Proceedings of CSAW’05

Ernest Cachia
University of Malta
ernest.cachia@um.edu.mt

Mark Micallef
University of Malta
mmica01@um.edu.mt

Abstract: This report contains the proceedings of the third Computer Science Annual Workshop (CSAW’05) — the research workshop held by the Department of Computer Science and AI of the University of Malta.
Preface

This is the third year that the annual Computer Science Annual Workshop (CSAW ’05) is being organised. Every year, the organisers try to improve the workshop in some way and this year it was decided that the workshop should reach out beyond the departmental boundaries. This is being done by means of the introduction of foreign speakers, speakers from prominent local companies and a government representative. In our opinion, such speakers can provide a stronger context for our research and also possibly inspire departmental researchers to take up projects which are relevant to the industries represented by the speakers and the business community in general.

Another addition to the workshop involves the introduction of an undergraduate short-paper competition whereby undergraduates can put an idea (founded on some basic research) to paper and present it in the workshop. This is meant to introduce undergraduate students to research at an early stage in their education.

We would like to thank departmental staff and research students who took time out of their summer to provide material for the workshop. Thanks are also due to GO Mobile for sponsoring the workshop, Gordon Pace, Joseph Cordina and Sandro Spina who organised the previous two workshops and who’s help proved invaluable for CSAW ’05, and ITSA who contributed their time to help the event in many ways (motivating undergraduates to participate, designing the front cover for the proceedings and looking for sponsors). Last but not least, we would like to thank the University of Strathclyde and Aalborg University for providing our two foreign keynote speakers.

Whilst hoping that your continuing interest will help improve the quality and reach of future CSAW events, we wish you an interesting and fruitful workshop.

September 2005

Ernest Cachia & Mark Micallef

CSAW ’05 Organisers
Table of Contents

A Framework for an Adaptive Virtual Learning Environment ........................................ 1
Ernest Cachia, Frankie Inguanez

Measuring the Functionality of Online Stores ................................................................. 3
Ernest Cachia and Mark Micallef

An Enterprise Integration Project Management Framework ............................................. 10
Ernest Cachia, Mark Vella

Source-level runtime validation through interval temporal logic .................................... 12
Karlston D’Emanuele, Gordon Pace

Semantically Annotate Everything? .............................................................................. 21
Matthew Montebello

Verifying Hardware Compilers ...................................................................................... 23
Gordon J. Pace, Koen Claessen

Describing and Verifying FFT circuits using SharpHDL ............................................. 31
Gordon J. Pace, Christine Vella

COLLEGE: A Collaborative On-Line Lecture Environment for Group and Individual
eLearning .................................................................................................................. 42
Chris Staff

Language Technology for eLearning ............................................................................. 46
Michael Rosner (Dept. CSAI, University of Malta)

SEMANTEXPLORER: A Semantic Web Browser ............................................................. 53
Simon Scerri, Charlie Abela, Matthew Montebello

Search Diversification Techniques for Grammatical Inference .................................... 62
Sandro Spina

Author Index ................................................................................................................. 67
A Framework for an Adaptive Virtual Learning Environment

Ernest Cachia and Frankie Inguanez
Software Engineering Process Improvement Research Group
Department of Computer Science and Artificial Intelligence
University of Malta
Malta

Abstract. This paper is actually an adapted M.Sc. proposal. Work on this degree has only just start so there are no results to be presented. However, during the CSAW presentation further details will be given about research problems and how they will be solved.

1 Research Problem

During the e-learning hype many Virtual Learning Environments (VLE) were developed and educational institutions started pushing forward towards offering E-Learning services. The problem that has been identified is that these environments are not meeting a good educational level, the majority of these solutions were being developed with the aim of rendering a profit. The solutions were filled with new technology features and gave less importance to the teaching techniques that a tutoring system should have.

A study has shown that 50% to 80% of students taking an online course tend to drop out before the actual termination of the course [2]. The speed to market, the focus on new technology and not on instructional design, boredom, lack of understanding of learning and teaching, and the lack of understanding of the unique teaching advantages of electronic media [1] can be identified as the main reason for which E-Learning is not being so successful.

When you analyse the learning process one identifies the following stages: a pupil forms a desire to acquire certain knowledge which a tutor possesses and is prepared to offer to the pupil; The tutor tries to deliver this knowledge to the pupil who is continuously assessed; The progress of the tuition is dependent on the learning progress of the pupil; The pupil acquires that knowledge when he/she proves to be able to solve a problem using the material and knowledge gathered during the tuition period.

Therefore a good VLE should, in my opinion, offer materials and support management services where tutors can manage the course material and offer support for such which students can acquire and use, offer research management services where students can research more on the course and finally offer assessment management services where the student is assessed on the his/her knowledge. With the use of these identified services the VLE should adapt the course delivery process so to give the ideal environment in which the student can acquire the desired knowledge.

This means that the way that the course is delivered to the student is dependant on the progress of the student. An essential part of the framework is the establishment of a standard for which any addition program interfacing with the framework should conform with. This standard would basically require that a foreign program should be able to related to the framework how much the student progresses in his/her understanding of a course by using the application.
Therefore a material delivery program would interface with the framework and report that the student received a certain level of knowledge through material. A research program would interface with the framework and report that the student was submitted to certain knowledge. An assessment program would interface with the framework and report how the student performed in solving a problem. With all this information gathered the framework would determine the suitable manner in how the course delivery should proceed.

References

Measuring the Functionality of Online Stores

Ernest Cachia and Mark Micallef
Software Engineering Process Improvement Research Group
Department of Computer Science and Artificial Intelligence
University of Malta
Malta

Abstract. This paper makes a case for the need of a framework which can be used to measure the functionality delivered by electronic commerce (e-commerce) systems. Such a framework would be helpful in areas such as cost prediction, effort estimation, and so on. The paper also goes on to propose such a framework, based on the tried and tested methods of function points [1] and object points [5].

Keywords: Software Metrification, e-Commerce, Function Points, Object Points, Software Quality Assurance

1 Introduction

Measuring the size of a system before it is even built has been an important issue in computer science ever since non-trivial systems started being developed. When the size of a system is known, other metrics can be calculated based on that size. A few examples are cost per unit, person months per unit or even errors per unit. Having a repository of projects with such metrics would help in estimating how much a system could cost (for example) before it is even built.

Initial attempts at measuring system size resulted in metrics revolving around lines of code (LOC). However, such metrics had two major flaws [7] [8]:

1. System size could not be measured before it was actually built
2. LOC metrics for the same system built with different programming languages can vary wildly

In the late 70s, Albrecht [1] proposed function points as a means of measuring the size of a system (later refined and extended by Arthur [2]). Since then, new application domains have been opened which Albrecht would not have considered (or even been aware of) at the time. Such application domains may encompass systems such as real time systems, embedded systems, e-commerce systems and so on. It was the advent of such new types of systems that led researchers to extend or modify function points according to the characteristics of the systems in question. A few examples include Jones’ Feature Points [3] for systems and engineering applications, Boeing’s 3D Function Points [4] for realtime systems, and Banker’s Object Points [5] for business and database application software. Although these extensions do suffice for measuring the functionality of their target domains, the same cannot be said for measuring the functionality of e-commerce systems as outlined in [10].

Given the increase in the popularity of online shopping [9], it has become an almost discounted fact for businesses to offer their products and services online. With so much activity in this area,
this paper proposes that a framework utilising methods similar to function points [1] and object points [5] be developed so as to provide an easy and effective way of measuring the functionality of e-commerce systems.

With regards to terminology, of the many definitions given to e-commerce, one of the broadest and most complete definitions is the one coined by the British government as “the exchange of information across electronic networks, at any stage in the supply chain, whether within an organisation, between businesses, between businesses and consumers, or between the public and private sectors, whether paid or unpaid” [6]. Throughout this paper, references to e-Commerce should be taken to imply a Business-to-Consumer (B2C) type model. This decision is mostly based on limiting the scope of our work to manageable chunks for the time-being.

2 The need for better gauging features of E-Commerce Systems

When Albrecht [1] developed function points, he naturally reasoned about systems as being a collection of inputs, outputs, inquiries, files and external interfaces. This was true then, and still is now. However, with the advent of new paradigms and system types, this may not always be the ideal way to describe a system. In 1994, Banker [5] developed a new methodology that allowed its users to reason about a system at a higher level of abstraction than they would if they used function points. Object points defined a system as being a series of screens and reports which operated on a number of tables in a database. Naturally, this is not necessarily true for all types systems. However, systems to which this reasoning applied could suddenly be evaluated much quicker.

In 2004, Cachia [10] showed that e-commerce systems differed significantly from systems of other types. So much so that he managed to identify and rank a number of quality attributes which were of particularly high importance to e-commerce systems. According to Cachia, e-commerce systems differ from other systems mainly because they are:

- content-driven
- exposed and vulnerable to security risks
- accessed through WWW browsers, thus limiting a programmer’s flexibility
- likely to have an enormous user base
- likely to change quite often

Based on these differences and a survey carried out amongst 350 online shoppers, Cachia identified the five most important quality attributes in e-commerce systems as being security, reliability, navigability, performance and portability.

Just as Banker [5] defined systems as being composed of a number of screens and reports, the authors of this paper are of the opinion that (based on Cachia’s [10] findings) e-commerce systems too warrant a framework that uses terminology tailored to their nature. One could (for example) refer to various e-commerce components as being online catalogues, shopping carts, and so on. This makes it less tedious for stake holders to carry out scientific cost and effort predictions and also reduces their tendency to simply ad-hoc it.

3 Components of Online Stores

The proposed framework will consider B2C e-commerce environments to be made up of the following types of components:
– Static Web Pages (welcome page, contact information page, etc)
– Dynamic Web Pages (online catalogue, registration form, etc)

A static web page is, put simply, a web page that is written once and constantly served to users as is. No matter how many users view such pages over and over again, they will always see the same content, layout, colours, etc. Static web pages have the advantage of being very fast, reliable and easy to create. However, they can be time consuming to update and it can be easy to have broken links and associated problems because a web-page or link was missed when they were updated [11]. It is worth mentioning that it is very difficult to measure the functionality of static pages since they are, in effect, simply content with embedded links to other pages.

Dynamic web pages on the other hand, are generated by the server each time a user requests them. They are able to process a number of parameters passed to them by the user’s browser and usually the resulting page is dependant on the nature of those parameters. Such capabilities have made it possible to show real-time information on websites, transform the WWW into an environment that can run applications and even offer the potential of personalisation. On the other hand, they are slow when compared to static web pages, require a much more powerful web server and are usually less reliable [11] since they contain program logic which may in turn contain errors. One must keep in mind that dynamic pages are usually supported by a number of backend services.

E-Commerce systems can also be said to be made up of a collection of objects of a higher abstraction. For example, one might examine a system and deduce that it is made up of an online catalogue, a product search page, an online ordering module and an online payment processing module. The proposed framework can also be used at this level of abstraction (see section 7) but at a core level, it will analyse static and dynamic pages. The reasoning behind this is that all the higher-level entities (online catalogue, product search page, etc) are in fact built out of pages.

4 Framework Overview

The proposed framework combines the ideas inherent in function point methodology [1][2] with Banker’s [5] idea of using terms of a higher abstraction than inputs, outputs, interfaces, etc when measuring a system’s functionality. Banker’s work also serves as a good starting point because it deals with business and database applications, a genre of applications which shares some characteristics with e-commerce systems. However, instead of using Banker’s terminology of screens and reports, the proposed framework will use such terms as static pages, dynamic pages, online catalogues, online ordering systems, and so on.

Calculating the size of the functionality of a system will be done in two steps. Firstly, a page-by-page analysis will be carried out. This will produce a so-called unadjusted count which is basically the sum of the points accrued by all individual pages in the system. However, this count must be adjusted according to certain criteria as discussed in section 6 below. This is necessary due to the fact that there may be certain system-wide requirements which are not countable at page-level but do in fact have an influence over the system-wide count.

5 The Unadjusted Count

During this phase, each page is individually analysed and assigned a count (or a number of points) according to the criteria defined below. The final unadjusted count is simply the summation of the counts of each individual page.
unadj\_count(system) = \sum_{i=1}^{n} unadj\_count(page_i)

5.1 Analysing Static Pages

Each static page is simply assigned a count of 1. This is because a static page is considered to be the simplest element in an e-commerce system which consists simply of content.

unadj\_count(static\_page) = 1

5.2 Analysing Dynamic Pages

Dynamic pages are assigned a count according to a number of different criteria which based on Cachia’s [10] findings, will affect the effort required to develop each individual page. Firstly, for each page, one needs to know the number of tables used or manipulated by that page. This reasoning is similar to the reasoning employed by Banker [5] and are also seen to e-commerce systems due to the fact that they are largely content-driven [10].

The second evaluated criteria requires the user to estimate/predict the expected popularity of the page in question. Some pages within the same site will be visited repeatedly by multiple users (e.g. a product search page) whilst others may not (e.g. a testimonial submission page). If a page is expected to be popular, more care must be taken to make the page’s internal workings efficient and able to cope with the repeated access, thus requiring more effort to develop.

If a page makes extensive use of dynamic features such as JavaScript\textsuperscript{TM}, VBScript\textsuperscript{TM} and so on, it would in effect be providing more functionality to the user. This is also reflected in the testing effort in terms of compatibility testing across browsers, operating systems and devices. [12][10]. Hence, the level of use of dynamic features within a page is also considered an important criteria in our analysis.

With the advent of powerful web and database servers, personalisation has become feasible and research in this field is very active. Levels of personalisation range from a page displaying the message “Hello Peter,” when you log in, to making personalised product recommendations, to changing the whole structure and content of pages to suite your user profile. It is the opinion of the authoers that the unadjusted count of personlised pages should be higher than those which are not, due to the fact that they offer more functionality.

Finally, we take the page’s visible algorithmic complexity into account. In some cases, what may seem like a simple form to the normal user may sometimes require vast amounts of processing in order to produce the required results. Hence, it is proposed that the unadjusted count for pages with a higher algorithmic complexity to be increased accordingly.

5.3 Points assigned to each criteria

The criteria defined above will be assigned points according to the following guidelines. The “amount of tables” criteria will be evaluated according to the table below:
The remaining criteria are to be assigned a value of *low*, *medium* or *high* by the person carrying out the calculations, and the unadjusted count incremented accordingly. If one considers an individual criteria as being non-existent, no increment should occur for that particular criteria.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>None</th>
<th>Low</th>
<th>Med</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected popularity of page</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Dynamic Features</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Personalisation</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Algorithmic Complexity</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

6 Calculating the Adjusted Count

The unadjusted count needs to be modified due to reasons explained in section 4 above. One can calculate the adjusted count by answering a number of questions about the presence of system-wide characteristics. Each question is assigned a relevance value between 0 (not important or applicable) and 5 (absolutely essential). At first glance, this is very similar to Arthur’s *complexity adjustment values* [2], however this is only true to a limited extent.

Firstly, following analysis of Arthur’s questions they were deemed to be inapplicable in the context of e-commerce systems. Also, Arthur’s formula can at most increase the function points of a system by 35% and the adjustments only increase the overall system count, never decrease. In some circumstances, the unadjusted count may need to be reduced. Finally, Arthur [2] assigns an equal weighting to the answer of each question whereas the proposed framework assigns different importance to different questions. For example, security considerations are considered to be more important than checking for dead links. Due to the very nature of e-commerce systems [10], the proposed framework poses different questions to those of Arthur [2] and also uses a different adjustment formula. The questions are as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Question</th>
<th>Max Adj.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Does the system use and/or manipulate complex data?</td>
<td>5%</td>
</tr>
<tr>
<td>2</td>
<td>Does the site in question require extensive security measures?</td>
<td>10%</td>
</tr>
<tr>
<td>3</td>
<td>Are components with the system being build in such a way as to allow them to be reused in other systems in future?</td>
<td>5%</td>
</tr>
<tr>
<td>4</td>
<td>Is the system meant to provide multilingual features?</td>
<td>5%</td>
</tr>
<tr>
<td>5</td>
<td>Is it important that the system produced personalised output and behaviour?</td>
<td>5%</td>
</tr>
<tr>
<td>6</td>
<td>Is the site expected to be popular and receive vast amounts of daily visitors?</td>
<td>5%</td>
</tr>
<tr>
<td>7</td>
<td>Are feature changes to the site planned after its release?</td>
<td>5%</td>
</tr>
</tbody>
</table>
Once the questions have been answered, the adjusted count can be calculated using the following formula:

\[ \text{AdjustedCount} = \text{UnadjCount} \times \text{adj\_factor} \]

where:

\[ \text{adj\_factor} = 100\% + \sum_{i=1}^{9} \left( \frac{A_i}{5} \times \text{MaxAdj}(Q_i) \right) \]

\[ - \left( \frac{A_{10}}{5} \times \text{MaxAdj}(Q_{10}) \right) \]

and:

\[ A_i = \text{Answer to question i} \]
\[ \text{MaxAdj}(Q_i) = \text{Maximum adjustment of question i} \]

Note that question 10 actually reduces the overall count because building the system out of reusable components would actually result in less effort being needed to develop it.

