Internship Report

Lesley Wevers
s0219843@student.utwente.nl

May 2nd 2011 - July 9th 2011
Department of Computer Science at the University of York
supervised by Dr. Detlef Plump

For the EWI Faculty at the University of Twente

September 25, 2011

Introduction

As part of the Twente Graduate School programme Dependable and Secure Computing I have done an internation research internship at the University of York. The internship started on the 2nd of May 2011, and ended 10 weeks later on the 9th of July 2011. During my internship I was supervised by Detlef Plump from the University of York, and my supervisor at the University of Twente was Arend Rensink.

During my internship I worked on a graph programming language called GP. The idea was to extend the GP language with results from earlier work on constant time graph transformations, and evaluate runtime complexity of programs in this new language. During the project I constructed a few algorithms within this new language, as well as creating a model for the runtime complexity of GP programs.

In this report I will provide an overview of my internship. In section 1 I will describe the organisation where I have been working. In section 2 I will quickly review some research which lay the foundations for my work. In section 3 I will describe my original assignment, and I will describe how I have performed the assignment. Finally, in section 4 I will provide an overview of the results, describe what still has to be done, and conclude this report.

Appended to this report is a report written for the Twente Mobility Fund which describes my non-work related experiences.

1 Organisation

During my internship I was a visiting research student at the Programming Languages and Systems Research Group, more commonly known as the PLASMA group, which is part of the Department of Computer Science at the University of York. The PLASMA group is working on programming languages, methods and tools and has a special interest in radical alternatives. During the time I was there, there where projects running on functional programming, graph transformations, secure software development, and perhaps more I was not aware of. Every project was headed by a senior member of the group, and most of them had PhD students working on sub-projects as well.

My supervisor in York was Detlef Plump, a senior lecturer at the PLASMA group. His research interests are mainly in graph rewriting and while I was there he was working on a project called Graph Programs (or GP), which I will explain in the next section. To put things in perspective, my work was an extension of the work already performed earlier in the GP project. And while I was there, there was also another internship student from India working on the project, as well as a PhD student working on seperation logic for GP.

When I arrived at the Department of Computer Science, they had just moved to a brand new building in a newly build part of the campus. During my internship I had a desk in a room together with all five PhD students working at the PLASMA group. The culture in the group was very informal and open. The office of my supervisor was just on the other side of the door, and communication was no problem.

Most weeks the group held their weekly seminar, one time inviting someone from the University of Leeds, but mostly work from within the group was presented to other members of the groups. These seminars not only served to inform the members of the group what was going on, but they also provided the possibility of presenting problems to the group and discuss them, hopefully getting some fresh ideas from people not tangled up in the problems.

There was also a weekly seminar hosted by the department, topics were very diverse, from a high-level overview of real-time systems, to model driven development from a business perspective, to very technical presentations on projects the speakers where working on. All-in-all this is a great way to broaden my knowledge in all kinds of subjects within Computer Science.

2 Preceeding Research

The research I have been doing was a continuation of earlier research performed by Detlef Plump on the graph programming language GP, and Mike Dodds on rooted graph rewriting. In this section, I assume the reader is familiar with common concepts in graph rewriting. I will start out by giving an overview of GP, followed by an overview of rooted graph transformations. For more details I refer the reader to the concept article.

The GP language extends graph rewriting systems in two ways: replacing

rules with rule schemata and introducing a command language. Rule schemata enhance rules with expressions on the right hand side over variables from the left hand side, as well as allowing each rule to have a condition over the variables from the left hand side which determines if the rule is applicable. When instantiating the variables of a rule schema with concrete values, you essentially get a (potentially infinite) set of normal graph rewriting rules. Another important ingredient of GP is the command language. Every GP program is associated with some command sequence, which determines which rule schemata are active at any point during the reduction process, as well as introducing an if-expression which allows conditional branching. The meaning of GP programs is defined by semantics which describe how a program is to be executed. GP is a non-deterministic language, which means that a program can produce multiple results. It is also Turing complete in the sense that any graph which is computable from a start graph, can be computed by GP.

