Straight ROBDDS are not the Best for *Pos*

Roberto Bagnara (bagnara@di.unipi.it)
Dipartimento di Informatica, Università di Pisa,
Corso Italia 40, 56125 Pisa, Italy

1 Introduction

The subject of groundness analysis for (constraint) logic programs has been widely studied, and interesting domains have been proposed. *Pos* has been recognized has the most suitable domain for capturing the kind of dependencies arising in groundness analysis. Its (by now standard) implementation is directly based on *reduced ordered binary-decision diagrams* (ROBDDs), a well-known symbolic representation for Boolean functions [4]. Even though several authors have reported positive experiences using ROBDDs for groundness analysis, in the literature there is no reference to the problem of the efficient detection of those variable which are deemed to be ground (or *true*, at the abstract level) in the context of a ROBDD. This is not surprising, since most currently implemented analyzers need to derive this information only *at the end* of the analysis and only for presentation purposes. Things are much different when this information is required *during* the analysis. This need arises when dealing with languages which employ some sort of *delay mechanism*, which are typically based on groundness conditions. In these cases, the *naïf* approaches are too inefficient, since the abstract interpreter must quickly (and often) decide whether a constraint is delayed or not. Fast access to ground variables is also necessary when aliasing analysis is performed using a domain not keeping track of ground dependencies.

We have shown that, by using an hybrid representation instead of the standard (purely ROBDD-based) one, it is possible to have constant-time, fast access to ground variables. Moreover, the new representation *improves* (not make worse, as it can be expected) the overall efficiency (on both time and space) of groundness analysis. Due to extreme space limitations we just give a rough idea of how the hybrid representation and its operations look like. We also report on the experimental results we obtained with the China analyzer. The interested reader is referred to [3] for a complete account of this work.

2 A New, Hybrid Implementation for *Pos*

The observation of many constraint logic programs shows that the percentage of variables which are found to be ground during the analysis, for typical invocations, is as high as 80%. This suggests that representing *Pos* elements simply by means of ROBDDs, as in [1, 8], is not the best thing we can do (whence the title).

Here we propose an hybrid implementation where each *Pos* element is represented by a pair: the first component is the set of true variables (just as in the domain used in early groundness analyzers [7, 6]); the second component is a ROBDD. In each element of this new representation there is no redundancy: the ROBDD component does not contain any information about true variables. This solution uses the more efficient representation for each kind of information: "surely ground variables" are best represented by means of sets (bit-vectors, at the implementation level), whereas ROBDDs are used only for "conditional" and "disjunctive" information.

The hybrid representation has two major advantages: (a) it gives fast, constant-time access to true variables; and, (b) allows for keeping the ROBDDs small, during the analysis, when many variables come out to be true, as it is often the case. Notice that, while having many true variables, in a straight ROBDD implementation, means that the *final* ROBDDs will be very similar to linear chain of nodes, the intermediate steps still require the creation (and disposal) of complex (and costly) ROBDDs. This phenomenon is avoided as much as possible in the hybrid representation. In fact, the operations on the hybrid representation make use of ROBDD's operations only when strictly necessary. When this happens, expensive operations like logical conjunction and disjunction are invoked with operands of the smallest possible size. In particular, we exploit the fact that the restriction operation (also called *valuation* or *co-factoring*) is relatively cheap. However, we cannot avoid searching for true variables. In programs where many variables are ground the ROBDDs generated will be kept small, and so also the cost of searching will be diminished.

3 Experimental Evaluation

The ideas presented in this paper have been experimentally validated in the context of the development of the China analyzer [2]. China is a data-flow analyzer for CLP($\mathcal{H}$, $\mathcal{N}$) languages (i.e. Prolog, CLP($\mathcal{R}$), `clp(FD)` and so forth) written in Prolog and C++. It performs bottom-up analysis deriving information on both call and success patterns by means of program transformations and optimized fixpoint computation techniques. The assessment of the hybrid domain has been done in a quite radical way. In fact, we have compared the standard, pure BDD-based implementation of *Pos* against the hybrid domain on the following problem: *deriving, once for each clause's evaluation, a boolean vector indicating which variables are ground and which are not.* This is a very minimal demand for each analysis requiring the knowledge about definitely ground variable *during* the analysis. We have thus performed the analysis of a number of programs on a domain similar to `Pat`(*Pos*) [5], switching off all the other domains currently supported by China[1]. `Pat`($\Re$) is a generic structural domain which is parametric with respect to any abstract domain $\Re$. Roughly speaking, `Pat`($\Re$) associates to each variable the following informaton: (1) a *pattern*, that is to say, the principal functor and subterms which are bound to

