_1 & A_4 = A_1; \\ f_2 & A_4 = A_2; \\ f_3 & A_4 = A_3; \\ f_4 & \text{TRUE}; \end{cases}
\]

where \( A_i \) is the term corresponding to \( ARG_1(F_i) \) in the translated formula.

In the following discussion the distinction between \( F_i, f_i, \) and \( F_i^* \) is crucial. \( F_i \) is the \( i \)th function instance of an EUF-formula \( \varphi^{EUF} \) (according to our predetermined numbering of the function instances of \( \varphi^{EUF} \)). \( f_i \) is the term-variable of \( T(\varphi^{EUF}) \) that was introduced by the reduction, and \( F_i^* \) is the term of \( T(\varphi^{EUF}) \) which replaces \( F_i \) in \( \varphi^{EUF} \).

8.1. Definitions

We start with some notations for an EUF-formula \( \varphi^{EUF} \), assignment \( \alpha \) to the variables of \( T(\varphi^{EUF}) \), and an A-graph \( G \uparrow \). For this purpose, we will use the following example assignment \( \gamma \) to \( T(\varphi^E_{9}) \)'s variables:

\[
\gamma(a) = 0, \quad \gamma(b) = 0, \quad \gamma(u) = 1, \\
\gamma(f_1) = 1, \quad \gamma(f_2) = 2, \quad \gamma(f_3) = 3, \quad \gamma(f_4) = 4.
\]
(1) For a term \( t \) of \( T(\varphi^{EUF}) \), define \( \alpha(t) \) to be the evaluation of \( t \) under \( \alpha \).

**Example 8.** In \( \varphi_9^{EUF} \),
\[
\begin{align*}
\gamma(A_1) &= \gamma(b) = 0, & \gamma(F_1^\bullet) &= \gamma(f_1) = 1, \\
\gamma(A_2) &= \gamma(a) = 0, & \gamma(F_2^\bullet) &= \gamma(f_1) = 1, \\
\gamma(A_3) &= \gamma(F_2^\bullet) = 1, & \gamma(F_3^\bullet) &= \gamma(f_3) = 3, \\
\gamma(A_4) &= \gamma(u) = 1, & \gamma(F_4^\bullet) &= \gamma(f_3) = 3.
\end{align*}
\]

(2) For a term-variable \( v \) of \( T(\varphi^{EUF}) \), define \( \text{source}_\alpha(v) \):
- If \( v \) is an original term-variable of \( \varphi^{EUF} \) then \( \text{source}_\alpha(v) = v \).
- If \( v = f_j \), then take the minimal \( i \) such that for all \( l \), \( \alpha(T(\text{arg}_l(F_j))) = \alpha(T(\text{arg}_l(F_j))) \), and set \( \text{source}_\alpha(v) = f_i \).

**Example 9.** In \( \varphi_9^{EUF} \):
\[
\begin{align*}
\text{source}_\gamma(a) &= a & \text{source}_\gamma(b) &= b & \text{source}_\gamma(u) &= u \\
\text{source}_\gamma(f_1) &= f_1 & \text{source}_\gamma(f_2) &= f_1 & \text{source}_\gamma(f_3) &= f_3 & \text{source}_\gamma(f_4) &= f_3
\end{align*}
\]

(3) Define the assignment \( \hat{\alpha} \) to be:
- For \( v \), a term-variable of \( \varphi^{EUF} \), set \( \hat{\alpha}(v) = \alpha(v) \).
- For \( v = f_i \), \( \hat{\alpha}(f_i) = \alpha(F_i^\bullet) \).

Notice that \( \hat{\alpha}(v) = \alpha(\text{source}_\alpha(v)) \). \( \hat{\alpha} \) can be seen as the real assignment induced by \( \alpha \), and for \( \varphi^{EUF} \) a sub-term or sub-formula of \( \varphi^{EUF} \) we have \( \alpha(T(\varphi^{EUF})) = \hat{\alpha}(\text{simp}(\varphi^{EUF})) \). In particular \( \alpha \models T(\varphi^{EUF}) \) iff \( \hat{\alpha} \models \text{simp}(\varphi^{EUF}) \).

**Example 10.**
\[
\hat{\gamma}(a) = 0, & \hat{\gamma}(b) = 0, & \hat{\gamma}(u) = 1, \\
\hat{\gamma}(f_1) = 1, & \hat{\gamma}(f_2) = 1, & \hat{\gamma}(f_3) = 3, & \hat{\gamma}(f_4) = 3.
\]

(4) For a term \( t \) of \( \varphi^{EUF} \), define \( \text{vals}(t) \) to be:
- If \( t \) is a term-variable then \( \text{vals}(t) = \{ t \} \).
- If \( t = F_i(\ldots) \) then \( \text{vals}(t) = \{ f_i \} \).
- If \( t = \text{ITE}(\text{cond}, t_1, t_2) \) then \( \text{vals}(t) = \text{vals}(t_1) \cup \text{vals}(t_2) \).

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

**Example 11.** In our example \( \varphi_9^{EUF} \):
- \( \text{vals}(F_3(\ldots)) = \{ f_3 \} \)
- \( \text{vals}(\text{ITE}(a \equiv b, u, a)) = \{ u, a \} \)
- \( \text{vals}(\text{arg}_1(F_2)) = \{ a, b \} \)

(5) For \( u, v \in \text{ITE}(G \uparrow) \), we mark \( u \equiv_\uparrow^* v \) iff \( u \equiv_\uparrow^* \text{flatEq}(G \uparrow) v \), and \( u \not\equiv_\uparrow^* v \) iff \( u \not\equiv_\uparrow^* \text{flatEq}(G \uparrow) v \). Recall that for an A-graph \( G \uparrow \), \( \text{flatEq}(G \uparrow) \) is \( G \uparrow \) where all assignment edges are replaced by equality edges.

We extend this definition to sets of vertices \( U_1 \) and \( U_2 \), and mark \( U_1 \equiv_\uparrow^* U_2 \) if there exists some \( u_1 \in U_1 \) and \( u_2 \in U_2 \) such that \( u_1 \equiv_\uparrow^* u_2 \).
8.2. A-graph construction

Given an EUF-formula $\varphi$ we construct an A-graph $G \uparrow$ as described by Algorithm 2.

