Extended Abstracts

Third International Workshop on Termination

May 21-23, 1997

Faculty of Mathematics & Computer Science

Department of Computer Science

Universiteit Utrecht

Third International Workshop on Termination

Ede, May 21-23, 1997

Abstracts

Program committee

Thomas Arts (Utrecht, The Netherlands)
Adam Cichon (Nancy, France)
Maria Ferreira (Lissabon, Portugal)
Pierre Lescanne (Nancy, France)
Ursula Martin (St.Andrews, Scotland)
Hans Zantema (Utrecht, The Netherlands)

Contents

Well-Quasi Orderings and Some Applications

Nachum Dershowitz

I will summarize the main results in wqo theory regarding finite and infinite trees and graphs. After reviewing past applications to termination arguments, I will speculate on possibilities for additional applications in the future.

A Refined Analysis for Termination of Combined Rewrite Systems

Bernhard Gramlich
INRIA Lorraine & CRIN, Nancy, France
e-mail: Bernhard.Gramlich@loria.fr
www: http://www.loria.fr/~gramlich

In this paper which is mainly based on [5] we investigate sufficient conditions for the preservation of the termination property under disjoint and more general combinations of term rewriting systems (TRSs). By means of a refined analysis of existing approaches we show how to prove several new asymmetric preservation results. The known positive results for modularity of termination are, roughly speaking, based on three different approaches concerning the essential ideas and proof structures (cf. [4]): **(I) a general approach** via an abstract structure theorem where the basic idea is to reduce non-termination in the union to non-termination of a slightly extended generic version of one of the systems ([2], [9]), **(II) a modular approach** via modularity of innermost termination where sufficient criteria for the equivalence of innermost termination and general termination are combined with the modularity of innermost termination ([8], [3]), and **(III) a syntactic approach** via left-linearity which in essence is based on commutation and uniqueness properties of collapsing reduction in left-linear systems ([12], [13], [11]). Virtually all proofs for showing preservation results for termination under disjoint unions (and more general combinations) rely on properties of minimal counterexamples of the following form: *If the union of two disjoint terminating systems (having some properties) is non-terminating, then a minimal counterexample in the union must enjoy certain properties and, consequently, the two systems must satisfy certain (additional) properties.* Since in general the role of both systems in minimal counterexamples need not be symmetric this may naturally entail corresponding (positive) symmetric and asymmetric preservation results. In the literature this observation has been systematically exploited for approach (I) above (starting with [7]), and partially also for (III). Here we shall show how to do this for (II) and how to considerably refine the existing analysis for (III). In (most of the) obtained new results (on the preservation of termination) one of the systems must be non-collapsing but not necessarily the other one. The refinement for (II) is based on particular properties of overlay systems, whereas for (III) a deeper analysis of collapsing reductions is necessary. For that purpose we introduce two interesting new properties of term rewriting systems, being *uniquely collapsing*

and *collapsing confluent*. These properties somehow capture essential structural aspects of rewriting in (disjoint and more general types of) unions of TRSs, and are used to obtain new preservation results by *abstraction techniques*. Furthermore we discuss these properties w.r.t. well-known consistency and normal form properties, and show that they are modular for left-linear systems, but not in general. Finally we discuss to what extent the results presented can be generalized to non-disjoint unions, e.g. of constructor-sharing, composable ([8], [10]) or hierarchical ([1], [6]) TRSs.

References

[1] N. Dershowitz. Hierarchical termination. In N. Dershowitz and N. Lindenstrauss, editors, *Proc. 4th Int. Workshop on Conditional and Typed Rewriting Systems, Jerusalem, Israel (1994)*, volume 968 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, 1995.

[2] B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 5:131–158, 1994.

[3] B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.

[4] B. Gramlich. *Termination and Confluence Properties of Structured Rewrite Systems*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Jan. 1996.

[5] B. Gramlich. Asymmetric preservation criteria for termination of combined rewrite systems. INRIA-Lorraine, France, Jan. 1997.

[6] M. Krishna Rao. Modular proofs for completeness of hierarchical term rewriting systems. *Theoretical Computer Science*, 151(2):487–512, Nov. 1995.

[7] A. Middeldorp. *Modular Properties of Term Rewriting Systems*. PhD thesis, Free University, Amsterdam, 1990.

[8] A. Middeldorp and Y. Toyama. Completeness of combinations of constructor systems. *Journal of Symbolic Computation*, 15:331–348, Sept. 1993.

[9] E. Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136:333–360, 1994.

[10] E. Ohlebusch. Modular properties of composable term rewriting systems. *Journal of Symbolic Computation*, 20(1):1–42, July 1995.

[11] M. Schmidt-Schauß, M. Marchiori, and S. Panitz. Modular termination of r-consistent and left-linear term rewriting systems. *Theoretical Computer Science*, 149(2):361–374, Oct. 1995.

[12] Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.

[13] Y. Toyama, J. Klop, and H. Barendregt. Termination for direct sums of left-linear complete term rewriting systems. *Journal of the ACM*, 42(6):1275–1304, Nov. 1995.

Recursively defined (quasi) orders on terms

M. C. F. Ferreira
Departamento de Informática, Faculdade de Ciências e Tecnologia,
Univ. Nova de Lisboa - Quinta da Torre - 2825 Monte da Caparica, Portugal,
tel: +351-1-294 85 36, fax: +351-1- 294 85 41, e-mail: cf@di.fct.unl.pt

Abstract

We consider the subject of recursively defining orders and quasi-orders on a set of terms. When giving such a recursive definition, we are confronted with several issues; one of them is the well-definedness of the (quasi) order or equivalently the existence of an object with the desired properties implied by the definition. Other aspects concern the properties that make a relation a partial or quasi-order, namely irreflexivity and transitivity and reflexivity and transitivity, respectively. In general irreflexivity or reflexivity (respectively) are not too difficult to establish, but verification of transitivity is usually not a trivial task. Another related topic is well-foundedness, a property which becomes particularly relevant if one is interested in using orders in termination proofs.

We discuss these issues and propose a general scheme (based in the approach of Kamin and Lèvy [1]) such that any recursive relation on terms defined according to our scheme will result in a well-defined partial or quasi-order on terms. Imposing some restrictions on the construction, we can also ensure that the resulting order will be well-founded.

We believe our approach has several advantages. Proving well-definedness, (ir)reflexivity and transitivity has to be done only once, namely for the scheme presented; then for any particular recursively defined relation on terms that we want to establish as a partial or quasi-order, we only need to check that the relation satisfies the properties required by the scheme, and this check is in general substantially simpler than establishing well-definedness, (ir)reflexivity and transitivity independently. The abstraction provided by the scheme allows for a better understanding of the mechanisms behind the definition of recursive path-like orders and the properties thay may be desirable for such orders (like subterm compatibility and closedness under contexts). Finally the scheme seems to cover most existing path orders while still allowing for the definition of new ones.

References

[1] KAMIN, S., AND LÉVY, J. J. Two generalizations of the recursive path ordering. University of Illinois, 1980.

Higher-Order Simplification Orderings*

Jean-Pierre Jouannaud† and **Albert Rubio**‡

† LRI, CNRS and Université de Paris-Sud
Bât 405, 91405 Orsay Cedex, France
Email: Jean-Pierre.Jouannaudlri.fr
http: //www.lri.fr/people/jouannau.html
FAX: + 33 01 69 15 65 86
Tel: + 33 01 69 15 69 05

‡ Technical University of Catalonia
Pau Gargallo 5, 08028 Barcelona, SPAIN
Email: Email: rubio@lsi.upc.es
http: //www.lsi.upc.es/~rubio.html

1 Introduction

Computer systems like *Saturate* [7] or CiME [1] make available semi-automated techniques for proving termination of first-order rewrite rules by comparing their left and right hand sides in some reduction ordering, of which the most popular one is the recursive path ordering [3]. Its principle is to generate recursively an ordering on terms from a user-defined ordering on function symbols called a precedence.

The recent development of proof assistants and logical frameworks using higher-order rewrite rules has created the need of powerful tools to reason about them, in particular, proving their local confluence and their termination. But higher-order rewrite rules are used with two different meanings. Some functional languages like ML or type theories like the calculus of inductive constructions use higher-order rewrite rules to define functions or recursors by first-order pattern matching. In this setting termination is known to be satisfied when all rules follow a generalized form of a primitive recursive schema of higher type. In functional languages like Elf, or theorem provers like Isabelle, higher-order rewrite rules define functions by higher-order pattern matching. Higher-order rewriting in this sense enjoys a theory which parallels the usual theory of first-order rewriting. In particular, the main property of first-order rewriting, the critical pair lemma, still holds for some restricted cases [6]. On the negative side, this framework lacks adequate termination proof methods.

The latter problem has been addressed recently from two different perspective: (i) by defining classes of interpretations operating on well-founded sets; (ii) by defining reduction orderings directly on higher-order terms. Originating from Gandy, the first approach has been studied thoroughly by Jaco Van de Pol [2]. The main drawback is that it is not quite suitable for automation: the user has to provide with well-founded sets in which the types and the function symbols can be interpreted, show that the orderings are monotonic and that the rules decrease in the orderings, and both may require induction. The second approach was initiated by Loria-Saenz and Steinbach [5], who tried to come up with a generalization of the recursive path ordering, restricted to so-called "pattern terms". A more recent attempt is due to the authors [4], in which typed first-order interpretations of arbitrary higher-order terms are compared in a typed recursive path ordering generated from a precedence and an ordering on types assumed to be *compatible* with the term structure, that is, types do not increase by taking subterms, a strong restriction indeed.