For his PhD thesis, Mike Dodds has performed research on fast graph transformations. One technique in particular is that of rooted graph transformations. In rooted graph rewriting every rule has one unique node called a root, and the host graph has one such node as well. Rooted transformations allow a slightly different algorithm for building graph morphisms than the conventional graph matching: start building a match by mapping the root from the rule to the root in the host graph, and then recursively extend the match by following edges from nodes which have been matched thus far until we find either a valid morphism and we are done, or we can not find a morphism and fail. The main result of rooted graph matching is that the complexity of constructing a morphism is bounded by the size of the rule and the maximum outdegree in the host graph. In practice rules are often fixed, if we also make sure that the maximum outdegree of rules in the host graph is bounded by some fixed integer, we essentially get a matching algorithm which runs in time independent of the host graph, i.e. in constant time.

3 Assignment

In this section I will provide an overview of the work I have been doing. I will start out with the initial assignment I receieved, followed by a chronological overview of the actual research work I have been doing, the work I have done on the article, and finally I will say something about a presentation I have done for the group in York.

3.1 Initial assignment

Initially my assignment was to extend the GP language with rooted graph transformations, perform a case study by implementing some algorithms in this new language, and to analyse the running time of these algorithms. The original assignment was stated as follows:

The graph programming language GP [1] allows to solve graph problems

at a high level of abstraction, based on nondeterministic applications of graph transformation rules. The nondeterminism, however, makes graph matching a costly operation which may cause poor execution times for graph programs.

To make graph programs fast, the PhD thesis [2], the paper [3] and the monograph [4] propose a technique which removes much of the nondeterminism in graph matching: viz. to contol the matching process by a unique root node in each rule and in the host graph. This allows constant-time graph matching on graphs of bounded node degree. The purpose of this project is to carry out a case study which tries out this technique on some selected graph algorithms. The running time of the resulting programs shall be determined theoretically - assuming a suitable implementation of GP - and compared practically with that of conventional GP programs using the current GP implementation [5].

References:

1. D. Plump: The Graph Programming Language GP. In Proc. Algebraic Informatics (CAI 2009), volume 5725 of Lecture Notes in Computer Science, pages 99-122. Springer-Verlag, 2009

2. M. Dodds: Graph Transformation and Pointer Structures. PhD thesis, The University of York, 2008.

3. M. Dodds and D. Plump: Graph Transformation in Constant Time. Proc. International Conference on Graph Transformation (ICGT 2006), Lecture Notes in Computer Science 4178, pages 367-382. Springer-Verlag, 2006

4. H. Doerr: Efficient Graph Rewriting and Its Implementation, Lecture Notes in Computer Science 922. Springer-Verlag, 1995

5. G. Manning and D. Plump: The GP Programming System. In Proc. Graph Transformation and Visual Modelling Techniques (GT-VMT 2008), Electronic Communications of the EASST 10, 2008

3.2 Research

As I had some time to prepare for the internship, I decided to study the paper about the graph programming language before I went to the University of York to make most of my time while I was there.

On the first day, Detlef and I had a meeting to discuss our plan of action. There was not really a clear research question at this time, but the plan was to explore the idea of rooted GP and the complexity of GP programs. We decided to proceed as the initial assignment stated by starting out with a case study. The idea was to see what came out, and decide later where to go from there. At this point the research was pretty much open ended.

As there was no implementation available of rooted GP, and it was outside the scope of my assignment to provide such an implementation, most of my work was done on paper.

After two weeks I had formulated eight algorithms in rooted GP and I had a simple complexity analysis of these algorithms. At that point we agreed that

rooted GP seemed to work on paper, and in the third week started work on an article to describe the results of the case study.