7 Quick Analysis

In some cases, it is simply not convenient (and may even be counter productive) to perform a detailed page-by-page analysis of a system. For this reason, the proposed framework also provides a way to perform a quicker (albeit less accurate) calculation. This is achieved by defining a number of higher level entities which one could analyse. For example, one could analyse a system as having 10 static content pages, an online catalogue, a product search page, and an online ordering page. The proposed framework provides a number of profiles whereby each high-level entity is pre-assigned the values for most criteria. So in the case of an online catalogue for example, the profile has a predefined popularity rating of high, a dynamic features rating of low, a personalisation rating of none, and an algorithmic complexity of none. With this information implemented in a software solution, estimates about a system’s size could be calculated very quickly. This can come in very handy when clients ask for a quick cost estimate of a system during meetings. These profiles will be published on the web shortly and a software solution is also under development.

8 Conclusions and future work

The proposed framework enables stake-holders in e-commerce systems to be able to make informed decisions about the future of a project. Its simplicity is such that it reduces motivation to use ad-hoc methods and can be seamlessly integrated with any software development process. The proposed framework can be applied to development effort prediction, cost prediction and even the prediction of testing efforts and bug-counts, much in the same way as function points.
However, the weights and values assigned to each criteria should be verified through extensive case-studies and industry interviews so as to refine them and reinforce their validity. Once this has been done, the proposed framework will form an integral part of a larger project which aims to provide a number of metrics, tools and methodologies for measuring the quality of e-commerce systems.

References

An Enterprise Integration Project Management Framework

Ernest Cachia and Mark Vella
Software Engineering Process Improvement Research Group
Department of Computer Science and Artificial Intelligence
University of Malta
Malta

Abstract. This paper is actually an adapted M.Sc. proposal. Work on this degree has only just start so there are no results to be presented. However, during the CSAW presentation further details will be given about research problems and how they will be solved.

1 Introduction

Software, is the brain behind today’s computerized world. From the number crunching applications running on main frames in the 70s, to today’s web applications driving massive data repositories, critical to the competitive advantage of an enterprise, software applications have gone a long way.

With many software systems still in use way beyond their expected lifetime, and automating nearly every aspect of an enterprise, these applications are victims of their own success, as the expectation from them could be more than they can handle. One modern approach taken to bridge this gap, is that of Enterprise Application Integration a fast growing software development approach, to aligning the existing independent vertical departmental software silos to the horizontal flows of business processes.

2 Research Question

Given the current available software project management methodologies or techniques for traditional software projects, and the feedback from the industry about the new nature and challenges project managers are encountering in EI projects during the first years of EI existence; is it possible to arrive to more rigorous EI project management framework, which could be proven to serve as the basis of more successful EI project life cycles, through the adoption of scientific identification of EI criteria and their subsequent classification and measurement of qualification attributes? If so, this scientific approach, would directly affect the quality of any associated deliverables in the applied business world.

3 Review of the relevant background research and theory to date

In his white paper for Wipro Technologies [3] makes a high-level overview of the EI challenges normally found throughout the life cycle of an EI project. [5] is a collection of EI principles from the EI methodology committee of the EI consortium, clearly showing the need of a more specific approach to EI rather than the available traditional methods. [4] and [2] present two attempts of presenting a complete project life cycle, specific for EI. [1] and [6] go further to propose attempts for process improvement and a capability maturity models for EI life cycles.
4 Tentative Research Procedure

The following is the research procedure as it is envisaged to proceed. Of course, this may be subject to change as different types of problems are encountered. One must keep in mind that this research procedure is based on a part-time commitment to the degree.

<table>
<thead>
<tr>
<th>Step</th>
<th>Timeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>What are the available traditional software management techniques? An indepth research of traditional software project management.</td>
<td>2 months</td>
</tr>
<tr>
<td>What unifies applications, and EI systems building blocks in general? A study on the nature of software, applications and more specifically business applications. An identification of the EI building blocks and their common attributes.</td>
<td>3 months</td>
</tr>
<tr>
<td>What is the purpose of EI? A research on the need of EI, the solutions delivered by EI, existent EI case studies analysis, and a forward look on to what EI is evolving to.</td>
<td>2 months</td>
</tr>
<tr>
<td>What are the new challenges imposed by EI? A study of EI project industry reports, as well as the first existent attempts of addressing these challenges</td>
<td>3 months</td>
</tr>
<tr>
<td>Why traditional Project methodologies fall short/are not enough for EI projects? A research of traditional project methodologies, and their analysis in an EI project context.</td>
<td>2 months</td>
</tr>
<tr>
<td>EI Management framework proposition Is it possible to bridge the gap between traditional software management methodologies and the new EI projects are posing?</td>
<td>4 months</td>
</tr>
<tr>
<td>Prove identified EI framework research results A proof of the results given in the previous step</td>
<td>2 months</td>
</tr>
<tr>
<td>How easy it is to build an automated management software tool with the identified EI management framework research results? Designing and building a software tool for helping project managers in the management of EI projects. Evaluation and testing of results.</td>
<td>6 months</td>
</tr>
</tbody>
</table>

References

Source-level runtime validation through interval temporal logic

Karlston D’Emanuele and Gordon Pace
University of Malta

Abstract. The high degree of software complexity achievable through current software development practices makes software more prone to failure. A number of work and work practices has evolved in order to reduce risks related to software correctness and reliability. One of which is validation, which monitors the system execution at runtime and verifies that the system states entered are valid according to the behavioural specification. This paper describes a framework providing an assertion like validation environment for integrating software properties specified in interval temporal logic. The framework consists in three parts. The first part provides a mechanism for converting the human readable assertion to a symbolic automata, which is then used by the second part of the framework that performs the validation.

1 Introduction

A common problem in software development or maintenance is that the software behaviour is not what expected. The main cause is that it is very difficult to map behavioural specifications in the system’s code. A solution can be to fully test the system but with the complexities in software that are being reached with current development techniques, it is infeasible.

Validation is another solution, which through logic based monitors the system is checked for correctness during runtime. Another important aspect of validation is that opposing to verification the system specifications do not need to be abstracted, especially if the monitors are placed local to the area effected. Validation monitors can be run synchronously with the system, in order to provide a more reliable solution for capturing errors on occurrence.

This paper presents a solution for integrating interval temporal logic inside a system core in order to perform runtime validation. The solution emphasises on guaranteeing that the memory and time necessary for the validation process can be estimated before commencing evaluation. These guarantees are attained by converting logical-based formulas into their equivalent Lustre [6] symbolic automata, where the memory and time required for evaluation are constants. Then a simple framework for the integration of the solution in a development environment is outlined.

The paper is organised as follows, in the next section quantified discrete duration calculus (QDDC) notation together with its semantics is introduced. Section 3 deals with the framework which provides validation mechanisms for integration into the system code. Finally concluding by mentioning some work that has been performed in the validations area.

2 QDDC notation

Quantified discrete duration calculus [9] is an interval temporal logic that checks satisfiability of properties within an interval rather than the more common techniques of sampling values or that of checking satisfiability at the end of intervals.
2.1 Syntax and Semantics

On the assumption that the system state has finite variability and that changes to the system state are performed sequentially, let \( \sigma \) be a non-empty sequence of state variables evaluations,

\[
\sigma \triangleq (\text{state variable} \mapsto \mathbb{B})^+.
\]

A state variable is a proposition and its syntax is defined as

\[
P ::= 0 \mid 1 \mid p \mid P \text{ op } P \mid \neg P \mid P + P
\]

where \( p \) is a proposition variable and \( \text{op} \in \{\land, \lor, \Rightarrow, \Leftrightarrow\} \).

The QDDC syntax is defined as,

\[
QDDC ::= \lceil P \rceil | \lceil P \rceil^0 | D_1 \hat{\text{op}} D_2 | D_1 \text{ b op } D_2 | \neg D | \exists p \cdot D | \eta c \text{ op } c | \Sigma P \text{ op } c
\]

where \( c \) is a constant, \( \text{b op} \in \{\land, \lor, \Rightarrow, \Leftrightarrow\} \) and \( c \text{ op} \in \{<, \leq, =, \geq, >\} \).

Let \( \sigma[b,e] \models D \) to denote that a finite non-empty sequence \( \sigma \) satisfies the QDDC formula \( D \) between the two discrete time clocks noted as \( b \) and \( e \). Leaving propositions to have their standard mathematical definition and \( \#(\sigma) \) denote the number of evaluations in the sequence \( \sigma \), then the QDDC notation semantics is as follows

\[
\begin{align*}
\sigma_i \models P & \iff \text{P is true at time clock } i. \\
\sigma_i \models \neg P & \iff i > 0 \land \sigma_{i-1} \models P. \\
\sigma_i \models P & \iff i < \#(\sigma) - 1 \land \sigma_{i+1} \models P. \\
\sigma[b,e] \models \lceil P \rceil & \iff \forall i \in [b,e) \cdot \sigma_i \models P. \\
\sigma[b,e] \models \lceil P \rceil^0 & \iff b = e \land \sigma_b \models P. \\
\sigma[b,e] \models D_1 \text{ b op } D_2 & \iff \sigma[b,e] \models D_1 \land \sigma[b,e] \models D_2. \\
\sigma[b,e] \models \neg D & \iff \sigma[b,e] \not\models D. \\
\sigma[b,e] \models \eta c \text{ op } c & \iff (e - b) c \text{ op } c. \\
\sigma[b,e] \models \Sigma P \text{ c op } c & \iff \sum_{i=b}^{e} \sigma_i \models P \\
\sigma[b,e] \models \exists p \cdot D & \iff \exists \sigma' \cdot \sigma'[b,e] \models D \text{ and } \\
& \forall i \in [b,e), \forall q \in P \cdot q \neq p \land \sigma'(q) = \sigma_i(q). \\
\sigma[b,e] \models D_1 \hat{\text{op}} D_2 & \iff \exists i \in [b,e) \cdot \sigma[b,i] \models D_1 \land \sigma[i,e] \models D_2
\end{align*}
\]

A number of derivable operators simplify the use of the notation. The table below defines four of the mostly used operators.
\( \circ D \equiv \text{true} \land \neg \circ \neg \text{true}. \)
\( \square D \equiv \neg \circ \neg D. \)
\( P \xrightarrow{\delta} Q \equiv \square ((P \land \eta \geq \delta) \Rightarrow ((\eta = \delta) \land Q)). \)
\( P \xleftarrow{\delta} Q \equiv \square (\neg P \land \eta < \delta) \Rightarrow Q). \)

### 2.2 Syntactic Sugar

In order to interweave the QDDC formulas within the code, it is necessary to define the formulas semantics. Gonnord et al. [5] showed that some QDDC operators are non-deterministic and hence have to be removed or substituted in order to be evaluated in runtime. Non-determinism arise from the use of the next evaluation in propositions, \(+P\), the existential (\(\exists p \cdot D\)), and the chop operator (\(D_1 \circ D_2\)). While the first two has to be complete removed since they are completely non-deterministic. While the chop operator is restricted to its wide used deterministic version, in other words, the subintervals are bounded with deterministic occurrences of events.

As with the QDDC notation propositions are used as the underlying driving mechanism. Let \(A_P(I)\) be the evaluation of \(P\) for the interval \(I\).

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>after((P, b))</td>
<td>Returns true if (P) was true at the start of the interval (denoted as (b)). Equivalent to ([P]^0 \land \neg \text{true}).</td>
</tr>
<tr>
<td>strict_after((P, b))</td>
<td>Returns true if (P) was true at the start of the interval. However, on the start of the interval it returns false. Equivalent to ([\text{true}]^0 \land \neg [P]^0 \land \neg \text{true}).</td>
</tr>
<tr>
<td>always_since((P, b))</td>
<td>Returns true if (P) has been constantly true for the interval, starting at clock cycle (b). Formally, ([P]^0).</td>
</tr>
<tr>
<td>nb_since((P, b))</td>
<td>Returns the number of occurrences of (P). In QDDC equivalent to (\Sigma P).</td>
</tr>
<tr>
<td>age((P, b))</td>
<td>Returns the number of times (P) was true from the last time it evaluated to false in the interval.</td>
</tr>
<tr>
<td>first((P, b))</td>
<td>Returns true on the first occurrence of (P).</td>
</tr>
</tbody>
</table>
Given the above methods, the fragment of QDDC that can evaluate to false only as time passes is defined as

\[
G = G_{G}(I) \land G_{A}(I) \land G_{F}(I) \land G_{E}(I)
\]

<table>
<thead>
<tr>
<th>(G)</th>
<th>(A_{G}(I))</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin()</td>
<td>after((A_{P}(I) \land b))</td>
</tr>
<tr>
<td>end()</td>
<td>after((b) \land \text{pre}(\text{always}<em>\text{since}(A</em>{P}(I), b)))</td>
</tr>
<tr>
<td>(\eta \leq c)</td>
<td>nb\text{since}(true, b) \leq c</td>
</tr>
<tr>
<td>(\Sigma P \leq c)</td>
<td>nb\text{since}(A_{P}(I), b) \leq c</td>
</tr>
<tr>
<td>(\text{age}(P) \leq c)</td>
<td>\text{age}(A_{P}(I), b) \leq c</td>
</tr>
<tr>
<td>(G_{1} \land G_{2})</td>
<td>(A_{G_{1}}(I) \land A_{G_{2}}(I))</td>
</tr>
<tr>
<td>(G_{1} \lor G_{2})</td>
<td>(A_{G_{1}}(I) \lor A_{G_{2}}(I))</td>
</tr>
</tbody>
</table>

Note the introduction of the QDDC operator \(\text{age}\). The operator is useful in order to provide alternatives to some of QDDC derived operators. For example, let \(\text{then}\) operator to be the concatenation of two subintervals, then \(P \delta \rightarrow Q\) is expressed equivalently as,

\[
P \delta \rightarrow Q \equiv \text{age}(P) < \delta \text{ then } P \land Q.
\]

Finally, the full fragment of QDDC that can be evaluated in runtime, is

<table>
<thead>
<tr>
<th>(F)</th>
<th>(A_{F}(I))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(G)</td>
<td>(A_{G}(I))</td>
</tr>
<tr>
<td>end()</td>
<td>after((b) \land A_{P}(I))</td>
</tr>
<tr>
<td>(F_{1} \land F_{2})</td>
<td>(A_{F_{1}}(I) \land A_{F_{2}}(I))</td>
</tr>
</tbody>
</table>
| \(\neg F\) | not \(A_{F}(I)\) |}

The \(\text{then}\) operator denotes the deterministic version of chop, which state that the entire interval is satisfied if on the first failure to satisfy the first subexpression, the second subexpression is immediately satisfied.

### 2.3 Examples

Before commencing with further formalisms, this section provides two simple examples of how systems can be expressed in QDDC formulas.

**Guess a number example.** Consider the simple guess a number game, where the number of attempts are limited. Whenever the user attempts to guess the number, he or she can either try a number smaller or higher than the target or the target number. The game can easily be specified using QDDC logic, as

\[
((\text{Less xor Greater}) \land \eta < \delta)^\ast \text{\(\Rightarrow\)} \text{\(\neg\text{Less and \neg Greater}\}}^0
\]

where delta is the number of attempts the user has.

The first subexpression of the chop operator specifies the condition that the user is trying to guess the number within the allowed attempts, while the second subexpression verifies that the user
guessed the number within the provided limit. The Kleene closure is used to allow the formula to be used repeated whenever a new game is started.