The goal of this paper is to provide new, powerful tools for proving termination of higher-order rewrite rules operating on simply typed λ-terms in η-long β-normal form. First, we

*This work was partly supported by the ESPRIT Basic Research Action CCL and the EU Human Capital and Mobility Network *Console*.

introduce a new notion of *higher-order subterm*, in which strict subterms are of basic type, and prove that its monotonic extension, *higher-order embedding*, is a well order, thus generalizing Kruskal's theorem to this setting. Compatibility of higher-order subterm becomes a property of basic types, hence is easy to ensure. We then define a true generalization of the recursive path ordering to higher-order terms, the *typed recursive path ordering*, and prove that it is a simplification ordering on ground terms. Together with closure under instantiation, this ensures well-foundedness for arbitrary terms, and an appropriate notion of monotonicity allows us to use it for proving termination of higher-order rewrite systems. This ordering has the same conceptual elegance as Dershowitz's recursive path ordering: it recursively extends to higher-order terms two well-orders, a well-order on types and the precedence on function symbols. It therefore improves over the author's previous attempt to mimic the first-order situation [4] in two different ways: by relaxing the compatibility condition considerably; by being conceptually simpler, since higher-order terms need not be interpreted by first-order ones. Several examples are carried out which show its power as well as its limitations.

2 Conclusion

We have generalized the theory of simplification orderings to the higher-order case, under the assumption of a simple type discipline. The obtained ordering which generalizes the recursive path ordering has the same conceptual elegance. Although this ordering allows to deal with all exemples given in [4], as well as new ones, it fails orienting rules like *apply* or Gödel's recursors. This is not so surprising, since the termination of the typed λ-calculus or Gödel's T is known to be intrinsically difficult. The question now is to investigate to which extent our higher-order typed recursive path ordering is compatible with these calculi.

References

[1] Evelyne Contejean and Claude Marché. CiME: Completion Modulo E. pages 416–419, 1996. System Description available at `http://www.lri.fr/~demons/cime.html`.

[2] Jaco Van de Pol. *Termination of Higher-Order Rewrite Systems.* Phd thesis, Utrecht University, Utrecht, Netherlands, 1996.

[3] Nachum Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, March 1982.

[4] Jean-Pierre Jouannaud and Albert Rubio. A recursive path ordering for higher-order terms in η-long β-normal form. In Ganzinger H., editor, *Proc. 7th Rewriting Techniques and Applications, New-Jersey, LNCS 1103.* Springer-Verlag, 1996.

[5] C. Loría-Sáenz and J. Steinbach. Termination of combined (rewrite and λ-calculus) systems. In *Proc. 3rd Int. Workshop on Conditional Term Rewriting Systems, Pont-à-Mousson, LNCS 656*, volume 656 of *Lecture Notes in Computer Science*, pages 143–147. Springer-Verlag, 1992.

[6] T. Nipkow. Higher order critical pairs. In *Proc. IEEE Symp. on Logic in Comp. Science*, Amsterdam, 1991.

[7] Pilar Nivela and Robert Nieuwenhuis. Practical results on the saturation of full first-order clauses: Experiments with the saturate system. (system description). In C. Kirchner, editor, *5th International Conference on Rewriting Techniques and Applications*, LNCS 690, Montreal, Canada, June 16–18, 1993. Springer-Verlag.

A total AC-compatible ordering with RPO scheme*

Albert Rubio

Technical University of Catalonia
Pau Gargallo 5, 08028 Barcelona, Spain.
E-mail: rubio@lsi.upc.es

Rewrite-based methods with built-in associativity and commutativity (AC) properties for some of the operators are well-known to be crucial in theorem proving and programming. Therefore a lot of work has been done on the development of suitable *AC-compatible* reduction or simplification orderings, like [DHJP83, BP85, GL86, BCL87, KSZ90, NR91, Bac92, DP93, RN95]. Most of these orderings were attempts to adapt to the AC case the *recursive path ordering* of Dershowitz [Der82], which is simple and easy to automatize and use. In order to obtain AC-compatibility, one normally *flattens* terms wrt. AC symbols before comparing them with RPO. But then monotonicity is easily lost: for instance if f is an AC symbol greater than some other symbol g then $f(a,a) \succ_{rpo} g(a)$ but $f(a,g(a)) \succ_{rpo} f(a,a,a)$.

An essential additional property of the ordering that is needed in order to preserve the completeness of most rewrite-based theorem proving techniques for first-order clauses (modulo AC) is its totality on (AC-different) ground terms, what we will call *AC-totality*.

AC-RPO ([RN95]) was the first RPO-based AC-total and compatible reduction ordering without restrictions on the number of AC-symbols or on the precedence over de signature. Unfortunately, although being defined in terms of RPO, it does not behave like RPO; e.g. it cannot orient the distributivity rule in the "right" (i.e. distributing) way. The reason for this fact seems to be that a transformation on the terms is applied before using RPO (this approach, with different transformations, is also used in [BP85] among others). Therefore, a better approach could be to directly apply an RPO-like scheme, treating as the only special case the *AC-equal-top case*, i.e. when the head symbols of the two compared terms are the same AC symbol. Among all known orderings, only the one in [KSZ90] is such a real *RPO scheme* ordering but it is rather compicated and not AC-total.

Very recently, in [KS97] another such an RPO scheme ordering has been defined, which is moreover AC-total. On the negative side, its definition for the AC-equal-top case is very involved, which implies on one hand that the proofs are difficult and long (mainly the monotonicity proof) and, on the other hand, that the ordering cannot be easily lifted to terms with variables.

The contribution given here is a new RPO scheme AC-ordering with a much simpler AC-equal-top case. This allows one to get simpler (as well as much

* This work is partially supported by the Esprit basic research working group CCL-II and the HCM Network CONSOLE.

shorter) proofs and a more powerful ordering for terms with variables, making it more suitable in practice.

References

[Bac92] Leo Bachmair. Associative-commutative reduction orderings. *Information Processing Letters*, 43:21–27, 1992.

[BCL87] Ahlem Ben-Cherifa and Pierre Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9:137–160, 1987.

[BP85] L. Bachmair and D. A. Plaisted. Termination orderings for associative-commutative rewriting systems. *Journal of Symbolic Computation*, 1:329–349, 1985.

[Der82] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.

[DHJP83] N. Dershowitz, J. Hsiang, A. Josephson, and D. Plaisted. Associative-commutative rewriting. In *Int. Joint Conference on Artificial Inteligence*, pages 940–944, 1983.

[DP93] Catherine Delor and Laurence Puel. Extension of the associative path ordering to a chain of associative commutative symbols. In C. Kirchner, editor, *5th International Conference on Rewriting Techniques and Applications*, LNCS 690, pages 389–404, Montreal, Canada, June 16–18, 1993. Springer-Verlag.

[GL86] Isabelle Gnaedig and Pierre Lescanne. Proving termination of associative-commutative rewrite systems by rewriting. In Jörg H. Siekmann, editor, *Proc. 8th International Conference on Automated Deduction*, LNCS 230, pages 52–61, Oxford, England, July 27–August 1, 1986. Springer-Verlag.

[KS97] D. Kapur and G. Sivakumar. A total, ground path ordering for proving termination of ac-rewrite systems. In H. Comon, editor, *8th International Conference on Rewriting Techniques and Applications*, LNCS, 1997.

[KSZ90] D. Kapur, G. Sivakumar, and H. Zhang. A new method for proving termination of ac-rewrite systems. In *Conf. Found. of Software Technology and Theoretical Computer Science*, LNCS 472, pages 134–148, New Delhi, India, December 1990. Springer-Verlag.

[NR91] Paliath Narendran and Michael Rusinowitch. Any ground associative commutative theory has a finite canonical system. In *Fourth int. conf. on Rewriting Techniques and Applications*, LNCS 488, pages 423–434, Como, Italy, April 1991. Springer-Verlag.

[RN95] Albert Rubio and Robert Nieuwenhuis. A total AC-compatible ordering based on RPO. *Theoretical Computer Science*, 142(2):209–227, May 15, 1995.

Proving termination by counting things

Ursula Martin

A wide variety of termination proofs depend on counting occurrences of patterns, but proofs in this area often seem intricate and unrelated to the structure of the computation we are trying to prove terminating.

In this lecture, based on joint work with Duncan Shand, we outline some new work which attempts to produce a unified framework for such discussions, by extending to trees the combinatorial ideas based on Lie Algebras which we have previously studied for strings.

References

[1] Martin, Ursula On the diversity of orderings on strings. Fund. Inform. 24 (1995), no. 1-2, 25–46.

[2] Martin, Ursula; Scott, Elizabeth The order types of termination orderings on monadic terms, strings and multisets. Eighth Annual IEEE Symposium on Logic in Computer Science (Montreal, PQ, 1993), 356–363, IEEE Comput. Soc. Press, Los Alamitos, CA, 1993.