During our meeting at the start the fourth week we decided that we would like a formal description of complexity of rooted GP programs. The idea was that rewrites in rooted GP programs take constant time, so complexity could be defined in the number of rewrite steps. During this week I formulated a complexity model for non-deterministic GP programs by taking the idea of a state transition graph and extending it by adding a measure of cost to the state transitions. A model could be built from a starting state and inductively expanded by recursive rules defined over the semantics of GP.

In the fifth week I showed the model to Detlef, and we both agreed that the model was too complex. Also there was a problem in the way the model was defined using the semantics. Instead of resolving these problems, we decided to simplify our goal by not considering back-tracking. Detlef pointed me to an article about complexity in term rewrite systems. During this week I mostly studied this article, and also took a better look at the work of Dodds. In the meantime I was working on a summary of the work on fast graph transformations by Dodds for the article.

In the sixth week I had a pretty clear idea on how to formulate runtime complexity for rooted graph rewrite systems based on the ideas of derivational complexity in term rewrite systems. This week I worked on working out the details and writing it up in the article. I also got an idea to generalize the model a bit further in order to account for runtime complexity of general graph rewrite systems. Detlef advised me to first work out the details for the rooted case, as this was simpler.

In the eight week I had the opportunity to give a presentation of my work for the group, so during the seventh week I prepared for this presentation. This really forced me to give a clear definition of all the concepts we had up to then, as well as allowing me to create some examples of derivational complexity for graph rewrite systems. By the end of the week the presentation was pretty much done, and I had a rudamentary model of runtime complexity of rooted GP. However there where still some problems with this model.

At the start of the eight week, Detlef and I had a meeting where I discussed the problems. During the session we worked together on solving the problem. I had the idea of extending the small-step semantics relation to include derivation length. Detlef thought this was a good idea and he completed the idea by defining the transitive closure over this relation which seemed to neatly solve our problem.

In the ninth week I worked out the details of our complexity model in our paper. During the last week I was also trying to find a formal method to describe complexity proofs of rooted graph programs, however I was unable to find a solution to this yet.

In the tenth week, I was unable to do any more work, as I was arranging for my trip back home, as well as finishing up all my work so far.

3.3 Writing the Article

During all these weeks I was constantly working on the article. I started with the article in the second week, setting up the main structure and inserting preliminaries from earlier papers by Detlef. In the third week I started writing about the results of the case study by describing algorithms in the paper. During this week I also started work on a chapter about rooted graph rewriting to summarize the work by Dodds. Work on this chapter continued well into week six, because the ideas were not very intuitive, and there were quite a lot of concepts to explain. In the fourth week I described my initial complexity model in the article, however as this model was revised multiple times, I had to rewrite this chapter a lot. During the sixth week I wrote a chapter about the derivational complexity of graph rewrite systems as a model of runtime complexity. This chapter was revised in the ninth week to incorporate a model of runtime complexity for general graph rewrite systems, as well as adding conditions to ensure that bounds, neccesary for constant time graph transformations, are preserved over multiple rewrite steps.

By the end of the internship, the article was not yet completed. If the article is finished I hope it might be worthy of a publiccation. After my internship, during the writing of this report I worked on cleaning up the article in order to make it well readable. I completely rewrote the chapters about complexity of graph rewrite systems and complexity of GP to include the ideas I had which I was not yet able to write down during my internship. I also chose one algorithm from the case study, and worked it out up to a somewhat final version. I finished the concept article by writing an abstract, introduction and conclusion.

In the evaluation section I summarize the results in the article, as well as provide a summary of the work which still has to be done to complete it.

3.4 Presentation

In the eight week of my internship I got the opportunity to do a presentation of my work for the group. The idea of this presentation was to inform the group of my work, as well as to gather feedback on it. During the seventh week I prepared the presentation, in the process clarifying a lot of results, as well as getting a coherent storyline which I could use for the article. The presentation itself went quite smooth, and the audience was positive about the presentation.