**Algorithm 2** The A-graph construction algorithm, which is the first stage of MINIMAL-E.

1. Let the vertices be the set of term-variables of $T(\varphi)$.
2. Add as minimum all edges of $G(simp(\varphi))$ to $G \uparrow$.
3. For every $F_i$ and $F_j$ such that $i < j$, if for all $l \ vars(\arg_l(F_i)) = \text{vals}(\arg_l(F_j))$, then add the following edges:
   - Add $(f_i, f_j)$ to $E \uparrow (G \uparrow)$.
   - If $f_i \neq \text{vals}(f_j)$ then for all $v_i \in \text{vals}(\arg_l(F_i))$ and $v_j \in \text{vals}(\arg_l(F_j))$ add edge $(v_i, v_j)$ to $E_\neq(G \uparrow)$. Also mark $F_i$ and $F_j$ as critical.
4. For every critical $F_i$, if for some $l$ the term $t = \text{ITE}(\text{cond}, t_1, t_2)$ appears in $\text{simp}(\arg_l(F_i))$, then add all edges of $G(\text{cond})$ and $G(\neg \text{cond})$ to $G \uparrow$. Also add $\text{cond}$ to set $C$ (which is initially empty).
5. Repeat Steps 3 and 4 until convergence.
6. For every $u, v$, such that $\neg(u = \text{vals}(v))$, add edge $(u, v)$ to $E_\neq(G \uparrow)$. Denote all these edges free (in Section 10 we prove that they do not increase the state-space).

**Example 12.** For $\varphi$, the edges are added as follows:

1. In Step 2, we add $(f_3, f_4)$ to $E_\neq(G \uparrow)$ and $(u, f_2)$ to $E_=(G \uparrow)$. The state of $G \uparrow$ at this point is described in Fig. 4A. In this figure, under a vertex corresponding to function variable $f_i$, we have added the list of vertices in $\text{vals}(\arg_l(F_i))$.
2. In Step 3, we add the following edges:
   - $(f_1, f_2)$ to $E \uparrow (G \uparrow)$ since $b \in \text{vals}(\arg_1(F_1))$ and $b \in \text{vals}(\arg_1(F_2))$.
   - $(f_2, f_4)$ to $E \uparrow (G \uparrow)$ since $a \in \text{vals}(\arg_1(F_2))$ and $a \in \text{vals}(\arg_1(F_4))$.
   - $(f_3, f_4)$ is added to $E \uparrow (G \uparrow)$ since $f_2 = \text{vals}(u)$. Also, since $f_3 \neq \text{vals}(f_4), F_3$ and $F_4$ are marked as critical, and $(f_2, u)$ and $(f_2, a)$ are added to $E_\neq(G \uparrow)$
3. Since $F_4$ was marked as critical, $(a = b) \in C$. Therefore, in Step 4, we add $(a, b)$ to both $E_=(G \uparrow)$ and $E_\neq(G \uparrow)$.
4. Now (this was not the case before), since $a = \text{vals}(b)$, we add $(f_1, f_4)$ to $E \uparrow (G \uparrow)$. See part B of Fig. 4.
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 $E_\neq(G \uparrow)$.

---

2 Naturally one would take exactly $G(simp(\varphi))$, but we need this generalization for the proof.
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 critical. This is an example of the “cone of influence” effect we were aiming for.

**Soundness:** The following theorem states that the construction is sound.

**Theorem 1.** The A-graph $G \uparrow$ constructed for the EUF-formula $\psi_{EUF}$ is adequate for $T(\psi_{EUF})$.

The proof of this theorem appears in Appendix D.

9. **Transforming A-graphs to E-graphs**

In the previous section we showed how to construct an adequate A-graph $G \uparrow$ for $T(\psi_{EUF})$. Next, we would like to generate an adequate set of assignments for $G \uparrow$. Since the methods proposed in [15,19] calculate an adequate set of assignments only for an E-graph, we proceed in the following manner: given an A-graph $G \uparrow$, we construct an E-graph $G$ such that if assignment set $R$ is adequate for $G$ it will also be adequate for $G \uparrow$. In principle, we could have used flatEq ($G \uparrow$) as the E-graph corresponding to the A-graph $G \uparrow$. However, we can do somewhat better.

For two vertices $u$ and $v$, we denote $v \subseteq_G u$, if

- for every $(v, w) \in E_=(G)$, $(u, w) \in E_=(G)$,
- for every $(v, w) \in E_\neq(G)$, $(u, w) \in E_\neq(G)$.

That is, $u$ inherits any $(E_= \cup E_\neq)$-edge that departs from $v$. Algorithm 3 transforms A-graphs to E-graphs.

**Algorithm 3 Transforming A-graphs to E-graphs**

1. Initially, $G = \langle V(G \uparrow), E_=(G \uparrow), E_\neq(G \uparrow) \rangle$.
2. While there are vertices $u, v$, such that $(u, v) \in E \uparrow (G \uparrow)$ and neither $(u, v) \in E_=(G)$ nor $v \subseteq_G u$, choose one of the following options:
   a. add edge $(u, v)$ to $E_=(G)$,
   b. for every $(v, w) \in E_=(G)$ add $(u, w)$ to $E_=(G)$,
      - for every $(v, w) \in E_\neq(G)$ add $(u, w)$ to $E_\neq(G)$.
Theorem 2. If $R$ is adequate for $G$ then it is also adequate for $G \uparrow$.

**Proof.** Take some satisfiable $H \uparrow \leq G \uparrow$. We construct a satisfiable $H \leq G$ such that if $\beta \models H$ then $\beta \models H \uparrow$.

Since $H \uparrow$ is satisfiable, there is some $\alpha$ that satisfies $H \uparrow$ where all of $H \uparrow$’s assignment edges are replaced by equality edges (recall the definition of the satisfiability of an A-graph). We construct a subgraph $H \leq G$ and show that $\alpha \models H$.

