SizeChangeTool: A Termination Checker for Rewriting Dependent Types

Guillaume Genestier[1,2]

[1] LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay
[2] MINES ParisTech, PSL University

1 Introduction

SizeChangeTool [10] is a fully automated termination checker for the $\lambda\Pi$-calculus modulo rewriting. Its development became essential as various libraries were encoded in an implementation of this logic: the logical framework Dedukti [3].

A logical framework allows the user to define the logic they want to reason with and then use it to actually write proofs. To define a logic in Dedukti, the user provides a set of rewriting rules. Those rules do not only define functions, but can also define types. However, to ensure that the defined type system has good properties, like logical consistency or decidability, the rules must satisfy some properties: termination, confluence and type preservation.

Many criteria have been created to check termination of first-order rewriting. For instance, dependency pairs [2], which evolved in a complete framework [21] or size-change termination [18], just to mention those appearing in this work. The dynamism of this research area is illustrated by the numerous tools participating in the various first-order categories of the termination competition [20]. For higher-order rewriting too, criteria have been crafted, many of them can be found in [15] and a category exists in the competition. However, one can deplore the small number of participants in this category: Only 2 in 2019, including SizeChangeTool!

This lack of implementations is even more visible for rewriting with dependent types, for which criteria have been developed [5, 14], but as far as the author knows, none of them have been implemented.

Outline After presenting the logical system and examples of programs in Sec. 2, we present the criterion used by the tool in Sec. 3. Sec. 4 details the implementation choices of SizeChangeTool and Sec. 5 compares it with the others termination checkers.

2 The $\lambda\Pi$-calculus Modulo Rewriting

The $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$ for short) is an extension of the logical framework LF [12]. It is a system of dependent types where types are identified not only modulo the β-conversion of λ-calculus, but also by user-given rewriting rules.

Definition 1. *$\lambda\Pi/\mathcal{R}$ extends the Pure Type System λP [4] with a finite signature $\mathbb{F}$ and a set of rules $\mathcal{R} = (\Delta, f\,\bar{l} \to r)$ such that $f \in \mathbb{F}$, $\mathrm{FV}(r) \subseteq \mathrm{FV}(\bar{l})$ and Δ is a context associating a type to every variable of $\bar{l}$. $\to_{\mathcal{R}}$ is the closure by substitution and context of $\mathcal{R}$.*

The conversion rule is enriched to take into account rewriting rules:

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : s \qquad A \leftrightarrow^*_{\beta\mathcal{R}} B}{\Gamma \vdash t : B}\ (conv)$$

Note that the constraints on the rewriting rules are very loose. In particular, we do not enforce the rules to be orthogonal, meaning that overlapping or non-linear rules are allowed. Let us give two examples, highlighting the possibilities offered by the system. A more comprehensive example can be found in [7].

Example 2 (Summation of variable arity). *Rewriting rules at type level allow us for instance to define* `F n` *as the type* `Nat ⇒ Nat ⇒...⇒ Nat` *with* n *arrows. With it, we can type the function* `sum` *which is such that* `sum n` $l_1 \ldots l_n$ `=` l_1 `+...+` l_n[1].

```
symbol Nat : TYPE
symbol const 0 : Nat                    symbol const s : Nat ⇒ Nat

symbol plus : Nat ⇒ Nat ⇒ Nat           set infix "+" ≔ plus
  rule 0 + &y → &y                      rule (s &x) + &y → s (&x + &y)

symbol F : Nat ⇒ TYPE
  rule F 0 → Nat                        rule F (s &n) → Nat ⇒ F &n

symbol sum : ∀ n: Nat, F n
  rule sum 0          → 0               rule sum (s 0) → λx, x
  rule sum (s (s &n)) → λx y, sum (s &n) (x + y)
```

Example 3 (Simply-typed λ-calculus). *A simple instance of encoding of logic in* Dedukti *is the simply-typed* λ*-calculus, which is presented here with the type* `typ` *for code of types and* `T` *which decodes an element of* `typ` *into a type of* Dedukti.

```
symbol typ : TYPE              symbol arrow : typ ⇒ typ ⇒ typ
symbol T : typ ⇒ TYPE

symbol lambda : ∀(a b : typ), (T a ⇒ T b) ⇒ T (arrow a b)
symbol appli  : ∀(a b : typ), T (arrow a b) ⇒ T a ⇒ T b
  rule appli &a &b (lambda _ _ &f) &x → &f &x
```

We are interested in the strong normalization of $\rightarrow_{\beta\mathcal{R}=} (\rightarrow_\beta \cup \rightarrow_\mathcal{R})$.