[3] Martin, Ursula Linear interpretations by counting patterns. Rewriting techniques and applications (Montreal, PQ, 1993), 421–433, Lecture Notes in Comput. Sci., 690, Springer, Berlin, 1993.

[4] Martin, Ursula; Shand, Duncan Ordering trees by counting patterns, in preparation

On Total Simplification Orderings on Strings

Jörg Frank, Dieter Hofbauer
Technische Universität Berlin*
{jofrank,dieter}@cs.tu-berlin.de

Continuing work by de Jongh and Parikh, Martin and Scott, and others, we present results towards a classification of all total simplification orderings on strings over a finite alphabet. A complete classification should provide answers to at least the following questions:

- Which ordinals can occur as order types of such orderings? (Is $\omega^{\omega^3+\omega}$ possible, for instance?)
- Which orderings are actually needed to prove termination of finite string rewrite systems? If most orderings are avoidable (the number of orderings far exceeds the number of rewrite systems), are there avoidable order types?
- What about the connection between order types and upper bounds on derivation lengths for finite string rewrite systems? Is it possible to leave primitive recursion? (It is for the case of Post systems.)

*Fachbereich Informatik, TU Berlin, FR 6-2, Franklinstraße 28/29, D-10587 Berlin

Ordinals extracted from termination proofs of functions

François Monin *
Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
e-mail: monin@win.tue.nl

An automated procedure for proving termination of recursively defined functions has been recently proposed by P. Manoury and M. Simonot. This approach has given rise to a system based on proofs as program paradigm. In this way, they build formal intuitionistic proofs in order to extract lambda-terms as programs. It is then important to know whether there is a well-founded ordering corresponding to this method, especially if it has not yet been used in other procedures for termination.

We show that an ordinal measure can actually be extracted from each proof. This measure is characterized by the fact it satisfies a decreasing property: It decreases in recursive calls of the corresponding function. We show with these measures that, in particular on natural numbers, one can prove usual primitive recursion, unnested multiple recursion, general recursion and that it is limited neither by the number of recursions nor by their order. A feature of these extracted ordinals is that it produces a method released from some constraints due to formal proofs. Moreover, the ordinal measures can be given as a "hint" for the *NQTHM* theorem prover. Indeed this one accepts well founded orderings leading it in its research of the proof of termination. In our context, assuming a defined function of the first sytem translated into the second one. Then, thanks to the extracted ordinal, *NQTHM* is able to prove the termination of the function only using the rewrite rules. This is a joint work with M. Simonot, CNAM, Paris.

*This work was done as part of a research project at the University of Savoie, France.

A constructive proof of strong normalization for Gödel's T via a monotone interpretation

Andreas Weiermann
Institut für Mathematische Logik und Grundlagenforschung
der Westfälischen Wilhelms-Universität Münster
Einsteinstr. 62, D-48149 Münster, Germany

Abstract

Let T be Gödel's system T of primitive recursive functionals of finite type. We are going to define a monotone interpretation I for T, i.e. a function I from T into the set of natural numbers so that $t \to t'$ implies $I(t) > I(t')$ for $t, t' \in T$. The definition of I is based on a finestructure analysis of Howard's 1970 proof of weak normalization proof for T. The reference is: W. A. Howard: Assignment of ordinals to terms for primitive recursive functionals of finite type. Intuitionism and Proof Theory. North-Holland, Amsterdam 1970, 443-458.

For a given natural number n let T_n be the subsystem of T in which the levels of the recursors are bounded by $n + 2$. As a corollary of the interpretation theorem we get optimal derivation lengths classifications for T and its subsystems. For $n = 0$ we obtain that the T_0 derivation lengths are Kalmar elementary recursive in the Ackermann function. We answer a question of S. Feferman by showing that the derivation lengths function for any type one functional $f \in T_0$ is bounded by a primitive recursive function. Thus, as already known by a result of Parsons, f defines at most a primitive recursive function. For $n = 1$ we obtain a classification of T_1 in terms of multiple recursion. The interpretation theorem also yields a derivation lengths classification of Leivant's system of predicative recursion in higher types in terms of Kalmar elementary recursion.

STRONG NORMALISATION IN SECOND ORDER LAMBDA CALCULUS WITH EXPLICIT SUBSTITUTIONS

Pierre LESCANNE
Centre de Recherche en Informatique de Nancy (CNRS)
INRIA-Lorraine
Campus Scientifique, BP 239,
F54506 Vandœuvre-lès-Nancy, France

email: Pierre.Lescanne@loria.fr

Explicit substitution is a convenient tool for revealing the true behaviour of lambda calculus in term of computations. By the fact that substitution is considered an internal operation and fully integrated in the calculus, interesting properties could be investigated concerning the complexity of computations and some features of actual implementations. Indeed in this framework, substitution is not anymore an atomic one-step operation with some magic properties and an unknown computational cost.

It is known that some calculi of explicit substitution preserve strong normalisation of lambda-calculus. We say that such calculi of explicit substitution are PSN. That means that terms which are strongly beta normalising are still strongly normalising in the calculus of explicit substitution despite this calculus has more rewrites. Therefore by PSN typed terms, e.g., simply typed terms or terms typed in a second order lambda calculus like system F are strongly normalising in the lambda calculus enriched with explicit substitutions. A direct proof of this fact which does not rely on PSN is interesting in that it would give a better insight on why some calculi of explicit substitution preserve strong normalisation and others do not. For this, an adequate type system should be provided for the calculus of explicit substitution from which a proof of strong normalisation can be extracted.

In this lecture, we will present an ongoing work on a proof of strong normalisation for a system called Fv, which is a system F extended with explicit substitutions. We will present the problem, describe a type system for Fu and give a proof of strong normalisation which can be seen as an extension of the classical proof based on candidates of reducibility.

Strong Normalization of Explicit Substitutions via Cut Elimination in Proof Nets

Roberto Di Cosmo* Delia Kesner §

This work is about explicit substitutions and proof nets, two well established formalisms that have been used to gain a better understanding of the λ-calculus over the past decade. On one side, explicit substitutions provide an intermediate formalism that - by decomposing the β rule into more atomic steps - allows a better understanding of the execution models. On the other side, Linear Logic decomposes the intuitionistic logical connectives, like the arrow, into more atomic, resource-aware connectives, like the linear arrow and the explicit erasure and duplication operators given by the exponentials: this decomposition is reflected in Proof Nets, which are the natural deduction of Linear Logic, and provide a more refined computational model that the one given by the λ-calculus, which is the natural deduction of Intuitionistic Logic.

In this paper, we define a typed version of the calculus with explicit substitutions λx proposed in [Rose92, Bloo95], we show how to translate it into Proof Nets and how to establish, using this translation, a simulation of the reduction rules for explicit substitutions via cut elimination in Proof Nets. As an immediate consequence of this simulation, we prove that a simply typed version of λx is strongly normalizing, as well as all the typed calculi isomorphic to it such as λ_v [Les94a], λ_s [KR95b], λ_d [Kes96], and λ_f [FKP96]. The proof technique developed in the paper to simulate explicit substitution via proof nets gives a clear interpolation between typed λ-calculus and linear logic since calculi with explicit substitutions are already known to simulate β-reduction. This same remark holds for polymorphic λ-calculus as the technique in this paper can be easily extended to a polymorphic version of λx.

Another important fallout is an extended notion of reduction on Proof Nets, which is needed in the simulation, and is of interest on its own because it is still confluent and strongly normalizing, but it identifies more proofs which differ by uninteresting details than the original notion of reduction. Last, but not least, by providing a finer analysis of reduction in explicit substitutions, the translation in Proof Nets provides a tool of choice to study the problems still open in the field of explicit substitutions, like the quest for a well-behaved notion of composition and the like.

References

[Bloo95] Roel Bloo. Preservation of Strong Normalization for Explicit Substitution. Technical Report 95-08, Eindhoven University of Technology. 1996.

[FKP96] María C. Ferreira, Delia Kesner and Laurence Puel. λ-calculi with explicit substitutions and composition which preserve β-strong normalization (extended abstract). *ALP*, LNCS 1139. 1996.

[KR95b] Fairouz Kamareddine and Alejandro Ríos. A λ-calculus à la de Bruijn with explicit substitutions. *PLILP*, LNCS 982. 1995.

[Kes96] Delia Kesner. Confluence properties of extensional and non-extensional λ-calculi with explicit substitutions. *RTA*, LNCS 1103. 1996.

*DMI-LIENS (CNRS URA 1347) Ecole Normale Supérieure - 45, Rue d'Ulm - 75230 Paris France. Email:dicosmo@ens.fr
§LRI (CNRS URA 410) - Bât 490, Université de Paris-Sud - 91405 Orsay Cedex, France. Email:kesner@lri.fr

[Les94a] Pierre Lescanne. From λ_σ to λv, a journey through calculi of explicit substitutions. In *POPL* 1994.

[Rose92] Kristoffer Rose. Explicit cyclic substitutions. *CTRS*, LNCS. 1992.

Deciding Termination in CL(S) (Abstract)

Johannes Waldmann

Institut für Informatik
Friedrich-Schiller-Universität
D-07740 Jena

joe@informatik.uni-jena.de
http://www5.informatik.uni-jena.de/~joe/

February 12, 1997

We are considering CL(**S**): Combinatory Logic with the sole combinator $\mathbf{S}xyz \to xz(yz)$. Of course CL(**S**) is confluent, and terms that have a normal form are strongly normalizing.

