Electronic Notes in Theoretical Computer Science 2 (1995)
URL: http://www.elsevier.nl/locate/entcs/volume2.html 9 pages

Semagraph: the Theory and Practice of Term Graph Rewriting

Ronan Sleep

School of Information Systems
University of East Anglia
Norwich, United Kingdom

Abstract

Semagraph II is a working group (WG 6345) supported by the CEC basic research directorate. The main aim of Semagraph II has been to mature and promulgate the research results and prototypes developed during the Semagraph I Basic Research Action (BRA 3074). This note outlines the aims and achievements of Semagraph, and provides references to key results.

1 Introduction

At an Esprit technical workshop held late in 1987, a number of computational models were considered as a possible basis for a 'common virtual machine' to support European work on various programming languages and architectures, both conventional and novel (e.g. highly parallel). If a single computational model could be found which efficiently supported the various conventional and novel language paradigms, it could be the basis for common intermediate language and associated toolset, as well as a basis for reasoning about the behaviour and performance of hybrid systems built using many programming paradigms and realised on many architectures.

One conclusion from the workshop was that whilst no satisfactory common computational model was known, one based on graph rewriting might succeed. However, it was also noted that a considerable amount of foundational work was required even to establish the feasibility of such a model. Subsequent to the workshop, the Semagraph Basic Research Action (BRA 3074) was approved and ran from July 1989 to June 1992. One of its primary aims was to develop appropriate foundations for a graph rewriting model of computation.

Subsequently the CEC funded a follow-up activity in the form of a working group (WG6345) called Semagraph II, scheduled to run from October 1992 to September 1995.The central aim of Semagraph II is to mature and promulgate key results obtained from Semagraph I. This note briefly describes the aims, composition, activities and major results of Semagraph.

2 Project Partners and Aims

2.1 The Partners

The following organisations participated in either Semagraph I (BRA 3074) or Semagraph II (WG 6345) or both:

- Ecole Normale Superieure (CAIMENS: Paris, France)
- Centre for Mathematics and Computer Science (CWI: Amsterdam, Netherlands)
- European Computer-Industry Research Centre GmbH (ECRC: Munich, Germany)
- International Computers Ltd (ICL: Manchester, England)
- Imperial College of Science, Technology and Medicine (ICSTM: London, England)
- University of Nijmegen (KUN: Netherlands)
- University of Rennes (IRISA: France)
- University of East Anglia (UEA: Norwich, England)

2.2 Aims

The principal objective of Semagraph was to develop new foundational knowledge about graph rewriting models of computation paying special attention to the characteristics of practical models.

3 Rewriting as a Computational Model

Rewriting forms the basis of several specification languages and logics. The notion of 'Computation as Controlled Deduction' underlies much of the recent activity in computer science, ranging from advanced type theory (like that of Martin-Löf or Huet and Coquand's theory of constructions) to new generation languages and architectures, backed by many IT initiatives including Esprit. These various initiatives have stressed particular paradigms for symbolic processing, such as functional, logic and object oriented approaches.

A central concern of much recent work has been to discover some powerful means of programming the potentially massive parallelism offered by VLSI and later technologies which does not require the programmer to specify detailed control flow for every processing element. Hence the interest in essentially rule-based models of computation of which the families of functional, logic and actor/object oriented languages are examples.

Semagraph regards the various execution models on which such languages are based as particular examples of rewriting systems, and seek a more general underlying model. A general notion of computation as rewriting must take account of *sharing* and *parallelism*.

We adopted a parallel rewriting model in which one or more rewriting agents repeatedly performs a match-act cycle on some shared object represented by a *graph*. The earliest models developed by the Semagraph partners (DACTL [4], and LEAN [5]) were defined informally, but had precise enough operational semantics to act as reference models for industrial projects. Both DACTL and LEAN were very expressive, supporting for example notions such as global-state triggered multiple assignment.

This expressiveness came with a cost: it rapidly became clear that it would be hard to obtain the foundational theory we sought without resorting to simpler models, particularly given that much of the syntactic theory of *term rewriting* had made progress only by studying the special and relatively pleasant *orthogonal* term rewriting systems.

