Termination Verification and Complexity Analysis of Term Rewrite Systems

Nao Hirokawa

February 18, 2008

1 Aim

Verifying *termination property* and estimating *runtime complexity* (the longest derivation length) are central issues in program analysis. Among several computational models used for the analysis, *term rewriting* is a simple and powerful computational model. Generally speaking, termination and complexity methods in term rewriting are much more sophisticated than their counterparts used in other program analysis. The current status in term rewriting is as follows:

- **Termination**. Recently, many powerful automatic methods, including the dependency pair method (Arts and Giesl, TCS 2000), have been introduced, and the verifiable class by automatic *termination tools* has been rapidly enlarged. However, still further investigations are needed for handling rewrite systems that are obtained by a transformation from practical programs.
- **Complexity**. In contrast complexity analysis has been *manually* done for each individual system (not only in rewriting). Although there are several important analytical methods, it is fair to say that research toward its automation is sorely lacking.

The aim of my research is to refine automatic methods for verifying termination property, and to establish an automated complexity analysis for term rewrite systems.

2 Approaches

Termination verification and complexity analysis are closely related. Needless to say, complexity analysis presupposes termination property of systems. Often techniques for complexity analysis are obtained by analyzing termination methods. With this observation we investigate two independent issues:

- **New Transformations.** Functional programs (like Scheme and Haskell) are very similar to term rewrite systems, but there are several notorious differences: Programs are content-sensitive (e.g., `if`), allow higher-order functions, and functions are curried or varyadic. These differences are known as main obstacles for existing termination techniques. In order to

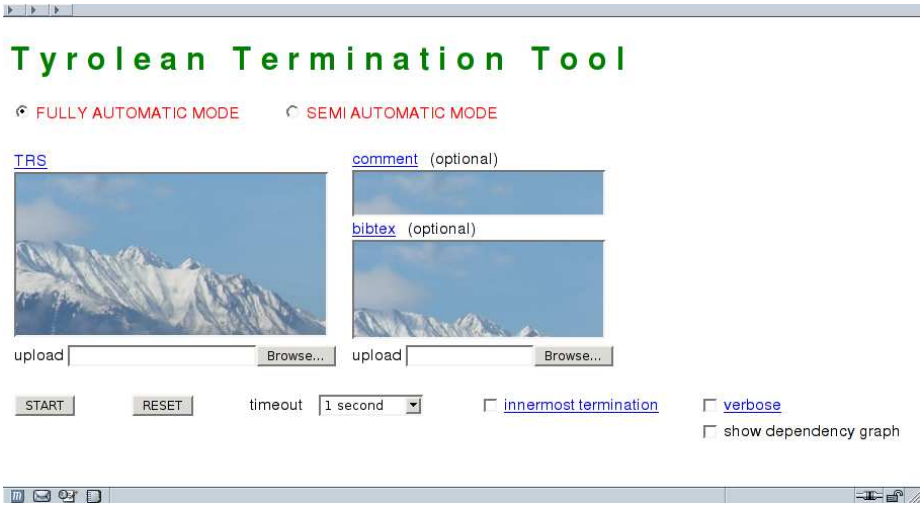

Figure 1: A screen shot of the fully automatic mode of TTT.

address this problem we develop new transformations from programs (or term rewrite systems) to term rewrite systems that are easier for existing termination methods. Especially, we try to design *complexity preserving* transformations to enable complexity analysis.

- **New Orderings.** Usually, after several transformations, one tries to find a *well-founded order* compatible with the resulting rewrite system, because the compatibility implies termination of the original rewrite system. Moreover, by careful analysis one may induce an upperbound of the complexity from the compatibility. That is why developing powerful well-founded orders is of particular interest in this research. Especially, we try to design orders so that their compatibilities induce *low* complexities.

For the research for termination verification I collaborate with Prof. Aart Middeldorp at University of Innsbruck, and collaborate with Dr. Georg Moser at University of Innsbruck for complexity analysis.

3 Progress in 2007

Our progress is summarized in the next two papers: the first was published in this year, and the second is submitted to a publication forum.

1. **Tyrolean Termination Tool: Techniques and Features** ([1]). The paper reports the Tyrolean Termination Tool (TTT) [2], which is a powerful tool for automatically proving termination of rewrite systems (Figure 1). It incorporates several new refinements of the dependency pair method (including the *subterm criterion* and *usable rules*) that are easy to implement, increase the power of the method, result in simpler termination proofs, and make the method more efficient. TTT employs *polynomial interpretations with negative coefficients*, like $x - 1$ for a unary function

Figure 2: Our techiniques that have been adopted in other termination tools.

	AProVE	CiME	Jambox	Muterm	TPA	TTT2
subterm criterion	✓	×	✓	✓	✓	✓
usable rules	✓	×	✓	✓	✓	✓
negative polynomials	✓	×	×	×	×	✓

symbol or $x-y$ for a binary function symbol, which are useful for extending the class of rewrite systems that can be proved terminating automatically.

These contributions have already influenced further research in the area of termination verification (e.g., Aoto and Yamada, RTA 2006; Giesl *et al*, JAR 2006; Lucas, IC 2006). Moreover, they has been adapted in many termination tools (Figure 1).

2. **Towards an Automatic Runtime Complexity Analysis of Scheme Programs by Rewriting**.[1] In the paper we study the runtime complexity of (a subset of) Scheme programs by a translation into rewrite systems. By designing the translation to be complexity preserving, the complexity of the initial Scheme program can be estimated by analyzing the complexity of the resulting rewrite system. In order to assess viability of our approach we have implemented a fast and powerful complexity analyzer.

4 Future Work

I am planing to investigate three techniques and to develop one system:

- **Orderings based on conditional polynomial interpretations.** Orderings induced by *polynomial* interpretations are very useful for termination verification. We refine the orderings by allowing conditional expressions in polynomial interpretations. We anticipate that the new orderings are also useful to show low runtime complexity.
- **Uncurrying.** We investigate a transformation from untyped applicative term rewrite systems to functional term rewrite systems that preserves termination, innermost termination, and complexity.
- **Dependency pairs for complexity analysis**. The dependency pair method is the most powerful method for proving termination. We investigate to adopt this method for complexity analysis.
- **Complexity analyzer.** We have been developing a new complexity analyzer to assess the viability of our techniques.

[1] A preliminary version is available at `http://cl-informatik.uibk.ac.at/~georg/publications/`.

5 Publications in 2007

5.1 Journal Paper

[1] Nao Hirokawa and Aart Middeldorp. *Tyrolean Termination Tool: Techniques and Features*. Information and Computation **205(4)**, pp. 474-511, 2007.

5.2 System Development

[2] T$_{\mathsf{T}}$T, a termination tool. Available at `http://colo6-c703.uibk.ac.at/ttt/`.