Bachelor Thesis

GEOMETRIC NON-TERMINATION ARGUMENT FOR INTEGER PROGRAMS

August 13, 2017

By Timo Bergerbusch
Rheinisch Westfälisch Technische Hochschule Aachen
Lehr- und Forschungsgebiet Informatik 2

First Supervisor: Jera Hensel
Second Supervisor: Prof. Dr. Thomas Noll

Acknowledgement

First, I would like to thank Prof. Dr. Jürgen Giesl for giving me the opportunity to work on a timely and relevant topic.

Secondly I would like to thank Jera Hensel, who supervised me during my thesis. I want to thank her for the many patience answers she gave my no matter how obvious the solution was and encouraging me during the whole process. Also I want to thank her for the possibility to write the underlying program the way I wanted to without any restrictions or limits regarding the way of approaching the topic.

Also I want to thank my girlfriend Nadine Vinkelau and all my friends, who encouraged me during my whole studies and not only accepting that I often was short on time, but also strengthen my back during the whole process. Especially I want to thank my good friend Tobias Räwer, who explained many topics to me over and over again to help me pass my exams without demanding anything in return. Thanks to his selfless behaviour I got this far within 3 years.

Finally I want to thank my parents for giving me the possibility to fulfil my desire to study at a worldwide known university. Without the financial support I would not have had this opportunity.

Erklärung Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den August 13, 2017

Abstract

Contents

Chapter 1

Introduction

1.1 Motivation

The topic of verification and termination analysis of software increases in importance with the development of new programs. Even though that for Touring Complete programming languages the Halting-Problem is undecidable, and therefore no complete and sound method can exist, a verity of approaches to determine termination are researched and still being developed. These approaches can determine termination on programs, which match certain criteria in form of structure, composition or using only a closed set of operations for example only linear updates of variables.
Given a tool, which can provide a sound and in many scenarios applicable mechanism to prove termination, a optimized framework could analyse written code and find bugs before the actual release of the software [VDDS93]. Contemplating that automatic verification can be applied to termination proved software the estimated annual US Economy loses of $60 billion each year in costs associated software could be reduced significantly [ZC09].

1.2 *AProVE*

One promising approach is the tool AProVE (<u>A</u>utomated <u>Pro</u>gram <u>V</u>erification <u>E</u>nvironment) developed at the RWTH Aachen by the Lehr- und Forschungsgebiet Informatik 2. The *AProVE*-tool (further only called AProVE) for a automatic termination and complexity proving works with different programming languages of major language paradigms like Java (object oriented), Haskell (functional), Prolog (logical) as well as rewrite systems. The conversion of these different languages into (integer) term rewrite systems ((int-)TRS) and subsequently applying various different approaches is what makes this tool strong in meanings of proofing [GAB+17].

Chapter 2

Preliminaries

In order to be able to explain the solution approach we have to declare to, which programs are considered within the Geometric Nontermination. Furthermore we have to define a few structures we work on.

2.1 Geometric Nontermination Argument (GNA)

Adapted from Jan Leikes and Matthias Heizmanns paper *Geometric Nontermination Arguments* [LH14] I will define the considered programs, define the *STEM*and *LOOP*and finally state the definition of Geometric Nontermination Arguments.

2.1.1 Considered Programs

The considered programs in the Geometric Nontermination are not bound to a special programming language. The paper works on so called Linear-Lasso Programs, which in fact are also used within AProVE to derive the so called (int-)TRS. Because of the, within the introduction stated, conversion of the language into *llvm*-code and further analysis the applicability of Geometric Nontermination Arguments are not bound to any program language.
In order to define the specific conditions under which we can use the approach, we take the language Java as an example.

2.1.2 Structure

The structure of the considered programs is quite simple. They contain an optional declaration of the used variables and a *while*-loop. Even though Java would not accept this the conversion to llvm would still be sound. An example of a fulfilling Java program is shown in Figure 2.1.

- The *STEM*:
  The initialization and optional declaration of variables used within the *while*-loop. In the example line 3 and 4 are considered the *STEM*. Also only b is declared.
- The guard:
  The guard of the *while*-loop is essential to restrict a as we will see in . With the restriction of $a + b \geq 4$ we can prove termination for $a < 3$ without further analysis, and also to prove termination assume that $a \geq 3$.