Start with $E_\equiv(H) = E_\equiv(H \uparrow)$, and $E_\neq(H) = E_\neq(H \uparrow)$. Clearly, at this stage $\alpha \models H$. For every edge $(u, v) \in E \uparrow (H \uparrow)$ (notice that $\alpha(u) = \alpha(v)$):

- If $(u, v) \in E_\equiv(G)$ add $(u, v)$ to $E_\equiv(H)$. Obviously $\alpha \models H$ after this addition.
- Otherwise, we have that $v \subseteq G u$. Add the following edges to $H$:
  1. for every $(v, w) \in E_\equiv(H)$ add $(u, w)$ to $E_\equiv(H)$.
  2. for every $(v, w) \in E_\neq(H)$ add $(u, w)$ to $E_\neq(H)$.

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

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

- If $(b, d) \in E_\equiv(H \uparrow)$ then $\beta(a) = \beta(c)$.
- If $(b, d) \in E_\neq(H \uparrow)$ then $\beta(a) \neq \beta(c)$.

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

- If $x =^*_{H \uparrow} y$ then $\beta(x) = \beta(y)$.
- If $x \neq^*_{H \uparrow} y$ then $\beta(x) \neq \beta(y)$.

We will therefore prove:

- If $(b, d) \in E_\equiv(H \uparrow)$ then $a =^*_{H \uparrow} c$.
- If $(b, d) \in E_\neq(H \uparrow)$ then $a \neq^*_{H \uparrow} c$.

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 $H \uparrow$ to $H$, the claim follows.

If this sum is greater than 0, then one of these paths is of length greater than 0. Without loss of generality, 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 $(b, d) \in E_\equiv(H \uparrow)$ then $a' =^*_{H \uparrow} c$.
- If $(b, d) \in E_\neq(H \uparrow)$ then $a' \neq^*_{H \uparrow} c$. 
Since \( (a, a') \in E_{\uparrow} (\mathcal{H}_{\uparrow}) \), we have one of the two cases:

1. \( (a, a') \in E_{=} (\mathcal{H}) \). But then clearly \( a \) satisfies the two claims, since we simply prolong the path \( a' =^*_c \) in the first case or \( a' \neq^*_c \) in the second by the equality edge \( (a, a') \).

2. Otherwise, we know that \( a' \subseteq a \). This time instead of prolonging these paths, we replace their first edge. For example, if \( (b, d) \in E_{\neq} (\mathcal{H}) \), then \( a' \neq^*_c \). 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 \( a' \subseteq a \). \( \square \)

We showed a general method for translating 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 EUF-formula \( \varphi_{EUF} \) is in Positive Equality then the A-graph constructed in Section 8.2 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 [3].

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 [3].

**Example 13.** We demonstrate the transformation with the help of \( \varphi_{EUF}^1 \), the EUF-formula that was first presented in Section 5. The resulting A-graph of \( \varphi_{EUF}^1 \) from the construction appears in Fig. 5, and the result of transforming it to an E-graph (using the greedy approach) is presented in Fig. 6. This graph has an adequate set of assignments that consists of two assignments.

10. **Free-Edges do not increase the state-space**

Recall the last step of our graph construction: If \( \neg(u =^*_G v) \) then the disequality edge \( (u, v) \) is added to \( G_{\uparrow} \). We call these edges free because they do not impose any increase in the size of the assignment set \( R \) that is adequate for \( G_{\uparrow} \).

Given an E-graph \( G \) (we will handle A-graphs later) and a set \( R \) that is adequate for \( G \), we construct an assignment set \( R' \) that is adequate for \( G' \), where \( G' \) is equal to \( G \) plus all Free-Edges. We will show that \( |R'| = |R| \).
Define $C(G) = \{C_1, C_2, \ldots\}$ to be the set of connected components of the graph $\langle V(G), E(G) \rangle$. Notice that $u =^*_{G} v$ 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 $v \in V(G)$, $N > 2 \cdot |\alpha(v)|$. For $\alpha \in R$ we define $\alpha'$ to be the assignment that satisfies:

\[ \forall i \forall v \in C_i, \alpha'(v) = \alpha(v) + i \cdot N. \]

Define $R' = \{\alpha' | \alpha \in R\}$.

**Claim 1.** $R'$ is adequate for $G'$.

**Proof.** Take $H' \leq G'$. $H'$ is composed of an $H \leq G$ plus some Free-Edges of $G$. Since $R$ is adequate for $H$, there exists $\alpha \in R$ such that $\alpha \models H$. We claim that $\alpha' \models H'$. Take some edge $(u, v)$ of $H'$. There are $C_i, C_j \in C(G)$ 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 $G$). 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 $\alpha$ satisfies edge $(u, v)$ (equality or disequality). Also $\alpha'(u) - \alpha'(v) = \alpha(u) - \alpha(v)$, and therefore $\alpha'$ also satisfies this edge. □

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 9, we see that if $\neg(u =^*_G v)$ in the original A-graph, then also $\neg(u =^*_G v)$ 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.

### 11. 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 $\varphi^{\text{EUF}}$, an EUF-formula with uninterpreted predicates, we transform $\varphi^{\text{EUF}}$ to $\varphi^{\text{EUF}'}$ without uninterpreted predicates.

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

The problem we are facing is that instead of adding a Boolean variable for each predicate instance (as suggested in [3]), we added a term-variable and possibly increased the state-space for checking $\varphi^{\text{EUF}}$. But a more careful examination proves that this is not the case.

Note that in the A-graph of $\varphi^{\text{EUF}'}$, 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 according to Section 9). In our case the graph of the component of some predicate $P$ cannot contain disequality edges other than $(f^P_i, \text{true}^P)$. An adequate set of assignments for this kind of E-graph is given by letting each variable range over $\{0, 1\}$, and setting $\text{true}^P$ 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.

12. Experimental results and conclusions

As was mentioned earlier, the graphs that are generated by MINIMAL-E are always contained or equal to the graphs that were generated in [15], and therefore result in a smaller state-space. Since both algorithms are polynomial, we can consider MINIMAL-E as dominant over [15].

