Worst-Case Groundness Analysis using Positive Boolean Functions

Michael Codish

Department of Computer Science and Software Engineering, The University of Melbourne, Australia

Abstract

This note illustrates a theoretical worst-case scenario for groundness analyses obtained through abstract interpretation over the abstract domain of positive Boolean functions. A sequence of programs is given for which any Pos-based abstract interpretation for groundness analysis follows an exponential chain. Another sequence of programs is given for which a state-of-the-art implementation based on ROBDD's gives a result of exponential size in only three iterations. The moral of the story is that a serious Pos analyser must incorporate some form of widening to protect itself from the inherent complexity of the underlying domain.

1 Introduction

Many analyses for logic programs use the lattice of positive Boolean functions ordered by implication (Pos) to express dependencies between program variables. One of the best known applications of this type of analysis involves reasoning about groundness dependencies in logic programs (e.g. [10]). For example, in this context the formula $x \wedge (y \leftarrow z)$ is interpreted to describe a program state in which x is definitely bound to a ground term and there exists a grounding dependency such that whenever z becomes bound to a ground term then so does y.

For a predicate p/n (of arity n), a program analysis using Pos consists in computing an increasing chain $\{\varphi_i\}_{i=1}^{i=k+1}$ of n-ary positive Boolean functions such that $\varphi_k \equiv \varphi_{k+1}$. The result of the analysis is φ_k which is the least fixed

[1] On leave from the Department of Mathematics and Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel, E-mail: `mcodish@cs.bgu.ac.il`.

Preprint submitted to Elsevier Preprint *10 February 1999*

point of a corresponding abstract semantic operator. The cost of a *Pos* analysis is related to the number of iterations to reach a fixed point (and one more to verify that it is a fixed point) and the cost of the operations performed in each iteration. These operations include: computing a join (disjunction), computing a meet (conjunction), projecting the formula (existential quantification), and renaming the variables in the formula. In addition, each iteration of the analysis involves a test for equivalence (to determine if a fixed point has been reached). While, in theory, the longest chain in *Pos* is exponential (in the number of variables) and the equivalence problem for *Pos* is in co-NP [1, Section 4], experimental results indicate that the use of this domain is viable.

One of the most promising techniques devised to represent, maintain and manipulate positive propositional formulae for program analyses involves the use of binary decision diagrams (BDD's) and their variants, such as reduced order binary decision diagrams (ROBDD's) [4]. BDD's were originally used in the field of circuit design and have been shown to lead to fast and accurate analyses of logic programs for a wide range of benchmarks [1,2,5,12,7].

The number of iterations required in a groundness analysis based on BDD's is typically small ($2-3$) and very rarely more than 4 or 5 per predicate. Analysing the Aquarius Prolog compiler (a benchmark with over 3000 clauses) is reported to encounter a strongly connected component which involves 10-12 iterations [7,9]. The size of the BDD's encountered in an analysis is however often witnessed to become quite large. Recent work suggests several techniques to reduce the size of BDD's with no loss of accuracy [2,3]. Other work has indicated that widening techniques [6] can be applied to restrict the size of BDD's [7].

Section 2 of this note illustrates that size is not the only problem facing *Pos* based program analyses. A series of pathological inputs is described for which any (accurate) *Pos* based groundness analysis (obtained as an abstract interpretation) must follow a chain that is exponential (in the number of symbols in the program). Section 3 illustrates an example which is problematic for an implementation based on BDD's.

While cases of these types are not likely to arise in real world situations, a serious *Pos* analyser must protect itself from the inherent complexity of the underlying domain. Widening techniques should be introduced for two reasons: both to restrict the size of the representation as well as to restrict the number of iterations. A first step in this direction is proposed in [8].

2 An Exponential Chain

The number of iterations in a *Pos* based analysis is essentially the length of a corresponding chain of formulae. The (*Pos* based) groundness analysis of the following predicate "*chain*/n" will iterate through a chain of maximal length consisting of 2^n elements. The program consists of a single predicate of arity n with $n+1$ binary clauses and contains no function symbols other than a single constant. Hence, the program is of size quadratic in n.

$$
\begin{array}{ll}
chain(c, c, c, \quad \ldots, \quad c, c, c). & \\
chain(V, c, c, \quad \ldots, \quad c, c, c) & \leftarrow chain(A, V, V, \ldots, V, V, V). \\
chain(X_1, V, c, \quad \ldots, \quad c, c, c) & \leftarrow chain(X_1, A, V, \ldots, V, V, V). \\
\quad \vdots & \\
chain(X_1, X_2, \ldots, X_{n-2}, V, c) & \leftarrow chain(X_1, X_2, \ldots, X_{n-2}, A, V). \\
chain(X_1, X_2, \ldots, X_{n-2}, X_{n-1}, V) & \leftarrow chain(X_1, X_2, \ldots, X_{n-2}, X_{n-1}, A).
\end{array}
$$

A groundness analysis of *chain*/n can be viewed as enumerating the models of the Boolean function *true* (in the reverse lexicographic order) which is the result of the analysis. Each iteration results in a function with one additional model. For example, with $n = 3$, the chain of formulae encountered in a bottom-up evaluation is

$$
\begin{array}{l}
false, \quad X_1 \wedge X_2 \wedge X_3, \quad X_1 \wedge X_2, \quad X_1 \wedge (X_2 \vee X_3), \quad X_1, \\
\qquad\qquad X_1 \vee (X_2 \wedge X_3), \quad X_1 \vee X_2, \quad X_1 \vee X_2 \vee X_3, \quad true
\end{array}
$$

The initial formula (*false*) has no models. The next has only one model: $\{(1,1,1)\}$, the next has two models: $\{(1,1,1),(1,1,0)\}$, then three: $\{(1,1,1),$ $(1,1,0),(1,0,1)\}$, and so forth. Note that each approximation to the final result (a Boolean formula for *chain*/n) "triggers" at least one clause in the program to derive the next approximation which has exactly one additional model. Since the number of clauses is linear in n, any fixed chaotic iteration strategy will involve an exponential number of iterations.