Now, let’s try to specify the same formula using the restricted logic to interweave it with the game code. Due to the constraints placed by the way a program is executed, a small modification is required. The modification is required since whenever the chop starts a new subinterval, the length of the entire interval is lost. Hence while leaving the limit of attempts to be checked programmatically and placing both Less and Greater variables are to true whenever the limit is exceeded to ensure that the chop operator fails, the formula to be integrated inside the code is:

\[
([\text{Less xor Greater}] \text{ then end(\neg\text{Less and } \neg\text{Greater}))}^*.
\]

**Memory allocation example.** Now consider a slightly more useful example. One of the major glitches in developing software in C/C++ is memory allocation. Freeing memory that was not allocated or already freed, or never freeing allocated memory give rise to misbehaviour in software which is very difficult to trigger or notice. Through the use of two simple QDDC formulas integrated with the memory de/allocation methods one can easily check for these problems.

Let \( P \leftrightarrow Q \) stand for \( P \) must hold one clock cycle before the occurrence of \( Q \). Given that the event of memory allocation is labelled as \( \text{Alloc} \) and that of memory deallocation labelled as \( \text{Free} \), then, the formulas required for the simple memory check program are,

\[
\Sigma\text{Alloc} < \Sigma\text{Free} \leftrightarrow \Sigma\text{Free}.
\]

(1)

\[
([\Sigma\text{Alloc} < \Sigma\text{Free}]^* [\Sigma\text{Alloc} = \Sigma\text{Free}])^*.
\]

(2)

Formula 1 is used to ensure that under no circumstances the number of freed memory exceeds the number of memory allocated. The second formula describes the behaviour in memory usage, that is, either some memory is still allocated or it is all free. The second subexpression in formula 2 ensures that at the end of program execution all memory has been freed. Defining these formulas for interweaving is straightforward.

\[
\Sigma\text{Free} \Rightarrow \Sigma\text{Alloc} \leq \Sigma\text{Free}.
\]

(\[
([\Sigma\text{Alloc} < \Sigma\text{Free}] \text{ then } [\Sigma\text{Alloc} = \Sigma\text{Free}])^*.
\]

It is important to note that more complex scenarios can be similarly handled using the restricted QDDC for integrating the formulas in the code.

### 3 Encoding QDDC syntax

Using the syntactics presented in Section 2.2 and Lustre representation for symbolic automata, in this section we present the solution adopted for encoding the restricted QDDC syntax. The solution is adopted from Gonnord et. al [5].

Consider it is required to evaluate the QDDC formula \([P]\). The formula is evaluated as \texttt{strict.after}(b) and \texttt{pre(always.since}(A_P(I), b)). In other words, we need to evaluate two methods because
the \texttt{pre()} is an operator over QDDC variables that returns the value they contained one clock cycle before.

First, let define the algorithms for the two methods. The \texttt{strict after(b)} states that when \( b \) is true the method returns false, but subsequently it must return true. Therefore, given that each different method invoked has a QDDC variable associated with it, for example \texttt{after(p)} has a variable named \texttt{after.p} to store its value history and on the assumption that all variables are initialised to false then,

\[
\texttt{strict after\_\textit{b}} = \texttt{false} \rightarrow (\texttt{pre(b)} \lor \texttt{pre(\texttt{strict after\_\textit{b}})}).
\]

The arrow sign after \texttt{false} is used to indicate that the variable \texttt{strict after\_\textit{b}} is initialised to false. One must also note that the or operator is lazily evaluated left-hand parameter first because on the first clock cycle the \texttt{pre()} might not be initialised.

As the \texttt{always since(p, b)} name suggests the method returns true if the variable parameter has been constantly true from the start of the interval.

\[
\texttt{always since\_\textit{P\_\textit{b}}} = \texttt{false} \rightarrow (\texttt{b} \lor \texttt{pre(\texttt{always since\_\textit{P\_\textit{b}}})}) \land \texttt{P}.
\]

Finally, the variable associated with the QDDC formula, \texttt{const\_\textit{P}}, is assigned the value,

\[
\texttt{const\_\textit{P}} = \texttt{strict after\_\textit{b}} \land \texttt{pre(\texttt{always since(P, b)})}.
\]

The use of the capital letter \( P \) within the algorithms is to indicate that the parameter value can either be a QDDC variable or an expression as in the case of evaluating \texttt{[Less xor Greater]} in the first example in Section 2.3. The methods in Section 2.2 are all evaluated using the same reasoning as in the above example-driven evaluation. For example, \texttt{after(P)} and \texttt{age(P, b)} are evaluated as:

\[
\texttt{after(P)} = \texttt{false} \rightarrow (\texttt{P} \lor \texttt{pre(\texttt{after(P)})}).
\]

\[
\texttt{nb since(P, b)} = \text{if \texttt{after(b)} and \texttt{P} then}
\]
\[
0 \rightarrow \texttt{pre(nb since(P, b))} + 1
\]
\[
\text{else}
\]
\[
0 \rightarrow \texttt{pre(nb since(P, b))}.
\]

\[
\texttt{age(P)} = \text{if \texttt{P} then}
\]
\[
0 \rightarrow \texttt{pre(age(P))} + 1
\]
\[
\text{else}
\]
\[
0.
\]
\[ \text{first}(P, b) = \text{if } \text{after}(b) \text{ then } \]
\[ P \land \neg(\text{strict after}(P)). \]

The basic operators in the algorithms above are the next operator, and the borrowed Lustre operators \( \text{pre}() \) and the initialisation operator \((\rightarrow)\). The latter consists in a simple initialisation process that assigns the formula variable, example \( \text{after}_P \), with the value preceding the operator.

Each variable that requires to use its previous value(s) is required to create additional memory space, the size of the variable data type, in order to store the old values. The process for keeping historical data regarding a variable is outlined in the algorithm below.

Variable assignment

Evaluate proposition

If keeping history

Assign variable old value to current value

Assign variable current value to the value returned in step 1.

In the assignment algorithm lines 2 and 3 ensure that if the variable is keeping history of its history, example \( \text{pre}(\text{pre}(P)) \), the algorithm is called recursively to update all history memory locations.

Being able to evaluate the restricted QDDC notation, the next operator is used to advance the basic clock by one clock cycle and perform the validation process outline below.

\[ \text{validate} = \forall \text{QDDC variables} \]
\[ \text{read data from the pointed memory location} \]
\[ \forall \text{QDDC formulas} \]
\[ \text{evaluate formula} \]
\[ \text{if formula fails then} \]
\[ \text{stop process and report error.} \]

4 Framework

The framework presented provides an assertion-like environment, which at user specified time intervals validates the system state with a set of interweaved QDDC formulas.

By leaving the user to perform the interweaving by placing the formulas as assertions within the code, the framework consists of three layers. The fundamental layer provides the validation engine that takes a set of automata representing the formulas and using the syntactic presented in Section 2.2, evaluates their satisfaction. To simplify the use of the engine, another layer is added on top of the engine. This layer allows the user to pass a well-formed QDDC formula and converts it into a symbolic automata, which is then fed to the underlying layer whenever the user requests the system to check its consistency. Finally in order to simplify the framework use and to abstract the user from the underlying system, an interface layer is provided.
**Engine.** The engine layer is responsible for validating the system with the formulas provided, as described in Section 3. The layer also provides a mechanism to report violations to the user because a formula might return false but will still be valid, like in the case of a `then` statement.

**Conversion** is achieved through the use of a parser, which checks for well-formed formula by analysing the string representing the formula with the grammar defined in Section 2.2. During the formula checking the parser also attempts to build a symbolic automaton for its representation. It is important to note out that since the conversion is performed at runtime, the symbolic automaton generated is not optimised but rather it consists of small automata concatenated together according to the formula.

### 5 Related work

The field of validation is gaining in importance, however, not a lot of work has been performed because the main focus is still on verification. Two of the major projects in the use of state variables for checking the system correctness are Bandera [1] and Temporal Rover [2, 4, 3].

The Bandera project is dedicated to the Java programming language and extracts a system model from the source code. The model extracted is minimised through the use of abstraction and slicing techniques. The project also provides a visualisation tool for checking the model for consistency, and allows the user to place behavioural constraints and check them through a simulated execution.

Bandera use of constraints over a model of the system is similar to our work. That is, both works use a description of the expected behaviour to check the system correctness. However, the main difference is that Bandera checks are performed over an abstracted model of the system that might not fit well in the system. While our framework performs the checks on the real system and during its execution. An advantage the Bandera project has over our solution is that the user is not required to be knowledgeable of how systems can be described formally. Nevertheless, our solution provides an environment that allows errors to be captured following the destined user feedback rather than using a pre-selected scenarios that the developer team thinks that might occur.

A major validation project is Temporal Rover by Time-Rover. Temporal Rover provides an assertion-based environment for the integrating specifications defined in temporal logic into programs [2, 4, 3]. The project consists in a precompiler that produces a file similar to the original with tangling code, representing the assertions and the validation engine.

Temporal Rover is far more powerful than the solution presented here. One of the advantages over our solution is that the system handles non-deterministic scenarios by creating instances for all possible solutions. Nevertheless, our solution lacking in non-determinism handling provides the user with a lightweight environment that reports errors as soon as they occur.

Another project is the integration of temporal assertions into a parallel debugger [8]. The parallel debugger project performs the validation by making use of a macrostep debugger, which while the program is being executed the assertions are checked for their validity. It is difficult to compare the project with our solution since there is lack of technical detail regarding the underlying work.

### 6 Conclusion and Future work

A major trouble in software development is verifying the system correctness. The concept of validation provides a solution for checking correctness during the execution. Through validation it
is also possible to trigger errors before they propagate and report their occurrence in more depth than it is possible with traditional error handling.

Using temporal logic formulas for defining the system specifications and allowing their integration into the system source code provides a robust environment for providing more reliable systems. The framework presented in this paper provides an environment for performing synchronous monitoring over system execution.

The grammar provided in Section 2.2 is enough expressive to handle the majority of QDDC formulas. Nonetheless, there are some commonly used QDDC expressions that require to be expressed in equivalent expressions. Since the framework does not handle optimisations and it is an inconvenience for the user, the framework can be enhanced to support common QDDC expressions directly.

One of the framework drawbacks is that formulas are converted at runtime since they are manually interweaved. Using an approach similar to that of aspect-oriented programming [7,11,10], where the formulas are supplied through a separate file and their observational interval defined through markups in the system code, the formulas can be converted through a pre-compiler which also handles the generation of tangled code.

Finally, another enhancement is to allow the validation to be performed asynchronously. This is fruitful in situations where immediate error reporting and handling is not critical and also in situations where time response during testing is important, especially if the framework is disabled in the system final version.

References
Semantically Annotate Everything?

Matthew Montebello

Software Engineering Process Improvement Research Group
Department of Computer Science and Artificial Intelligence
University of Malta
Malta

Abstract. Search as we know it will change and improve as new and enhanced methodologies to accommodate the ever expanding semantically annotated World-Wide Web will be developed and fruitfully employed. While a lot of basic infrastructure has already been developed to prepare for the onset of the Semantic Web, like meta-languages, editors, inference engines, and others, other critical developments need to be put in place too. This short paper discusses the issues as well as the best application areas to ‘Semantically Annotate Everything’ and transform on a mass scale the wealth of knowledge on the web into information as semantic metadata on the Semantic web.

1 Introduction

With the introduction of semantics in the second generation of the web as we know it, the Semantic Web, came the need to develop and make good use of tools and technologies in order to take advantage of such a richer resource. The Semantic web is strongly based on conceptual description of resources by means of semantic annotations, and the need to correctly map the available syntactic information onto a set of relevant conceptual entities is essential to correctly model the knowledge domain to which a resource belongs. This will have major repercussions primarily on searching as well as on related areas like indexing, retrieval and information extraction. The issue of describing all content semantically, thereby annotating everything over the World-Wide Web (WWW) is no simple task and so the need of upgrading the actual Web to the Semantic Web by means of automated semantic annotation strategies is very much desirable. The basic infrastructure has already been developed to prepare for the onset of this Semantic Web [1], like meta-languages (RDF [2], OWL [3]), editors [4], inference engines [5], and others, yet the main area that needs major input is the raw transformation of the mass of information over the existent WWW as well as the continuous flow of unstructured and semi-structured content appended automatically or humanly processed. The best path to the success of a technology is its flourishing use and universal adoption / application of the same technology. Any conceivable area within the existent WWW is a possible application area to semantically annotate the content and reveal the benefits and advantages such a technology can bring about, together with the functional strengthening and the intensified capabilities offered by existent technologies like search engines, information retrieval and data extraction / mining.

2 Conclusion

Such work is still in its infancy and plenty of work still needs to be done not just to get the technology perform up to its expectations, but also for its global acceptance and adoption. Several problems need still to be overcome among which stand out Ontology issues (mapping, versioning, instantiations), fully automatic annotation, meta-language expressivity and standardisation.
References

Abstract. The use of hardware compilers to generate complex circuits from a high-level description is becoming more and more prevalent in a variety of application areas. However, this introduces further risks as the compilation process may introduce errors in otherwise correct high-level descriptions of circuits. In this paper, we present techniques to enable the automatic verification of hardware compilers through the use of finite-state model checkers. We illustrate the use of these techniques on a simple regular expression hardware compiler and discuss how these techniques can be further developed and used on more complex hardware-description languages.

1 Introduction

The size, and level of complexity of hardware has increased dramatically over these past years. This has led to the acceptance of high-level hardware synthesis — allowing the compilation of program-like descriptions into hardware circuits [10]. As in the case of software compilers, correctness of synthesis tools is crucial.

Hardware description languages embedded in general purpose languages have been given proposed and used with various languages and areas of application [4, 1, 2, 7]. In [3], we proposed a framework in which different hardware synthesis languages can be combined and compiled together within the framework of a single embedded hardware description language, Lava [4]. The high-level synthesis languages provide a number of complex composition operators, the interaction of which can be difficult to understand and ensure the correctness of. The verification of the synthesis procedures proved to be tedious, and in some cases very difficult to demonstrate. When combining languages, this proved to be even more difficult since each language has its own underlying compilation invariants which need to be satisfied for the compilation to be correct.

Lava is linked to a number of model-checking tools, which one can use to verify properties via the use of synchronous observers. We propose a technique using which we can verify the correctness of our compilers using finite-state model checkers to verify the compilation techniques. Compositional compilation techniques are usually verified using structural induction over the language constructs, the individual cases of which usually turn out to be of a finite nature.

2 Circuit Descriptions in Lava

Circuit descriptions in Lava correspond to function definitions in Haskell. The Lava library provides primitive hardware components such as gates, multiplexors and delay components. We give a short introduction to Lava by example.

Here is an example of a description of a register. It contains a multiplexer, \texttt{mux}, and a delay component, \texttt{delay}. The delay component holds the state of the register and is initialised to \texttt{low}. 

Verifying Hardware Compilers

Gordon J. Pace and Koen Claessen

1 University of Malta
2 Chalmers University, Gothenburg, Sweden
setRegister (set, new) = now
    where
      old = delay low now
      now = mux (set, (old, new))

Note that setRegister is declared as a circuit with two inputs and one output. Note also that definitions of outputs (now) and possible local wires (old) are given in the where-part of the declaration.

After we have made a circuit description, we can simulate the circuit in Lava as a normal Haskell function. We can also generate VHDL or EDIF describing the circuit. It is possible to apply circuit transformations such as retiming, and to perform circuit analyses such as performance and timing analysis. Lava is connected to a number of formal verification tools, so we can also automatically prove properties about the circuits.

Should we want to decompose the multiplexor into more basic logic gates, one could define in terms of negation (inv), disjunction (<|>) and conjunction (<&>) gates:

mux (set, (case0, case1)) =
    (case0 <&> inv set) <|> (case1 <&> set)

2.1 Verification of Circuit Properties

Lava is connected through a number of model-checking tools which allow the verification of properties of circuits. To avoid introducing yet another formalism for property specification, An observer based approach as advocated in [5] is used to specify safety properties.

Given a circuit C, the property is specified using a separate circuit, called the observer reading the inputs and outputs of C, and outputting a single bit.

The circuit is then passed onto the model-checker to ensure that it outputs a constant high value.

For example, to check that the value in a register does not change if set is low, we can use the following observer:

checkRegister (set, new) = ok
    where
      current = setRegister (set, new)
      ok = inv set ==> (current <==> new)

Note that ==> and <=> denote boolean implication and equivalence respectively. To check that it always holds using external model-checkers from within the Lava environment:

Lava> verify checkRegister
    Proving: ... Valid.