MINIMAL-E is also dominant over Positive Equality and Robust Positive Equality [3,13], since it always assigns a single constant to the same terms that are assigned single constant by their analysis, but much smaller ranges to other terms. Given the variables $\{v_0, \ldots, v_n\}$ belonging to the non-positive part, the Positive-Equality algorithm in [3,13] assign variable $v_i$ the range $\{1, \ldots, i\}$, resulting in a state-space of $n!$, while we use the Range-Allocation algorithm, which searches this state-space only in the worst case.
We implemented MINIMAL-E and combined it with the Range-Allocation algorithm of [15] to construct a new procedure for checking satisfiability of EUF-formulas in the cvt tool, which performs Translation Validation of both DC+ to C and Sildex to C optimizing compilers.\(^3\) We compared our decision procedure with that of [15] on dozens of industrial examples generated as part of the European SACRES and SafeAir projects (cvt is part of the tool-set that was developed in these projects). The results appear in Table 1, where the prefix ME denotes the results obtained by MINIMAL-E, and [15] the results of Range-Allocation as in [15]. Space denotes the resulting assignment set size. Since in all cases the verification procedure either proves that the formula is valid in less than 1 s, or runs out of memory, we do not write the exact running time. Instead we write ✓ if the run completed, and × if it did not. Num. vars denotes the number of variables in the example. There are also many examples for which both methods generate very small domains that take almost no time to explore, which we omit from the table. As can be seen from the table, MINIMAL-E has a significant effect on the state-space size. Indeed, by using this method we were able to verify formulas which we could not with the previous method. In fact, these examples were generated by decomposing large Translation Validation verification tasks. Now MINIMAL-E verifies these tasks without decomposition at all in a few seconds.

We also ran these benchmarks with six of the leading decision procedures to date: ICS [11], UCLID [7,14], CVC [21], and SIMP.\(^4\) None of these tools performed as well as MINIMAL-E on these benchmarks (Table 2).

All of these tools solve richer logics than just Equality Logic, and therefore the comparison cannot be considered as entirely fair: combining theories imposes an overhead which our tool does not have. Another difference is that our tool is the only one from this set, as far as known, that is based on (multi-typed) BDDs rather than on SAT.

### 12.1. Conclusions

We presented MINIMAL-E, an algorithm for deciding EUF formulas, which generalizes and improves the Positive Equality method of Bryant et al. [3] and the Range Allocation method of Pnueli

---

\(^3\) DC+ and Sildex are popular synchronous languages in the European avionics industry.

\(^4\) In fact most of these benchmarks were run as part of an independent research by De-Moura and Ruess [10].
We proved that the new method is both theoretically and empirically dominant over these two methods.

Appendix

A. Positive-Equality and other related work

The traditional approach for deciding EUF (in fact, as far as we know the only approach until the work of [12] in 1998) in the verification community was not to reduce Uninterpreted Functions to Equality Logic explicitly, but rather solve equalities while maintaining congruence closure with respect to the Uninterpreted Functions. Briefly, they worked as follows: given a conjunction of equalities with Uninterpreted Functions, they maintained an equivalence class for each set of variables that is interpreted as having the same value. Then, they enforce functional consistency of the Uninterpreted Functions by computing the congruence closure of these classes, i.e., if the (pair-wise) arguments of two function instances are in the same equivalence class, then so are the function instances themselves. The main disadvantage of such methods is that they only work with a conjunction of terms, while disjunctions are treated via syntactic case-splitting.\footnote{There is a large body of literature on how to do splitting efficiently, for example by using Semantic-Tableaux, or in more recent years with SAT [11,2,21], which makes these methods far more competitive.}

In the past few years several different procedures for checking satisfiability of such formulas without case splitting have been suggested, all of which rely on a reduction to Equality Logic. This reduction can be performed, for example, by using Ackermann’s reduction [1]. Ackermann suggested to replace each Uninterpreted Function instance with a new variable of the same type as the return type of the function, and for every pair of function instances add a constraint to $\varphi^e$ that enforces functional consistency: if the arguments of the two function instances are equal, then the variables that replace them are equal as well. For example, checking for the satisfiability of the formula $\varphi_{\text{EUF}}^{x}: f(x) \neq f(y)$ is reduced to the question of satisfiability of the E-formula $\varphi^e : (x = y \rightarrow f_1 = f_2) \land f_1 \neq f_2$. As a second step, different procedures can be used for checking the satisfiability of $\varphi^e$. Hence, efficient decision procedures for equalities are more important as they indirectly solve the logic of Equalities with Uninterpreted Functions.

Goel et al. suggest in [12] to solve Equality Logic by replacing all comparisons in $\varphi^e$ with new Boolean variables, and thus create a new Boolean formula $\varphi^{e'}$. The BDD [8] of $\varphi^{e'}$ is calculated while ignoring transitivity of equality. They then traverse the BDD, searching for a satisfying assignment that also satisfies these constraints. Bryant et al. in [5] suggested instead to explicitly compute the set of necessary transitivity constraints. By checking $\varphi^{e'}$ conjoined with these constraints using a standard BDD package they were able to verify larger designs than [12]. In the worst case this method requires a search in a space spanned by $O(n^2)$ variables, $n$ being the number of original variables in $\varphi^e$, and a formula whose size is larger (yet still linear) than the size of $\varphi^e$ due to the additional constraints.
A.1. Positive Equality

The main motivation for introducing Bryant et al.'s translation scheme in [6,3], and not using the more traditional Ackermann's reduction (Appendix B contains an elaborated discussion of this alternative translation and its implications on our method) is that it allows to exploit what they call the Positive Equality structure of formulas.

Informally, a term in an EUF-formula \( \varphi^{\text{EUF}} \) is said to be of positive equality if it is only adjacent to solid edges in the corresponding E-graph. The accurate definition is more complicated than this, and refers separately to variables and function applications, as we will soon see through an example.

