Termination of Recursive Functions by Lexicographic Orders of Linear Combinations

Raphael Douglas Giles
raphaeldouglasgiles@gmail.com
UNSW Sydney
Australia

Abstract

This paper presents an improvement to Isabelle/HOL's lexicographic termination algorithm. This paper also shows how to encode positive vector-component maximisation as a linear program.

CCS Concepts: • **Theory of computation → Linear programming**; *Pattern matching*.

Keywords: termination checking, pattern matching, functional programming, linear programming

ACM Reference Format:
Raphael Douglas Giles. 2022. Termination of Recursive Functions by Lexicographic Orders of Linear Combinations. In *Companion Proceedings of the 2022 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion '22), December 5–10, 2022, Auckland, New Zealand*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3563768.3563958

Introduction

The widely used proof assistant Isabelle/HOL has a simple yet effective algorithm [3] for generating proofs that certain recursive functions terminate. Isabelle/HOL's termination checker works by finding a lexicographic ordering on the measures generated by each argument of the function using the structural ordering [2] which decreases at every recursive call. This project seeks to extend this algorithm to check for linear combinations of these measures with coefficients in the natural numbers to prove that a broader class of functions terminate.

SPLASH Companion '22, December 5–10, 2022, Auckland, New Zealand

ACM ISBN 978-1-4503-9901-2/22/12…$15.00
https://doi.org/10.1145/3563768.3563958

Related Work

Isabelle/HOL's lexicographic [3] and size-change [6] termination checkers represent the current state of the art in termination checking for theorem provers, being strictly more powerful than the termination checkers found in Agda [1], [2] and Coq [5]. Our extension of Isabelle/HOL's lexicographic algorithm [3] both maintains the complexity class of and solves a superset of the termination problems solved by this algorithm.

Background

We begin by giving a brief overview of Isabelle/HOL's lexicographic algorithm [3] [2]. Firstly, this algorithm compares the measures of each argument on every recursive call between the input value and the value passed to the recursive call. For example, consider the function $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, expressed using the successor operation $S : \mathbb{N} \to \mathbb{N}$:

$$g(S(x), y) = g(x, S(y))$$
$$g(x, S(y)) = g(x, y)$$

We assume that g, and any function written in this paper, outputs 0 if none of the specified rules apply (i.e. $g(0, 0) = 0$). To represent the size-change information symbolically, let $<$ mean the measure decreased, ? mean we either don't know or the measure increased and $=$ mean the measure didn't change. We can then summarise the changes in size of measures of g in each recursive call with the following matrix [3],

$$\begin{bmatrix} < & ? \\ = & < \end{bmatrix}$$

where each row corresponds to a recursive call and each column corresponds to the measure of each argument. The algorithm then works in the following way: for every column c of our matrix with only $<$ and $=$ entries, remove every row of the matrix with a $<$ entry in c. If we eventually remove all the rows of the matrix from repeating this process, then the function must terminate.

For our example, if we represent one iteration of the algorithm with $\rightsquigarrow$ and the empty matrix as $\varnothing$, this algorithm would look like this:

$$\begin{bmatrix} < & ? \\ = & < \end{bmatrix} \rightsquigarrow \begin{bmatrix} = & < \end{bmatrix} \rightsquigarrow \varnothing$$

The problem

This procedure works for many functions, including the merge function and the Ackermann function; functions which do not obviously terminate. However, consider the following function f:

$$f(S(S(x)), y) = f(x, S(y))$$
$$f(x, S(S(y))) = f(S(x), y)$$

We know that this function must terminate because the sum of the arguments is clearly always decreasing. However, we can't show this using Isabelle/HOL's algorithm. We can see this directly by generating the matrix,

$$\begin{bmatrix} < & ? \\ ? & < \end{bmatrix}$$

which can not be reduced at all using the above procedure since there is no column which only has < and = entries.