4 Term Graph Rewriting

In fact the pressure from the theory to select simple models was supported by a number of language design and implementation issues, such as the difficulty of designing type systems for generalised graph rewrite rules, and the need to implement parallel agent matching with as little global locking as possible. With both pressures in the same direction it was not hard to decide to adopt the simplest model possible. We adopted the *term graph rewriting* model of computation as the interface between theoretical and practical work in Semagraph. This decision was not intended, and indeed did not, prevent continued practical or theory work within Semagraph on other rewriting models, but it did have the important effect of synchronising many Semagraph activities. Over half the papers in the book containing much of the Semagraph output [18] deal either explicitly or implicitly with the Term Graph Rewriting (TGR) model.

The model of term graph rewriting adopted was essentially that studied by earlier workers such as Wadsworth [1], Staples [2] and Turner [3]: a term graph is rooted directed graph with labelled nodes. Of course, simply defining a set of structured objects is not sufficient to characterise a computational model: the notion of a graph rewrite rule, and a definition of what it means to perform a term graph rewriting (both sequeampl and parallel) are also required. For an introductory presentation of a restricted form of Term Graph Rewriting see [6].

The significance in adopting a term graph model lies in its obvious connection to classical term rewriting. Each term graph unravels, in an obvious sense, to a term: the unravelling starts at the root of the term graph, and proceeds recursively. The unravelling transformation gives us a basis for studying the relationship between classical term rewriting and term graph rewriting, and also a basis for designing axioms which a notion of term graph rewriting must satisfy. Most obviously, term-graph-rewritability should imply term-rewritability of the unravelled term graphs. The converse does not hold in general, because a term rewriting is more general than term graph rewriting. The great advantage of pointer sharing of subterms is that rewriting a single shared subterm in a term graph may correspond to rewriting many copies of the subterm in the term in the same way. The disadvantage is that the correspondence is forced: there is no possibilities of treating separate copies of a subterm differently if these are represented by a single shared copy.

5 Selected Activities and Results

Semagraph activities fall broadly under the following headings:

Theory focused on the relationship between term rewriting and term graph rewriting, but also studied alternative models e.g. categorical and equational notions of graph rewriting.

Implementation focused on developing industrial quality prototype languages and compilers based on term graph rewriting. This resulted in the major practical output from Semagraph, namely the Concurrent CLEAN software development system from Nijmegen. The techniques underlying this system are the subject of a major textbook [19].

Analysis focused on issues such as typing and compile time analysis.

A representative picture of Semagraph activities with preliminary versions of many results was collected in [18], which also contains chapters from our sister project COMPUGRAPH. Some of the key results from Semagraph are highlighted below.

5.1 Theory

Transfinite term rewriting and its relation to term graph rewriting Some studies of the correspondence between term rewriting and term graph rewriting had been made before Semagraph, e.g. [6] and earlier related papers. In particular it was known that in the special case of orthogonal systems, *acyclic* term graph rewriting is a sound implementation of term rewriting, and for normal forms also a complete one. It proved a major exercise to extend this result to cyclic graphs [14], not least because unravelling cyclic graphs generates infinite terms and it proved necessary to extend classical term rewriting [8] to obtain a reference model for our work with cyclic term graphs.

Categorical models of graph rewriting The traditional double pushout definition of graph rewriting can be simplified to a single pushout in a category of partial graph morphisms [15]. When hereditary pushouts exist in the base category of total morphisms (as is the case for the usual category of hypergraphs), all pushouts exist in the category of partial morphisms, without any need for the usual 'gluing conditions'.

Substitution calculi In [23] a substitution calculus was developed to demonstrate the correctness properties of a relatively space efficient combinator encoding of lambda terms. A later Semagraph study [21] used a similar but more direct approach by defining an extended lambda calculus containing additional substitution rules, and proving various confluence properties.