3 Dependency Pairs and Size-Change Termination

Dependency pairs are at the core of all the state-of-the-art automated termination provers for first-order term rewriting systems. Arts and Giesl [2] proved that a first-order rewriting relation is terminating if and only if there is no infinite chain, that is sequence of dependency pairs interleaved with reductions in the arguments. This notion of dependency pair has been extended to higher order [6, 9], however those extensions do not include dependent types, which is a compulsory feature when we are developing a logical framework.

Definition 4 (Dependency pairs). *Let* $f\,\bar{l} > g\,\bar{m}$ *iff there is a rule* $f\,\bar{l} \rightarrow r \in \mathcal{R}$, g *is the head of the left-hand side of a rule and* $g\,\bar{m}$ *is a subterm of* r *maximally applied.*

$f\,t_1 \ldots t_p \tilde{>} g\,u_1 \ldots u_q$ *iff there are a dependency pair* $f\,l_1 \ldots l_i > g\,m_1 \ldots m_j$ *with* $i \leqslant p$ *and* $j \leqslant q$ *and a substitution* σ *such that, for all* $k \leqslant i$, $t_k \rightarrow^*_{\beta\mathcal{R}} l_k\sigma$ *and, for all* $k \leqslant j$, $m_k\sigma = u_k$.

[1] & is used in Dedukti to identify pattern variables in rewriting rules.

One criterion for first-order rewriting is Lee, Jones and Ben-Amram size-change termination criterion (SCT) [18]. It consists in following the arguments through sequences of recursive calls and checking that, in every potential loop, one of them strictly decreases.

Definition 5 (Size-Change Termination)**.** *Let $\triangleright$ be a well-founded order on terms. The call graph $\mathcal{G}(\mathcal{R}, \triangleright)$ associated to $\mathcal{R}$ is the directed labeled graph on the symbols of $\mathbb{F}$ such that there is an edge between f and g iff there is a dependency pair $f\, l_1 \dots l_p > g\, m_1 \dots m_q$. This edge is labeled with the matrix $(a_{i,j})_{i \leq ar(f), j \leq ar(g)}$ where:*

- *if $l_i \triangleright m_j$, then $a_{i,j} = -1$;*
- *if $l_i = m_j$, then $a_{i,j} = 0$;*
- *otherwise $a_{i,j} = \infty$ (in particular if $i > p$ or $j > q$).*

$\mathcal{R}$ is size-change terminating for $\triangleright$ if, in the transitive closure of $\mathcal{G}(\mathcal{R}, \triangleright)$ (using the min-plus semi-ring to multiply the matrices labeling the edges), all idempotent matrices labeling a loop have some -1 on their diagonal.

In [7], we present an adaptation of dependency pairs to $\lambda\Pi/\mathcal{R}$ and prove that (under some conditions) the absence of infinite chains implies the termination of $\rightarrow_{\beta\mathcal{R}}$. After Wahlstedt [22], we used an adaptation of SCT to check the absence of infinite chains of dependency pairs.

Definition 6 (Well-Structured System)**.** *We consider a pre-order $\succeq$ on $\mathbb{F}$ such that if g occurs in the type of f or in the right-hand side of a rewriting rule defining f, then $f \succeq g$. $\mathcal{R}$ is well-structured if for every rule $(\Delta, f\, \bar{l} \rightarrow r)$, if f is of type $\Pi(\bar{x} : \bar{T}).U$, then $\Delta \vdash r : U[\bar{x} \rightarrow \bar{l}]$ is derivable using only symbols smaller or equal to f.*

The result of [7] is:

Theorem 7. *The relation $\rightarrow_{\beta\mathcal{R}}$ terminates on terms typable in $\lambda\Pi/\mathcal{R}$ if $\rightarrow_{\beta\mathcal{R}}$ is locally confluent and preserves typing, $\mathcal{R}$ is well-structured, size-change terminating for the subterm ordering and plain-function passing.*

where "plain-function passing" is a quite restrictive condition on the variable allowed to occur in the right-hand side of rules.

In SizeChangeTool, the criterion used is a (still unpublished) extension of this result where we replace the plain-function passing hypothesis by a condition analogous to strict positivity of inductive types and use the structural ordering introduced in [8] for checking size-change termination.

Extension 8. *The relation $\rightarrow_{\beta\mathcal{R}}$ terminates on terms typable in $\lambda\Pi/\mathcal{R}$ if $\rightarrow_{\beta\mathcal{R}}$ is locally confluent and preserves typing, $\mathcal{R}$ is well-structured, is size-change terminating for the structural ordering and there is a pre-order between types such that for every rule $(\Delta, f\, \bar{l} \rightarrow r)$ and every $c \in \mathbb{F}$ occurring in a l_i, the type of c is strictly positive for this pre-order.*

4 Implementation and interaction with the type-checker

SizeChangeTool takes as input Dedukti files or XTC files, the format of the termination competition [20]. However, XTC does not include dependent types now, hence we proposed a backward compatible extension of the format. In fact, the tool accepts this format extension.

