The SCT Analyser

An analysis tool based on size-change termination

Arne John Glenstrup

DIKU, University of Copenhagen
panic@diku.dk
http://www.diku.dk/topps/Research.html#Products

Abstract. The Size-Change Termination Analyser is a prototype tool for identifying potentially non-terminating loops, or for guaranteeing termination, in a given first-order functional program. The size-change termination principle is general, allowing subject programs to be written using mutual and general recursion, yet the analyser is able to detect termination in many programs. The analyser is implemented with a publicly available web interface for demonstration purposes.

1 Background

The Size-Change Termination (SCT) Analyser is a prototype tool for identifying all potentially non-terminating loops in a given program, as part of the program's verification. It has been jointly developed by Neil D. Jones, Amir Ben-Amram and Chin Soon Lee at TOPPS, DIKU [1]. The web implementation is due to Carl C. Frederiksen [2].

2 Size-change termination

In an size-change terminating program, all infinite call sequences must contain some argument values that are never increased from one call to the next, and are decreased infinitely often. We assume function arguments range over a well-founded domain, so for a size-change terminating program infinite call sequences are impossible, implying termination under normal evaluation for any input.

Although this seems like a rather simple approach, as the value of tests in conditional expressions are not considered, it can detect termination of many programs, including Ackermann's function [2]. The method is general: it does not restrict subject programs to primitive recursion, does not rely on lexicographical ordering of argument values, and handles mutual recursion without any special treatment. The set of size-change terminating functions is large: it has been shown to be identical to Péter's multiple recursive functions [3, 4].

3 The analyser tool

The SCT Analyser takes as input a subject program written in the first-order functional language shown in Fig. 1, and displays as output some function call loops that might not terminate. If no such loops are detected, the program is guaranteed to terminate.

The analyser works in 4 steps:

1. First the size of the return values of functions, relative to the sizes of their input arguments, are approximated.
2. Based on these approximations, a size-change graph (SCG) is generated for each call site.
3. The closure of the set of SCGs under composition is computed.
4. The closure set is checked: each idempotent SCG in this set must have an in situ decreasing parameter $x \xrightarrow{\downarrow} x$.

Any SCG G in Step 4 without an in situ decreasing parameter represents one or more *critical multipaths*, that is, the list of call sites for the SCGs that were composed in Step 3 to yield G.

A web based user interface for the SCT Analyser is publicly available at the DIKU TOPPS pages for demonstration purposes. The user can enter a program or select an example program, and when the program has been analysed, the results are displayed in the browser as shown in Fig. 2. Output diagnostics include SCGs generated for individual call sites, critical multipaths, and the SCG set closure.

4 Further reading

The size-change termination principle was originally published at POPL [1] and is discussed in depth in Lee's PhD Dissertation [5]. Details on the SCT Analyser implementation are described in Frederiksen's Master's Thesis [2].

$$
\begin{array}{rcl}
\mathit{Prog} & ::= & \mathit{Def}_1 \ldots \mathit{Def}_n \\
\mathit{Def} & ::= & \mathit{fn}(x_1, \ldots, x_n)\ \texttt{=}\ \mathit{Expr}^{\mathit{fn}} \\
\mathit{Expr} & ::= & x \\
& \mid & \mathit{con} \\
& \mid & \mathit{con}(\mathit{Expr}_1, \ldots, \mathit{Expr}_n) \\
& \mid & \mathit{des}(\mathit{Expr}) \\
& \mid & \texttt{if}\ \mathit{Expr}_1\ \texttt{then}\ \mathit{Expr}_2\ \texttt{else}\ \mathit{Expr}_3 \\
& \mid & \texttt{let}\ x_1\ \texttt{=}\ \mathit{Expr}_1 \ldots x_n\ \texttt{=}\ \mathit{Expr}_n\ \texttt{in}\ \mathit{Expr}_0 \\
& \mid & \mathit{fn}(x_1, \ldots, x_n) \\
& \mid & \mathit{op}(x_1, \ldots, x_n) \\
x & := & \text{identifier beginning with lower case} \\
\mathit{fn} & := & \text{identifier beginning with lower case, not in } \mathit{op} \\
\mathit{con} & := & \text{capitalized identifier} \\
\mathit{des} & := & \{\texttt{1st}, \texttt{2nd}, \texttt{3rd}, \ldots\} \\
\mathit{op} & := & \text{primitive operator } \{\texttt{eq}, \texttt{equal}, \ldots\}
\end{array}
$$

Fig. 1. First-order functional language syntax treated by the analyser

References

1. Lee, C.S., Jones, N.D., Ben-Amram, A.M.: The size-change principle for program termination. In: ACM Symposium on Principles of Programming Languages. Volume 28., New York, ACM Press (2001) 81–92