Extending the algorithm

Our solution to this problem is to first change how we represent size-change information. Instead of just saying a particular measure decreased or stayed the same and so on, we now use specific, numeric values by which the sizes changed (when we can find such values) as our matrix entries. When we can't say, we put a ? entry. For instance, the example above would become:

$$\begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}$$

This allows us to add the columns together yielding the matrix $\begin{bmatrix} -1 \\ -1 \end{bmatrix}$ which clearly terminates. We provide a Haskell implementation which demonstrates how to generate such a matrix for functions expressed in a simple lambda calculus where functions must be defined by pattern matching (making the calculation of differences in structural size significantly easier).

This operation of adding together columns of numeric values always yields a new measure. For the same reason, taking any linear combination of numeric columns with non-negative natural coefficients is now a valid matrix reduction operation/will yield a valid measure. Finding such a linear combination with natural coefficients is equivalent to doing so with positive rational coefficients, since we can multiply out by a common denominator to get a corresponding natural linear combination. Analogous observations have been made in the context of logic programming [7], [4], though the methods developed there are not directly applicable in our context.

The lexicographic algorithm can still be used, but now finds columns with only non-positive entries and removes rows with negative entries in those columns. So, our idea for extending the algorithm is to find a positive rational linear combination of the numeric columns with only non-positive entries and maximal number of negative entries, which we call the maximal negative entries problem (MNE), then to reduce using the lexicographic algorithm and repeat. We have proven that the function terminates if we can reduce to the empty matrix via this process.

We have proven the MNE problem for a rational $n \times m$ matrix A equivalent to the following linear programming problem:

Maximise:

$$\sum_{i=0}^{n-1} b_i$$

subject to:

$$\begin{aligned} Ax + b + z &= 0 \\ b_i, z_i, x_i &\in \mathbb{Q} \\ 0 &\leq x_i \\ 0 &\leq b_i \leq 1 \\ 0 &\leq z_i \end{aligned}$$

This formulation allows us to use a linear solver to solve the MNE problem in polynomial time. Since the lexicographic algorithm also runs in polynomial time, this means our extended algorithm runs in polynomial time. We also provide a Haskell implementation of this procedure.

Linearity of solutions to this linear program implies that these solutions are unique up to scaling by positive rational numbers. This further implies that this termination algorithm is deterministic. We've also proven that the algorithm will say a function with matrix M terminates if and only if every function with corresponding matrix M terminates. This further implies that given a function f with associated matrix M, if there exists some lexicographic order of linear combinations of the measures in M that decreases at every recursive call, then the algorithm will give a "terminates" result for f.

Conclusion

By extending the way size-change information is represented and the operations used to combine measures, we have improved the capability of the termination checking algorithm underlying Isabelle/HOL's termination checker. We have provided a Haskell implementation of this algorithm in the context of checking the termination of functional programs.

Acknowledgements

This work was done in collaboration with Vincent Jackson and Christine Rizkallah.

References

[1] Andreas Abel. 1998. foetus–termination checker for simple functional programs. *Programming Lab Report* 474 (1998).

[2] Andreas Abel and Thorsten Altenkirch. 2002. A predicative analysis of structural recursion. *Journal of functional programming* 12, 1 (2002), 1–41.

[3] Lukas Bulwahn, Alexander Krauss, and Tobias Nipkow. 2007. Finding lexicographic orders for termination proofs in Isabelle/HOL. In *International Conference on Theorem Proving in Higher Order Logics*. Springer, 38–53.

[4] Samir Genaim, Michael Codish, John Gallagher, and Vitaly Lagoon. 2002. Combining norms to prove termination. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 126–138.

[5] Eduarde Giménez. 1994. Codifying guarded definitions with recursive schemes. In *International Workshop on Types for Proofs and Programs*. Springer, 39–59.

[6] Alexander Krauss. 2007. Certified size-change termination. In *International Conference on Automated Deduction*. Springer, 460–475.

[7] Kirack Sohn and Allen Van Gelder. 1991. Termination detection in logic programs using argument sizes. In *Proceedings of the tenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 216–226.