Equational graph rewriting A natural direction for Semagraph work is to attempt to develop a more general theory of term graph rewriting which allows bound variables. One approach is to regard the formal definition of a term graph as a system of equations from which new equations may be deduced using substitution rules. A variety of substitution rules may be defined, each corresponding to the basic instructions of some substitution machine. [20] presents an early study of the properties of such calculi in the context of modelling cyclic lambda graphs.

Notions of undefinedness During our studies of various rewriting models we repeatedly found ourselves using distinct syntactic characterisations of undefined terms. In [25] we gather together these notions, and study their relations and properties.

The relationship between Curried and Uncurried rewriting systems Currying is a transformation of term rewrite systems which may contain symbols of arbitrary arity into systems which contain only nullary symbols, together with a single binary symbol called application. It is frequently used by language implementors. We studied the properties of this transformation and its converse in [27].

Implementing Mobile Processes using Graph Rewriting There is increasing interest in fine grain process models of computation, and it is interesting to examine the extent to which a graph rewriting model can be used to implement fine grain process models. [26] studies this problem, and presents some new translation techniques. This work relates to the powerful state of the art mobile process system called Facile, developed by ECRC.

Fundamental Properties of Expression Reduction Systems The Combinatory Reduction Systems of Klop have been alternatively formulated as *Expression Reduction Systems*, and various properties fundamental properties developed [28].

5.2 Implementation

The major practical result from Semagraph is Nijmegen's Concurrent CLEAN software development system. At the heart of Concurrent CLEAN is a functional language built on a term graph rewriting model of computation. Key features of this system [1] are outlined below:

Functional programming: Concurrent CLEAN can be regarded as a lazy functional programming language and as such it supports pattern matching, higher order functions, a polymorphic type system, a lazy evaluation scheme and automatic memory management.

Uniqueness type system for safe use of destructive update: By keeping reference counts to nodes in a term graph undergoing reduction, it is possible to recognise at run time cases where an in place update implementation of a rewrite is safe. But such dynamic monitoring is costly, and compile time analysis is only practicable for special cases. In Concurrent

[1] http://www.cs.kun.nl/ clean contains information about obtaining Sun, Macintosh and OS/2 versions of the Concurrent CLEAN system via FTP

CLEAN, the type system is extended [29] to allow the programmer to declare instances of objects which will at run time have at most one reference. This allows the programmer to write various input/output methods in a functional style, and also enables the programmer to benefit from in place update of array or list elements in appropriate cases. For example, a functionally specified quick sort algorithm applied on a spine-unique list can effectively be implemented as an in situ sort algorithm with the same time and space efficiency as an imperatively written program in C.

Abstract interfaces to industry standard APIs: By using a combination of uniqueness types and higher order functions, it is possible to define various standard Application Programmers Interfaces in Concurrent CLEAN. This means, for example, that on a Macintosh all the usual GUI toolbox calls may be used by the Concurrent CLEAN programmer.

Support for parallel and distributed programming: CLEAN supports implicit and explicit control of parallel evaluation and granularity allowing both parallel and distributed applications to be programmed. Control is expressed by annotating appropriate sections of a term graph rewrite rule to indicate for example whether or not parallel evaluation is permitted, and whether or not movement to another processor is allowed. Examples of parallel algorithms expressed in CLEAN are given in [16] and examples of communicating distributed processes expressed in CLEAN are given in [7].

Advanced compiler technology: The raw function call rate per second is comparable with that obtained via C, and sophisticated interactive windowing applications run in real time. For suitable applications, and appropriate use of Concurrent CLEAN's concurrency control mechanisms, near linear speedups have been obtained [16,17].

Stand-alone application generation: The CLEAN system can generate stand alone applications on a range of industry standard personal workstations.

5.3 Analysis

There are many features of Concurrent CLEAN which are novel, most notable ca:

- the use of term graphs rather than terms
- the uniqueness type system
- experimental annotations for control
- abstract reduction

A considerable amount of study needed before these new techniques are well understood, especially in combination. However, theory work within Semagraph [14] has laid some foundations, and other work within Semagraph has examined more directly some of the CLEAN techniques.