Which allows us to conclude that ∀set, new · checkRegister (set, new).
2.2 Generic and Parametrized Circuit Definitions

We can use the one bit register to create an \( n \)-bit register array, by putting \( n \) registers together. In Lava, inputs which can be arbitrarily wide are represented by means of lists. A generic circuit, working for any number of inputs, can then be defined by recursion over the structure of this list.

\[
\text{setRegisterArray} \ (\text{set}, \ []) \ = \ []
\]
\[
\text{setRegisterArray} \ (\text{set}, \ \text{new}:\text{news}) \ = \ \text{val}:\text{vals}
\]

where
\[
\text{val} \ = \ \text{setRegister} \ (\text{set}, \ \text{new})
\]
\[
\text{vals} \ = \ \text{setRegisterArray} \ (\text{set}, \ \text{news})
\]

Note how we use pattern matching to distinguish the cases when the list is empty (\([]\)) and non-empty (\(x:xs\), where \(x\) is the first element in the list, and \(xs\) the rest).

Circuit descriptions can also be parametrized. For example, to create a circuit with \( n \) delay components in series, we introduce \( n \) as a parameter to the description.

\[
\text{delayN} \ 0 \ \text{inp} = \ \text{inp}
\]
\[
\text{delayN} \ n \ \text{inp} = \ \text{out}
\]

where
\[
\text{inp}' = \ \text{delay low} \ \text{inp}
\]
\[
\text{out} = \ \text{delayN} \ (n-1) \ \text{inp}'
\]

Again, we use pattern matching and recursion to define the circuit. Note that the parameter \( n \) is \emph{static}, meaning that it has to be known when we want to synthesise the circuit.

A parameter to a circuit does not have to be a number. For example, we can express circuit descriptions which take other circuits as parameters. We call these parametrized circuits \emph{connection patterns}. Other examples of parameters include truth tables, decision trees and state machine descriptions. In this paper, we will talk about circuit descriptions which take behavioural hardware descriptions, or \emph{programs}, as parameters.

2.3 Behavioural Descriptions as Objects

In order to parametrize the circuit definitions with behavioural descriptions, we have to embed a behavioural description language in Lava. We do this by declaring a Haskell datatype representing the syntax of the behavioural language. To illustrate the concepts with a small language, we will use simplified regular expressions without empty strings\(^3\). The syntax of regular expressions is expressed as a Haskell datatype:

\[
data \text{RegExp} = \text{Input} \ \text{Sig}
\]
\[
| \ Plus \ \text{RegExp}
\]
\[
| \ \text{RegExp} :+* : \text{RegExp}
\]
\[
| \ \text{RegExp} :-> : \text{RegExp}
\]

The data objects belonging to this type are interpreted as regular expressions with, for example, \(a(b + c)^+\) being expressed as:

\(^3\) This constraint can be relaxed, but it allows us to illustrate the concepts presented in this paper more effectively.
Input a :>: Plus (Input a :+: Input c)

Note that the variables a, b and c are of type Sig — they are signals provided by the programmer of the regular expression. They can either be outputs from another existing circuit, or be taken as extra parameters to the definition of a particular regular expression. We interpret the signal a being high as the character ‘a’ being present in the input.

Since regular expressions are now simply data objects, we can generate these expressions using Haskell programs. Thus, for example, we can define a power function for regular expressions:

```haskell
power 1 e = e
power n e = e :>: power (n-1) e
```

Similarly, regular expressions can be manipulated and modified. For example, a simple rewriting simplification can be defined as follows:

```haskell
simplify (Plus e :>: Plus e) =
  let e' = simplify e
  in e' :>: Plus e'

simplify (Plus (Plus e)) =
  simplify (Plus e)
...
```

3 Compiling Regular Expressions into Hardware Circuits

Following the approach presented in [8], it is quite easy to generate a circuit which accept only input strings which are admitted by a given regular expression. The circuits we generate will have one input start and one output match: when start is set to high, the circuit will start sampling the signals and set match to high when the sequence of signals from a received start signal until just before the current time is included in the language represented by the regular expression.

The type of the circuit is thus:

```
type Circuit = Signal Bool -> Signal Bool
```

The compilation of a regular expression is a function from regular expressions to circuits:

```
compile :: RE -> Circuit
```

Signal input

The regular expression Input a is matched if, and only if the signal a is high when the circuit is started.

```
compile (Input a) s = delay low (s <&> a)
```

Sequential composition

The regular expression e :>: f must start accepting expression e, and upon matching it, start trying to match expression f.

```
compile (e :>: f) start = match_f
  where
    match_e = compile e start
    match_f = compile f match_e
```
Loops
The circuit accepting regular expression \texttt{Plus e} is very similar to that accepting \texttt{e}, but it is restarted every time the inputs match \texttt{e}.

\[
\text{compile } (\texttt{Plus e}) \text{ start } = \text{match} \\
\text{where} \\
\text{start’ } = \text{start } \text{<|> match} \\
\text{match } = \text{compile e start’}
\]

Non-deterministic choice
The inputs match regular expression \texttt{e :+: f} exactly when they match expression \texttt{e} or \texttt{f}.

\[
\text{compile } (\texttt{e :+: f}) \text{ start } = \text{match} \\
\text{where} \\
\text{match}_\text{e } = \text{compile e start} \\
\text{match}_\text{f } = \text{compile f start} \\
\text{match } = \text{match}_\text{e } \text{<|> match}_\text{f}
\]

4 Model Checking Compilers

As we have seen in the regular expression example, we will be using embedded language techniques to represent programs as instances of a datatype:

\[
data \text{ Program } = \\
\text{ Variable } : = \text{ Expression} \\
| \text{ Program } : > \text{ Program } \quad \text{-- Sequential composition} \\
| \ldots
\]

In general, a synthesis procedure is nothing but a function from a program to a circuit:

\[
\text{compile } : : \text{ Program } \rightarrow (\text{Circuit}_\text{Ins } \rightarrow \text{Circuit}_\text{Outs})
\]

To reason about individual programs is no different from reasoning about circuits. For example the following function generates an observer to verify whether a given program satisfies a given property (with some constraints on the environment):

\[
\text{observer } (\text{environment}, \text{property}) \text{ program } = \\
\text{ins } \rightarrow \text{let outs } = \text{compile ins program} \\
\text{in } \text{environment } (\text{ins}, \text{outs}) \rightarrow \text{property } (\text{ins}, \text{outs})
\]

Similarly, we can compare programs by generating an appropriate observer:

\[
\text{p }===\text{ q } = \\
\text{ins } \rightarrow \text{let outs}_\text{p } = \text{compile ins p} \\
\text{outs}_\text{q } = \text{compile ins q} \\
\text{in } \text{outs}_\text{p } \leftrightarrow \text{outs}_\text{q}
\]
However, we can do more than this. We have identified a number of levels in which we can use model checking to reason about the synthesis procedure itself.

- We would like to be able to quantify over programs, to be able to write properties like:

```plaintext
forallPrograms $ \ e ->
forallPrograms $ \ f ->
forallPrograms $ \ g ->
e :> (f :> g) === (e :> f) :> g
```

To do this, we add a new component to the `Program` datatype, which represents a circuit:

```plaintext
data Program =
...
| Circuit (Circuit_Ins -> Circuit_Out)
```

We can now quantify over the outputs of the circuit to obtain an observer for the quantification:

```plaintext
forallPrograms fprogram =
\ (outs_c, ins) -> fprogram (Circuit (\ _ -> outs_c)) ins
```

Using this approach, we can verify various properties of our regular expression compilation function:

```plaintext
plusCommutative ins =
forallPrograms $ \ e ->
forallPrograms $ \ f ->
e :+: f === f :+: e
```

Lava> verify plusCommutative
Proving: ... Valid.

Using this technique, we managed to prove that the compilation procedure satisfies standard axioms of regular expressions, hence effectively verifying the compiler.

- However, most interesting language properties can only be proved using structural induction. It is usually impossible to prove properties of a program unless one assumes that the subcircuits satisfy these properties. This can be encoded inside the synthesis procedure, by adding an extra output which confirms whether the sub-components of the compiled circuit satisfy the invariant:

```plaintext
compile2 (p :+: q) invariant ins = (ok, outs)
where
  (ok_p, outs_p) = compile2 p invariant ins
  (ok_q, outs_q) = compile2 q invariant ins
  outs = combine (outs_p, outs_q)
  inv = invariant (ins, outs)
  ok = ok_p && ok_q ==> inv

compile2 (Circuit c) invariant ins = (invariant (ins, outs), outs)
where
  outs = c ins
```
To prove the invariant inductive case for an operator, it would suffice to prove the following observer holds:

\[
\text{proveAlt invariant } (\text{outs}_c1, \text{outs}_c2, \text{ins}) = \text{ok}
\]

where

\[
c1 = \text{Circuit } (\_ \rightarrow \text{outs}_c1) \\
c2 = \text{Circuit } (\_ \rightarrow \text{outs}_c2) \\
(\text{ok}, \_ ) = \text{compile2 } (c1 :+: c2) \text{ invariant } \text{ins}
\]

Similar inductive step observers can be constructed for the other language operators:

\[
\text{proveStructuralInduction invariant } (o1, o2, \text{ins}) = \text{ok}
\]

where

\[
\text{ok} = \text{ok1 <&> ok2 <&> ok3 <&> ok4}
\]

\[
\begin{align*}
\text{ok1} &= \text{proveSeq invariant } (o1, o2, \text{ins}) \quad :>:\n\text{ok2} &= \text{proveAlt invariant } (o1, o2, \text{ins}) \quad :+: \\
\text{ok3} &= \text{provePlus invariant } (o1, \text{ins}) \quad \text{-- Plus} \\
\text{ok4} &= \text{proveInput invariant } \text{ins} \quad \text{-- Input}
\end{align*}
\]

One such property we can prove only through the use of structural induction is:

\[
\text{noEmptyString } (\text{start}, \text{match}) = \text{start } \Rightarrow \text{ inv } \text{match}
\]

Lava> verify (proveStructuralInduction noEmptyString)
Proving: ... Valid.

This confirms all the cases of the structural induction, allowing us to confirm its truth for all regular expressions.

- One weakness with the above proof, is that if the sub-circuits ‘break’ for some time but then resume to work correctly, the top-level circuit is expected to resume correctly. This is a strong property which compilation procedures which encode some form of state usually fail to satisfy.

To strengthen induction to deal with this adequately we need to add temporal induction – assuming that the sub-components always worked correctly, the top-level component works as expected:

\[
\text{compile3 } (p :+: q) \text{ invariant } \text{ins} = (\text{inv}, \text{outs})
\]

where

\[
\begin{align*}
(\text{ok}_p, \text{outs}_p) &= \text{compile3 } p \text{ invariant } \text{ins} \\
(\text{ok}_q, \text{outs}_q) &= \text{compile3 } q \text{ invariant } \text{ins}
\end{align*}
\]

\[
\begin{align*}
\text{outs} &= \text{combine } (\text{outs}_p, \text{outs}_q) \\
\text{inv} &= \text{invariant } (\text{ins}, \text{outs})
\end{align*}
\]

\[
\text{ok} = \text{always } (\text{ok}_p <&> \text{ok}_q) \Rightarrow \text{inv}
\]

Where:

\[
\text{always } s = \text{ok}
\]

where

\[
\text{ok} = \text{delay high } (s <&> \text{ok})
\]

- Most compiler invariants assume that the environment satisfies certain conditions. At first sight, this could be expressed as \text{environment } \Rightarrow \text{property}. However, we run into the same problem that we had with sub-circuits breaking and then starting to work once again. The solution is to add an environment condition to the condition verified:
compile4 (p :+: q) ins = (ok, outs)
where
(ok_p, outs_p) = compile4 p ins
(ok_q, outs_q) = compile4 q ins
outs = combine (outs_p, outs_q)
inv = invariant (ins, outs)
env = environment (ins, outs)
env_p = environment (ins, outs_p)
env_q = environment (ins, outs_q)
ok = always (env <&> ok_p <&> ok_q)
     ==> (inv <&> env_p <&> env_q)

5 Conclusions

In this paper, we have outlined how finite state model-checkers can be used to verify properties of hardware compilers, and we have illustrated the use of these techniques on a simple regular expression compiler. Through the (external) use of structural induction, we decompose the general property into a number of finite state cases. This works on hardware compilers thanks to the fact that the size of the data path can be determined at compile time. In languages where this is not possible, our techniques clearly fail to work as presented. All related work we have identified use pen-and-pencil proofs (eg [6]) or theorem provers (eg [9]) to verify the correctness of the synthesis procedure. Our approach, although narrower in scope, has the distinct advantage of being a (relatively speaking) ‘push-button’ approach to compiler verification.

We plan to apply our techniques on more complex languages. In particular, we would like to investigate the correctness of Esterel compilation and standard Verilog and VHDL synthesis techniques. Furthermore, the use of these techniques on other generic and parametrised circuits can also prove to be effective.

References

Describing and Verifying FFT circuits using SharpHDL

Gordon J. Pace and Christine Vella
Department of Computer Science and AI, University of Malta

Abstract. Fourier transforms are critical in a variety of fields but in the past, they were rarely used in applications because of the big processing power required. However, the Cooley’s and Tukey’s development of the Fast Fourier Transform (FFT) vastly simplified this. A large number of FFT algorithms have been developed, amongst which are the radix-2 and the radix-2². These are the ones that have been mostly used for practical applications due to their simple structure with constant butterfly geometry. Most of the research to date for the implementation and benchmarking of FFT algorithms have been performed using general purpose processors, Digital Signal Processors (DSPs) and dedicated FFT processor ICs but as FPGAs have developed they have become a viable solution for computing FFTs. In this paper, SharpHDL, an object oriented HDL, will be used to implement the two mentioned FFT algorithms and test their equivalence.

1 Introduction

Embedded domain specific languages have been shown to be useful in various domains. One particular domain in which this approach has been applied is hardware description languages. SharpHDL is an example of such a language. It is embedded in C#, a modern object-oriented programming language, enabling us to describe structurally large regular circuits in an intuitive way. Its structure makes it easily extendible and helps the user produce clear and elegant circuit descriptions. These descriptions can then be used by Verilog simulators and SMV model checkers.

This paper introduces us to SharpHDL and describes how it is used to build and verify FFT circuits. Section 2 gives a brief overview of the SharpHDL syntax and some of its features. General information about FFTs and derivation of two FFT algorithms are given in section 3 while section 4 describes how the circuits of these algorithms are implemented using SharpHDL. It also describes how their equivalence was verified. Section 5 gives an account of some works related to this paper while section 6 draws up some conclusions.

2 SharpHDL — An Objected-Oriented Hardware Description Language

SharpHDL [3] is an HDL embedded in C#, an objected-oriented language. By embedded we mean that SharpHDL uses the same syntax, semantics and other tools belonging to C#. By objected-oriented we mean that ideas are expressed in terms of objects, which are entities that have a set of values and perform a set of operations [8].
2.1 Syntax Overview

As an introduction to the syntax of SharpHDL we see how a half-adder circuit is built. A half-adder accepts two inputs \( x \) and \( y \) and outputs the sum \( x \text{ XOR} y \) and the carry \( x \text{ OR} y \). Below, is the SharpHDL code for this circuit followed by a quick explanation.

```csharp
//<1>
using System;
using SharpHDL;

//<2>
public class HalfAdder: Logic {
    //<3>
    public HalfAdder(Logic parent): base(parent) {}
    //<4>
    protected override void DefineInterface(){...}
    //<5>
    public void gate_o(LogicWire in0, LogicWire in1, ref LogicWire sum, ref LogicWire carryOut)
    {
        //<6>
        Wire[] input = {in0, in1};
        Wire[] output = {sum, carryOut};
        this.Connect(input, output);
        //<7>
        new And(this).gate_o(in0, in1, ref carryOut);
        new Xor(this).gate_o(in0, in1, ref sum);
    }
    //<8>
    public LogicWire gate(LogicWire in0, LogicWire in1, out LogicWire sum){...}
}
```

To use the main C# functionalities we call the `System` namespace, which contains classes that implement basic functionalities. The `SharpHDL` library is also invoked so as to be able to make use of its facilities. **<2>** Since we are going to code a circuit, the `HalfAdder` class should inherit from the class `Logic`, an abstract class representing the general logic circuit. An instance of a class is initialized using a constructor which is defined in **<3>**. In `SharpHDL`, a user needs to specify the parent of the new circuit. By parent we mean the circuit to which the new circuit is a direct sub-circuit. Therefore, a half-adder circuit will be initialized as follows:

```csharp
HalfAdder ha = new HalfAdder(parent_circuit)
```

**<4>** The ports needed in the circuit are specified by overriding `DefineInterface`, which is a virtual method. Although this is an optional step it may be necessary when using the circuit as input to a generic circuit. **<5>** The behavior of the circuit is defined in the method `gate_o`\(^1\). It accepts four wires, two of which are the inputs and the last two are the outputs. The keyword `ref` indicates that the parameter is a Reference Parameter. In other words, the values are supplied by reference and therefore can be read and modified. **<6>** A compulsory step is to connect the wires to the respective ports. When the ports are not specified as explained in **<4>**, the ports are usually created automatically. **<7>** As already mentioned, the circuit needs two other circuits, a XOR gate and an AND gate. An instance of each is created by calling their respective constructor and specifying that the `HalfAdder` is the parent circuit by using the keyword `this`. The `gate_o` method of each circuit is invoked, passing to it the required parameters.