The original definition of [3] of these terms was a syntactic one, not through an E-graph. It categorized terms as 'negative' if none of its occurrences in \( \varphi^{\text{EUF}} \) appear under positive polarity.\(^6\) In fact a negative term can only appear as part of a disequality or as argument of functions. Further, the categorization of functions as positive or negative is done as per function symbol, not function application, and therefore all function instances of a function should be negative to categorize it as negative (see remark on an improvement called robust positive equality below that overcomes this problem). The example below will hopefully clarify these informal definitions:

**Example 14.**

\[
\begin{align*}
\varphi_{2}^{\text{EUF}} & := x_1 \neq x_2 \land f(x_1) \neq f(x_2) \land g(x_1) = g(x_2), \\
\varphi_{3}^{\text{EUF}} & := x_1 \neq x_2 \land f(\text{ITE}(x_1 = x_2, x_3, x_4)) \neq f(x_2), \\
\varphi_{4}^{\text{EUF}} & := f(x_1) = f(x_2) \land f(x_3) \neq f(x_4).
\end{align*}
\]

In the first formula \( x_1, x_2 \), and the function \( f \) appear under negative polarity while \( g \) does not. In the second formula \( f \) is under negative polarity but \( x_1, x_2 \) are not, because they appear in a guard of an ITE expression (which implicitly represent both \( x_1 = x_2 \) and \( x_1 \neq x_2 \)). In the third formula \( f \) is not considered to be under negative polarity, because one of its instances is under positive polarity.

The big benefit of Positive Equality is that terms that are marked as having negative polarity can be assigned a single unique constant. Recall that these terms are never adjacent to dashed edges in the E-graph, which means that they do not need to be equal to any other term to satisfy the formula. Thus, a unique constant is sufficient because if there is a satisfying assignment, then there is also a satisfying assignment in which all the inequalities hold, and surely they hold when their terms are assigned different values. Again, let us demonstrate this with an example:

**Example 15.** We can check the satisfiability of \( \varphi_{2}^{\text{EUF}} \) from Example 14 by replacing \( x_1, x_2, f_1, f_2 \) with 0, 1, 2, 3 in \( T(\varphi_{2}^{\text{EUF}}) \), respectively, which satisfies this formula.

\(^6\) The confusion between the term 'positive equality' and our definition of them as 'negative' is because in [3], where this term was introduced, the analysis referred to validity checking, rather than satisfiability as in this paper.
When the formula is mixed, like \( \varphi^\text{EUF}_3 \), only the terms that are marked as having negative polarity are replaced with constants, and the others are allocated a full range (the range 1 \ldots i to the /th such variable). So in the case of \( \varphi^\text{EUF}_3 \), we can replace \( f_1 \) and \( f_2 \) in \( T(\varphi^\text{EUF}_3) \) with the constants 0,1 respectively, and allocate the range, e.g., \([2, 2, 3] \) to \( x_1, x_2 \), respectively.

Finally, in the case of \( \varphi^\text{EUF}_4 \), \( f_1, \ldots, f_4 \) are positive, hence are assigned a full range 1 \ldots i, while their arguments \( x_1, \ldots, x_4 \) are assigned a single unique value each.

B. Ackermann’s reduction

The reduction scheme shown in Section 5 due to Bryant et al. is not the only one possibility. The more traditional method is due to Ackermann [1]. Ackermann’s reduction has a disadvantage that it cannot be combined with the Positive Equality optimization, for reasons that we will soon describe in detail. We would like to describe this reduction here nevertheless, because this was the underlying reduction technique used in [15], so therefore we need to describe it to compare it to MINIMAL-E.

We will denote the resulting E-formula using Ackermann’s reduction by \( T^A(\varphi^\text{EUF}) \). Denote by \( F \) the set of function symbols in \( \varphi^\text{EUF} \). \( T^A(\varphi^\text{EUF}) \) is then defined by:

\[
\left[ \bigwedge_{F \in F} \bigwedge_{i,j} \left( \bigwedge_l \text{simp}(\arg_l(F_i)) = \text{simp}(\arg_l(F_j)) \rightarrow f_i = f_j \right) \right] \wedge \text{simp}(\varphi^\text{EUF}).
\]

The reduction replaces every function instance by a new term-variable (\( F_i \) is replaced by \( f_i \) in \( \text{simp}(\varphi^\text{EUF}) \)), and then require functional consistency of all the new variables. The functional consistency constraints are given in the big conjunction on the left of \( T^A(\varphi^\text{EUF}) \), requiring that \( f_i = f_j \) if the arguments of their corresponding function instances \( F_i \) and \( F_j \) are equal.

Example 16.

\[
T^A(\varphi^\text{EUF}_1) : \begin{align*}
x_1 = x_2 & \rightarrow f_1 = f_2 \wedge \\
x_1 = f_1 & \rightarrow f_1 = f_3 \wedge \\
x_1 = f_3 & \rightarrow f_1 = f_4 \wedge \\
x_2 = f_1 & \rightarrow f_2 = f_3 \wedge \\
x_2 = f_3 & \rightarrow f_2 = f_4 \wedge \\
f_1 = f_3 & \rightarrow f_3 = f_4 \\
\end{align*}
\wedge ((f_4 \neq f_3) \wedge (f_3 \neq f_2) \wedge (x_2 = f_1)).
\]

Given this translation scheme, we can now compare Range-Allocation as done in [15] to MINIMAL-E, through \( \varphi^\text{EUF}_1 \). In Example 13 and Figs. 5 and 6 the E-graph for \( \varphi^\text{EUF}_1 \) was presented according to MINIMAL-E, which results in a domain allocation with a state-space of 2. With Ackermann’s reduction, however, all edges in \( T^A(\varphi^\text{EUF}) \) are added to the E-graph, resulting in the E-graph shown in Fig. 7. In this E-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 (\( x_1, x_2, f_1, f_3 \)). If we ignore the Free-Edges that are added by Rule 3 (since they do not increase the size of the assignment set
needed), then we get that $G \leq G(T^A(\varphi_{EUF}))$, where $G$ is the graph generated by Minimal-E. Generally speaking, if assignment set $R$ is adequate for an E-graph $H$, then it is adequate for any E-graph $H'$ such that $H' \leq H$. Therefore the assignment set size that results from Minimal-E should generally be smaller. In the case of this example, we claim without proof that there cannot be a range allocation with a size of less than 16 for $G(T^A(\varphi_{EUF}))$.