However, CL(**S**) contains terms without normal form, for instance **S(SS)SSS** (found by Dobue), **SSS(SSS)(SSS)** (by Barendregt).

We show a procedure that decides whether a given $X \in$ CL(**S**) has a normal form. In order to achieve this, we investigate certain reduction properties. It turns out that the sets having these properties are regular tree languages.

Termination and Innermost Normalisation of Term Rewriting Systems

Thomas Arts[1] and Jürgen Giesl[2]

[1] Dept. of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands, E-mail: thomas@cs.ruu.nl

[2] FB Informatik, TH Darmstadt, Alexanderstr. 10, 64283 Darmstadt, Germany, E-mail: giesl@inferenzsysteme.informatik.th-darmstadt.de

We present sound and complete criteria for termination and innermost normalisation of term rewriting systems (TRSs). Based on these criteria, two techniques for automatically proving termination and innermost normalisation respectively are developed. The techniques transform a TRS into a set of inequalities. Subsequently, standard methods like the recursive path ordering and polynomial interpretations are applied to find a well-founded quasi-ordering satisfying the inequalities. Although our techniques are based on these existing methods, our techniques can be applied to prove termination of TRSs that are not simply terminating, which cannot be done by a direct application of the recursive path ordering or polynomial interpretations. On the other hand, any TRS that can automatically be proved terminating by these existing methods, can automatically be proved terminating by our technique.

The technique for automatically proving innermost normalisation can be applied to prove innermost normalisation of TRSs that are not terminating. As far as we know, this is the first automatic technique with that property. Moreover, this technique can also be used to prove termination for TRSs of a special form. For example, non-overlapping TRSs and locally confluent overlay systems can be proved terminating by proving innermost normalisation. By this approach it is possible to automatically prove termination of TRS for which other automatic techniques fail.

The motivation for introducing the criteria stems from regarding the operational semantics of term rewriting systems, i.e. TRSs are considered to be 'programs'. Intuitively, such a program is terminating if any recursive call is terminating. As an example to clarify this intuition, consider the following TRS, which can be used for division of natural numbers

$$\begin{array}{ll} \mathsf{minus}(x, \mathsf{0}) & \to x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to \mathsf{minus}(x, y) \\ \mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \to \mathsf{0} \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))). \end{array}$$

Intuitively it is clear that termination of an instantiation of $\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y))$ depends on the termination of the instantiation of $\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y)))$ in the right-hand side of the last rewrite rule. The function symbol s around this subterm of the right-hand side has no influence on the termination behaviour. This way of looking at termination of TRSs motivates that only subterms of the right-hand sides of rewrite rules that have a defined symbol as root symbol are considered for the examination of the termination behaviour. The defined symbols are the root symbols of the left-hand sides, whereas the other symbols in the signature are constructors.

Infinite reductions originate from the fact that defined symbols are introduced by the right-hand side of rewrite rules. By tracing the introduction of these defined symbols information is obtained about the termination as well as the innermost normalisation of the term rewriting system.

If a term $f(s_1, \ldots, s_n)$ rewrites to another term $C[g(t_1, \ldots, t_m)]$ (where C is a context and g is a defined symbol), then to prove termination the argument tuples $s_1, \ldots, s_n$ and $t_1, \ldots, t_m$ are compared. For a formal definition and to avoid the

handling of tuples, a special symbol F, not occurring in the signature of the TRS, is introduced for every defined symbol f in D. Instead of comparing tuples, the *terms* $F(s_1, \ldots, s_n)$ and $G(t_1, \ldots, t_m)$ are now compared.

Definition 1. If $f(s_1, \ldots, s_n) \to C[g(t_1, \ldots, t_m)]$ is a rewrite rule of R with g a defined symbol, then $\langle F(s_1, \ldots, s_n), G(t_1, \ldots, t_m)\rangle$ is called a *dependency pair* of $\mathcal{R}$.

The dependency pairs in the above example are

$$\langle \mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{MINUS}(x, y)\rangle$$
$$\langle \mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{MINUS}(x, y)\rangle$$
$$\langle \mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y))\rangle.$$

Dependency pairs are the basis for both the termination and the innermost normalisation criterion. By regarding a sequence of these dependency pairs, the introduction of defined symbols can be traced.

Definition 2. An *$\mathcal{R}$-chain* is a sequence of dependency pairs such that there exists a substitution σ with $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ for every two consecutive pairs $\langle s_i, t_i\rangle$ and $\langle s_{i+1}, t_{i+1}\rangle$ in the sequence.

We always assume that two occurrences of dependency pairs have disjoint variables. Then, for example,

$$\langle \mathsf{MINUS}(\mathsf{s}(x_1), \mathsf{s}(y_1)), \mathsf{MINUS}(x_1, y_1)\rangle\langle \mathsf{MINUS}(\mathsf{s}(x_2), \mathsf{s}(y_2)), \mathsf{MINUS}(x_2, y_2)\rangle$$

is an $\mathcal{R}$-chain, because $\mathsf{MINUS}(x_1, y_1)\sigma \to_{\mathcal{R}}^* \mathsf{MINUS}(\mathsf{s}(x_2), \mathsf{s}(y_2))\sigma$ for the substitution σ that replaces x_1 by $\mathsf{s}(x_2)$ and y_1 by $\mathsf{s}(y_2)$.

Theorem 3. *A TRS $\mathcal{R}$ is terminating if and only if no infinite $\mathcal{R}$-chain exists.*

For the innermost normalisation criterion only restricted sequences of dependency pairs are considered. Instead of an arbitrary reduction in between two dependency pairs, the reduction should be an innermost reduction and additionally only a specific set of substitutions is allowed.

Definition 4. An *innermost $\mathcal{R}$-chain* is a sequence of dependency pairs such that a substitution σ exists with for every two consecutive pairs $\langle s_i, t_i\rangle$ and $\langle s_{i+1}, t_{i+1}\rangle$ in the sequence $t_i\sigma \stackrel{\mathrm{i}}{\to}_{\mathcal{R}}^* s_{i+1}\sigma$ and both $s_i\sigma$ and $s_{i+1}\sigma$ are normal forms.

Theorem 5. *A TRS $\mathcal{R}$ is innermost normalising if and only if no infinite innermost $\mathcal{R}$-chain exists.*

For the automation of both criteria we introduce (innermost) dependency graphs. The nodes in an (innermost) dependency graph are the dependency pairs and an arc between dependency pairs $\langle s_1, t_1\rangle$ and $\langle s_2, t_2\rangle$ exists if these two dependency pairs form an (innermost) chain. Cycles in these graphs correspond to infinite chains. We developed techniques for finding the dependency pairs that occur on such a cycle. These dependency pairs together with certain rules of the TRS are transformed into a set of inequalities. Standard techniques are used to find a suitable quasi-ordering satisfying these inequalities. If such an ordering is found, then termination or innermost normalisation of the TRS is proved.

References

[AG97a] T. Arts and J. Giesl. Automatically proving termination where simplification orderings fail. *Proceedings of CAAP'97*, LNCS, Lille, France, April 1997.

[AG97b] T. Arts and J. Giesl. Proving Innermost Normalisation Automatically. *Proceedings of RTA-97*, LNCS, Sitges, Spain, June 1997.

Transformational techniques for proving termination of term rewriting

Hans Zantema
Dept. of Computer Science, Universiteit Utrecht,
E-mail: hansz@cs.ruu.nl

Basic techniques for proving termination of term rewriting include recursive path order and interpretations like polynomials. Much work has been done in extending these techniques in such a way that termination of a wider class of term rewite systems can be proved. Two main approaches can be distinguished: generalizing the orders involved and applying transformations. In this talk we try to give an overview of the latter approach. The main idea is as follows. Let Ψ be a transformation transforming some rewrite system R into a rewrite system $\Psi(R)$ which is *non-termination preserving*, i.e., termination of R follows from termination of $\Psi(R)$. Then termination of R has been proved as soon as we can apply one of the basic techniques to prove termination of $\Psi(R)$, for a suitable choice of Ψ. Hence by this approach the main step is not finding a suitable order, but finding a suitable non-termination preserving transformation. Examples of this approach include

- transformation ordering;
- dummy elimination;
- semantic labelling;
- dependency pairs.

Most of the talk will be an overview of earlier techniques; the only new result to be presented is the following. Let S be a terminating non-erasing recursive program scheme. Write $S(t)$ for the normal form of a term T with respect to S. Let R be any TRS and let

$$R_1 = \{l \to r \in R \mid S(l) = S(r)\},$$

$$R_2 = \{S(l) \to S(r) \mid l \to r \in R \wedge S(l) \neq S(r)\}.$$

If both R_1 and R_2 are terminating TRSs, then R is terminating.

Demand-driven and constraint-based automatic left-termination analysis for Logic Programs

Stefaan Decorte* Danny De Schreye*

Abstract

Current automatic termination analysis techniques can be split up into different components: inference of mode or type information, derivation of models, generation of well-founded orders and verification of the termination conditions themselves. Although providing high precision results, these techniques suffer from an efficiency point of view as several of these analyses are often performed through abstract interpretation. Moreover, because of performing the different analyses independently, viable information that should be shared through all phases is not taken advantage of.

