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 CiaoPP1, 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 involves solving 
constraint 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 novelities 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 specialization 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.

```c
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