- The linear Updates:
  The updates of the variables within the *while*-loop are the most essential part for termination, since their value determine if the guard still holds. The approach works with only linear updates of the variables, so for every variable v_i where $1 \leq i \leq n$ we can have a $f(v_i) = a_1 * v_1 + ... + a_n * v_n$ with $n \in \mathbb{N}$. Note since we work on int-TRS it is sufficient for n to be in $\mathbb{N}$.

```java
1  int main(){
2
3    int a;
4    int b=1;
5
6    while(a+b>=4){
7      a=3*a+b;
8      b=2*b;
9    }
10 }
```

the *STEM*

the guard

the linear update

Figure 2.1: A Java program fulfilling the conditions to be applicable

The guard and linear updates together form the so called *LOOP*.

After ... we finally receive the equivalent int-TRS shown in Figure 2.2. As we can see the original program can be recognized quite easily. The first rule in line 1 denotes the *STEM*, while the second line equals the loop *LOOP*.

$$
\begin{array}{rl}
1 & \overbrace{f_1 -> f_2(1+3*c, 2) : | : c > 2\&\&8 < 3*c}^{\textit{STEM}} \\
2 & \underbrace{\underbrace{f_2(a,b) \rightarrow f_2(3*a+b, 2*b)}_{\texttt{linear update}} : | : \underbrace{3*a > 29\&\&a+b > 11\&\&31 < 3*a+b\&\&3 < 2*b}_{\texttt{guards}}}_{\textit{LOOP}}
\end{array}
$$

Figure 2.2: The int-TRS corresponding to the Java program in Figure 2.1

Neglecting the conditional terms for now the declaration of b is set in line 1 obviously to 2, because of the one circle the GRAPH has to compute in order to find a loop. The definition of a is more difficult and will be shown within REF . Also the update within line 2 is the same as in Figure 2.1 line 7 and 8.

Definition 2.1.1 (Guard Matrix, Guard Constants)**.** *Let $n \in \mathbb{N}$ be the number of distinct variables, v_i $1 \leq i \leq n$ the i-th distinct variable name , $m \in \mathbb{N}$ be the number of guards, r_i $1 \leq i \leq m$ the i-th guard, $a_{i,j} \in \mathbb{N}$ $1 \leq i \leq n$, $1 \leq j \leq m$ the factor of v_i in g_j and c_i be the constant term within r_j.*

Then the Guard Matrix $G \in \mathbb{N}^{m \times n}$ is defined as $G_{i,j} = a_{i,j}$ and Guard Constants $g \in \mathbb{N}^m$ are defined as $g_i = c_i$.

Example 1. *The corresponding Guard Matrix to Figure 2.2 is* $G = \begin{pmatrix} 3 & 0 \\ 1 & 1 \\ 3 & 1 \\ 0 & 2 \end{pmatrix}$ *and the Guard Constants is* $g = \begin{pmatrix} 29 \\ 11 \\ 31 \\ 3 \end{pmatrix}$

2.2 Reverse-Polish-Notation-Tree

2.3 *SMT*-Problem

Chapter 3

Geometric Non-Termination

3.1 Derivation of the *STEM*

3.2 Derivation of the *LOOP*

3.2.1 The Update Matrix

3.2.2 The Guard Matrix

3.2.3 The Iteration Matrix

3.3 Derivation of the *SMT*-Problem

3.3.1 The Domain Criteria

3.3.2 The Initiation Criteria

3.3.3 The Point Criteria

3.3.4 The Ray Criteria

3.4 Verification of the Geometric Non-Termination Argument

Chapter 4

Benchmarks

Chapter 5

related work

Bibliography

[GAB+17] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, et al. Analyzing program termination and complexity automatically with aprove. *Journal of Automated Reasoning*, 58(1):3–31, 2017.

[LH14] Jan Leike and Matthias Heizmann. Geometric series as nontermination arguments for linear lasso programs. *arXiv preprint arXiv:1405.4413*, 2014.

[VDDS93] Kristof Verschaetse, Stefaan Decorte, and Danny De Schreye. Automatic termination analysis. In *Logic Program Synthesis and Transformation*, pages 168–183. Springer, 1993.

[ZC09] Michael Zhivich and Robert K Cunningham. The real cost of software errors. *IEEE Security & Privacy*, 7(2), 2009.