Current research has led to a different strategy to termination analysis. We first introduce a new termination condition, which has both the advantage of being very natural in the sense that it considers only the recursive structure of the clauses and of being able to start the analysis from any specific set of calls. Where most approaches in practice initiate the analysis from one fixed (semi-linear) norm, levelmapping and model, we take a different leap. We introduce a symbolic version of our termination condition, by parameterising the concepts of norm, levelmapping and model. Next, we show how this condition can be easily translated into a system of constraints on the values of the introduced symbols only. To each solution of the constraint system, there corresponds one (different) termination proof. It is this use of, and more importantly, this solving of constraint sets as the final phase that enables the different components of a termination proof to communicate with one another and to direct the proof towards success (if there is). In a final part, we show that the generated constraint system is not hard to solve, as each constraint involves only the introduced symbols and is of a particularly simple form.

The proposed method is both efficient and precise. The use of constraint sets enables the propagation of information over all different phases while the need for multiple analyses is eliminated. We have been experimenting with the technique on a lot of examples and we have obtained positive results. At this moment, we strongly believe that our technique is applicable in almost any practical case where termination can be proved through semi-linear norms.

*Department of Computer Science, K.U.Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium. e-mail : {stefaan,dannyd}@cs.kuleuven.ac.be

What Norms are Useful for Logic Programs?

N. Lindenstrauss, Y. Sagiv and N. Dershowitz

TermiLog is a powerful system for automatically checking termination of queries to logic programs. Its three main components are: instantiation analysis, inference of constraints, and generation of a finite set of "query-mapping" pairs. Currently, the system uses term size, list length, and other user-supplied linear functions to determine if every possible cycle of calls is decreasing. We'll explore the power and limitations of these norms.

On Normalizing, Non-terminating One-rule String Rewriting Systems

Alfons Geser, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, D-72076 Tübingen, Phone: +49 7071 29-78961,
E-mail: geser@informatik.uni-tuebingen.de

This contribution is concerned with the set of one-rule string rewriting systems (SRSs) which are *normalizing* but *not terminating*, i.e. one-rule SRS where every string can be reduced into a normal form, but the SRS also admits some infinite reductions. It is obvious that the two-rule SRS $\{0 \to 1, 0 \to 0\}$ has these properties. Continually ignoring the second rule gives a normal form. For instance, the string 00 reduces to normal form 11 by $00 \to 01 \to 11$. On the other hand the string 0 starts the infinite reduction $0 \to 0 \to 0 \to \cdots$.

It is not quite obvious whether a similar effect is achievable by only one rule, $u \to v$. Church already noted that a left-linear, non-erasing, non-overlapping term rewriting system is normalizing if and only if it is terminating. As SRSs are always left-linear and non-erasing, we know that u must overlap with itself.

We will demonstrate that

Theorem 1. *The SRS R over the alphabet $\{0,1\}$, consisting of the single rule*

$$1010 \to 010110$$

is normalizing but not terminating. Moreover no normalizing, non-terminating one-rule SRS $u \to v$ where $|u| < 4$ or $|v| < 6$ exists, and R is the only normalizing, non-terminating SRS where $|v| = 6$.

It is by no means easy to find this system by trial and error, let alone its proof of minimality; as Kurth [1] mentions there are, modulo renaming of letters and reversal of strings, 96029 one-rule SRSs $u \to v$ with $|u| < |v| = 6$.

[1] KURTH, W. *Termination und Konfluenz von Semi-Thue-Systemen mit nur einer Regel.* Dissertation, Technische Universität Clausthal, Germany, 1990.

Computing the Domains of Partial Functions

Jürgen Brauburger

FB Informatik, TH Darmstadt, Alexanderstraße 10, 64283 Darmstadt, Germany
E-mail: brauburger@informatik.th-darmstadt.de

We present a method for automatically computing non-trivial subsets of the domains of partial functions specified by functional procedures. Assume the free data type nat whose objects are built with the constructors 0 and $\mathsf{s}(x)$ and whose equality is defined by the functional procedure

$$\begin{array}{lll} \textbf{function } \mathsf{eq}(x,y:\mathsf{nat}):\mathsf{bool} \Leftarrow & \textbf{if } x=0 \wedge y=0 & \textbf{then } \mathsf{true} \\ & \textbf{if } x=0 \wedge y=\mathsf{s}(w) & \textbf{then } \mathsf{false} \\ & \textbf{if } x=\mathsf{s}(v) \wedge y=0 & \textbf{then } \mathsf{false} \\ & \textbf{if } x=\mathsf{s}(v) \wedge y=\mathsf{s}(w) & \textbf{then } \mathsf{eq}(v,w), \end{array}$$

where $=$ denotes syntactical equality. We prove the termination of eq with polynomial orderings. A *polynomial ordering* $\prec$ is defined by associating each n-ary function symbol f (including the 'tuple symbol' $\langle..\rangle$) with an n-ary polynomial over the integers and each variable with a variable over the naturals [3]. The termination of eq is proved if we find a polynomial ordering satisfying the *termination hypothesis*

$$x=\mathsf{s}(v) \wedge y=\mathsf{s}(w) \;\rightarrow\; \langle v,w\rangle \prec \langle x,y\rangle \wedge c \preceq \langle x,y\rangle, \tag{1}$$

where c is a new nullary function symbol [2]. The first inequality in (1) ensures that the arguments of the recursive call $\mathsf{eq}(v,w)$ decrease wrt. $\prec$ and the second inequality guarantees that $\prec$ is well founded. (1) is *polynomially satisfied*, e.g., if $\mathsf{s}(x)$ is associated with the polynomial $x+1$, the tuple symbol $\langle x,y\rangle$ with x, and c with 0 thus denoting the implication $x=v+1 \wedge y=w+1 \;\rightarrow\; v <_{\mathbb{Z}} x \wedge 0 \leq_{\mathbb{Z}} x$ which is true for all naturals v,w,x,y. Since a polynomial ordering satisfying *ordering constraints* like (1) can be computed by machine, the termination of eq can be proved automatically [2].

Now consider the functional procedure $\mathsf{minus}(x,y)$ that computes the difference of x and y.

$$\begin{array}{lll} \textbf{function } \mathsf{minus}(x,y:\mathsf{nat}):\mathsf{nat} \Leftarrow & \textbf{if } \mathsf{eq}(x,y) & \textbf{then } 0 \\ & \textbf{if } \neg\mathsf{eq}(x,y) & \textbf{then } \mathsf{s}(\mathsf{minus}(x,\mathsf{s}(y))) \end{array}$$

The procedure $\mathsf{minus}(x,y)$ terminates iff x is greater than or equal to y. We characterize subsets of inputs where procedures like minus terminate by termination predicates [1]. Given a function $f : s_1 \times..\times s_n \rightarrow w$, a function $\theta_f : s_1 \times..\times s_n \rightarrow \mathsf{bool}$ is a *termination predicate* for f iff θ_f is total and $\theta_f(x_1,..,x_n) = \mathsf{true}$ implies that the evaluation of $f(x_1,..,x_n)$ halts. In [1] rules are presented for synthesizing a functional procedure for θ_f if a well-founded relation $\prec$ is *given*. For each recursive call in f, the generated procedure θ_f checks also recursively if the arguments decrease wrt. $\prec$. For example, given the procedure for minus and a well-founded relation $\prec$ on pairs of naturals, the following procedure is synthesized.

$$\begin{array}{lll} \textbf{function } \theta_{\mathsf{minus}}(x,y:\mathsf{nat}):\mathsf{bool} \Leftarrow & \textbf{if } \mathsf{eq}(x,y) & \textbf{then } \mathsf{true} \\ & \textbf{if } \neg\mathsf{eq}(x,y) \wedge \langle x,\mathsf{s}(y)\rangle \prec \langle x,y\rangle & \textbf{then } \theta_{\mathsf{minus}}(x,\mathsf{s}(y)) \\ & \textbf{if } \neg\mathsf{eq}(x,y) \wedge \langle x,\mathsf{s}(y)\rangle \not\prec \langle x,y\rangle & \textbf{then } \mathsf{false} \end{array}$$

We aim at the synthesis of polynomial orderings for termination predicates, too. For that purpose we again consider the termination hypotheses. For instance, the termination hypothesis of minus is

$$\neg\mathsf{eq}(x,y) \;\rightarrow\; \langle x,\mathsf{s}(y)\rangle \prec \langle x,y\rangle \wedge c \preceq \langle x,y\rangle. \tag{2}$$