---

\(^1\) In `SharpHDL`, the name `gate_o` is a standard method name for the method that specifies the structure and behavior of the given circuit, given the necessary input and output wires. Likewise is the method `gate` (**<8>**), which, although it has the same function, it creates new output wires.
2.2 Using Generic Circuits

Generic Circuits are circuits which accept another circuit as input and use it to construct a more complex circuit with a regular structure. Hence, such circuits are called higher-order circuits [10]. SharpHDL provides a library of various generic circuits. This section briefly describes some of these circuits that will be referred to later.

- The generic circuit TwoN accepts a list of wires and recursively divides it in two for \( n \) times, applying the input circuit to the resultant lists.
- OneN is similar to TwoN but instead of applying the circuit to each resultant half, it applies it to the most bottom half only.
- A Butterfly circuit is another circuit that is built recursively. The input circuit is a component having two inputs and two outputs. After riffling the list, it uses the TwoN generic circuit to divide a given list of wires for \( \log_2 \text{size of list} \) times and applies the input circuit to the resultant subsets of wires. At the end, the list is unriffling. By riffling we mean shuffling the list such that the odd-indexed items are separated and followed by the even-indexed, while unriffling is the reverse operation.

3 Fast Fourier Transforms

The Fast Fourier Transform is one of the most important topics in Digital Signal Processing. It is a computationally efficient way to calculate the Discrete Fourier Transform (DFT) [2, 7]. A DFT’s problem involves the computation of the sequence \( \{X(k)\} \) of \( N \) complex numbers given a sequence of \( N \) \( \{x(n)\} \), according to the formula:

\[
X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} \quad 0 \leq k \leq N - 1
\]

where

\[
W_N^{kn} = e^{-j2\pi/N} = \cos(\frac{2\pi}{N} \cdot nk) - j \sin(\frac{2\pi}{N} \cdot nk)
\]

The latter is also known as the twiddle factor. When symmetric and periodicity properties of the twiddle factor are taken into consideration, an efficient computation of DFTs is carried out. The properties specify that

\[
W_N^{k+N} = -W_N^k \quad \text{(Symmetry property)}
\]
\[
W_N^{k+N} = W_N^k \quad \text{(Periodicity property)}
\]

FFT’s use these properties, making them efficient algorithms for computing DFT’s.
3.1 Radix-2 Decimation in Time FFT Algorithm

The Radix-2 DIT FFT algorithm [2] is the simplest and most written about form of the Cooley-Tukey algorithm. It works on an input sequence having length to the power of two. It splits the input into the odd-indexed numbers and the even-indexed numbers, hence making it a decimation-in-time algorithm. Therefore,

\[ f_1(n) = x(2n) \]

\[ f_2(n) = x(2n + 1) \quad n = 0, 1, \ldots, \frac{N}{2} - 1 \]

It follows that

\[ X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} \]

\[ = \sum_{n, \text{even}} x(n)W_N^{kn} + \sum_{n, \text{odd}} x(n)W_N^{kn} \]

\[ = \sum_{m=0}^{(\frac{N}{2})-1} x(2m)W_N^{2mk} + \sum_{m=0}^{(\frac{N}{2})-1} x(2m + 1)W_N^{k(2m+1)} \]

But \( W_N^\frac{N}{2} = W_\frac{N}{2} \)

Therefore,

\[ X(k) = \sum_{m=0}^{(\frac{N}{2})-1} f_1(m)W_\frac{N}{2}^{km} + W_N^k \sum_{m=0}^{(\frac{N}{2})-1} f_2(m)W_\frac{N}{2}^{km} \]

\[ = F_1(k) + W_N^k F_2(k) \quad k = 0, 1, \ldots, N - 1 \]

where \( F_1(k) \) and \( F_2(k) \) are the \( \frac{N}{2} \)-point DFTs of the sequences \( f_1(m) \) and \( f_2(m) \) respectively.

Using the symmetry property, we know that \( W_N^k = -W_N^{\frac{N}{2}} \). We also know that \( F_1(k) \) and \( F_2(k) \) are periodic, having period \( \frac{N}{2} \). Therefore,

\[ F_1(k + \frac{N}{2}) = F_1(k) \quad \text{and} \quad F_2(k + \frac{N}{2}) = F_2(k) \]

Hence,

\[ X(k) = F_1(k) + W_N^k F_2(k) \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \]

\[ X(k + \frac{N}{2}) = F_1(k) - W_N^k F_2(k) \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \]

This is known as the Radix-2 FFT Butterfly, better illustrated in Figure 1. In this diagram, multiplications are represented by numbers on the wires and crossing arrows are additions.

The decimation of the data is repeated recursively until the resulting sequences are of length two. Thus, each \( \frac{N}{2} \)-point DFT is computed by combining two \( \frac{N}{4} \)-point DFTs, each of which is computed using two \( \frac{N}{8} \)-point DFTs and so on. The network for this algorithm can be seen in Figure 2.

One should note that the indexes of the input sequence are re-ordered. The technique used is called Bit Reversal [1, 2, 9]. As it suggests, it basically involves the reversing of the bits such that the MSB becomes the LSB and vice-versa. Therefore, bit-reversed indexes are used to combine FFT stages.
### 3.2 Radix-2² Decimation in Time FFT Algorithm

The Radix-2² FFT Algorithm [4] is a less popular algorithm than the one described above. It is used by an input sequence of length to the power of 4, so that the N-point DFT formula can be broken down into four smaller DFTs.

Therefore, the DFT can be calculated as follows:

\[
X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn}
\]

\[
= \sum_{n=0}^{N/4-1} x(n)W_N^{kn} + \sum_{n=N/4}^{N/2-1} x(n)W_N^{kn} + \sum_{n=3N/4}^{3N/2-1} x(n)W_N^{kn} + \sum_{n=N}^{N-1} x(n)W_N^{kn}
\]

\[
= \sum_{n=0}^{N/4-1} x(n)W_N^{kn} + W_N^{N/2} \sum_{n=0}^{N/4-1} x(n+N/4)W_N^{kn}
\]

\[
+ W_N^{N/2} \sum_{n=0}^{N/4-1} x(n+N/4)W_N^{kn} + W_N^{3N/4} \sum_{n=0}^{N/4-1} x(n+3N/4)W_N^{kn}
\]

We know that

\[
W_N^{kn} = (-j)^k, \quad W_N^{kN} = (-1)^k, \quad W_N^{3kn} = (j)^k
\]
Hence,

\[ X(k) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) + (-j)^k x(n + \frac{N}{4}) + (-1)^k x(n + \frac{N}{2}) + (j)^k x(n + \frac{3N}{4})] W_N^{kn} \]

To get the radix-2\(^2\) decimation-in-frequency DFT, we subdivide the DFT sequence into four \(\frac{N}{4}\)-point sub-sequences:

\[ X(4k) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) + x(n + \frac{N}{4}) + x(n + \frac{N}{2}) + x(n + \frac{3N}{4})] W_N^{kn} W_N^{k\frac{N}{4}} \]

\[ X(4k + 1) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) - jx(n + \frac{N}{4}) - x(n + \frac{N}{2}) + jx(n + \frac{3N}{4})] W_N^{kn} W_N^{k\frac{N}{4}} \]

\[ X(4k + 2) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) - x(n + \frac{N}{4}) + x(n + \frac{N}{2}) - x(n + \frac{3N}{4})] W_N^{2kn} W_N^{k\frac{N}{4}} \]

\[ X(4k + 3) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) + jx(n + \frac{N}{4}) - x(n + \frac{N}{2}) - jx(n + \frac{3N}{4})] W_N^{3kn} W_N^{k\frac{N}{4}} \]

Note that the property \( W_N^{4kn} = W_N^{kn} \) is used. This procedure, illustrated in Figure 3, is repeated for \( \log_4 N \) times.

![Fig. 3. A Radix-2\(^2\) procedure](image)

The corresponding network for this algorithm can be seen in Figure 4. One can note that the output needs to be re-ordered using a bit-reversal permutation, as explained in the previous section.

### 4 FFTs in SharpHDL

This section explains how SharpHDL is used to implement FFT circuits.

FFT\(s\) work with complex numbers, therefore, a new type of wire \texttt{ComplexNumber} was created in SharpHDL which represents such numbers. \texttt{ComplexList} is a list object that accepts \texttt{ComplexNumber} objects.

Although being two different approaches to solving FFT\(s\), the algorithms described above have common components:
From the network designs one can easily notice the common butterfly operation, known as the FFTComponent in SharpHDL. This takes two inputs \( x_1 \) and \( x_2 \), and returns \( x_1 + x_2 \) and \( x_1 - x_2 \). This is the main operator of the algorithms as it calculates the 2-point DFT.

Another important operation is the twiddle factor multiplication. This component takes a complex number \( x \) and multiplies it with the \( W_k^N \) constant. The Radix-2\(^2\) FFT algorithm also uses multiplication with \(-j\) which is equal to the constant \( W_4^1 \).

Both algorithms use the bit reversal permutation, though in different times as described previously. This operation can be carried out by riffling the sequence recursively [10].

Using the above components and the generic circuits described in a previous section we can construct the circuits for the two FFT algorithms using SharpHDL. A Radix-2 FFT algorithm is described as follows:

```csharp
//BitReversal
int0 = (ComplexList)new BitReversal(this).gate(int0);

int exponent = (int)Math.Log(int0.Count, 2);
for (int i = 0; i < exponent; i++)
{
    int n = (exponent - i) - 1;
```
// Use TwoN to divide the list for n times and apply  
// the Radix2FFTStage to each resultant set  
ComplexList output0 = ComplexList)new TwoN(this,  
    new Radix2FFTStage(null), n).gate(in0);  
in0 = output0;  
}

where a \textbf{Radix2FFTStage} circuit is described as follows:

...

// Apply twiddle using OneN generic circuit  
BusWire one = new OneN((this, new FFT2Twiddle(null), 1).gate(in0);  

// Call Butterfly using the FFTComponent  
new Butterfly(this, new FFTComponent(null)).gate_o(one, ref outBus);  

A \textbf{Radix-2^2} FFT algorithm is described as follows:

...

int exponent = (int)Math.Log(in0.Count, 4);  
int n;  
for (int i = exponent; i > 0; i--)  
{
    n = (exponent - i)*2;  
    // Use TwoN to divide the list for n times and apply  
    // the Radix4FFTStage to each resultant set  
    ComplexList output0 = (ComplexList)new TwoN(this,  
        new Radix4FFTStage(null), n).gate(in0);  
    in0 = output0;  
}

// BitReversal  
new BitReversal(this).gate_o(twoN, ref output0);  

where the \textbf{Radix4FFTStage} circuit is described as follows:

...

// Use Butterfly generic circuit using the FFTComponent  
BusWire bflys = new Butterfly(this, new FFTComponent(null)).gate(in0);  

// Use OneN to multiply with -j  
BusWire ones = new OneN(this, new MinusJ(null), 2).gate(bflys);  

// Use TwoN to divide the list of wires and apply a Butterfly to it  
// The Butterfly circuit uses the FFTComponent  
ComplexList twoBflys = (ComplexList)new TwoN(this,  
    new Butterfly(null, new FFTComponent(null)), 1).gate(ones);  

// Multiply with twiddle constant  
new FFT4Twiddle(this).gate_o(twoBflys, ref out0);  

4.1 Verifying the equivalence of the FFT circuits using SMV

One of the tools offered by SharpHDL is that of converting a circuit description to SMV to verify
safety properties. SMV \cite{SMV} is a tool used for verifying that a finite system conforms to a set of CTL
specifications using symbolic model checking. Safety properties state that a condition is always true
or never false. Using this tool, we can verify that two circuits are equivalent, i.e. for any sequence
of input they generate the same output. Therefore, the equivalence relation $R$ can be defined as
Describing and Verifying FFT circuits using SharpHDL

\[ R = \lambda x, y. \forall z. (\gamma(x, z) = \gamma(y, z)) \]

where \( \gamma(x, z) \) is the function that determines the output from circuit \( x \) given input \( z \) [11]. Using SharpHDL, this can be done by constructing a Comparison Circuit which calls two circuits using the same input and compares the outputs. The circuit outputs true if all the outputs are equal [13].

The equivalence of the two FFT circuits can be verified using this methodology. The circuits are given the same input list of complex numbers and the two outputs \( \text{output}_{\text{radix}=2} \) and \( \text{output}_{\text{radix}=2^2} \) are compared. If they are all equal the FFT circuits are equivalent. On giving the generated SMV code of this circuit to the model checker, it proved that the FFT circuits of size 4 are equal.

4.2 Related Works on FFT circuits description and verification

The equivalence of two FFT algorithms using a Hardware Description Language has been shown using Lava in [9]. Lava is an HDL embedded in Haskell, a functional programming language. Although the programming paradigm used is different, the two algorithms and the verification approach are more or less the same as used in this paper.

5 Related Work

In this section we discuss some works related to other HDLs.

Synchronous programming languages provide tools that easily construct automatonsthat describe reactive systems, where the main tool they provide is the concept of synchronous concurrency. Lustre is one such language. Besides programming reactive systems, Lustre is used to describe hardware and express and verify their properties which makes it similar in concept to SharpHDL.

Lustre programs define their outputs as functions of their inputs. Each variable is defined using equations such that \( A = B \) means that the variable \( A \) is always equal to expression \( B \). Expressions consist of variable identifiers and constants, arithmetic, boolean and conditional operators and another two special operators: previous and followed-by operators. Taking \( E_n \) and \( F_n \) to denote the values of \( E \) and \( F \) at instant \( n \), the previous operator \( \text{pre}(E) \) returns the previous value of \( E \), such that \( (\text{pre}(E))_n = E_{n-1} \). The followed by operator is used to define initial values, such that \( (E \rightarrow F) \) is defined as follows:

\[
\begin{align*}
- (E \rightarrow F)_0 &= E_0 \\
- (E \rightarrow F)_n &= F_n \text{ for } n > 0
\end{align*}
\]

On the other hand, SharpHDL defines a circuit using instances of objects representing wires and circuits thus making the language a structural hardware description language. One must point out that the difference in concept and paradigm does not effect SharpHDL’s power or efficiency. Nevertheless, it should be noted that the various studies [12–14] conducted by Halbwachs et al concerning synchronous observers and the use of Lustre were of great help in understanding and
using verifications techniques in SharpHDL. Further use of such techniques in SharpHDL is already in the pipeline.

Another HDL with verification capabilities is the already-mentioned Haskell-based Lava [9, 10]. One of Lava’s high points is its simplicity and elegance, much aided by its functional-oriented host language. It also offers the facilities of connection patterns, which basically are the generic circuits found in SharpHDL. Another similar tool to SharpHDL is that a user can analyze circuits using both simulation and model checking.

Lava is used as an experimental tool for formal verification of hardware at Chalmers. It is also being successfully used in the development of FPGAs such as filter, Bezier curve drawing circuits and digital signal processing [10].

6 Conclusion

In this paper, we have described and made use of SharpHDL, an object-oriented HDL. We have seen how the object-oriented paradigm can be used to create a powerful HDL. This success can especially be seen in the easy implementation of generic circuits. The inheritance and polymorphism properties of OOP makes it possible for any generic circuit to accept any other circuit of the right structure, even itself, and build a bigger circuit, without the need to know what it actually does.

