ICT-ENERGY LETTERS

Energy analysis of C program by analyzing its Horn clauses representation

Bishoksan Kafle and John P. Gallagher Roskilde University, Denmark

***Abstract*— In this paper, we present an approach to automatic resource (energy) analysis of C programs. Our approach combines ideas from abstract interpretation, ranking function synthesis and complexity analysis in a unified framework. Instead of acting on a source program, the analysis uses an intermediate language known as Constrained Horn clauses (CHCs) and computes bound in terms of the program's input parameters. Our approach is based on the method described by Sinn et al. where a program is abstracted to a lossy vector addition system with states (VASS). A VASS can be conveniently described by a set of CHCs where the constraints have a particular shape. Our analysis derives a VASS from the input Horn clauses using abstract interpretation and ranking function synthesis. Then it computes a lexicographic ranking function, which proves termination of the VASS. Finally a bound is computed directly from the VASS ranking functions. Resource consumption is obtained by multiplying the bound by a suitable resource measure.**

I. INTRODUCTION

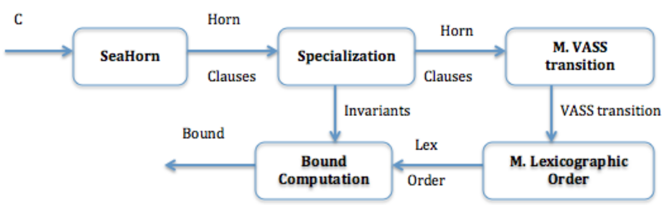
Static bound analysis or resource analysis of programs [1, 2, 3, 4, 5, 10, 12, 14, 15] is an active area of research. As a result of this, several tools and techniques with different flavors have emerged. The use of abstract interpretation [2, 10, 14], symbolic execution [12], computer algebra [1] and typing rules [3, 7, 8] for bound analysis are common in the literature. Some of these techniques are scalable but compute loose bounds while others compute tight bounds but are not scalable. Finding a balance between scalability of analysis and tightness of a bound is a matter of ongoing research. There are tools such as CiaoPP[1], SPEED [4], PUBS [1], Rank [2], LOOPUS [11, 12] which derive impressive bounds. Some of these tools are language dependent and others are not. So, we would like to purpose CHCs as an intermediate representation language for bound analysis since it provide a suitable intermediate form for expressing the semantics of a variety of programming languages (imperative, functional, concurrent, etc.) and computational models (state machines, transition systems, big- and small-step operational semantics, Petri nets, etc.). This makes us possible to reuse several years of works on constraint logic programming and exploit this representation for our purpose. In general, automatic complexity analysis has gained little attention compared to program verification and termination analysis though several interesting and useful properties about programs can be derived by bound analysis such as loops upper bounds, number of visits to a control location or instruction, the amounts of resources (time, energy, memory etc.) consumed by a program etc.

In this paper, we present an automatic approach to resource analysis of C programs, which uses CHCs as an intermediate language. It can be seen as an adaptation of the method described by Sinn et al. in [12]. We chose this method because it is more flexible than others for example CiaoPP, which involve solving cost equations and they might not have solutions, especially for mutually recursive loops. The program is first translated to a set of CHCs. The analysis then derives a VASS from it using abstract interpretation and ranking functions synthesis. Then it computes a lexicographic ranking function, which proves termination of the VASS. Finally a bound is computed directly from the VASS ranking function. To compute a bound for a program, each such step should be precise/succeed, therefore the choice of the techniques matter a lot. Our work, which is an adaptation of [12] has the following characteristics:

- Our analysis is based on Horn clause representation, which allows reuse of several state of the art tools developed for it, e.g. Horn specialisation, computing over-approximation of the set of Horn clauses etc.;
- Program abstraction: instead of symbolic execution and simple invariant generation as in [12] we use abstract interpretation, which is scalable and computes good invariants. Good invariants seem to be crucial for bound analysis;
- Ranking function generation: instead of guessing ranking functions from the generated invariants and checking if they are local ranking functions for a given transition [12], we use a complete approach based on [9, 13] for computation of ranking functions. They are complete in the sense that if a linear ranking function exists then they will find one. In [12] this is not guaranteed.
- Control flow abstraction and bound computation: furthermore we still benefit from the novelties of [12] for control flow abstraction and bound computation.

[1] Summary of our approach

The architecture of our tool-chain is depicted in Figure 1. The boxes represent different modules/ components and the label on the arrows shows the output or input to and from these components. Given a program written in *C*, we obtain a set of Horn clauses using SeaHorn [6]. The resulting set of clauses are specialised by a *specialzation module, which* also computes invariants for the clauses using abstract interpretation. These set of clauses are fed into the module of *VASS transition*, which computes a VASS. A lexicographic order for the given VASS is computed by the module *lexicographic order*. Then a bound is computed by the bound *computation module* from this lexicographic ordering.