Since minus does not terminate for each input, there does not exist a polynomial ordering $\prec$ satisfying (2) for *all* pairs of naturals. Hence our goal is to find those pairs for which the termination hypothesis is polynomially satisfiable. For that purpose we transform (2) into the ordering constraints (7)–(9) using induction theorem proving techniques; cf. [2]. The premise $\neg\mathsf{eq}(x,y)$ 'suggests' in (2) a case analysis according the cases of eq. Thus we obtain

$$\neg\mathsf{eq}(0,0) \;\rightarrow\; \langle 0,\mathsf{s}(0)\rangle \prec \langle 0,0\rangle \wedge c \preceq \langle 0,0\rangle, \tag{3}$$

$$\neg\mathsf{eq}(0,\mathsf{s}(w)) \;\rightarrow\; \langle 0,\mathsf{s}(\mathsf{s}(w))\rangle \prec \langle 0,\mathsf{s}(w)\rangle \wedge c \preceq \langle 0,\mathsf{s}(w)\rangle, \tag{4}$$

$$\neg\mathsf{eq}(\mathsf{s}(v),0) \;\rightarrow\; \langle \mathsf{s}(v),\mathsf{s}(0)\rangle \prec \langle \mathsf{s}(v),0\rangle \wedge c \preceq \langle \mathsf{s}(v),0\rangle, \tag{5}$$

$$\neg\mathsf{eq}(\mathsf{s}(v),\mathsf{s}(w)) \;\rightarrow\; \langle \mathsf{s}(v),\mathsf{s}(\mathsf{s}(w))\rangle \prec \langle \mathsf{s}(v),\mathsf{s}(w)\rangle \wedge c \preceq \langle \mathsf{s}(v),\mathsf{s}(w)\rangle. \tag{6}$$

The implication (3) can be omitted, since $\neg\mathsf{eq}(0,0)$ evaluates to false. In (4) and in (5), the premises can be eliminated, since they evaluate to true. Thus we get instead of (3)–(5) the ordering constraints

$$\langle 0, \mathsf{s}(\mathsf{s}(w))\rangle \prec \langle 0, \mathsf{s}(w)\rangle \wedge c \preceq \langle 0, \mathsf{s}(w)\rangle, \tag{7}$$

$$\langle \mathsf{s}(v), \mathsf{s}(0)\rangle \prec (\langle \mathsf{s}(v), 0\rangle \wedge c \preceq \langle \mathsf{s}(v), 0\rangle. \tag{8}$$

In (6) we use that we have already proved that eq terminates or that $\langle v, w\rangle$ is the predecessor of $\langle \mathsf{s}(v), \mathsf{s}(w)\rangle$ wrt. a well-founded relation. Hence we may use as *induction hypothesis* that (2) holds for $\langle v, w\rangle$, i.e. $\neg\mathsf{eq}(v,w) \rightarrow (v, \mathsf{s}(w)) \prec (v,w) \wedge c \preceq (v,w)$. Since the premise of (6) can be simplified to the premise of the induction hypotheses, we get instead of (6) the ordering constraint

$$[\langle v, \mathsf{s}(w)\rangle \prec \langle v, w\rangle \wedge c \preceq \langle v, w\rangle] \rightarrow [\langle \mathsf{s}(v), \mathsf{s}(\mathsf{s}(w))\rangle \prec \langle \mathsf{s}(v), \mathsf{s}(w)\rangle \wedge c \preceq \langle \mathsf{s}(v), \mathsf{s}(w)\rangle]. \tag{9}$$

The ordering constraints (7), (8), and (9) are not polynomially satisfiable, since otherwise the termination of minus was proved. However, if we omit (7), then the remaining two constraints (8) and (9) are polynomial satisfiable, e.g., by associating c and 0 with 0, $\mathsf{s}(x)$ with $x+1$, and $\langle x, y\rangle$ with $x - y$. Hence we have found some cases for which (2) is polynomially satisfiable. But how can we represent those pairs $\langle x, y\rangle$ for which the transformation of (2) into (8) and (9) represents a sound abduction?

Regard the transformation of the termination hypothesis (2) into the ordering constraints (7)–(9) as an attempt to *verify* (2) by induction. Furthermore, regard the selected ordering constraints (8) and (9) as axioms for $\prec$ or as verified subgoals in the proof attempt, and (7) as an unproved subgoal. In [4] a method is presented to analyze a *failed* proof attempt P for a conjecture $\varphi(x_1, .., x_n)$. Depending on which subgoals have been verified and which induction hypotheses have been used in P, the procedure of a total *proof predicate* $p(x_1, .., x_n)$ is generated which represents exactly those data objects – i.e. it returns true – for which $\varphi(x_1, .., x_n)$ is verified by P. A proof predicate $p(x, y)$ for the above transformation then represents those pairs $\langle x, y\rangle$ for which (2) has been verified assuming (8) and (9). As (8) and (9) are polynomially satisfiable, (2) is polynomially satisfiable for all $\langle x, y\rangle$ satisfying $p(x, y)$.

We construct the below functional procedure for $p(x, y)$ according to [4]. The *conditions* of the cases in p are given by the case analysis performed in (2). Since (3) has been verified, the *result* true is generated for the condition $x = 0 \wedge y = 0$. Since inequality (7) of the second case has been omitted, the 'proof failed' for $x = 0 \wedge y = 0$ thus yielding the result false in p. Analogously, selection of (8) results in the value true for $x = \mathsf{s}(v) \wedge y = 0$. As the constraint (9) of the fourth case has also been selected and since it has been created assuming the induction hypothesis for $\langle v, w\rangle$, the result $p(v, w)$ is generated.

$$\begin{array}{lll} \textbf{function } p(x, y{:}\mathsf{nat}){:}\mathsf{bool} \Leftarrow & \textbf{if } x = 0 \wedge y = 0 & \textbf{then } \mathsf{true} \\ & \textbf{if } x = 0 \wedge y = \mathsf{s}(w) & \textbf{then } \mathsf{false} \\ & \textbf{if } x = \mathsf{s}(v) \wedge y = 0 & \textbf{then } \mathsf{true} \\ & \textbf{if } x = \mathsf{s}(v) \wedge y = \mathsf{s}(w) & \textbf{then } p(v, w) \end{array}$$

The procedure for p returns true iff x is greater than or equal to y. As $p(x, y)$ entails (2) for a well-founded relation $\prec$, in θ_{minus} the inequality $\langle x, \mathsf{s}(y)\rangle \prec \langle x, y\rangle$ may be replaced by $p(x, y)$. Then θ_{minus} defines the entire domain of minus, because it returns true iff x is greater than or equal to y.

Our method does not solve the halting problem, but it works successfully in many examples which could not be treated automatically up to now. It has a choice point when a subset M of the ordering constraints stemming from a termination hypothesis must be determined. However, since the number of constraints is finite, we can backtrack if M cannot be shown polynomially satisfiable. Efficiency can be improved with the successful heuristic presented in [2] which determines 'probably polynomially satisfiable' inequalities.

References

1. J. Brauburger and J. Giesl. Termination analysis for partial functions. In *Proceedings of the 3rd International Static Analysis Symposium, Aachen, Germany*, LNCS 1145, pages 113–127. Springer, 1996.
2. J. Giesl. *Automatisierung von Terminierungsbeweisen für rekursiv definierte Algorithmen.* PhD thesis, Technical University of Darmstadt, 1995.
3. D.S. Lankford. On proving term rewriting systems are noetherian. Memo MTP-3, Mathematics Department, Louisiana Technical University, Ruston, USA, 1979.
4. M. Protzen. Patching faulty conjectures. In *Proceedings of the 13th International Conference on Automated Deduction, New Brunswick, USA*, LNAI 1104, pages 77–91. Springer, 1996.

Proving Termination of Sequential Reduction Relation using Tree Automata

Thomas Genet*

In the context of rewriting logic, a Term Rewrite System (TRS for short) is a program, and the *normalisation* of a term by a TRS is the execution of the program. Thus, we can take advantage of methods and tools developed in rewriting to prove termination of programs. Unfortunately, proving termination of a TRS is undecidable and automatising TRS termination proof methods is hard. Moreover, it is difficult to apply a "divide and conquer" approach to termination proof of TRSs, since termination is not modular in general. Termination of the union of two TRSs is ensured only for restricted classes of disjoint or constructor sharing TRSs.

On the other hand, for termination of rewrite programs, we often do not need to prove *universal termination* of the TRS, i.e. termination on any argument, but only *existential termination*, i.e. termination on a set of "possible" arguments for which the program has been designed. Similarly, we do not need to prove termination of rewriting under any strategy of application of the rules, but only under a "normalising strategy" that rewrites any term to a normal form. However, existential termination and termination of rewriting under strategies are difficult to capture with orderings.

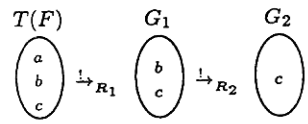
We propose here another approach to termination proofs of rewrite programs. We start from a restricted relation enjoying good modularity properties for termination: the *modular reduction relation* defined by Kurihara and Kaiji. The modular reduction relation preserves termination and completeness for disjoint systems and even for constructor sharing systems. The intuition behind this relation is to normalise a term w.r.t. a set of TRS by separate normalisations w.r.t. to each TRS.