Another powerful tool discussed is the model-checker language SMV. Being able to convert SharpHDL descriptions into SMV allows the verification of circuit properties. This tool was used to verify that the circuits of two FFT algorithms are equivalent. The mathematical derivation of these two algorithms — the Radix-2 FFT algorithm and the Radix-2² FFT algorithm — were explained and then used to build their circuits using SharpHDL. The results proved that the algorithms produce the same results.

References

3. Christine Vella, SharpHDL: A Hardware Description Language Embedded in C#, University of Malta, Final Year Project, June 2004
7. I. S. Uzun, A. Amira & A. Bouridane, FPGA Implementations of FFT for real-time signal and image processing, School of Computer Science, The Queen's University of Belfast, BT7 1NN Belfast, United Kingdom.
COLLEGE: A Collaborative On-Line Lecture Environment for Group and Individual eLearning

Chris Staff
University of Malta,
Department of Computer Science and AI,
Malta

Abstract. COLLEGE is a platform for the development and delivery of interactive learning content for individual students or groups and will be built during 2005-2007. Phase I will deliver primarily video- and audio-based learning content together with tools to provide automated assistance and assessment of student progress. Phase II will increase the options for the learning content to include non-time-based media and will increase the level of Just-in-Time support for students. The COLLEGE toolset will be based around virtual metaphors corresponding to traditional tools for learning, recording, interacting with the source of the learning material, and assessment.

1 Introduction

Modern digital environments and affordable broadband Internet connections enable us to re-visit solutions for delivering video- and audio- based learning content over the Internet. We can now also provide automated assistance to students to construct individualised interactive learning experiences. These can also be extended to group and collaborative learning experiences.

COLLEGE, which is being developed at the University of Malta, builds upon work that has been on-going since 2001. Four BSc IT (Hons) Final Year students and one MSc student have researched and worked on various aspects of a collaborative on-line lecture environment and automatic question-answering system. In 2005, the University of Malta awarded funds to COLLEGE to engage a part-time developer and award a grant to an MSc studentship. The project has two phases. The first phase (2005-2006) will consist of combining existing work ([1], [2], [3], [4], and [5]) under a unifying project and evaluating the result. The second phase (2006-2007), if the funding is renewed, will involve rolling out learning material selected from a variety of certificate, diploma, and undergraduate degree programmes offered by the University of Malta. Further evaluation studies will be carried out, by comparing the progress of different students following the same course through COLLEGE and traditionally. Additionally, alternative non-time-based learning content material will be incorporated into the eLearning delivery platform to reach students who may not have high bandwidth access to the Internet; for learning content that is not best suited to audio- or video-based delivery; and to analyse alternative delivery methods.

2 Further details about COLLEGE

The initial motivation for COLLEGE and its predecessors was to recreate a virtual environment that would mimic a ‘real’ lecture environment. We wanted the lecture to follow a time-line, with lecture materials displayed to the student at the appropriate time. We also wanted to give students
the ability to interact with the lecturer or his/her proxy, including asking questions, but also taking advantage of the benefits offered by on-demand lecture service.

In COLLEGE, a lecture is primarily driven by a media stream that has a time-line (e.g., video or audio) against which events can be triggered to occur ([3]). A lecture can be subdivided into sections, and the designer of the lecture can decide how students may progress from one section to the next. For instance, although a student’s progress will normally be unhindered, it may be the case that a student is required to successfully perform some task in order to proceed. This could take the form of a pop quiz or question that the student must answer correctly. Through these and similar mechanisms it is also possible to monitor individual students’ progress in the course.

A lecture may be supported by visual aids, animations, on-line documents, lecture notes, study aids, etc., which are triggered to be made available to the learner at the appropriate time during the lecture.

![Fig. 1. The role of the lecturer in the classroom (left) is replaced by a video stream (right) (from [3]).](image)

The video stream takes on the lecturer’s role of delivering content and controlling the flow of the lecture (fig. 1). However, just as a student may be asked questions, so may a student ask a question. In an educational programme for which the content is relatively stable over a number of years (with different students passing through the course each time it is given), we expect that over time the number of previously unheard (or unseen) questions that students ask will become minimal. For instance, the first time a course is given all questions are previously unseen. Of the questions that are asked in the second year of the course, some may have been asked the previous year. Galea’s MaltaQA “stores” answers to new questions and then supplies the same answer to similarly phrased questions in the future [4].

Borg [2] converted Ellul’s proprietary lecture delivery platform to a Web-based platform using current W3C (World Wide Web Consortium) standards for synchronised multimedia (e.g., SMIL), and has extended it to support virtual classrooms that can contain students distributed, potentially, around the globe. In this case, a (digitally recorded) lecture is advertised for delivery at a particular time and students can register to virtually attend it. It is also possible for the lecturer to attend, in which case the lecturer may elect to take student questions himself or herself. At the advertised time, the lecture is broadcast to participating students. The toolset required to support these students is extended so that when a student asks a question, all participating students will hear (or read) the question. The lecture must then be suspended for all students so that the question can be answered. The lecturer has the choice of answering the question, asking the audience to provide the answer, or requiring all students to attempt to answer. Unlike non-interactive Internet radio which does not need to cater for listeners to be hearing the same thing at the same time, it is necessary to ensure that all students are synchronised with respect to each other, because otherwise students could be receiving questions which relate to a part of the lecture they have not yet seen, or which was seen several minutes earlier.

Bezzina’s PS4School [1] is a Just-in-Time Support System for Curriculum based Learning. This
eLearning platform assumes that the main learning environment is a real (or virtual) classroom, and that the student using PS4School is currently performing a learning task outside of the classroom. Students and tutors connect to PS4School. PS4School knows in which areas each tutor is proficient. A student who requires assistance is automatically connected to an on-line tutor who is in a position to help. PS4School uses policy mechanisms to determine the degree to which a student can seek help. Policies may also be used to ensure a fair distribution of requests for assistance among all on-line tutors who are capable of providing that specific assistance. Spiteri built an open source Web-based version of PS4School and extended the policy mechanisms [5].

3 Future Work

Phase I of the COLLEGE project is expected to run from October 2005 to September 2006. In this phase we aim to consolidate the work that has already been done in [1], [2], [3], [4], and [5], and make it available as a common eLearning platform for delivering video- and audio-based lectures to individual students or groups of students simultaneously. COLLEGE will provide an editor for the development of COLLEGE-based learning content, and toolkits for students to interact with a question-answering system, assessment exercises, the lecturer/tutor, the lecture itself, each other, and to seek and obtain assistance both inside and outside of the virtual classroom. A motivating short course will be developed for delivery through COLLEGE for evaluation purposes.

Phase II of COLLEGE will run from October 2006 to September 2007. COLLEGE will be further developed to include support for non-time-based learning material. Although time-based material, such as video and audio, provides a convenient time-line against which events (such as changing a visual aid) may be triggered, non-time-based material does not. Additionally, some students may not have broadband access to the internet, and some course material may be better suited to a non-time-based delivery mode. Students will still require just-in-time support, mechanisms for collaboration, and interaction with a question-answering system and the tutors, which can all continue to be provided as in Phase I. However, the delivery of the learning content will now centre around knowledge concepts to be learned, the interdependence of concepts, the mechanism for knowledge communication and transfer, and guiding students through a knowledge space consistent with their aims and experience.

4 Conclusion

The availability of relatively cheap broadband access to the Internet makes on-demand lecture delivery viable. COLLEGE will be an on-line environment that mimics a real lecture environment, with a toolset to enable students to participate in lectures either as individuals or as part of a group.

By the end of Phase II (September, 2007), we expect COLLEGE to be a robust, evaluated platform that can be used to develop and deliver University of Malta learning content to students who are unable to regularly physically attend classes; prefer to use alternative mechanisms to learn at their own pace; are resident in other countries; or require additional support to supplement their regular attendance.

References

Language Technology for eLearning

Michael Rosner

Department of Computer Science and AI
University of Malta
mike.rosner@um.edu.mt

Abstract. Given the huge amount of static and dynamic contents created for eLearning tasks, the major challenge for their wide use is to improve their retrieval and accessibility within Learning Management Systems.

This paper describes the LT4eL project which addresses this challenge by proposing Language Technology based functionalities to support semi-automatic metadata generation for the description of the learning objects, on the basis of a linguistic analysis of the content. Semantic knowledge will be integrated to enhance the management, distribution and retrieval of the learning material. We will employ ontologies, key elements in the architecture of the Semantic Web initiative, to structure the learning material within Learning Management Systems, by means of the descriptive metadata. We will also explore the use of Latent Semantic Indexing techniques for the matching of the learning objects with the user information requirements.

Acknowledgement and Disclaimer

1

1 Introduction

In the context of enlarged Europe, eLearning (including smooth exchange, transformation of contents) will play a central role in education, information exchange and life-long training.

Given the huge amount of high-quality content - teaching materials and courses - one of the major challenges for their wide use is to improve their retrieval and accessibility. This includes:

- production and dissemination of useful and standardized metadata which describes the contents adequately;
- multilingual information retrieval and access of the learning material;
- appropriate matching of well-defined learning need with the relevant content;
- description and retrieval of dynamic contents, i.e. contributions made by learners in an interactive learning environment (e.g. Contributions to fora, chat room discussions)

1 This article is a highly abbreviated and lightly edited version of project proposal no. 027391 LT4eL as submitted to Brussels under FP6 in March 2005. The project was accepted in July. Although the present author obviously had some role in the preparation of that proposal, the bulk of the text derives from others in the consortium listed in section 6. The present author therefore wishes to acknowledge their contribution and disclaim originality of authorship.

The project coordinator responsible for the original submission is Paola Monachesi (Paola.Monachesi@let.uu.nl) of the University of Utrecht.
Innovative solutions are needed in order to solve the problems which are beyond content production: raising wider awareness of the existence of these contents, personalising eLearning processes with adaptive Learning Management Systems (LMSs) and supporting active learning. Language Technology (LT) and the Semantic Web (SW) are areas which can offer effective solutions to these problems. They offer new technologies which open the possibility to integrate intelligent techniques in eLearning platforms and transcend the current situation in which “eLearning = e-versions of course materials + communication-tool”.

Significant research has been carried out in the area of LT and SW: the aim of this project is to enhance eLearning with these technologies in order to develop innovative applications for education and training. We will improve on open source LMSs by integrating the relevant technology. In particular, we will enhance the retrieval of static and dynamic learning objects by employing LT resources and tools for the semi-automatic generation of descriptive metadata while semantic knowledge will be integrated to enhance the management, distribution and search of the learning material.

The new functionalities will be integrated and validated within the ILIAS LMS, developed at the University of Cologne. ILIAS is an open-source web-based learning management system that allows users to create, edit and publish learning and teaching material in an integrated system with a normal web browser. However, we plan to adopt a modular approach so that the functionalities developed within the project will be compatible with different open source platforms.

Several initiatives have been launched within LT both at national and international level aiming at the development of resources and tools in the areas of Parsing, Tagging and Corpus Linguistics. However, their integration in enhancing eLearning platforms has not yet been fully exploited. It seems feasible to draw on existing and mature language processing techniques and to integrate them into LMSs. We believe that language resources and tools can be employed to facilitate tasks which are typically performed in a LMS such as searching for learning material in a multilingual environment, summarizing discussions in fora and chatrooms and generating glossary items or definitions of unknown terms.

Several tools and techniques are under development within the Semantic Web initiative which could play a significant role also within eLearning. In particular, ontologies can be employed to query and to navigate through the learning material which can improve the learning process. The topics which constitute the object of learning are linked to ontologies allowing for the creation of individualised courses. There is thus the possibility to develop a more dynamic learning environment with better access to specific learning objects. We will investigate the best way to include ontologies to structure and to query the learning material within an LMS. In particular, we will investigate the extent to which Latent Semantic Indexing (Landauer Foltz and Laham [2]), techniques could be employed for the matching of learning objects with the requirements of the user.

The starting point of this project were the workshops on Language Resources: Integration and Development in eLearning and in teaching Computational Linguistics (http://nats-www.informatik.uni-hamburg.de/view/Main/LrecWorkshop) and on eLearning for Computational Linguistics and Computational Linguistics for eLearning (http://www.cogsci.uni-osnabrueck.de/vreuer/ws_elearning_cl/) organized during the conference on Language Resources and Evaluation (LREC), and the conference on Computational Linguistics (COLING), respectively. They created the basis for the present collaboration which includes centres from Bulgaria, Czech Republic, Germany, Malta, Netherlands, Poland, Portugal, Romania, United Kingdom and Switzerland.

From these workshops it transpired that not enough attention has been dedicated, within eLearning, to the commercially less attractive languages (Bulgarian, Czech, Dutch, Maltese, Polish, Portuguese, Romanian) which are represented in the consortium: our project aims at filling this gap by

\[2\] The ILIAS project website is at http://www.ilias.de
integrating the expertise and the results gained for more commercially attractive languages such as German and English. The new functionalities introduced will be developed for all the languages represented in the consortium.

eLearning applications are very much an emerging field, and there are no standard, general methodologies that can be used to validate effectiveness of the learning process in our specific context. We will therefore develop a suitable validation methodology that will allow us to assess the enhancement of the ILIAS Learning Management System with the added functionalities.

The remaining parts of the paper are structured as follows. Section 2 discusses the main aims and objectives of the project. Sections 3 and 4 describes the implementation philosophy and individual work packages. Section 5 attempts to show how the division of labour is organised, whilst 6 mentions the other members of the consortium.

## 2 Aims and Objectives

The project will focus on the following scientific and technological objectives:

1. **Integration of LT resources and tools in eLearning:** the project will employ open source LT resources and tools, produced in the context of other projects, for the development of new functionalities that will allow the semi-automatic generation of metadata for the description of learning objects in a LMS.

2. **Integration of semantic knowledge in eLearning:** the project will integrate the use of ontologies, a key element in the SW architecture, to structure and retrieve the learning material within LMSs. Furthermore, it will investigate, through a pilot study, the use of Latent Semantic Indexing for the retrieval of the required learning object.

3. **Supporting multilinguality:** the project will support the multilingual character of enlarged Europe. Special attention is dedicated to the commercially less attractive languages which are represented in the consortium, such as Dutch, Polish, Portuguese, Czech, Romanian, Maltese and Bulgarian. The results already obtained for more studied languages such as German and English will be integrated. The new functionalities will be developed for the eight languages represented in the consortium.

4. **Development and validation of enhanced LMS:** the new functionalities will be integrated in the existing open source ILIAS Learning Management System and validated in a realistic learning environment. The developed prototype will be made available to the community and activities will be planned to stimulate its use within academia and schools.

5. **Expertise dissemination:** the project will stimulate information flow within and beyond the parties of the consortium by means of research and development activities, workshops and seminars, thus strengthening the integration of IST Research in Europe.

6. **Awareness raising:** the project will draw attention, within the eLearning community to the significant potential of Language Technology and emerging technologies such as the SW. Our aim is to bring together parties across discipline boundaries.

7. **Knowledge transfer:** the project will encourage the flow of knowledge from academia to industry. We wish to strengthen the cooperation with industrial partners in the area of eLearning and to encourage their participation in research activities through the organization of a user panel which will monitor the project throughout.

## 3 Implementation Principles

Contents for eLearning tasks are created either inside a learning management system, e.g. through built-in authoring tools, or outside a particular LMS. They will reside in a particular LMS or they
will be distributed on various platforms. Modern LMSs must therefore allow for import and export of contents. This includes not only static content, which is supplied by authors and does not change frequently, but also dynamic content which is generated by the participants through the learning process, ranging from short notes, contributions to discussion fora and emails to more elaborate texts such as homework assignments.

In this project, we will address one of the major problems users of ever expanding LMSs will be confronted with: how to retrieve learning content from an LMS. We want to tackle this problem from two different, but closely connected, perspectives: content and retrieval.

On the content side, the fast growing content cannot be easily identified in the absence of systematic metadata annotation. It should thus be common practice to supply metadata along with the contents, however this is a tedious activity which is not widely accepted by authors as part of their tasks.

