Program Analysis using BDDs

by

Debasish Das    Prahladavaradan Sampath
Ashok Sreenivas

Tata Research Development and Design Centre    TRDDC–??–3

Tata Consultancy Services
54B Hadapsar Industrial Estate
Pune    411-013
India

TRDDC Report Series    TRDDC–??–3
Tata Research Development and Design Centre    July 2003

Copyright © 2003, TRDDC
Program Analysis using BDDs

Debasish Das       Prahladavaradan Sampath       Ashok Sreenivas
Tata Research Development and Design Centre
Tata Consultancy Services

July 31, 2003

Abstract

The use of BDDs is wide-spread in the program verification community where they are used as compact and efficient representations of state spaces and transition relations. In this paper we investigate the use of BDDs in program analysis. We exploit the efficient representation that BDDs allow for finite relations and formulate analysis algorithms that exploit the properties of BDDs.
1 Introduction

BDDs are a representation of Quantified Boolean Formulae, and can be used to encode relations on finite domains. BDDs also exhibit efficient encodings of various operations on these relations. The traditional application of BDDs is in the program verification community, where they are used to provide compact encodings of very large state spaces and transition relations. The use of BDDs has enabled the verification of very large circuits having hundreds of millions of states.

The use of BDDs in program analysis has however remained largely in the fringes. Recent work [?] has used BDDs to efficiently represent data-structures used in analysis; however as far as we are aware [?] is the first attempt to give BDDs a more central role in program analysis. In this work, BDDs have been used to perform points-to analysis.

The points-to analysis in [?] is a context-free and flow-insensitive analysis similar to the analysis formulated by Steensgaard [?]. The analysis formulated as a non-standard type-system. The type-system is itself fairly simple and intuitive, and consists of just three inference rules (c.f. Figure 1)

![Figure 1: Type inference rules for points-to analysis](main.tex)

2 Literature

3 Data-Flow Frameworks

4 Using BDDs to Solve Gen-Kill Problems

We notice that analyses expressed in terms of data-flow frameworks have a very regular structure. Let us first restrict ourselves to problems where the solution of the analysis can be represented as sets, and further to problems that can be represented in terms of Gen and Kill functions on sets.

In the above setting, let the meet semi-lattice we consider be \( L \) (or more completely \( \langle L, \sqsubseteq_L, \sqcap_L, \top_L \rangle \)). In this case, the result of the analysis of a program \( P \) can be represented as a relation from the set of program-points of \( P \) to the lattice \( L \).

\[
\text{Analysis}_P : \text{Pt}_P \leftrightarrow L
\]

Furthermore, we can consider the control-flow information as a relation, \( \text{CFG}_P \), from program-points to program points, where a point \( l_1 \) is related to a point \( l_2 \) by \( \text{CFG}_P \) if there is an edge in the control-flow graph of \( P \) taking node \( l_1 \) to node \( l_2 \).

\[
\text{CFG}_P : \text{Pt}_P \leftrightarrow \text{Pt}_P
\]

We can also represent the initialization required for the problem as a relation, \( \text{Init}_P \), relating the entry (or exit) program-point of the program with the \( \top_L \) (or \( \bot_L \)) value of the semi-lattice.

\[
\text{Init}_P : \text{Pt}_P \leftrightarrow L
\]
Let us initially restrict ourselves to analysis problems where

- the $\cap_L$ operator is set-union
- the analysis is a *backward* analysis

An example of such a problem is the *live-variables* analysis problem. The analysis can therefore be expressed in a data-flow framework as the equation:

$$LV_{entry}(pt) = (\bigcup_{pt' \in \text{succ}(pt)} LV(pt') - \text{Kill}_{pt}) \cup \text{Gen}_{pt}$$

Given a relation, $S$, representing an approximation to the analysis solution, and the relation for the control-flow graph, we can represent the expression

$$\bigcup_{pt' \in \text{succ}(pt)} LV(pt')$$

in the data-flow framework as the relational composition

$$\text{CFG}_P ; S$$

Intuitively, relational composition with $\text{CFG}_P$ represents the *backward* flow of information in the analysis. Now, the entire analysis equation can be represented by the expression

$$((\text{CFG}_P ; S) - \text{Kill}_P) \cup \text{Gen}_P \cup \text{Init}_P$$

and the analysis itself can be expressed as the fix-point of the function

$$\lambda S . ((\text{CFG}_P ; S) - \text{Kill}_P) \cup \text{Gen}_P \cup \text{Init}_P \quad (1)$$

This expression of the analysis can be verified by the proposition

**Proposition 4.1 Soundness**

**Proof:**

Note that the only part of equation 1 that depends on the analysis being a backward analysis, and having set union as the $\cap_L$ operator is

$$(\text{CFG}_P ; S)$$

Only this part of the equation needs to be changed to address analyses that are not backward analysis or those that have set intersection as the $\cap_L$ operator.

It is quite straightforward to see that in order to consider forward analysis, it is sufficient to consider the *transpose*, $\text{CFG}_P^{-1}$, of the relation representing the control-flow graph for relational composition.
Proposition 4.2  *Soundness*

*Proof:*

The case of problems where the $\cap_L$ operator is set intersection is slightly more complex. In this case, the relational composition expression has to be of the form:

$$\text{(CFG}_P \cup S)$$

where $\overline{S}$ represents the *inverse* of the set $S$ (with respect to some universal set).

Proposition 4.3  *Intersection over Finite Domains*

*Proof:*

The above results are tabulated in Table 1. Note that the relational approach to analysis requires the solution of different equations for different problems – it, in effect, results in four different frameworks for different classes of analysis problems. This is in contrast to data-flow frameworks where the same framework and equation can be used to solve a large class of analysis problems.

<table>
<thead>
<tr>
<th></th>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\cup$</td>
<td>(CFG$_P \cup S$)</td>
<td>(CFG$_P^{-1} \cup S$)</td>
</tr>
<tr>
<td>$\cap$</td>
<td>(CFG$_P \cap \overline{S}$)</td>
<td>(CFG$_P^{-1} \cap \overline{S}$)</td>
</tr>
</tbody>
</table>

Table 1: Relational analysis frameworks

5  **Beyond Gen-Kill Problems**

6  **Interprocedural Analysis**

7  **Case Study**

7.1  **Alias Analysis**

7.2  **Analysis using BDDs**

7.3  **Alternative use of BDDs**

8  **Conclusion and Future Work**