**B.1. Comparing the translation methods**

Our improved graph construction, as well as the original work on Positive Equality, cannot be applied with Ackermann’s reduction. The reason is subtle, and is best explained with an example.

**Example 17.** Suppose that we want to check the satisfiability of the following (satisfiable) formula:

$$\varphi_{10}^{EUF} := x_1 = x_2 \lor (f(x_1) \neq f(x_2) \land \text{FALSE})$$  \hspace{1cm} (B.2)

Using Bryant et al.’s reduction we get:

$$T(\varphi_{10}^{EUF}) := x_1 = x_2 \lor (\overline{F_1} \neq \overline{F_2} \land \text{FALSE})$$ \hspace{1cm} (B.3)

$$\overline{F_1} := f_1 \quad \overline{F_2} := \begin{cases} f_1 & x_1 = x_2; \\ f_2 & \text{TRUE}. \end{cases}$$

The resulting graph after applying Minimal-E appears in Fig. 8.

An adequate range for this graph can be

$$R: x_1 \mapsto \{0\}, \ x_2 \mapsto \{0,1\}, \ f_1 \mapsto \{2\}, \ f_2 \mapsto \{3\}.$$  \hspace{1cm} (B.4)

Fig. 8. The graph corresponding to Example 17 after applying Minimal-E (but before adding Free-Edges, for the sake of clarity of the graph).
Clearly these domains are adequate for $T(\varphi_{EUF}^{10})$, since we can choose the satisfying assignment:

$$(x_1, x_2, f_1, f_2) \leftarrow (0, 0, 2, 3).$$

Now suppose that we want to use these domains with $T^A(\varphi_{EUF}^{10})$:

$$T^A(\varphi_{EUF}^{10}) := (x_1 = x_2 \rightarrow f_1 = f_2) \land (x_1 = x_2 \lor (f_1 \neq f_2 \land \text{false})). \tag{B.5}$$

To satisfy $T^A(\varphi_{EUF}^{10})$ it must hold that $x_1 = x_2$, which implies that $f_1 = f_2$ must hold as well. But the domains allocated in Eq. (B.4) do not allow an assignment in which $f_1$ is equal to $f_2$, which means that the graph is not adequate for $T^A(\varphi_{EUF}^{10})$.

So why does Ackermann’s reduction not work with our graph construction method (and also not with the Positive Equality optimization)?

The reason is that when two function instances, say $F(x_1)$ and $F(x_2)$, have equal arguments, in Ackermann’s reduction the two variables representing the functions, say $f_1$ and $f_2$, are constrained to be equal. But if we force $f_1$ and $f_2$ to be different (by giving them a singleton domain comprised of a unique constant), this forces the functional consistency constraints to be false, and consequently $T^A(\varphi_{EUF}^{10})$ to be false. On the other hand in Bryant et al.’s reduction, when the arguments $x_1$ and $x_2$ are equal, the terms $F_1^\star$ and $F_2^\star$ that represent the two functions are both assigned the value of $f_1$. So even if $f_2 \neq f_1$, it does not necessarily make $T^A(\varphi_{EUF}^{10})$ false.

C. Optimal elimination order

In [13] it was shown how controlling the elimination order of functions (the indices that are assigned to function instances) can affect the number of terms that can be replaced with a unique constant. Further, they showed that this is an NP-complete optimization problem, and suggested a greedy heuristic to solve it that turns out to find the optimal solution in all the examples that they tried. Although our goals seem different (their goal is to maximize the number of variables that can be declared as negative and hence replaced with a constant, and our goal is to find an ordering which minimizes the size of the allocated domain), we claim that by applying their ordering technique we generalize their result. This means that we can do even better if we choose a heuristic that directly minimizes the allocated domain instead. But in this appendix we just want to emphasize the generalization.

Where does elimination ordering have an effect on Minimal-E? In Rule 2, if we choose the second option, the function ordering matters, since we copy the constraints over the function instance with the higher index to the lower one. We claim that by using their optimal ordering Minimal-E constructs an E-graph in which those variables are only adjacent to solid edges, and hence allocated a single value as well. Briefly, the reason is that such terms are adjacent to solid edges in the E-graph corresponding to $\text{simp}(\varphi_{EUF}^{10})$. Additionally, they schedule them to be eliminated last. Applying Rule 1 only adds solid edges, and when applying Rule 2 we can always choose the second option, which again only copies solid edges (since they have the highest index, their constraints are copied to other function instances). A more detailed proof requires a far more detailed description of their ordering strategy, which is beyond the scope of this article.
D. Soundness proof

**Theorem 1.** The A-graph $G \uparrow$ constructed for the EUF-formula $\varphi^{EUF}$ is adequate for $T(\varphi^{EUF})$.

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 $T(\varphi^{EUF})$. The base case where $C = \emptyset$ is the difficult part, so we start with the induction step.

If $\text{cond} \in C$ then there is $F_i$ and $l$ such that a term $t = \text{ITE}(\text{arg}_l(F_i))$ appears in $\text{simp}(\text{arg}_l(F_i))$, where $F_i$ is marked critical. This term also appears in $\varphi^{EUF}$ as $t' = \text{ITE}(\text{cond}', t'_1, t'_2)$ where $t = \text{simp}(t')$ (we use $'$ to mark the corresponding un-simplified term appearing in $\varphi^{EUF}$).

