NTI+cTI: a Logic Programming Termination Analyzer

Fred Mesnard[1]
LIM, university of La Réunion, France

Étienne Payet[2]
LIM, university of La Réunion, France

Abstract

We describe NTI+cTI, our logic programming termination analyzer that takes part in the Termination Competition 2023. The tool is built from two separate components, NTI for *Non-Termination Inference* and cTI for *constraint-based Termination Inference*, plus an overall main process.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases Termination, Non-termination, Logic Programming

1 NTI

NTI [9] is fully written in Java. It performs automated non-termination proofs of logic programs. It implements a technique that consists in unfolding [6] the program under analysis and in checking whether the produced unfolded clauses satisfy some non-termination criteria [11]. When a proof is successful, NTI provides an example of a non-terminating query. Two kinds of criteria are used.

- The first kind [11] relies on an extension of the "is more general than" relation. It is able to detect infinite derivations that consist of the repeated application of the same sequence ω of clauses, *i.e.*, of the form $Q_0 \Rightarrow_\omega Q_1 \Rightarrow_\omega \cdots$. If the body of an unfolded clause is more general than the head up to some predicate arguments in *neutral position*, then non-termination is detected; more precisely, every query obtained from replacing the neutral arguments of the head with ground terms is non-terminating. So, if such a non-terminating query belongs to the mode of interest then the proof is successful.
- The second kind [10] is able to detect infinite derivations that rely on two sequences ω_1 and ω_2 of clauses, *i.e.*, that have the form $Q_0(\Rightarrow^*_{\omega_1} \circ \Rightarrow_{\omega_2})Q_1(\Rightarrow^*_{\omega_1} \circ \Rightarrow_{\omega_2})\cdots$. It consists in detecting pairs (c_1, c_2) of unfolded clauses of a particular form. Intuitively, c_1 and c_2 are mutually recursive and, in c_1, a context is removed from the head to the body while, in c_2, it is added again.

2 cTI

Termination analysis starts with applying termination inference as presented in [8] [3]. If the mode given in the moded query of interest of the analyzed program implies the inferred termination condition, termination is ensured. This first analysis relies on the *term-size* norm to abstract the logic program and on linear ranking functions, see e.g., [3] for a review. If necessary, termination inference is restarted using the same tool but by combining both the term-size norm and the *list-size* norm, as proposed in [5]. Combining these two norms

[1] Corresponding author
[2] Corresponding author
[3] Source code available here: `https://github.com/FredMesnard/cTI`

doubles the arity of each predicate, hence increases the analysis time so we use it in a second step.

In case these first attemps fail, we switch to BinTerm, the termination analyzer we've built for Java bytecode termination analysis [12]. BinTerm includes various termination tests: linear and eventual ranking functions [2], multi-dimensional linear ranking [1] and the size-change principle [7]. BinTerm analyzes binary Constrained Horn Clauses. Here how we map the original moded query and the original logic program to a binary Constrained Horn Clauses. The original logic program goes through a tabled left-to-right top-down mode analysis starting from the original moded query of interest. An abstract numeric constraint logic program is built using the term-size norm. A numerical model is computed [4]. From these three pieces, a binary Constrained Horn Clauses program is created by a tabled left-to-right top-down interpreter, which keeps only the input arguments of the predicates. Finally, this binary Constrained Horn Clauses program is analyzed by BinTerm.

Again, if necessary, a similar analysis is done by combining both the term-size and the list-size norms.

At last resort, a left-to-right top-down meta-interpreter with occurs-check computes a time-bounded SLD tree for the most general query. If the tree is finite and because we deal with logic programs, any query from the set of concrete queries abstracted by the original moded query terminates.

Otherwise, the termination analyzer cannot conclude.

3 NTI+cTI

The main process of our analyzer performs non-termination and termination analyses in parallel. It launches a thread that runs NTI and another thread that runs cTI. If one thread terminates successfully then the other one is stopped.

References

1 Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In Radhia Cousot and Matthieu 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. URL: `https://doi.org/10.1007/978-3-642-15769-1_8`, `doi:10.1007/978-3-642-15769-1\_8`.

2 Roberto Bagnara and Fred Mesnard. Eventual linear ranking functions. In Ricardo Peña and Tom Schrijvers, editors, *15th International Symposium on Principles and Practice of Declarative Programming, PPDP '13, Madrid, Spain, September 16-18, 2013*, pages 229–238. ACM, 2013. URL: `https://doi.org/10.1145/2505879.2505884`, `doi:10.1145/2505879.2505884`.

3 Roberto Bagnara, Fred Mesnard, Andrea Pescetti, and Enea Zaffanella. A new look at the automatic synthesis of linear ranking functions. *Inf. Comput.*, 215:47–67, 2012. URL: `https://doi.org/10.1016/j.ic.2012.03.003`, `doi:10.1016/j.ic.2012.03.003`.

4 Florence Benoy and Andy King. Inferring argument size relationships with CLP(R). In John P. Gallagher, editor, *Logic Programming Synthesis and Transformation, 6th International Workshop, LOPSTR'96, Stockholm, Sweden, August 28-30, 1996, Proceedings*, volume 1207 of *Lecture Notes in Computer Science*, pages 204–223. Springer, 1996. URL: `https://doi.org/10.1007/3-540-62718-9_12`, `doi:10.1007/3-540-62718-9\_12`.

5 Maurice Bruynooghe, Michael Codish, John P. Gallagher, Samir Genaim, and Wim Vanhoof. Termination analysis of logic programs through combination of type-based norms. *ACM Trans. Program. Lang. Syst.*, 29(2):10, 2007. URL: `https://doi.org/10.1145/1216374.1216378`, `doi:10.1145/1216374.1216378`.

6 Michael Codish and Cohavit Taboch. A semantic basis for the termination analysis of logic programs. *Journal of Logic Programming*, 41(1):103–123, 1999. `doi:10.1016/S0743-1066(99)00006-0`.

7 Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 81–92. ACM, 2001. URL: `https://doi.org/10.1145/360204.360210`, `doi:10.1145/360204.360210`.

8 Fred Mesnard and Roberto Bagnara. cTI: A constraint-based termination inference tool for iso-prolog. *Theory Pract. Log. Program.*, 5(1-2):243–257, 2005. URL: `https://doi.org/10.1017/S1471068404002017`, `doi:10.1017/S1471068404002017`.

9 NTI (Non-Termination Inference). `http://lim.univ-reunion.fr/staff/epayet/Research/NTI/NTI.html` and `https://github.com/etiennepayet/nti`.

10 Étienne Payet. Binary non-termination in term rewriting and logic programming. In Akihisa Yamada, editor, *Submitted to the 19th International Workshop on Termination (WST'23)*, 2023.

11 Étienne Payet and Fred Mesnard. Non-termination inference of logic programs. *ACM Transactions on Programming Languages and Systems*, 28(2):256–289, 2006. `doi:10.1145/1119479.1119481`.

12 Fausto Spoto, Fred Mesnard, and Étienne Payet. A termination analyzer for java bytecode based on path-length. *ACM Trans. Program. Lang. Syst.*, 32(3):8:1–8:70, 2010. URL: `https://doi.org/10.1145/1709093.1709095`, `doi:10.1145/1709093.1709095`.