Abstract reduction A Concurrent CLEAN program is expressed as a collection of term graph rewrite rules, and the computational model is a (dynamically varying) collection of term graph rewriting agents operating in parallel on a global graph. The compiler must respect this model, but otherwise is allowed to perform quite dramatic transformations in the interests of either speed or space efficiency or both. For example simple recursive functions should generate fast iterative machine code which builds no graph structures at run time and which passes evaluated, unboxed small arguments in registers where possible. The Concurrent CLEAN compiler employs a number of new and old techniques [19] in its analysis, most notably a new technique called *abstract reduction*. Some preliminary results concerning abstract reduction appear in [24,11,13].

Semantics Some early studies of mathematical structures for representing semantics are described in [9,12,30].

Typing Concurrent CLEAN has a powerful, expressive and very novel type system which appears to work very well in practice but is not so well understood theoretically. One approach is to use the transfinite bridge to map type systems for terms to type systems for term graphs. However, it turns out that there are significant issued to be resolved even within the restricted world of terms. A study of one such issue appears in [10].

6 Conclusion

The main result of Semagraph has been to show that it is indeed possible to build a modern, efficient and highly expressive software development environment (Concurrent CLEAN) based on a term graph rewriting model of computation. Further, using new techniques discovered within Semagraph, it is possible for the programmer to express safe in place update operations using uniqueness types, and to express both parallel and distributed solutions to problems using annotations. Last but not least, Concurrent CLEAN is available now on a variety of industry standard platforms, and generates very efficient stand alone applications.

The pragmatic advances made by the designers and implementors of CLEAN have left the theoreticians somewhat behind. But several important foundations stones have been laid, most notable the transfinite bridge connecting term graph rewriting and term rewriting. It is to be hoped that the growing industrial interest in CLEAN will lead to support for researching the many new issues raised by CLEAN.

7 Acknowledgements

The author would like to thank, on behalf of Semagraph, our sister working group COMPUGRAPH for organising the joint final workshop (SeGraGra 95), and arranging for these electronic preprints. The support of the CEC in the form of Basic Research Action No. 7034 and Working Group No. 6345 is gratefully acknowledged.

References

[1] Wadsworth C.P., Semantics and pragmatics of the lambda-calculus, Ph.D. thesis, University of Oxford, 1971.

[2] Staples J., A graph-like lambda calculus for which leftmost-outermost reduction is optimal, in Graph grammars and their application to Computer Science and Biology, ed. V.Claus, H.Ehrig, and G.Rozenberg, LNCS, 73 (Springer 1979).

[3] Turner, D.A., A new implementation technique for applicative languages, Software: Practice and Experience, vol.9, 1979.

[4] Glauert J.R.W., Kennaway J.R. and Sleep M.R. DACTL: A Computational Model and Compiler Target Language Based on Graph Reduction. ICL Technical Journal, V5, Issue3, May 1987 pp 509-540

[5] Barendregt H.P., Eekelen M.C.J.D. van, Glauert J.R.W., Kennaway J.R.,Plasmeijer M.J. and Sleep M.R. Towards an Intermediate Language based on Graph Rewriting. Journal of Parallel Computing 9, North Holland, 1989

[6] Barendregt H.P., Eekelen M.C.J.D. van, Glauert J.R.W., Kennaway J.R.,Plasmeijer M.J. and Sleep M.R., Term Graph Rewriting, Proc. PARLE conference, Vol 2, June 1987 LNCS 259, Springer-Verlag. pp141-158

[7] van Eekelen M.C.J.D. and Plasmeijer M.J., Process Annotations and Process Types, in [18] John Wiley April 1993, pp346-362

[8] Kennaway, J.R., Klop, J.W., Sleep, M.R., and de Vries, F.J., Transfinite reductions in orthogonal term rewriting systems, Information and Computation, V119, No. 1, May 1995 pp18-38

[9] Hunt S., PERs, Logical Relations and Abstract Interpretation, DoC Technical Report 91/27, Imperial College

[10] van Bakel S., Smetsers S. and Brock S., Partial Type Assignment in Left Linear Applicative Term Rewriting Systems, in [18].