Define:

$$\rho_1 = \text{cond}' \land \varphi^{EUF}[t' \leftarrow t'_1],$$

$$\rho_2 = \neg \text{cond}' \land \varphi^{EUF}[t' \leftarrow t'_2].$$

Clearly, $\varphi^{EUF} = \rho_1 \lor \rho_2$. Also, by calculating $T(\rho_1)$ in a way similar to the calculation of $T(\varphi^{EUF})$, it is easy to see that if there is some $H \uparrow \subseteq G \uparrow$ s.t. $H \uparrow \models T(\rho_1)$, then $H \uparrow \models T(\varphi^{EUF})$, since if $\alpha \models T(\rho_1)$ then $\alpha \models T(\varphi^{EUF})$. Therefore to show that $G \uparrow$ is adequate for $T(\varphi^{EUF})$ it is sufficient to show that $G \uparrow$ is adequate for both $T(\rho_1)$ and $T(\rho_2)$.

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

For this we need to show that $G(\text{simp}(\rho_i)) \subseteq G \uparrow$. For example, if we consider $\rho_1$, $G(\text{simp}(\rho_1))$ is $G(\text{simp}(\varphi^{EUF})) \cup G(\text{simp}(\text{cond}'))$. Clearly $G(\text{simp}(\rho_1)) \subseteq G(\text{simp}(\varphi^{EUF})) \subseteq G \uparrow$. Also, since $\text{cond} \in C$ then $G(\text{cond}) \subseteq G \uparrow$ (Step 8.2). Since $\text{simp}(\text{cond}') = \text{cond}$, we have that $G(\text{simp}(\rho_1)) \subseteq G \uparrow$.

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 $\text{vals}(\text{arg}_l(F_i))$ for some $F_i$, $i$ and $l$. Therefore $G \uparrow$ is a graph that can be constructed by our procedure for $\rho_1$ and $\rho_2$.

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

We proceed to prove the case where $C = \emptyset$. By the fact that $C$ is empty, we know that for every critical $F_i$, for every $l$, $\text{vals}(\text{arg}_l(F_i))$ contains exactly one term-variable. In this case we mark this variable by $\text{Arg}_l(F_i)$.

Assume there is a satisfying assignment $\alpha$ to $T(\varphi^{EUF})$. Without loss of generality we can assume that $\alpha = \hat{\alpha}$ because $\hat{\alpha}$ is also a satisfying assignment to $\varphi^{EUF}$.

We will construct a satisfiable A-graph $H \uparrow \subseteq G \uparrow$, such that if $\beta \models H \uparrow$ then $\hat{\beta}$ satisfies all equality and disequality edges of $H \uparrow$. $H \uparrow$ will also contain all the edges of $G \uparrow$ that $\alpha$ satisfies. Since $G(\text{simp}(\varphi^{EUF})) \subseteq G \uparrow$, and $\alpha \models \text{simp}(\varphi^{EUF})$ then by Proposition 1 $\beta \models \text{simp}(\varphi^{EUF})$, and so $\beta \models T(\varphi^{EUF})$. This will prove that $H \uparrow \models T(\varphi^{EUF})$, proving the desired result.
Denote by \( \mathcal{H} \uparrow_0 \leq \mathcal{G} \uparrow \) the graph of all equality and disequality edges of \( \mathcal{G} \uparrow \) that \( \alpha \) satisfies.

Create \( \mathcal{H} \uparrow_1 \) starting from \( \mathcal{H} \uparrow_0 \) and using the following rule successively until now new edges are added. If for all \( l, \) \( \text{vals} (\text{Arg}_l(F_i)) =^*_\mathcal{H} \text{vals} (\text{Arg}_j(F_j)) \) then add the edge \((f_i, f_j)\) to \( E \uparrow (\mathcal{H} \uparrow)_1 \).

Create \( \mathcal{H} \uparrow_2 \) from \( \mathcal{H} \uparrow_1 \). For every pair \( F_i \) and \( F_j \) where \( i < j \), proceed if the following conditions are true:

\( (f_i, f_j) \notin E \uparrow (\mathcal{H} \uparrow)_1 \).
\( f_i \neq ^*_\mathcal{H} f_j \).
\( (f_i, f_j) \in E \uparrow (\mathcal{G} \uparrow) \).

The last two conditions imply that \( F_i \) and \( F_j \) are critical. Therefore \( \text{Arg}_l(F_i) \) and \( \text{Arg}_l(F_j) \) are defined. We also know that \((\text{Arg}_l(F_i), \text{Arg}_l(F_j)) \in E_\neq(\mathcal{G} \uparrow)\). Since \((f_i, f_j) \notin E \uparrow (\mathcal{H} \uparrow_1)\), there is some \( l \) such that \( \neg(\text{Arg}_l(F_i) = ^*_\mathcal{H} \text{Arg}_l(F_j)) \). Add \((\text{Arg}_l(F_i), \text{Arg}_l(F_j))\) to \( E_\neq(\mathcal{H} \uparrow_2) \).

Create \( \mathcal{H} \uparrow_3 \) from \( \mathcal{H} \uparrow_2 \) by successively adding assignment edges from \( \mathcal{G} \uparrow \) to \( \mathcal{H} \uparrow_3 \) that do not render it unsatisfiable.

Add all Free-Edges of \( \mathcal{G} \uparrow \) to \( \mathcal{H} \uparrow_3 \) to create \( \mathcal{H} \uparrow_4 \).

We define \( \mathcal{H} \uparrow \) to be \( \mathcal{H} \uparrow_4 \). By the construction, \( \mathcal{H} \uparrow \) is satisfiable and \( \mathcal{H} \uparrow \leq \mathcal{G} \uparrow \).

**Lemma 1.** For \( \beta \models \mathcal{H} \uparrow \):

1. If \( \hat{\beta}(v) = \hat{\beta}(u) \) then \( v = _{\mathcal{G} \uparrow} u \).
2. Let \( u = \text{source}_\beta(v) \). If \( u \neq v \), then \( (u, v) \in E \uparrow (\mathcal{G} \uparrow) \).

**Proof.** The proof proceeds by induction on the pairs \((u, v)\). For two variables \( x, y \) we denote \( x < y \), if \( y = f_i \), and \( x \neq f_i \) is in the fan-in cone of \( F_i^* \) in \( T(\phi^{RV}) \) \((x \) appears as a sub-term \( F_i^* \))

This is the standard pre-order on terms in a formula. Note that if \( \text{source}_\beta(a) \neq a \), then \( \text{source}_\beta(a) < a \).