When considering a *hierarchical combination* of TRSs, i.e. a constructor sharing combination where some defined symbols of a TRS (the base system) may appear in the right hand side of rules of another TRS (the extension), termination is not necessarily preserved by modular reduction relation. We choose here to define the *sequential reduction relation*, which is a deterministic version of the modular reduction relation, and to study a sufficient criterion for its termination. Let F be a signature, (F, R) be a TRS and $s, t \in T(F, X)$. We note $s \xrightarrow{!}_R t$ if $s \to_R^* t$ and if $t \in NF(R)$ (i.e. t is irreducible by R). Let

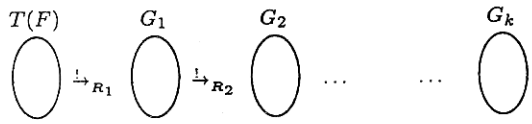
*INRIA Lorraine & CRIN CNRS, BP 101, 54602 Villers-lès-Nancy Cedex, France, E-mail: Thomas.Genet@loria.fr

$(F_1, R_1), \ldots, (F_n, R_n)$ be TRSs and $F = \bigcup_{j=1}^{n} F_j$. The sequential reduction relation, denoted by $\rightarrow_{R_1;\ldots;R_n}$, is defined by: $s \rightarrow_{R_1;\ldots;R_n} t$ if and only if s is reducible by the TRS $(F, R_1 \cup \ldots \cup R_n)$ and $\exists s_1, \ldots, s_{n-1} \in T(F,X)$ s.t. $s \xrightarrow[R_1]{!} s_1 \xrightarrow[R_2]{!} s_2 \ldots s_{n-1} \xrightarrow[R_n]{!} t$.

We are here interested in the termination of the transitive closure of the sequential reduction relation. Since this relation is a particular case of modular reduction relation, it also preserves termination for disjoint and constructor sharing systems. For the hierarchical and the general combination of systems, we propose a criterion based on a schematisation of all possible derivations on $T(F)$. In order to sketch the method on a very simple example (on a finite set of terms), let us consider $R_1 = \{a \rightarrow b\}$ et $R_2 = \{b \rightarrow a, a \rightarrow c\}$ be two terminating TRSs defined on $F = \{a, b, c\}$. Termination of $\rightarrow_{R_1;R_2}$ on $T(F)$ can be proved in the following way. By rewriting each term of $T(F)$ with R_1 into a normal form, we obtain the set G_1. Then, if we rewrite each term of G_1 with R_2 into a normal form, we obtain the set G_2. From G_2, neither R_1 or R_2 applies, hence $\rightarrow_{R_1;R_2}$ terminates on $T(F)$.



When considering signatures with some m-ary symbols ($m > 0$) and then, possibly infinite sets of terms, it is possible to generalise this approach thanks to a tree automaton representation. Sets of terms recognised by tree automata are called *regular tree languages*. From the tree automata theory, we know that G_i is not necessarily a regular tree language, even if G_{i-1} is. As a result, we choose here to consider an *approximation* of G_i, i.e. an automaton A_i s.t. the regular tree language recognised by A_i is a superset of G_i, i.e. $\mathcal{L}(A_i) \supseteq G_i$. Starting from $T(F)$, we first deduce an automaton A_1 recognising the set (or the language) $\mathcal{L}(A_1) \supseteq G_1 = \{t \in NF(R_1) \mid \exists s \in T(F) \text{ s.t. } s \rightarrow_{R_1}^* t\}$, then we deduce an automaton A_2 recognising $\mathcal{L}(A_2) \supseteq G_2 = \{t \in NF(R_2) \mid \exists s \in G_1 \text{ s.t. } s \rightarrow_{R_2}^* t\}$, and so on. If $R_1, \ldots, R_n$ are at least weakly terminating, and if there exist a natural k and an automaton A_k s.t. every term of $\mathcal{L}(A_k) \supseteq G_k$ is irreducible by $R_1 \cup \ldots \cup R_n$, then every term of G_k is irreducible and thus, $\rightarrow_{R_1;\ldots;R_n}$ is terminating.



When termination cannot be proven on the whole set $T(F)$, it is also possible to start the termination proof of $\rightarrow_{R_1;\ldots;R_n}$ from an automaton A_0 such that $\mathcal{L}(A_0) \subseteq T(F)$ is the set of initial terms we want to prove termination on. In order to automatically deduce an automaton A_1 from A_0 (or $T(F)$), then A_2 from A_1, and so on, we propose a specific matching and rewriting technique on tree automata, schematising matching and rewriting on sets of terms.

Proving Termination of Term Graph Rewriting

— Abstract —

Detlef Plump
Universität Bremen*

Term graph rewriting differs from term rewriting in that common subexpressions can be shared, improving the efficiency of rewriting in time and space. It is known that this model of computation terminates more often than term rewriting. In this talk, first the relationship between termination of term and term graph rewriting is discussed. Then a notion of simplification order for term graph rewriting is proposed, based on an extension of Kruskal's Tree Theorem to term graphs.

*Author's address: Fachbereich Mathematik und Informatik, Universität Bremen, Postfach 33 04 40, D-28334 Bremen, Germany. e-mail: det@informatik.uni-bremen.de.

Termination of Single-Pushout Graph Rewriting Systems — Forward Closures

Jürgen Müller *

Technical University of Berlin

To prove that a program is terminating for all inputs is an important task but in general a difficult one. In fact, the problem is undecidable in general. If we see graph rewriting systems as a generalization of term rewriting systems, we find proof methods for termination of term rewriting systems (a comprehensive theory of termination can be find in Dershowitz's survey [Der87]), but for graph rewriting systems we just found one method for the double-pushout approach [Ehr79] established in [Plu95]. The single-pushout graph rewriting approach established in [Löw93] is a generalization of the double-pushout approach in the sense that each double-pushout graph rewriting system can be translated into a corresponding single-pushout graph rewriting system so that each derivation possible induces a derivation with the translated rule. But the method presented in [Plu95] cannot be adapted by translating double-pushout graph rewriting systems into single-pushout graph rewriting systems. This is due to the fact, that the translation of the rules does not reflect the application condition of the rules in the double-pushout graph rewriting approach, which guarantee that the application of a rule does not leave dangling edges. This application condition is translated in deletion of unknown context which means the deletion of dangling edges. So, it is possible, that a terminating double-pushout graph rewriting system can be translated into a non-terminating single-pushout graph rewriting system, where the termination or non-termination is caused by the application condition for the rules only.

The method to prove termination of a graph rewriting system introduced in [Plu95] can be used for the single-pushout graph rewriting approach, too. The method allows to show termination of a graph rewriting system by

*Author's address: Fachbereich Informatik, Technische Universität Berlin, Sekr. FR 6-1, Franklinstr. 28/29, 10587 Berlin, Germany. E-mail: jolli@cs.tu-berlin.de

proving termination of certain restricted derivations. These so-called forward closures are derivations in which each step depends on previous steps. They are minimal in the sense that all items in a graph are used or generated by the rules. Forward closures can be inductively generated from the rules, providing a method to prove termination by showing that only finite forward closures are possible. Together with the notion of a consumptive graph rewriting system, that is, each rule of the system must at least consume one item of a graph, we can show the following characterization of termination. A graph rewriting system is terminating if, and only if, it is consumptive and does not admit an infinite forward closure.

References

[Der87] Nachum Dershowitz. Termination of Rewriting. In *Journal of Symbolic Computation*, number 3, pages 69 – 116, 1987.

[Ehr79] H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *1st Graph Grammar Workshop, Lecture Notes in Computer Science 73*, pages 1–69. Springer Verlag, 1979.

[Löw93] M. Löwe. Algebraic approach to single-pushout graph transformation. *TCS*, 109:181–224, 1993.

[Plu95] D. Plump. On termination of graph rewriting. In *Proc. 21st Workshop on Graph-Theoretic Concepts in Computer Science (WG '95)*. Lecture Notes in Computer Science, 1995. to appear.

3rd INTERNATIONAL WORKSHOP ON TERMINATION

Utrecht (The Netherlands)

Wednesday, May 21

8:30 - 8:40 **Opening of WST97**

8:40 - 9:30 **Nachum Dershowitz**
Well-Quasi Orderings and Some Applications

9:30 - 9:50 BREAK

9:50 - 10:30 **Bernhard Gramlich**
A Refined Analysis for Termination of Combined Rewrite Systems

10:30 - 11:10 **Maria Ferreira**
Recursively defined (quasi) orders on terms

11:10 - 11:30 BREAK

11:30 - 12:10 **Jean-Pierre Jouannaud** and **Albert Rubio**
Higher-Order Simplifcation Orderings

12:10 - 12:50 **Albert Rubio**
A total AC-compatible ordering with RPO scheme

12:50 - 14:20 LUNCH

14:20 - 15:10 **Ursula Martin**
Proving termination by counting things

15:10 - 15:50 **Jörg Frank** and **Dieter Hofbauer**
On Total Simplification Orderings on Strings

15:50 - 16:20 BREAK

16:20 - 17:00 **François Monin**
Ordinals extracted from termination proofs of functions

17:00 - 17:40 **Andreas Weiermann**
A constructive proof of strong normalization for Gödel's T via a monotone interpretation

18:30 - DINNER

3rd INTERNATIONAL WORKSHOP ON TERMINATION

Utrecht (The Netherlands)

Thursday, May 22

8:30 - 9:20 **Pierre Lescanne**
Strong Normalisation in Second Order Lambda Calculus with Explicit Substitutions

9:20 - 9:40 BREAK

9:40 - 10:20 **Roberto Di Cosmo** and **Delia Kesner**
Strong Normalization of Explicit Substitutions via Cut Elimination in Proof Nets

10:20 - 11:00 **Johannes Waldmann**
Deciding Termination in CL(**S**)

11:00 - 11:20 BREAK

11:20 - 11:50 **Thomas Arts**
Termination and Innermost Normalisation of Term Rewriting Systems (Termination)

11:50 - 12:30 **Jürgen Giesl**
Termination and Innermost Normalisation of Term Rewriting Systems (Innermost Normalisation)

12:30 - 13:30 LUNCH

13:30 - 14:20 **Hans Zantema**
Transformational techniques for proving termination of term rewriting

14:50 - EXCURSION

3rd INTERNATIONAL WORKSHOP ON TERMINATION

Utrecht (The Netherlands)

Friday, May 23

8:30 - 9:20 **Stefaan Decorte** and **Danny De Schreye**
Demand-driven and constraint-based automatic left-termination analysis for Logic Programs

9:20 - 9:40 BREAK

9:40 - 10:20 **Nachum Dershowitz, N. Lindenstrauss, Y. Sagiv**
What Norms are Useful for Logic Programs?