Checking that the provided rules are confluent with β is left to the user. To check it automatically, Dedukti offers an export to the format of the confluence competition. SizeChangeTool performs check of the 4 remaining hypotheses, to use the extension 8 of Thm. 7.

1. *type preservation* is checked by Dedukti assuming that the provided rewrite rules are confluent with β. This algorithm can be found in [3].

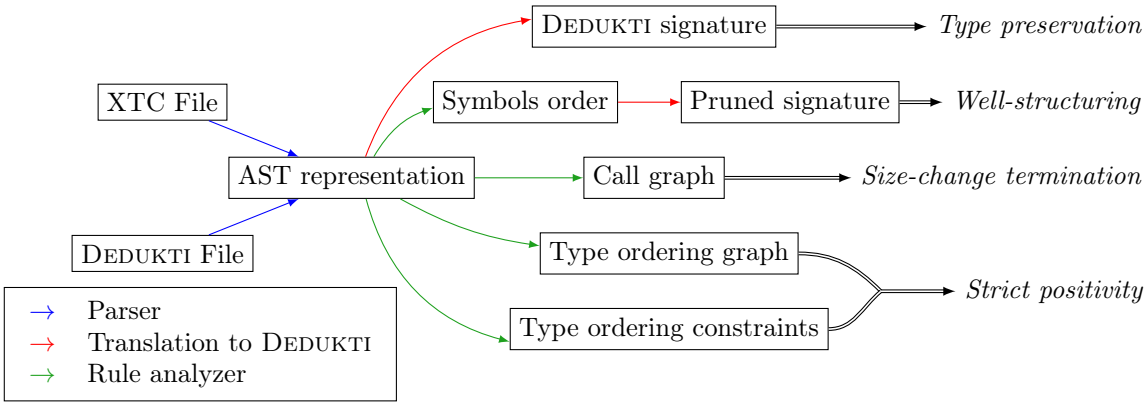

Figure 1: SizeChangeTool Workflow

2. *well-structuring* requires to construct the pre-order described in Def. 6. Once this pre-order is computed, Dedukti is reused to check if it is possible to type the right-hand side of every rule using the pruned signature where symbols greater than the one defined are removed.

3. *size-change termination* requires to analyze every rule in order to extract the dependency pairs. Then the call-graph is constructed. To perform size-change termination checking, one must compute the transitive closure of the call graph and verify the presence of a -1 on the diagonal of every idempotent matrix labeling a loop. This check has been implemented by Lepigre and Raffalli for the language PML_2 [19]. SizeChangeTool reuses their work to analyze the call graph.

4. *the strict positivity condition* requires to have a pre-order on type constructors. The user is not asked to provide this order. While analyzing the rules, SizeChangeTool constructs a graph whose vertices are type constructors and arrows means "is smaller or equal to" as well as a list of constraints of the form "Type constructor A is strictly greater than type constructor B." To check that this relation is a pre-order, one checks that for every constraint "A is strictly smaller than B." there is no arrow between A and B in the transitive closure of the graph.

For the sake of simplicity, representation of terms and rules are mainly shared between Dedukti and SizeChangeTool. So the red arrows on Fig. 1 are (almost) the identity. However those translation functions are made explicit, since one could imagine plugging another type-checker on the rule analyzer offered by SizeChangeTool.

5 Comparison with other tools

As far as the author knows, there are no other termination checker combining dependent types and non-orthogonal rewriting rules. However, dropping one of these features and restricting ourselves to simply-typed higher-order rewriting systems or to dependently-typed orthogonal systems permits comparison with existing tools.

For simply-typed systems, the termination competition [20] proposes a category "higher-order rewriting union beta". In 2019, there were only two tools competing in this category: SizeChangeTool and Wanda [17]. Wanda uses multiple techniques to prove termination:

dependency pairs, polynomial interpretations, HORPO... [15]. Unsurprisingly, the sole criterion used in SizeChangeTool cannot prove as many examples as this wide range of techniques.

However, on the bench of the competition, SizeChangeTool is 11 times faster than Wanda. The speed of SizeChangeTool permits it to show in less than 0.1 second termination of 3 examples on which Wanda is unable to answer with a timeout of 300 seconds: `Mixed_HO_10/deriv.xml` encodes derivation of usual mathematical functions, like:[2]

```
rule der (λx, (&F x) + (&G x)) → λx: real, (der &F x) + (der &G x)
rule der (λx, ln (&F x))       → λx: real, (der &F x) / (&F x)
```

`Hamana_17/churchNum.xml` and `Hamana_17/churchNum2.xml`, contain the Church encoding of natural numbers, with rules like:

```
rule two (λx, &I x) (λx, &J x) (λx, &F1 x) &Y1
     → &I (&I (λy, &J y)) (λy, &F1 y) &Y1
```

The very low time consumption of the presented criterion suggests that Wanda would improve significantly its efficiency by implementing this technique.