[1]Namely, numerical bounds and relations, aliasing, and polymorphic types

	Analysis time (sec)			N. of BDD nodes		
Program	STD	HYB	S/H	STD	HYB	S/H
`CS`	1.06	0.6	1.77	12387	391	31.7
`Disj`	1.06	0.6	1.77	72918	176	414.3
`DNF`	5.17	4.4	1.18	5782	111	52.1
`Gabriel`	1.13	0.74	1.53	28634	10472	2.73
`Kalah`	3.92	2.02	1.94	43522	645	67.5
`Peep`	6.13	5.52	1.11	176402	128332	1.37
`PG`	0.37	0.25	1.48	3732	86	43.4
`Plan`	0.59	0.5	1.18	1736	65	26.7

Table 1: Experimental results obtained with the China analyzer.

the variable; and (2) the "properties" of the variable, which are delegated to the $\Re$ domain (the two implementations of *Pos*, in our case). As reported in [8], `Pat`(*Pos*) is a very precise domain for groundness analysis.

The experimental results are reported in Table 1. The table gives, for each program, the analysis times and the number of ROBDD nodes allocated for the standard implementation (STD) and the hybrid one (HYB), respectively. It also shows the ratio STD/HYB for the above mentioned figures (S/H). The computation times have been taken on a 80486DX4 machine running Linux 1.3.64. The tested programs have become standard for the evaluation of data-flow analyzers. The results indicate that the hybrid implementation outperforms the standard one in both time and space efficiency. The systematic speed-up obtained was not expected. Indeed, we were prepared to content ourselves with a moderate slow-down which would have been recovered thanks to the fast access to ground variables. The space figures show that we have achieved significant (and sometimes huge) savings in the number of allocated ROBDD nodes. With the hybrid domain we are thus able to keep the ROBDDs which are created and traversed during the analysis as small as possible. This phenomenon is responsible for the speed-up. It seems that, even for programs characterized by not-so-many ground variables, there are always enough ground variables to make the hybrid implementation competitive. This can be observed, for instance, in the case of the `Peep` program, which was analyzed with a non-ground, most-general input pattern. The following observations are important for a full understanding of Table 1:

1. we are not comparing against a poor standard implementation of *Pos*, as can be seen by comparing the analysis times with those of [8]. The ROBDD package we are using is fine-tuned: it employs separate chaches for the main operations (with hit-rates in the range 95%–99% for almost all the programs we have tried), specialized and optimized versions of the important operations over ROBDDs, as well as aggressive memory allocation strategies. Indeed, we were led to the present work by the apparent impossibility of further optimizing the standard

implementation. Moreover, the hybrid implementation has room for improvement, especially for what concerns the handling of bit-vectors.

2. We are not taking into account the cost of garbage-collection for ROBDD nodes. In particular, the sizes of the relevant data-structures were chosen so that the analysis of the tested programs can run to completion without any node deallocation or reallocation.

3. In a truly reactive combination of domains, the set of ground variables is not needed only at the end of each clause's evaluation (this is the optimistic hypothesis under which we conducted the experimentation), but at each body-atom evaluation for each clause. In this context the hybrid implementation, due to its incrementality, is even more favoured with respect to the standard one (which is not incremental at all).

In conclusion, the experimental results indicate that the hybrid domain outperforms, from any point of view, the standard implementation based on ROBDDS only. Surprisingly enough, we have thus been able to assess the superiority of the hybrid domain even for those cases where fast access to ground variables is not important.

References

[1] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of boolean functions for dependency analysis. Technical Report 94/211, Dept. Computer Science, Monash University, Melbourne, 1994.

[2] R. Bagnara. On the detection of implicit and redundant numeric constraints in CLP programs. In *Proceedings GULP-PRODE '94*, pages 312–326, Peñíscola, Spain, September 1994.

[3] R. Bagnara. A reactive implementation of *Pos* using ROBDDs. Technical Report TR-96-19, Dipartimento di Informatica, Università di Pisa, 1996.

[4] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[5] A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Combinations of abstract domains for logic programming. In *Conference Record of POPL '94*, pages 227–239, Portland, Oregon, January 1994.

[6] N. D. Jones and H. Søndergaard. A Semantics-based Framework for the Abstract Interpretation of Prolog. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*, pages 123–142. Ellis Horwood Ltd, 1987.

[7] C. Mellish. Some Global Optimizations for a Prolog Compiler. *Journal of Logic Programming*, 2:43–66, 1985.

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