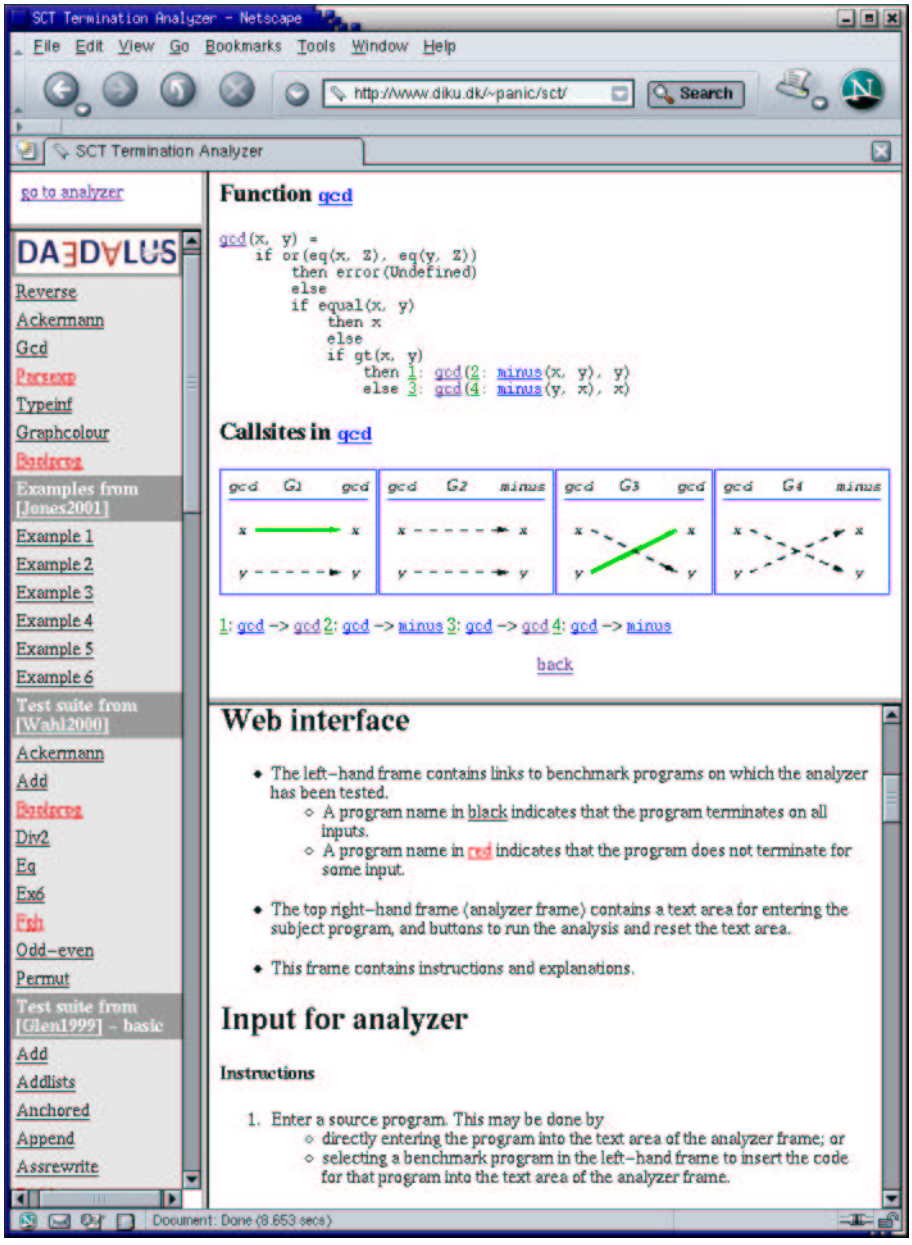

Fig. 2. Screenshot of the SCT Analyser web interface

2. Frederiksen, C.C.: Automatic runtime analysis for first order functional programs. Master's thesis, DIKU, University of Copenhagen, Denmark (2002)
3. Ben-Amram, A.M.: General size-change termination and lexicographic descent. In Mogensen, T., Schmidt, D., Sudborough, I.H., eds.: The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones. Volume 2566 of Lecture Notes in Computer Science. Springer-Verlag, Berlin (2002) 3–17
4. Péter, R.: Rekursive Funktionen (Recursive Functions). Académiai Kiadó, Budapest (1951 (1976)) (Academic Press, New York).
5. Lee, C.S.: Program Termination Analysis and Termination of Offline Partial Evaluation. PhD thesis, University of Western Australia (2002)