If we restrict ourselves to orthogonal systems, it is then possible to compare our technique to the ones implemented in Coq and Agda. Coq essentially implements a higher-order version of primitive recursion [11], whereas Agda uses subterm criterion (a criterion very similar to size-change termination) [1]. Hence, Coq cannot handle function definitions with permuted arguments in function calls, which is not a problem for Agda and SizeChangeTool. Agda recently added the possibility of declaring rewriting rules but this feature is highly experimental and no check is performed on the rules. In particular, Agda termination checker does not handle rewriting rules.

6 Conclusion and future work

For now on, the accepted input files are restricted to Dedukti and XTC files. One could imagine extending it to other input formats, for instance the rewriting rules offered in Agda.

Following the approach adopted by Wanda, one could also just study truly higher-order rules, use a state-of-the-art first-order prover for the remaining rules and then rely on a modularity theorem to conclude. This strategy would improve the performance of SizeChangeTool in the competition, since, according to C. Kop: "about half the benchmarks now do little more than test the strength of the first-order back-end that some higher-order tools use." [16].

One could also think of various enhancement of the criterion, for instance to handle rules with a local increase of the size of the arguments like in:

```
rule f &x → g (s &x)          rule g (s (s &x)) → f &x
```

Hyvernat proposed such an extension of SCT for constructor-based first-order languages [13].

Adapting other so-called "dependency pairs processors" [9] to the $\lambda\Pi/\mathcal{R}$ is of course another active subject of study and would improve the tool.

Now that a higher-order rewriting with dependent types termination prover has been developed, one can hope such development will emulate other researches. The adoption of an extension of XTC and the creation of a category for $\lambda\Pi/\mathcal{R}$ in the competition, would probably support the creation of such new implementations.

[2] For sake of readability, examples are presented in Dedukti syntax and some η-reduction are performed.

References

[1] A. Abel. foetus – Termination Checker for Simple Functional Programs. 1998.

[2] T. Arts, J. Giesl. Termination of term rewriting using dependency pairs. *TCS* 236:133–178, 2000.

[3] A. Assaf, G. Burel, R. Cauderlier, D. Delahaye, G. Dowek, C. Dubois, F. Gilbert, P. Halmagrand, O. Hermant, R. Saillard. Dedukti: a Logical Framework based on the λΠ-Calculus Modulo Theory.

[4] H. Barendregt. Lambda calculi with types. In *Handbook of logic in computer science. Volume 2. Background: computational structures*, p. 117–309. Oxford University Press, 1992.

[5] F. Blanqui. Definitions by rewriting in the calculus of constructions. *MSCS* 15(1):37–92, 2005.

[6] F. Blanqui. Higher-order dependency pairs. WST, 2006.

[7] F. Blanqui, G. Genestier, O. Hermant. Dependency Pairs Termination in Dependent Type Theory Modulo Rewriting. FSCD, 2019.

[8] T. Coquand. Pattern matching with dependent types. TYPES, 1992.

[9] C. Fuhs, C. Kop. A Static Higher-Order Dependency Pair Framework. ESOP, LNCS, 2019.

[10] G. Genestier. SizeChangeTool. `https://github.com/Deducteam/SizeChangeTool`, 2018.

[11] E. Giménez. Codifying Guarded Definitions with Recursive Schemes. TYPES 1994.

[12] R. Harper, F. Honsell, G. Plotkin. A framework for defining logics. *JACM* 40(1):143–184, 1993.

[13] P. Hyvernat. The size-change termination principle for constructor based languages. *LMCS* 2014.

[14] J.-P. Jouannaud, J. Li. Termination of Dependently Typed Rewrite Rules. TLCA, 2015.

[15] C. Kop. *Higher order termination*. PhD thesis, VU University Amsterdam, 2012.

[16] C. Kop. Mail to the termtools list. higher-order union beta category in the TPDB. 19/03/2019.

[17] C. Kop. Wanda. `http://wandahot.sourceforge.net/`.

[18] C. S. Lee, N. Jones, A. Ben-Amram. The size-change principle for program termination. POPL'01.

[19] R. Lepigre. PML_2. `https://github.com/rlepigre/pml`, 2017.

[20] Termination Competition. `http://termination-portal.org/wiki/Termination_Competition`.

[21] R. Thiemann. *The DP framework for proving termination of term rewriting*. PhD thesis, RWTH Aachen University, 2007. Technical Report AIB-2007-17.

[22] D. Wahlstedt. *Dependent type theory with first-order parameterized data types and well-founded recursion*. PhD thesis, Chalmers University of Technology, 2007.