Finding Relations Among Linear Constraints

Jun Yan\textsuperscript{1,2}, Jian Zhang\textsuperscript{1}, and Zhongxing Xu\textsuperscript{1,2}

\textsuperscript{1} Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
\textsuperscript{2} Graduate University, Chinese Academy of Sciences
\{yanjun, zj, xzx\}@ios.ac.cn

Abstract. In program analysis and verification, there are some constraints that have to be processed repeatedly. A possible way to speed up the processing is to find some relations among these constraints first. This paper studies the problem of finding Boolean relations among a set of linear numerical constraints. The relations can be represented by rules. It is believed that we can not generate all the rules in polynomial-time. A search based algorithm with some heuristics to speed up the search process is proposed. All the techniques are implemented in a tool called MALL which can generate the rules automatically. Experimental results with various examples show that our method can generate enough rules in acceptable time. Our method can also handle other types of constraints if proper numeric solvers are available.

1 Introduction

Constraints play an important role in various applications, and constraint solving has been an important research topic in Artificial Intelligence. A useful technique for constraint solving is to add some redundant constraints so as to improve the algorithms’ efficiency \cite{1,2}. However, there is not much work on the systematic discovery of such constraints.

When we study constraint solving techniques, it is usually helpful if we take the form of constraints into account. This often depends on the application domain. One domain that is quite interesting to us is program analysis and verification. To analyse a program and generate test data for it, we may analyze the program’s paths. For each path, we can derive a set of constraints whose solutions represent input data which force the program to be executed along that path \cite{3}. Such a path-oriented method is often used in software testing, and it may also be used in infinite loop detection \cite{4}.

Generally speaking, the constraints encountered in program analysis and testing can be represented as Boolean combinations of arithmetic constraints \cite{3,5}. Here each constraint is a Boolean combination of primitive constraints, and a primitive constraint is a relational expression like $2x + 3y < 4$. In other words, a constraint is a Boolean formula, but each variable in the formula may stand for a relational expression. To solve such constraints, we developed a solver called BoNuS which combines Boolean satisfiability checking with linear programming.

* Supported in part by the National Science Foundation of China (grant No. 60125207 and 60421001).
In program analysis, we often need to solve many sets of constraints. They may contain some common primitive constraints. So if we can find some logic relationship (i.e., rules) between the primitive constraints (especially those occurring frequently) and add them as lemmas, the search space can be reduced.

Formal verification is also an important way to maintain program quality. Model checking is an effective verification method. But traditionally it is based on the propositional temporal logic. To scale it up to more real programs, one may use abstraction[6,7]. It is based on the observation that the specifications of systems that include data paths usually involve fairly simple relationships among the data values in the system. A mapping between the actual data values in the system and a small set of abstract data values are given according to these relationships. In fact, since the abstract values are not always independent, these abstractions can be regarded as some rules deduced from the predicates. Therefore we need to find out some logic relations of the predicates before using abstraction.

In this paper, we try to employ a linear programming solver called lp_solve[8] to find all the logic relations among a set of linear arithmetic constraints automatically. We implemented a tool and used it to analyze how the attributes of a constraint set affect the number of rules. Since most of our techniques do not rely on any special characteristics of linear constraints, our method can be generalized to other types of constraints such as non-linear constraints if a proper solver is provided.

This paper is organized as follows. The next section will briefly introduce the problem of finding rules from numerical constraints and analyze its complexity. Then section 3 will present the main idea of our algorithm and some improving techniques. Experimental results and some analysis are given in Section 4. Then our approach is compared with some related works in Section 5, and some directions of future research are suggested in the last section.

2 The Problem and Its Complexity

2.1 Linear Arithmetic Constraints

In this paper, a numerical constraint is a Linear Constraint in the following form:

\[ a_1 x_1 + \ldots + a_n x_n \succ b \]

where \( a_i \) is a coefficient, \( x_i \) is a variable, and \( \succ \in \{=, <, >, \leq, \geq, \neq \} \) is a relational operator. A conjunction of linear constraints

\[ \varphi : \begin{bmatrix} a_{11} x_1 + \cdots + a_{1n} x_n \succ b_1 \\ \vdots \cdots \vdots \\ a_{m1} x_1 + \cdots + a_{mn} x_n \succ b_m \end{bmatrix} \]

can be written concisely in matrix form as \( Ax \succ b \) where the bold \( x \) and \( b \) are \( n \)-dimensional and \( m \)-dimensional vectors, respectively.

Example 1. Here is an example of a set of numerical constraints.