3 An Exponential BDD

The next example takes advantage of the fact that some functions are known to give a poor representation using BDD's. As an example Bryant notes in [4] that the BDD representation of

$$(X_1 \wedge X_{n+1}) \vee (X_2 \wedge X_{n+2}) \vee \cdots \vee (X_n \wedge X_{2n})$$

requires 2^{n+1} vertices (although with the variables numbered in their order of occurrence $2n+2$ vertices suffice). In [11] this function (termed an AND–OR circuit) is given as Figure 1.2.9. A (*Pos* based) groundness analysis of the following program results in such a function for the last argument of p/(2n+1):

$$
\begin{array}{ll}
p(X_1,\ldots,X_n,Y_1,\ldots,Y_n,Z) \leftarrow & \qquad append([\,],X,X). \\
\quad append(X_1,Y_1,Z_1), & \qquad append([X|Xs],Ys,[X|Zs]) \leftarrow \\
\quad append(X_2,Y_2,Z_2), & \qquad \quad append(Xs,Ys,Zs). \\
\qquad \cdots & \qquad select(Z_1,Z_1,\ldots,Z_n). \\
\quad append(X_n,Y_n,Z_n), & \qquad select(Z_2,Z_1,\ldots,Z_n). \\
\quad select(Z,Z_1,Z_2,\ldots,Z_n). & \qquad \cdots \\
 & \qquad select(Z_n,Z_1,\ldots,Z_n).
\end{array}
$$

Note that the groundness analysis of *append* is obtained in 3 iterations resulting in terms of the form $Z_i \leftrightarrow (X_i \wedge Y_i)$; and that the call to *select* results in $Z \leftrightarrow (Z_1 \vee Z_2 \vee \cdots \vee Z_n)$ (which is obtained in the first iteration).

4 Conclusion

Both of the pathological examples described in this note have proven fatal for a state-of-the-art *Pos* based groundness analyser which uses a BDD representation and combines both top-down and bottom-up analysis techniques. Analysis of both types of programs cannot be improved using techniques such as those described in [2,3] to restrict the size of the BDD's encountered in the course of an analysis.

On one hand, this should come as no surprise given the complexity of the underlying domain. On the other hand the need to incorporate widening techniques in any *Pos* based analysis has yet to achieve proper attention by the designers of BDD based *Pos* analysers.

Acknowledgement

Many thanks to Peter Schachte for gracefully helping me to break his BDD *Pos* based groundness analyser.

References

[1] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Boolean functions for dependency analysis: Algebraic properties and efficient representation.

Science of Computer Programming, 31(1):3–45, 1998.

[2] R. Bagnara. A reactive implementation of *Pos* using ROBDDs. In *Programming Languages: Implementations, Logics and Programs*, pages 107–121. Springer, 1996. LNCS 1140.

[3] R. Bagnara and P. Schachte. Factorizing equivalent variable pairs in ROBDD-based implementations of *Pos*. In *Proceedings of the Seventh International Conference on Algebraic Methodology and Software Technology*. Springer, January 1999. (to appear).

[4] R. Bryant. Graph based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[5] M.-M. Corsini, K. Musumbu, A. Rauzy, and B. Le Charlier. Efficient Bottom-up Abstract Interpretation of Prolog by means of Constraint Solving over Finite Domains. In *Programming Language Implementation and Logic Programming*, pages 75–91. Springer-Verlag, 1993. LNCS 714.

[6] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In *Programming Language Implementation and Logic Programming*, pages 269–295. Springer-Verlag, 1992. LNCS 631.

[7] C. Fecht. *Abstrakte Interpretation logischer Programme: Theorie, Implementierung, Generierung.* PhD thesis, Universität des Saarlandes, 1997.

[8] A. Heaton, M. Abo-Zaed, M. Codish, A. King, and P. Hill. Widening *pos* for efficient and scalable groundness analysis of logic programs. Technical report, Ben-Gurion University of the Negev, 1998. ftp://ftp.cs.bgu.ac.il/pub/people/mcodish/wpos.ps.gz.

[9] A. King, J-G. Smaus, and P. Hill. Quotienting share for dependency analysis. In Doaitse Swierstra, editor, *Proceedings of the European Symposium on Programming*, LNCS. Springer-Verlag, March 1999. (to appear).

[10] K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM Lett. Program. Lang. Syst.*, 2(4):181–196, 1993.

[11] S. Minato. Graph-based representations of discrete functions. In Tsutomu Sasao and Masahiro Fujita, editors, *Representations of Discrete Functions*, chapter 1. Kluwer Academic Publishers, 1996.

[12] P. Van Hentenryck, A. Cortesi, and B. Le Charlier. Evaluation of the domain *Prop*. *Journal of Logic Programming*, 23(3):237–278, 1995.