10:20 - 11:00 **Alfons Geser**
On Normalising, Non-Terminating One-rule String Rewriting Systems

11:00 - 11:20 BREAK

11:20 - 12:00 **Jürgen Brauburger**
Computing the Domains of Partial Functions

12:00 - 12:40 **Thomas Genet**
Proving Termination of Sequential Reduction Relation using Tree Automata

12:40 - 14:00 LUNCH

14:00 - 14:50 **Detlef Plump**
Proving Termination of Term Graph Rewriting

14:50 - 15:10 BREAK

15:10 - 15:50 **Jürgen Müller**
Termination of Single-Pushout Graph Rewriting Systems -- Forward Closures

15:50 - 16:30 **Adam Cichon**
???

List of Participants Workshop on Termination 1997

Thomas Arts

Department of Computer Science
Universiteit Utrecht
Padualaan 14
3584 CH Utrecht
the Netherlands
+31-30-2532585
+31-30-2513791
thomas@cs.ruu.nl

Stefaan Decorte

K.U. Leuven
Celestijnenlaan 200 A
3001 Heverlee
Belgium
+32-16-327544
+32-16-327996
Stefaan.Decorte@cs.kuleuven.ac. be

Juergen Giesl

FB Informatik
TH Darmstadt
Alexanderstrasse 10
D-64283 Darmstadt
Germany
+49-6151-164494
+49-6151-165326
giesl@inferenzensysteme.informatik.th-da

Roel Bloo

Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
the Netherlands
+31-40-2475004
none
bloo@win.tue.nl

Nachum Dershowitz

Hebrew University / Univ. of Illinois
16/4 Tifereth Israel Rd.
97500 Jerusalem
Israël
+972-2-627-3440
none
nachum@uiuc.edu

Richard Gilles

University of Orleans
Rue Leonard de Vinci, BP 6759
45067 Orleans Cedex 2
France
+33-2-3841-7298
+33-2-3841-7137
richard@lifo.univ-orleans.fr

David Bosley

Univ. of Northumbria at Newcastle
Ellison Terrace
Newcastle-upon-Tyne, NE1 8ST
United Kingdom
+44-191-227-3979
+44-191-227-4561
david.bosley@unn.ac.uk

Maria Ferreira

Universidade Nova de Lisboa
Quinta da Torre
2825 Monte de Caparica
Portugal
+351-1-294-8536
+351-1-194-8541
cf@di.fct.unl.pt

Bernard Gramlich

CRIN
INRIA Lorraine
615 Rue du Jardin Botanique, BP 101
54602 Villers-Les-Nancy Cedex
France
+33-3-83-593015
+33-3-83-278319
Bernhard.Gramlich@loria.fr

Jürgen Brauburger

Technische Hochschule Darmstadt
Alexanderstrasse 10
D-64283 Darmstadt
Germany
+49-61-51166658
+49-61-51165326
brauburger@informatik.th-darmstadt.de

Jörg Frank

Turmstrasse 44
D-10551 Berlin
Germany
+49-30-3966129
none
jofrank@cs.tu-berlin.de of jfrank@cscplo

Dieter Hofbauer

Technische Universität Berlin
Franklinstrasse 28/29, FR 6-2
D-10587 Berlin
Germany
+49-30-314-22624
+49-30-314-73500
dieter@cs.tu-berlin.de

Adam Cichon

CNRS-CRIN
INRIA Lorraine
Campus scientifique, 615 rue du jardin bota
54602 Villers-Les-Nancy Cedex
France
+33-3-83-593014
cichon@loria.fr

Thomas Genet

CRIN
INRIA Lorraine
615 Rue du jardin botanique
54602 Villers-Les-Nancy
France
+33-3-83-593018
+33-3-83-278319
genet@loria.fr

Jean-Pierre Jouannaud

Lab. de Recherche en Informatique
Bat 490 - Universite Paris XI
91405 Orsay cedex
France
+33-1-69-156905
none
Jean-Pierre.Jouannaud@lri.fr

Evelyne Contejean

Laboratoire de Recherche en Informatique
Bat 490 - Universite Paris XI
91405 Orsay cedex
France
+33-1-69-156476
none
Evelyne.Contejean@lri.fr

Alfons Geser

Universität Tübingen
Sand 13
D-72076 Tübingen
Germany
+49-7071-29-77476
+49-7071-29-67540
geser@informatik.uni-tuebingen.de

Delia Kesner

Laboratoire de Recherche en Informatique
Bat 490 - Universite Paris XI
91405 Orsay cedex
France
+33-1-69-156592
none
Delia.Kesner@lri.fr

List of Participants Workshop on Termination 1997

Pierre Lescanne

CRIN
INRIA Lorraine
615 Rue du Jardin Botanique, BP 239
54602 Villers-Les-Nancy Cedex
France
+33-3-83-593007
+33-3-83-278319
lescanne@loria.fr

Vincent van Oostrom

Vrije Universiteit Amsterdam
De Boelelaan 1081a
1081 HV Amsterdam
the Netherlands
+31-20-4447735
+31-20-4447653
oostrom@cs.vu.nl

Helene Touzet

CRIN
INRIA Lorraine
615 Rue du Jardin Botanique, BP 101
54602 Villers-Les-Nancy Cedex
France
+33-3-83-592043
+33-3-83-278319
touzet@loria.fr

Salvador Lucas-Alba

Universidad Politecnica de Valencia
Camino de Vera, s/n
E-46071, Valencia
Spain
+34-6-387-7353
none
slucas@dsic.upv.es

Detlef Plump

Universität Bremen
Postfach 33 04 40
D-28334 Bremen
Germany
+49-421-218-4854
det@informatik.uni-bremen.de

Xavier Urbain

Lab. de Recherche en Informatique
Bat 490 - Universite Paris XI
91405 Orsay cedex
France
+33-1-69-156485
none
Xavier.Urbain@lri.fr

Claude Marche

Laboratoire de Recherche en Informatique
Bat 490 - Universite Paris XI
91405 Orsay cedex
France
+33-1-69-156485
none
Claude.Marche@lri.fr

Laurence Puel

Laboratoire de Recherche en Informatique
Bat 490 - Universite Paris XI
91405 Orsay cedex
France
+33-1-69-156460
none
Laurence.Puel@lri.fr

Johannes Waldmann

Institut für Informatik
Friedrich Schiller University, Jena
Ernst-Abbe-Platz 3
D-07743 Jena
Germany
+3641-638684
+3641-638726
joe@informatik.uni-jena.de

Ursula Martin

Computer Science Devision
University of St. Andrews
North Haugh, St. Andrews
Fife KY16 9SS
United Kingdom
+133-4-463252
+133-4-463278
um@dcs.st-and.ac.uk

Alberto Rubio

Universitat Politecnica de Catalunya
Jordi Girona, 1-3, Campus Nord (C6)
08034 Barcelona
Spain
+34-3-4017988
+34-3-4017014
rubio@lsi.upc.es

Andreas Weiermann

Westfällische Wilhelms-Universität Münst
Einsteinstrasse 62
D-48149 Münster
Germany
+49-251-83-3-3762
+49-251-83-3-3078
weierma@math.uni-muenster.de

François Monin

Department of Computer Science
Universiteit Eindhoven
Den Dolech 2
5612 AZ Eindhoven
the Netherlands
+31-40-2475020
+31-40-2463992
monin@win.tue.nl

Duncan Shand

Division of Computer Science
University of St. Andrews
North Haugh, St. Andrews
FIFE KY16 9SS
United Kingdom
+44-1334-463271
+44-1334-463278
ddshand@dcs.st-and.ac.uk

Hans Zantema

Vakgroep Informatica
Universiteit Utrecht
Postbus 800089
3508 TB Utrecht
the Netherlands
+30 2534116
+30 2513971
hansz@cs.ruu.nl

Jürgen Müller

Institute for Communication and Software
Technical University of Berlin
Franklinstrasse 28/29
D-10587 Berlin
Germany
+49-30-314-24189
+49-30-314-23516
jolli@cs.tu-berlin.de

Elias Tahnan Bittar

LLAIC1
Université d'Auvergne
IUT Informatique, BP 86
63172 Aubiere Cedex
France
+473-407605
+473-407733
tahhan@llaic.univ-bpclermont.fr