Our running example is presented in Figure 2. It consists of two

nested loops, whose counters depend on the input parameters (*c* and *d*). The program terminates and has a linear bound since the outer loop can only be executed at most the initial value of *a* (that is *c*) times since *a* is decremented by 1 in each iteration of the outer loop. The counter variable *b* of the inner loop is conditionally increased by the outer loop, however this can only be done at most the initial value of *a* (that is *c*) times. So the inner loop can be executed at most *c+d* times. To derive a linear bound, a path sensitive reasoning is necessary. Given this program, our tool extracts three single path linear constraints (SLC) loops, two of them corresponding to the outer loop and one to the inner. Then it computes the local ranking functions for each such loops separately. Since the outer loop has two SLC loops it computes *a* as ranking function for both of the SLC loops and *b* for the SLC loop corresponding to the inner loop. Then the lexicographic ordering *(a,a,b)* is derived, which proves the termination of this program. But during bound computation the tool takes into account the effect of the outer loop on the ranking function of the inner one, that is, the local ranking function of the inner loop is increased by the outer at most by 1 in each iteration. So the inner loop can be executed at most *c+d* times. So the total bound is *c+d*. Let *E* be the worst case energy consumed by this program in one iteration, then *(c+d)*E* is the total energy consumed by this program in the worst case.

```
main( uint c, uint d){
  int a=c,b=d;
  while (a>0){
    if (*)
      b++;
    else
      while (b>0)
        b--;
    a--; }
  return 0;}
```

Figure 2: example program

References

[1] E. Albert, P. Arenas, S. Genaim, M. Gómez-Zamalloa, G.Puebla, D.V. Ramírez-Deantes, G. Román-Díez, and D. Zanardini. Termination and cost analysis with COSTA and its user interfaces. Electr. Notes Theor. Comput. Sci., 258(1):109–121, 2009.

[2] C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In R. Cousot and M. Martel, editors, Static Analysis - 17th International Symposium, SAS 2010, Perpignan, France, September 14-16, 2010. Proceedings, volume 6337 of Lecture Notes in Computer Science, pages 117–133. Springer, 2010.

[3] Q. Carbonneaux, J. Hoffmann, T. Ramananandro, and Z. Shao. End-to-end verification of stack-space bounds for C programs. In M. F. P. O'Boyle and K. Pingali, editors, ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014, page 30. ACM, 2014.

[4] S. Gulwani. SPEED: symbolic complexity bound analysis. In A. Bouajjani and O. Maler, editors, Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings, volume 5643 of Lecture Notes in Computer Science, pages 51–62. Springer, 2009.

[5] S. Gulwaniand, F. Zuleger. The reachability-bound problem. In B. G. Zornand, A. Aiken, editors, Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, Toronto, Ontario, Canada, June 5-10, 2010, pages 292–304. ACM, 2010.

[6] A. Gurfinkel, T. Kahsai, and J. A. Navas. Seahorn: A framework for verifying C programs (competition contribution). In C. Baier and C. Tinelli, editors, Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings, volume 9035 of Lecture Notes in Computer Science, pages 447–450. Springer, 2015.

[7] J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate amortized resource analysis. ACM Trans. Program. Lang. Syst., 34(3):14, 2012.

[8] M. Hofmann and S. Jost. Static prediction of heap space usage for first-order functional programs. In A. Aiken and G. Morrisett, editors, Conference Record of POPL 2003: The 30th SIGPLAN-SIGACT Sympo- sium on Principles of Programming Languages, New Orleans, Louisisana, USA, January 15-17, 2003, pages 185–197. ACM, 2003.

[9] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In B. Steffen and G. Levi, editors, Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, January 11-13, 2004, Proceedings, volume 2937 of Lecture Notes in Computer Science, pages 239–251. Springer, 2004.

[10] A. Serrano, P. López-García , and M.V. Hermenegildo. Resource usage analysis of logic programs via abstract interpretation using sized types. TPLP, 14(4-5):739–754, 2014.

[11] M. Sinn and F. Zuleger. LOOPUS - A tool for computing loop bounds for C programs. In A. Voronkov, L. Kovács, and N. Bjørner, editors, Second International Workshop on Invariant Generation, WING 2009, York, UK, March 29, 2009 and Third International Workshop on Invariant Generation, WING 2010, Edinburgh, UK, July 21, 2010, volume 1 of EPiC Series, pages 185–186. EasyChair, 2010.

[12] M. Sinn, F. Zuleger, and H. Veith. A simple and scalable static analysis for bound analysis and amortized complexity analysis. In A. Biere and R. Bloem, editors, Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings, volume 8559 of Lecture Notes in Computer Science, pages 745–761. Springer, 2014.

[13] K. Sohn and A. V. Gelder. Termination detection in logic programs using argument sizes. In D. J. Rosenkrantz, editor, Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 29-31, 1991, Denver, Colorado, USA, pages 216–226. ACM Press, 1991.

[14] F. Zuleger, S. Gulwani, M. Sinn, and H. Veith. Bound analysis of imperative programs with the size-change abstraction. In E. Yahav, editor, Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings, volume 6887 of Lecture Notes in Computer Science, pages 280–297. Springer, 2011.

[15] S. K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin. Lower Bound Cost Estimation for Logic Programs. In 1997 International Logic Programming Symposium, pages 291–305. MIT Press, Cambridge, MA, October 1997.