[11] van Eekelen M.C.J.D., Goubault E., Hankin C.L. and Nocker E., Abstract Reduction: Towards a Theory via Avstract Interpretation, in [18].

[12] Kennaway, J.R., Klop, J.W., Sleep, M.R., and de Vries, F.J., Event Structures and Orthogonal Term Graph Rewriting, in [18].

[13] Goubault E. and Hankin C.L., A Lattice for the Abstract Interpretation of Term Graph Rewriting Systems, in [18].

[14] Kennaway, J.R., Klop, J.W., Sleep, M.R., and de Vries, F.J., The Adequacy of Term Graph Rewriting for Simulating Term Rewriting, ACM Transactions of Programming Languages and Systems, V16 No3 May 1994 pp493-523.

[15] Kennaway J.R., Graph Rewriting in a Category of Partial Morphisms, Proc 4th International Workshop on Graph Grammars and their Applications, ed. Ehrig et al, Springer LNCS 532, June 1991

[16] Goldsmith R.G, McBurney D.L. and Sleep M.R., Parallel Execution of Concurrent Clean on ZAPP, in [18] John Wiley April

[17] McBurney D.L. and Sleep M.R., Graph Rewriting as a Computational Model, McBurney D.L. and Sleep M.R., in: Concurrency: Theory, Language and Architecture, ed Yonezawa A. and Ito T., Springer-Verlai LNCS 491, 1991

[18] ed. Sleep M.R., Plasmeijer M.J. and van Eekelen M.C.J.D., Term Graph Rewriting: Theory and Practice, John Wiley, April 1993

[19] Plasmeijer M.J. and van Eekelen M.C.J.D., Functional Programming and Parallel Graph Rewriting, Addison Wesley International Computer Science Series, 1993

[20] Ariola Z.M. and Klop J.W., Cyclic lambda graph rewriting, 9th Annual IEEE Symposium on Login in Computer Science, July 1994, Paris

[21] Hardin T., How to get Confulence for Explicit Substitutions, [18]

[22] Clark D. and Hankin C., A lattice of abstract graphs, Programming Language Implementation and Logic Programming, PLILP, Tallin, Estonia 1993

[23] Kennaway J.R. and Sleep M.R., Director Strings as Combinators, ACM Transactions on Programming Languages and Systems, V10, No.4, Oct. 1988

[24] Hankin C. L., Static Analysis of Term Graph Rewriting Systems, Proceedings of PARLE ‘91, Volume 2, Springer LNCS 506.

[25] Ariola Z, Kennaway, J.R., Klop, J.W., Sleep, M.R., and de Vries,F.J., Defining the Undefined, Int. Symp. on Theoretical Aspects of Computer Software, Sendai Japan, Springer Lecture Notes in Computer Science, vol. 789

[26] Glauert J.R.W., Asynchronous Mobile Processes and Graph Rewriting, Proceedings PARLE’92, Paris, Springer Lecture Notes in Computer Science, Vol.605

[27] Kennaway, J.R., J.R., Klop, J.W., Sleep, M.R., and de Vries,F.J., Comparing Curried and Uncurried Rewriting, J. Symbolic Computation, 1995 to appear.

[28] Khasidashvilli Z. and , On higher order recursive program schemes, Proc. of the Colloquium on Trees in Algebra and Programming, CAAP’94, Springer LNCS, vol.787 1994.

[29] Smetsers J.E.W., Barendsen E., Eekelen M.C.J.D. van and Plasmeijer M.J., Guaranteeing safe destructive updates through a type system with uniqueness information for graphs. Technical report 93-4 1993, University of Nijmegen

[30] Raoult J-C and Voisin F., Set-theoretic graph rewriting, INRIA research report RR 1663, INRIA Rennes, 1992

[31] Giacalone A., Mishra P. and Prasad S., Facile: A symmetric integration of concurrent and functional programming, IJPP V18, No.2 1989