The solution we offer is to provide LT based functionalities that take care of the metadata annotation semi-automatically on the basis of a linguistic analysis of the (static or dynamic) content. To our knowledge no such functionalities have been integrated in LMSs yet, which makes the project quite innovative. One other thing that makes our project unique is that we want to provide these functionalities for all the nine languages represented in our project. All partners will participate in the research and development of the new functionalities for their language by providing their resources, as well as their computational and eLearning expertise.

On the retrieval side, we can observe that standard retrieval systems based on keyword matching will only look at the queries and not at systematic relationships between the concepts denoted by the queries and other concepts that might be relevant for the user. In this project, we will use ontologies as an instrument to express and exploit such relationships, which should result in better search results and more sophisticated ways to navigate through the learning objects. An ontology of at least 1000 concepts for the relevant domain will be developed as well as an English vocabulary and English annotated learning objects. All partners will contribute to the development of the language vocabularies which will be linked to the ontology as well as to the annotation of the learning objects. We will implement multilingual retrieval of learning objects, focussing on the languages of the NMS and ACC and on the language families represented in the consortium (Romance, Germanic, Slavic).

### 4 Work Packages

In order to achieve the goals of the project, the following work packages have been envisaged:

#### 4.1 WP 1: Setting the scene

This involves making a survey of the state of the art in the fields which are relevant for the tasks in WP2 and WP3, in particular information extraction. The survey will cover published research as well as available systems (open source or ILIAS compatible) which perform these tasks at least partially. In addition, we will make a survey of the sort of information people are typically looking for in the domain under consideration. The main tasks are as follows:

- Survey of the state of the art for the fields relevant to WP2 and WP3 including information extraction, inventorisation and classification of existing tools and resources;
collection and normalization of the learning material in the area of information society technologies, dealing with the use of the computer in a particular application area to be defined, possibly business or humanities. We aim at a corpus of 200,000 words (i.e. 1000 pages) for each language. We will try to look for material dealing with the same topic in the various languages. IPR issues will be considered.

adoption of relevant standards (e.g. OLAC [1] and IMDI [3]) for linguistic annotation of learning objects;

development of a glossary for eLearning and LT terms to be used project-wide;

dissemination of the results through the Web portal;

4.2 WP 2: Semi-automatic metadata generation driven by LT resources

The aim of this workpackage is to improve the retrieval and accessibility of content through the identification of learning material using descriptive metadata. LT and resources for the languages addressed in the project will be employed to develop functionalities to facilitate the semi-automatic generation of metadata. Authors and managers of learning objects will be provided with a set of candidate keywords (for the keyword field). Terms and definitions will be detected and provided for glossary compilation as well as input to the ontology construction which is the main target of WP3. Here the tasks are:

Annotation of the training and test corpora for the languages covered;

definition of use cases;

implementation of new functionalities: keyword extractor (at least 1000 keywords), glossary candidate detector;

testing evaluation and feedback (2 cycles);

possible optimisation of the functionalities after testing;

documentation of the new functionalities.

4.3 WP 3: Enhancing eLearning with semantic knowledge

Ontologies will be adopted to structure, query and navigate through the learning objects which are part of the LMS.

Two groups of users will be taken into account: Educators and authors of teaching material who want to compile a course for a specific target group and who want to draw on existing texts, media etc., and learners who are looking for contents which suit their current needs, e.g. for self-guided learning.

The ontology will play two roles. (i) In the classification of learning objects, each learning object will be connected to a set of concepts in the ontology. This classification will allow ontological search, i.e. search based on concepts and their interrelations within the ontology. (ii) In multilingual search for learning objects, the ontology acts as an interlingua between the different languages. Thus the user might specify the query in one language and retrieve learning objects in other language(s).

The main tasks in this WP are

Definition of use cases;

domain ontology development based on existing resources;

creation of English vocabulary for the ontology;

ontological annotation of learning objects for the various languages;

creation of vocabularies for the various languages and their mapping to the ontology;

multilingual retrieval: evaluation of problems and tuning;
4.4 WP 4: Integration of the new functionalities within ILIAS

ILIAS does not provide semantic web based functionalities but it does offer the possibility of reusing learning objects like media objects or glossary items in the process of creating learning material. Ontology-based retrieval of learning objects will considerably improve the task of reusing learning objects since ontologies will allow for intelligent searching and navigation in huge amounts of data.

Metadata annotation and ontology-driven search and navigation will allow for individual content assemblance for learners. Learners will be able to build individual learning paths by entering key terms of concepts they need to learn.

The work package includes

- Technical integration of the functionalities and ontology tools into the ILIAS LMS;
- testing of integrated functionalities
- technical and user documentation
- optimization of functionalities after testing and validation

4.5 WP 5: Validation of new functionalities in ILIAS

The aim of this work package is to validate the extent to which LMSs are improved by adding new functionalities based on language technology tools and ontology-based retrieval. The objective is to assess the extent to which the integration of these new functionalities affects the effectiveness of the eLearning process. The work includes the following tasks:

- Development of a suitable validation methodology for eLearning;
- Preparation of experiments and questionnaires;
- Pilot experiments and questionnaires;
- Execution experiments and questionnaires;
- Report results.

5 Time Frame

In the table below, numbers have been rounded for the sake of readability.

<table>
<thead>
<tr>
<th>WP</th>
<th>WP Name</th>
<th>Man Months</th>
<th>Malta MM</th>
<th>Start</th>
<th>Finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inventory</td>
<td>14</td>
<td>1.2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Metadata Generation</td>
<td>77</td>
<td>7.5</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Semantic Retrieval</td>
<td>62</td>
<td>6.0</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>Integration</td>
<td>24</td>
<td>1.6</td>
<td>16</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>Validation</td>
<td>28</td>
<td>1.6</td>
<td>16</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>Dissemination</td>
<td>23</td>
<td>1.0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>Management</td>
<td>18</td>
<td>0.2</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td></td>
<td><strong>250</strong></td>
<td><strong>18</strong></td>
<td><strong>1</strong></td>
<td><strong>30</strong></td>
</tr>
</tbody>
</table>
6 Partners

1. University of Utrecht (Coordinator)
2. University of Hamburg
3. University of Iasi
4. University of Lisbon
5. Charles University, Prague
6. Bulgarian Academy of Sciences
7. University of Tübingen
8. Polish Academy of Sciences
9. Zürich Academy of Applied Sciences, Winterthur
10. University of Malta
11. University of Cologne
12. Open University, UK

7 Conclusion

In this paper we have given a brief survey of the assumptions, goals, methodology and implementation underlying the LT4el project. At the time of writing, the project lies in the future, so there is a sense in which any conclusion has to be speculative. However, it will start soon. The projected kickoff date is October 2005, and therefore the proof of the pudding will soon be in the eating.

Apart from the intrinsic goals as described above, it is worth mentioning that the project provides a great opportunity to begin the hard task of integrating LT with other areas of CS that are of interest to the Department.

From this author's perspective, one of the success criteria will be measured in terms of the degree of synergy with other eLearning projects that are either under way or about to start in the Department.

References

SEMANTEXPLORER: A Semantic Web Browser

Simon Scerri, Charlie Abela, and Matthew Montebello

Department of Computer Science and Artificial Intelligence
University of Malta
Malta

Abstract. The Semantic Web [17] will be the keystone in the creation of machine accessible domains of information scattered around the globe. All information on the World Wide Web will be semantically enhanced with metadata that makes sense to both human and intelligent information retrieval agents. For the Semantic Web to gain ground it is therefore very important that users are able to easily browse through such metadata. In line with such philosophy we are presenting semantExplorer, a Semantic Web Browser that enables metadata browsing, provides visualization of different levels of metadata detail and allows for the integration of multiple information sources to provide a more complex and complete view of Web resources.

Keywords: Semantic Web Browsing, Metadata, RDF Triple Store

1 Introduction

In the Semantic Web, classes of objects and their relationships are described in accessible Ontologies. In turn, resources in a Web document are defined as instances of the objects in the applicable Ontologies. Creating relationships between the resources is possible with the use of the Web Ontology Language [18], an Ontology Language that is built on top of the Resource Description Framework [13], the associated schema [14] and the eXtensible Markup Language (XML). The ultimate goal of the Semantic Web is to achieve a semantically enabled World Wide Web, by annotating online documents and services with semantic meaning. In this way it will be possible to relate between resources on the Web, thus making it easier for software agents to understand the content of the Web and ultimately for people to have better access to concept-oriented data.

Metadata Annotation is the process of attaching semantic descriptions to Web resources by linking them to a number of classes and properties defined in Ontologies. In general, metadata annotation methods fall under two categories: Internal and External annotation. Internal annotation involves embedding mark-up elements inside the HTML documents. On the other hand, external annotation involves storing the metadata in a separate location and providing a link from the HTML document. W3C recommends the use of external annotations [18]. Other methods are increasing in popularity, one of which promotes the inclusion of the external annotation reference within the HTTP response header.

With more RDF metadata being created, the need for persistent metadata storage and query facilities has emerged. The Semantic Web could enable structured searches for search engines and automated agents, given a large database to manage metadata efficiently. With such a database, related resources are easily connected, irrelevantly of their location and provider. RDF triple stores can be used to store RDF triples so that document metadata becomes more accessible. This would result in quicker and more efficient metadata querying. A number of experimental RDF triple stores have been set up, however none handle provenance, that is, the original source of the triples is not stored.
Currently there are two main approaches to creating Semantic Web Browsers [5]. A Semantic Web Browser has been described as a browser that explores the Semantic Web in its own right, and is able to relate and aggregate information about resources located in different Web documents. On the other hand a Semantic Web Browser can be described as a special Web browser, which augments standard Web browsers with the ability to visualize hidden metadata. Although quite a number of projects have tackled these approaches singularly, few have attempted to merge them together and develop an appropriate tool that can browse and visualize the Semantic Web at the same time. The aim behind this project is to create a tool in the form of a Resource Oriented Browser that will help with the visualization of hidden metadata layers associated with resources in a Web document, as well as aggregate information related to a singular resource of interest from different locations. The latter possibility needs to be based on a unique RDF triple store, which stores RDF triples for each accessed Web document.

The objective in this paper is to show how the two named approaches are bridged to achieve a Semantic Web Browser, which is able to:

- Access a required Web document and return the list of resources described within it, if any.
- Navigate from resource to resource, irrelevantly of the Web document they are defined in, as well as standard document to document navigation.
- Collect metadata from all visited locations and store it in an appropriate database for future use.
- Aggregate information related to a resource of interest from multiple locations, and displays it to the user.

The rest of this paper is divided as follows. In section 2 we provide some insight into the work carried out to bridge these techniques, where we discuss semantExplorer’s architecture and major modules. In the evaluation section we discuss the ability or inability to reach the set objectives. Section 4 presents a discussion and comparism of semantExplorer with similar current technologies, while ideas for future work are discussed in section 5, after which we give some concluding remarks.

2 System Overview

Figure 1 depicts the overall architecture of the semantExplorer. This includes the most important components, their interactions and the resulting output given to a user who is browsing the Semantic Web. The developed system is composed of four subsystems which we describe below.

The Navigation subsystem caters for document location, verification and accession or download. Navigation to particular URIs can be requested by the user. Contrary to standard web browsers, this subsystem provides a resource-oriented mechanism apart from the standard document-oriented navigation mechanism. Hence navigation requests can include locations pointing to resources defined through RDF data, and not just to web documents.

The Document Processing subsystem handles document processing and information extraction. When this subsystem receives a file, it is passed on to the RDF Extractor, which extracts any available RDF descriptions. The descriptions are then passed on to the parser to be transformed to a set of RDF triples and associated namespaces. Apart from being used by the Data Viewing subsystem, the generated triples are sent to the Cache Generator, and to the remote RDF triple store for storage. This is a unique, remote database, to which users can contribute freshly discovered RDF triples, or update them accordingly. The triple store caters for provenance, which is
The Data Viewing subsystem is responsible for all user output. After receiving the generated set of RDF triples, it creates a corresponding list of available resources, which the user can use to request information about some particular resource. In such a case, three different views are generated and presented to the user. The Table Builder gathers information in a document concerning the selected resource, and provides different and simplified ways of displaying it to the user. This data is presented as a set of properties and property values relevant to a resource. Users can navigate to resources that are connected to the selected resource. The Graph Builder processes information as in the table builder, with the difference that such information is displayed to the user as a colour-coded graph. This component also provides an option to extract further relevant background data. This is achieved by processing ontologies that are linked to the current document by namespaces. Data from these ontologies that is relevant to the resource of interest is attached to the basic data, to obtain a more detailed graph. Some basic reasoning is performed by one of the Data Viewing classes. Triple predicates are applicable to a domain and a range. For example, a predicate...
'isLocatedIn' could have a domain of 'Building' and a 'Place' as a range. The resource 'University-OfMalta' could be linked to 'Malta' by such a predicate. Although 'Malta' could be untyped and defined only as a datatype, it can be inferred that it represents a 'Place' by checking the range of 'isLocatedIn'. Although simple, this reasoning can enhance information about resources. The Lens Builder extracts data related to the resource of interest from the underlying triple store. These are then displayed to the user as a collection of 'lenses'. A lens can be described as a particular conceptual aspect of the required resource, which after being located can be 'focused'. Such lenses can give the user a broader vision of the resource of interest. The user can view each lens separately as a graph similar to the one generated by the graph builder. Users can navigate to any generated Lens, since in reality such lenses are nothing other than resources. Before displaying RDF triples in the three generated views, triples are shortened and simplified as required by the Triple Processor. Some triples are irrelevant to the average user and therefore the user is presented with the option to simplify the triple set before it is processed by the Table, Graph and Lens builders for output.

The User Options subsystem handles customizable user options that are retained when the user exits the application. This class library is also responsible for managing collections. The collector component saves a selected resource for future reference. When such a saved collection is selected, the system navigates to that resource and as a result, the table, graph and lens builders process and present the relevant data.

3 Evaluation of the system

In this section we describe the capabilities of semantExplorer through the use of an example scenario. We will consider the situation where a user visits the Web page associated with this project [15] through a standard Web browser. A lot of information would be available on the project, but information on concepts behind the various terms and titles in the page are not available, unless given explicitly in the standard Web content. If on the other hand, the visitor is an automated computer agent, it is probable that it would not make heads or tails of what the project is about. Nevertheless, semantExplorer’s first provided view is the 'Web Browser' view, showing standard Web content in standard format. In order for the average internet users to be introduced to the Semantic Web, we believe that viewing standard web content alongside its semantic context is crucial. Although the power of the Semantic Web is much greater than that of the Web, much headway has been made on the latter for it to be completely replaced by the former. However, hidden metadata associating terms in the page with concepts needs to be displayed to the user. Since these concepts are nothing other than resources, the user can request further information about them, or navigate to them. For the page in question, a list of annotated resources is drawn up and shown to the user. One such resource is 'DepartmentOfCSAI'. If the user clicks on this resource, the three available views display the relevant information.

The 'Item Description' view, Figure 2, shows data extracted from metadata in the page about the resource. A set of item properties and associated property values are listed. The user can navigate to any of the latter values. In this case, the presented data has not been processed and is not very readable. However the data could have been simplified if the user chose any number of available data fixes.

The 'Graph Viewer' view displays the data given by the 'Item Description' as a directed graph. Additionally this view can be used to extract more data from underlying ontologies that are referenced in the namespaces. This data is then attached to the basic set to get a more detailed graph.
Figure 3 shows the resulting graph output for the 'DepartmentOfCSAI' resource. Comparing this with the data seen in Figure 2, besides the fact that data is output in the form of a colour-coded graph, two other differences can be noted. First, the output has been simplified. Blank nodes within basic constructs, as well as references to namespaces, have been removed. In particular, the 'Bag' of objects included in Figure 2 is simply shown as a multi-object node in Figure 3. The other difference concerns the inclusion of a lot of extra information on concepts somehow related to the resource of interest. These are attached to the basic data using dashed arcs.
Finally the 'Lens Viewer' view responds to the selection by extracting a number of lenses related to the selected resource from the RDF Triple store. Lenses are resources which are directly or indirectly related to the selected resource. These lenses are categorized into lens categories. When a lens is selected, semantExplorer obtains the available information concerning this lens from the database and displays it to the user as a graph. The lens would be opening up on a particular conceptual aspect of the selected resource.

