Range Allocation for Equivalence Logic

Amir Pnueli\textsuperscript{1}, Yoav Rodeh\textsuperscript{1,2}, and Ofer Shtrichman\textsuperscript{1,2}

\textsuperscript{1} Weizmann Institute of Science, Rehovot, Israel
\textsuperscript{2} IBM Haifa Research Laboratory
\{amir | yrodeh | ofers\}@wisdom.weizmann.ac.il

Abstract. The range allocation problem was recently introduced as part of an efficient decision procedure for deciding satisfiability of equivalence logic formulas with or without uninterpreted functions. These type of formulas are mainly used when proving equivalence or refinement between systems (hardware designs, compiler’s translation, etc.). The problem is to find in polynomial time a small finite domain for each of the variables in an equality formula $\varphi$, such that $\varphi$ is valid if and only if it is valid over this small domain. The heuristic that was presented for finding small domains was static, i.e. it finds a small set of integer constants for each variable. In this paper we show new, more flexible range allocation methods. We also show the limitations of these and other related approaches by proving a lower bound on the size of the state space generated by such procedures. To prove this lower bound we reduce the question to a graph theoretic counting question, which we believe to be of independent interest.

1 Introduction

The range allocation problem was introduced in [PRSS98] as part of an efficient decision procedure for equivalence logic formulas with or without uninterpreted functions. These type of formulas are mainly used when proving equivalence or refinement (abstraction) between systems. Deciding satisfiability (and validity) of formulas with uninterpreted functions is of major importance due to their broad use in abstraction. We refer the reader to [BD94] and [PSS98], where these type of formulas are used for proving equivalence between hardware designs (former) and for translation validation, a process in which the correctness of a compiler’s translation is proven by checking the equivalence of the source and target codes (latter).

In the past few years several different BDD-base procedures for checking satisfiability of such formulas have been suggested. (in contrast to earlier decision procedures that are based on computing congruence closure [BDL96] in combination with case splitting). Typically the first step of these procedures is the translation of the original formula $\varphi$ to a function-free formula in equivalence logic $\psi$ such that $\psi$ is satisfiable iff $\varphi$ is. Then, a procedure for checking satisfiability of E-formulas is used for deciding $\psi$. This second procedure is the focus of this paper.
Goel et al. suggest in [GSZAS98] to replace all comparisons in \( \psi \) with new Boolean variables, and thus create a new Boolean formula \( \psi' \). The BDD of \( \psi' \) 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 \( \psi' \) conjuncted with these constraints using a regular BDD package they were able to verify larger designs.

The method which we will present here, extends the method first presented in [PRSS98], where \( \psi' \)’s satisfiability is decided by allocating a small domain for each variable, such that \( \psi \) is satisfiable if and only if it is satisfiable over 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 \( \psi \). 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 over the computed domain. In [RS01] we elaborate on this by generating a smaller graph than [PRSS98]. This is achieved by examining the original formula with uninterpreted function \( \phi \), instead of its translated version \( \psi \).

In this paper, we extend the second part of [PRSS98], by suggesting a more general method of allocating finite domains to variables. Using the information in the graph generated by [PRSS98] or [RS01], we suggest and analyze different procedures for generating a small state space that is adequate for checking \( \psi' \).

One of our main results is a general lower bound on the size of the state space generated by any method using only the information in this graph. The suggests the need for a more in depth investigation of the formula at hand, rather than only examining which atomic equalities appear in it.

2 Equivalence Logic Formulas

An equivalence logic formula (called an E-formula) has the following syntax:

\[
\langle \text{Formula} \rangle \leftarrow \langle \text{Term} \rangle = \langle \text{Term} \rangle \mid \neg \langle \text{Formula} \rangle \mid \langle \text{Formula} \rangle \lor \langle \text{Formula} \rangle
\]

\[
\langle \text{Term} \rangle \leftarrow \langle \text{Variable} \rangle \mid \text{ITE}(\langle \text{Formula} \rangle, \langle \text{Term} \rangle, \langle \text{Term} \rangle)
\]

\text{ITE}(f, t, e) \text{ stands for if } f \text{ then } t \text{ else } e. \text{ The E-formula } \varphi \text{ is said to be satisfiable if there is some assignment of values to } \varphi \text{’s variables that satisfies } \varphi. \text{ Therefore, an E-formula } \varphi \text{ with variables } V \text{ is a function } \varphi : \mathbb{N}^V \rightarrow \{0, 1\}. \text{ However, not all such functions can be realized as E-formulas, for example } \varphi(a, b) = a > b. \text{ Therefore, we will try to make a more accurate definition.}

\textbf{Definition 1. (partition)}: Given a set } V, \text{ we say that } \alpha = \{\alpha_1, \ldots, \alpha_k\} \text{ is a partition of } V \text{ if:}