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

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

Following are the proofs of the two claims:

1. We want to prove that if \( \hat{\beta}(u) = \hat{\beta}(v) \) then \( u = _{\mathcal{G} \uparrow} v \).

   If \( \beta(u) = \beta(v) \) then \( u = _{\mathcal{G} \uparrow} v \), since if \( \neg(u = _{\mathcal{G} \uparrow} v) \) then by Step 6 of the A-graph construction, disequality edge \((u, v)\) will be in \( E_\neq(\mathcal{G} \uparrow) \) and it will always be added to \( \mathcal{H} \uparrow_4 \).

   If \( \hat{\beta}(u) = \hat{\beta}(v) \) then \( \beta(\text{source}_\beta(u)) = \beta(\text{source}_\beta(v)) \). By the same argument as above we have that \( \text{source}_\beta(u) = _{\mathcal{G} \uparrow} \text{source}_\beta(u) \).

   If \( \text{source}_\beta(u) \neq u \), then according to our induction hypothesis (1), \((\text{source}_\beta(u), u) \in E \uparrow (\mathcal{G} \uparrow) \), meaning \( \text{source}_\beta(u) = _{\mathcal{G} \uparrow} u \). Clearly, if \( \text{source}_\beta(u) = u \) then also \( \text{source}_\beta(u) = _{\mathcal{G} \uparrow} u \). For the same reason, \( \text{source}_\beta(v) = _{\mathcal{G} \uparrow} v \).
Since $=_{G↑}^*$ is an equivalence relation, $source_{β}(u) =_{G↑}^* source_{β}(v)$ (as we have shown above), $source_{β}(u) =_{G↑}^* u$ and $source_{β}(v) =_{G↑}^* v$ we get that $u =_{G↑}^* v$.

(2) Let $u = source_{β}(v)$. If $u \neq v$ then for some function $F$, $v = f_j$ and $u = f_i$, where $i < j$. Also, for all $l$, $β(T(arg_{i}(F_i))) = β(T(arg_{j}(F_j)))$. Equivalently, $β(simp(arg_{i}(F_i))) = β(simp(arg_{j}(F_j)))$. This means that for every $l$, there is some $v_i \in \text{vals}(arg_{i}(F_i))$ and $u_i \in \text{vals}(arg_{i}(F_i))$ such that $β(u_i) = β(v_i)$. Using our induction hypothesis (1), for all $l$, $u_i =_{G↑}^* v_i$, implying $vals(arg_{i}(F_i)) =_{G↑}^* vals(arg_{i}(F_j))$, This means that $(f_i, f_j) \in E↑(G↑)$. \qed

To conclude we prove:

**Claim 2.** For $β \models \mathcal{H}↑:$

1. $β$ satisfies all equality and disequality edges of $\mathcal{H}↑$.
2. Let $u = source_{β}(v)$. If $u \neq v$. then $(u, v) \in E↑(\mathcal{H}↑)$.

**Proof.** We use the same induction strategy as in the proof of Lemma 1:

1. Recall that we marked $a \xrightarrow{↑G↑} b$ if there exists a directed assignment edge path (possibly of length 0) from $a$ to $b$ in $G↑$. Using our induction hypothesis (2), $source_{β}(u) \xrightarrow{↑H↑} u$ and $source_{β}(v) \xrightarrow{↑H↑} v$. Since $β \models \mathcal{H}↑$, we get that if $(u, v) \in E= (\mathcal{H}↑)$ then $β(source_{β}(u)) = β(source_{β}(v))$ and if $(u, v) \in E≠ (\mathcal{H}↑)$ then $β(source_{β}(u)) ≠ β(source_{β}(v))$. Since always $β(a) = β(source_{β}(a))$, we conclude.

2. Let $u = source_{β}(v)$. From Lemma 1, we know that $(u, v) \in E↑(G↑)$. We split to two cases:

- If $¬(u ≠_{G↑}^* v)$ then the edge $(u, v)$ of $E↑(G↑)$ will always be added to $\mathcal{H}↑$ since it will never make $\mathcal{H}↑$ unsatisfiable.
- Assume $u ≠_{G↑}^* v$. We know that $v = f_j$ and $u = f_i$ where $i < j$, and that $f_i$ and $f_j$ are critical. Therefore, for every $l$, $Arg_{i}(F_i)$ and $Arg_{j}(F_j)$ are defined, and since $(f_i, f_j) \in E↑(G↑)$ and $f_i ≠_{G↑}^* f_j$ then $(Arg_{i}(F_i), Arg_{j}(F_j)) \in E≠(G↑)$.

If for some $l'$, $(Arg_{i}(F_i), Arg_{j}(F_j)) \in E≠(\mathcal{H}↑)$ then using our induction hypothesis (2), we get $β(Arg_{i}(F_i)) ≠ β(Arg_{j}(F_j))$, but this contradicts the fact that $source_{β}(f_j) = f_i$.

Therefore for all $l$, $(Arg_{i}(F_i), Arg_{j}(F_j)) \not\in E≠(\mathcal{H}↑)$ but then already in $\mathcal{H}↑$ for all $l$, $Arg_{i}(F_i) =_{H↑1} arg_{i}(F_j)$. If this is not true then for some $l$ the disequality edge $(Arg_{i}(F_i), Arg_{j}(F_j))$ would be taken in $\mathcal{H}↑2$.

Now, since for all $l$, $Arg_{i}(F_i) =_{H↑1} arg_{i}(F_j)$, then $(f_i, f_j) \in E↑(\mathcal{H}↑)$). Therefore $(u, v) \in E↑(\mathcal{H}↑)$. \qed

**References**


Aided Verification (CAV’02), 2002.

[22] M. Velev, R. Bryant, EVC: a validity checker for the logic of equality with uninterpreted functions and memories,
exploiting positive equality, and conservative transformations, in: G. Berry, H. Comon, A. Finkel (Eds.), Proc.
13th Intl. Conference on Computer Aided Verification (CAV’01), Lecture Notes in Computer Science, vol. 2102,