4 Evaluation

In this section I will summarize the results written in the concept article, summarize the work to be done to complete the article, and reflect on my work during the internship.

4.1 Summary of results

The following is a quick summary of results described in the article:

- A short overview of work by Dodds on rooted graph rewriting. (incomplete, missing algorithm descriptions, missing proofs)
- A definition of derivational complexity for graph rewrite systems adapted from that of term rewrite systems.
- An extension of derivational complexity to runtime complexity for graph rewrite systems.
- Description of the link between derivational complexity and runtime complexity of rooted graph rewrite systems.
- A definition of rooted GP.
- A justification that rules in rooted GP can be applied in constant time.
- A definition of derivational complexity for GP.
- A definition of runtime complexity for GP.
- A case study showing how to derive complexity for a few selected algorithms, only one in the concept article, and seven more algorithms on paper.

In term rewriting, work has been done to derive complexity bounds from termination proofs. I hope these results may provide a basis for further work in a similar directions for graph rewriting and extensions such as GP.

4.2 Remaining work

There are still some rough edges in the article, some missing parts, and most parts need to be reviewed. Concretely, the following is a summary of the work still to be done:

- Algorithms and proofs have to be provided in the Fast Graph Transformations section.
- An example has to be provided for the Complexity of Graph Rewrite Systems section.
- The section Constant Time Graph Matching needs to be rewritten, at the moment this is basically a brain dump.
- The section Runtime Complexity of Rooted GP needs to be written.
- An example has to be provided for the Complexity of Graph Programs section.
- The article has to be reviewed thoroughly.
- The case study needs to be extended with more cases, especially with cases from normal GP.

4.3 Reflection

This was my first time doing research on an academic level, in this section I would like to reflect on some aspects of my work. In general it seemed Detlef was very happy with my work, and he had no concrete points of improvement for me to work on.

First of all there was no long-term plan of action for the project. I have done two internships in the past, and both of them had a somewhat predefined goal to work towards. Because of the nature of the work this was not really possible here, but in retrospect this ad-hoc approach worked quite well. The other side of the coin is that because there was no real plan, sometimes I was a bit lost, exploring solutions to problems which where actually out of scope. In retrospect it seemed Detlef had quite a clear idea on where he wanted to go with the research from the beginning, and if there is one point of improvement it would be to gather more information about the bigger picture at the start of a reseach project.

In terms of the process of researching and writing the article, I have found that these two things can not really be seen in seperation. At the start I had the idea of working out the details on paper, and then writing it down. In practice I noticed that I actually worked out the results while I wrote the article, because then I was really forced to do it right, and I actually noticed the same in the process of preparing my presentation.

The presentation itself was actually a very good idea, not only because is forced me to get a clear formulation of my ideas, but also because it presented me a deadline for doing this. I noticed that under the pressure of a deadline I work more efficiently than I do without, also when writing this report and completing the concept article. So a point of improvement might be to create more short-term deadlines to put a bit of pressure behind the work. One way of doing this might be to adopt a light-weight project management method such as SCRUM, while also involving my supervisor in this process.

During the internship I also got the opportunity to identify where my interests are. I am quite theoretically oriented, but this work was a bit too theoretical for my taste. This project in particular was more geared towards laying the foundations of further research, with a long way to go before any practical results will appear. In future projects I would also like to do some implementation work as I think this is one of my stronger points.

4.4 Conclusion

During my internship I learned a lot on many different subjects. First of all the process of doing research and writing an article, but also about some of the theory behind term rewriting, graph rewriting and complexity. Also I got more experience in doing typesetting work in Latex and I got the chance to improve my English skills.

In the time following my internship I am planning to complete the article, while maintaining regular communication with Detlef by email. In the end

I hope the article can be published, which should also be an interesting new experience for me.

Also I hope there will be further research on mechanized forms of termination proofs in graph rewriting, so that this research can be extended to automatically derive complexity bounds of graph rewrite systems.

5 Appendix A: Experience Report