In our case, suppose that the user has navigated to the resource linked to the 'xmlns:cs#Department' from the 'Item Description'. All three views display information on the new resource 'Department'. The 'Lens Viewer' shows the three categories available to the user. The first contains a list of URLs containing data on the resource. The second contains other instances of the resource, that is, other resources defined as 'Department', while the third category contains a number of concepts related to the resource. The resource 'Entity' is in fact a super class of 'Department'. Figure 4 shows the resulting information after the user selected the resource 'Department' and 'Entity' lens. The user can decide to navigate to any lens, which in this case would trigger semantExplorer to navigate to the resource 'Entity'.

The 'Graph Viewer' and the 'Item Description' fulfil the Semantic Web browser approach to Semantic Web browsing. The 'Lens Viewer' caters for the Semantic Web browser facet of semantExplorer. Through these three views, semantExplorer enables Semantic Web data to be collected, visualized and browsed alongside the displaying and browsing of standard Web content.
4 Related work and comparisons

Existing projects and tools on Semantic Web Browsing have been given their due importance. A number of tools related to the subject are already available. Some of them are very basic, others are very promising and quite complex. Different applications take a different approach to such browsing and metadata visualization. The main ideas behind our research where taken from these projects. The following is a brief introduction to the tools that were most relevant to our work, and a consecutive comparism of these tools in relation to semantExplorer.

Magpie [8] is a tool that extends standard web browsers with the ability to visualize hidden metadata. It provides two methods for browsing [6]. The first provides browsing to physically linked content while the other method provides the semantic context of a particular resource. semantExplorer provides both these methods in its Semantic Web Browser facet. In fact, while browsing to physically linked content is provided by the 'Web Browser' view, the semantic context is available to the user through the given list of available resources. This context can be visualized by selecting a desired resource, upon which, the 'Item Description' and 'Graph Viewer' views will display the relevant data. The advantages of semantExplorer over Magpie are the following. semantExplorer can simplify semantic data to improve interpretability. Secondly, the 'Graph Viewer' can be set to extract further data from higher ontology levels, and in this way enhance the basic semantic data available for a resource in a Web page. Magpie provides trigger services called Magpie Collectors, which collect relevant data while the user is browsing. Alternatively, semantExplorer collects RDF data from all over the Semantic Web and sends it to a unique RDF store, which can be subsequently used by any instance of semantExplorer. On the downside, semantExplorer does not provide any semantic data annotation mechanisms, and it does not tackle Semantic Web services.

Brownsauce [4] is a Java servlet application for browsing generic RDF data using a web interface through HTML and CSS. It breaks the problem into Coarse-Graining, where data is broken down into usable chunks consisting of a set of triples relating a singular resource, and Aggregation, where data chunks relating an underlying resource from multiple sources are merged together. The latter feature however, has not yet been implemented. semantExplorer’s 'Item Description’ is basically an improvement over BrownSauce’s Coarse-Graining approach. The output given by the 'Item Description’ is in fact similar to that given by BrownSauce, with the difference that the latter does not cater for blank nodes and no data simplification options are available. The Aggregation problem proposed by the BrownSauce developers has been implemented in semantExplorer through the use of an RDF triple store as discussed in the previous sections.

Haystack [11] employs a Semantic Web Browser that browses the actual metadata and is not just an extension to visualize metadata in conventional URIs. A person’s haystack can be described as a repository of all information that the person has come across. RDF metadata concerning a resource from multiple locations is collected and the unified data is presented to the user after converting it to a human readable form. The user in turn can navigate from some piece of Semantic Web data to other related pieces. In this way, separate pieces of information about the same resource that used to require browsing through several different Web sites can be unified into one display. In semantExplorer we have adopted this strategy to achieve a Semantic Web browser. In fact, the 'Graph Viewer’ and 'Lens Viewer’ are both based on such ideas. The 'Graph Viewer’ attempts to gather further related semantic data than is originally associated with a URL, by gathering more information from ontologies whose definitions are being used to annotate data. In this way, the user is presented with unified information sets while being spared the tedious task of browsing to related resources to achieve better understanding of a resource of interest. Haystack’s approach to Semantic Web browsing is based on the notion of the Semantic Web being almost a completely different technology than the present Web. In fact, it is perhaps too complex for the average internet user to consider using it instead of standard Web browsers. semantExplorer is better designed to facilitate
such ease of use by users, while integrating the key innovative ideas presented by the Haystack project. The latter has introduced the concept of Lenses, which was subsequently adopted in the design of semantExplorer. A Lens in Haystack is defined as a list of properties that make sense being shown together, and is like a window opening up on certain aspects of a resource of interest, displaying a list of appropriate properties. Similarly, semantExplorer generates Lenses by aggregating information on a resource by querying the RDF triple store. A number of Lenses are in this way generated within four possible Lens Categories. The first contains a number of URLs that contain semantic information directly related to the resource of interest. The second category contains a number of resources similar to the one of interest. For example, if the user requested information about an object whose type is ‘Student’, this category will display a list of other instances of the class ‘Student’. The third category displays a number of definitions related to the chosen resource. In the previous example scenario, this category could yield the Lenses ‘Student’, ‘Person’, ‘UnderGraduateStudent’ and ‘PostGraduateStudent’. The fourth category is based on the ‘rdfs:seeAlso’ property, and URIs defined to be related to the selected resource will be included within. When the user selects one of the generated Lenses within any of these categories, the information gathered from the RDF triple store is conveniently merged and displayed as a colour-coded graph. Another notion adopted by semantExplorer from Haystack is the idea of Collections. In both applications, Collections are the Semantic Web browsers equivalent to the standard Web browser’s Favourites. When a user selects a previously saved Collection, all semantExplorer’s views will focus on that one specific resource. Haystack provides alternative naming schemes other than URLs, and it is based on presentation Recommendations, themselves defined in RDF. At this stage, these ideas where deemed unnecessary for semantExplorer. Contrary to the latter, Haystack also caters for Semantic Web Services.

Piggy-Bank [16] is an extension to the Mozilla Firefox browser, which enables the collection and browsing of Semantic Web data linked from ordinary Web pages. Users can collect and save useful metadata from within the browser pages which in turn can be browsed and searched through an appropriate built-in faceted browser. Piggy-Bank is similar in principle to the Magpie tool. In effect, the same ideas where used in the implementation of our Semantic Web browser. A basic difference in both Piggy-Bank and Magpie vis-à-vis semantExplorer is that while the former two are extensions to standard Web browsers, semantExplorer is a singular tool with its own independent Web browser, providing Semantic Web browsing and mechanisms for the gathering, simplification, integration and visualization of metadata.

5 Future Work

A number of ideas for future work have been brought up at the end of this project. Various components in the application can be improved to significantly make them more efficient.

In particular, the Lens Viewer could be extended to fully extract data from a bottom-up triple search. Although the system handles document and parses result caching, this cache is very primitive. Proper caching mechanisms should be developed.

A component which could be included in the system is an RDF Reasoner. The Graph Viewer does provide some basic reasoning. Although simple, this reasoning infers some indirect information about resource, which would otherwise have been missed. With a full-fledged reasoner, data about resources could be significantly enriched for the benefit of the user.

Another idea involves creating a stand-alone Graph Viewer plug-in for standard web browsers. The Graph Viewer is the focal point of the whole application, and it is the component which provides the most important and easily interpretable data visualization. Since the Graph Viewer and
Item Descriptor are intrinsically very similar, these two components can be integrated into one, resulting in a Graph Viewer which is also able to navigate to the resources it displays. In this way, all the powerful functions implemented in the Semantic Web Browser facet of this project, could be implemented into a single plug-in application that can be attached to a standard Web browser. This would augment such browsers with the ability to show hidden metadata and browse to related resources.

6 Conclusion

The idea to merge the two Semantic Web Browsing approaches was successfully realized. The integration of these two approaches, together with the useful external components and a suitable resource oriented navigation mechanism, resulted in semantExplorer: a Semantic Web Browser. Our browser can be useful both to Semantic Web beginners, to help them learn about the potential of this new generation of the Web, as well as to Semantic Web developers, to help them visualize, analyse and integrate the metadata they are annotating or working with.

References

7. Graph Visualization Software, (Online), Available from: http://www.graphviz.org/
9. MbUnit, QuickGraph - Generic Graph Library for .NET, (Online), Available from: http://mbunit.tigris.org/
11. Haystack Project, (Online), Available from: http://haystack.lcs.mit.edu/
14. RDFS, 2003, RDF Vocabulary Description Language 1.0: RDF Schema, (Online), Available from: http://www.w3.org/TR/rdf-schema/
Search Diversification Techniques for Grammatical Inference

Sandro Spina
Department of Computer Science and Artificial Intelligence
University of Malta

Abstract. Grammatical Inference (GI) addresses the problem of learning a grammar \( G \), from a finite set of strings generated by \( G \). By using GI techniques we want to be able to learn relations between syntactically structured sequences. This process of inferring the target grammar \( G \) can easily be posed as a search problem through a lattice of possible solutions. The vast majority of research being carried out in this area focuses on non-monotonic searches, i.e. use the same heuristic function to perform a depth first search into the lattice until a hypothesis is chosen. EDSM [?] and S-EDSM [6] are prime examples of this technique. In this paper we discuss the introduction of diversification into our search space [5]. By introducing diversification through pairwise incompatible merges, we traverse multiple disjoint paths in the search lattice and obtain better results for the inference process.

1 Introduction

Grammatical Inference (GI) is an instance of inductive inference and can be described as the algorithmic process of discovering syntactic patterns from a corpus of training data. GI addresses the following problem:

**Given a finite set of strings that belong to some unknown formal language \( L \), and possibly a finite set of strings that do not belong to \( L \), we require a learning algorithm that infers \( L \).**

Machine learning of grammars finds a variety of applications in syntactic pattern recognition, diagnosis, computational biology, systems modeling, prediction, natural language acquisition, data mining and knowledge discovery.

One of the main challenges in DFA learning algorithms is that of learning grammars from as sparse as possible training set of that particular grammar. In real-life scenarios the amount of data available for the learning task might be very limited. Both the Evidence Driven State Merging (EDSM) [?] and the Shared Evidence Driven State Merging (S-EDSM) [6, 1] heuristic aim at improving target convergence when using sparse data. Sharing the evidence gathered from different potential merges has in effect improved the convergence rate of grammatical inference. The interested reader is referred to [6] for a detailed description of how the S-EDSM heuristic works.

A number of DFA learning tasks still remain very difficult to solve. This difficulty measure depends both on the size of the target automata and the size of the training set. Figure 1 refers to the current Gowachin map for DFA problems \(^1\). The graph plots the size of the target automata against the number of training strings. The sparser the training set the more difficult it is for a learning algorithm to identify the target grammar.

\(^1\) http://www.irisa.fr/Gowachin/
Search Diversification Techniques for Grammatical Inference

2 Search Diversification

This section discusses the motivations behind our introduction of search diversification techniques. It is being assumed that the reader is already familiar with the definitions and results in set theory and formal languages, as well as the area of DFA learning in particular state merging algorithms. If this is not the case the interested reader is referred to [3] and [6] for an exposure to formal languages and grammatical inference respectively. We shall start by defining what state compatibility means followed by pairwise compatibility of merges. We then explore how search diversification can be obtained by taking into account pairwise incompatibility of merges.

2.1 State Compatibility and Merges

States in a hypothesis DFA are either unlabeled or labeled as accepting or rejecting. Two state labels \(A, B\) are compatible in all cases except when, \(A\) is accepting and \(B\) is rejecting, or, \(A\) is rejecting and \(B\) is accepting. Two states are state compatible if they have compatible labels. The set of all possible merges is divided between the set of valid merges, \(M_V\), and that of invalid merges, \(M_I\). A valid merge is defined in terms of the transition trees mapping operation [6] as follows:

**Definition 1 (Valid Merge)** A valid merge \(M_V\) in a hypothesis DFA \(H\) is defined as \((q, q')\), where \(q\) and \(q'\) are the states being merged, such that, the mapping of \(q'\) onto \(q\) results in a state partition \(\pi\) of \(H\), with a number of blocks \(b\), such that for each block \(b \in \pi\), all states in \(b\) are state compatible with each other.

Choosing which states to merge next depends on the heuristic adopted. Both EDSM and S-EDSM searches for a target DFA within a lattice of hypotheses (automata) enclosed between the augmented prefix tree acceptor (APTA) and the Universal Acceptor Automaton (UA) [?]. It is assumed that the target DFA lies in the search space of EDSM. It therefore follows, that at least one sequence of merges exists that will lead to the target DFA.

With search diversification we explore more than one sequence of merges. Our goal is to order these sequences in such a way such that the optimal one (i.e. the one producing the target grammar) is produced in the first few sequences.
2.2 Diversification through Pairwise Incompatible Merges

The most basic interaction between two merges is pairwise compatibility. Merge $M$ is said to be pairwise compatible to merge $M'$ if after performing merge $M$, $M'$ remains a valid merge in the hypothesis automaton as changed by $M$. More formally two merges are pairwise compatible, if the following property holds:

**Definition 2 (Pairwise Compatible)** Let $\pi_1$ and $\pi_2$ be the state partitions resulting from the application of the map operator to the two merges $M_1$ and $M_2$ on hypothesis $H$. Let $H_1$ and $H_2$ be the hypotheses resulting from $\pi_1$, $\pi_2$ respectively. $M_1$ and $M_2$ are pairwise compatible if for each state $s \in H$, $s \in H_1$ is state compatible with $s \in H_2$.

Suppose we have a set $S$ of high scoring merges generated by the EDSM or S-EDSM algorithms. Each merge $M$ in $S$ might not necessarily be pairwise compatible with all the other merges in the same set $S$. Let merges $M_1$ and $M_2$ be two pairwise incompatible merges from the set of merges $S$. As a result of their incompatibility we know that the merge path followed after $M_1$ is carried out in the DFA lattice can never cross the merge path which follows $M_2$. Thus we can state that pairwise incompatible merges guarantee diversification of the search paths.

![Diagram of DFA Lattice](Fig. 2. Paths in DFA Lattice)

Figure 2 gives a simple illustration of how performing merges $M_1$ and $M_2$ produces two disjoint paths in the DFA lattice. Now suppose that after performing $M_2$ there are two further merges (at the same level), we can further diversify our search paths by performing these two merges. When doing so we are increasing our probabilities of inferring the target grammar.

Clearly, by increasing the degree of diversification (i.e. the number of disjoint lattice paths explored) we are also increasing the amount of computation required. For this reason a distributed application has been developed in [5]. A central authority would determine which path in the lattice each daemon connected to the system has to traverse. Finally each daemon sends back its hypothesis and the central authority of the system decides which of the received hypothesis is closest to the
target DFA (by comparing the number of states of the hypotheses with the number of states of the target DFA).

The following example illustrates the notation that has been developed to describe diversification strategies.

\[
\begin{align*}
M_1 &: M_1 \\
M_2 &: \text{PI}(M_1) \\
M_3 &: \text{PI}(M_1 \land M_2) \\
M_4 &: \text{F}(M_2) \\
M_5 &: \text{F}(M_2) \land \text{PI}(M_4)
\end{align*}
\]

In this example \(M_1\) is carried out first. \(M_2\) is then carried out at the same level of \(M_1\). Note that \(M_2\) is pairwise incompatible with \(M_1\). \(M_3\) is pairwise incompatible with both \(M_1\) and \(M_2\) and is carried out at their same level. The algorithm then performs further diversification on the node following merge \(M_2\). The two paths followed are \(M_4\) and \(M_5\). Note that \(M_4\) simply follows from \(M_2\), while \(M_5\) follows after \(M_2\) and is also pairwise incompatible with \(M_4\). Various search diversification strategies can easily be described through this notation.

3 Results

This section presents some preliminary results of our search diversification strategies. The training sets of all the problems used for experimentation were downloaded from the Gowachin server. 60 state DFAs with 4000 training examples have been used with the following diversification strategy.

\[
\begin{align*}
M_1 &: M_1 \\
M_2 &: \text{PI}(M_1) \\
M_3 &: \text{PI}(M_1 \land M_2)
\end{align*}
\]

The EDSM heuristic was used to order the valid merges. The following table illustrates the classification rate for five DFA learning problems. Three daemons were used to traverse the three different search paths. In these examples diversification has been carried out on the APTA node of the DFA lattice.

<table>
<thead>
<tr>
<th>Problems</th>
<th>Daemon 1</th>
<th>Daemon 2</th>
<th>Daemon 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>99.1</td>
<td>98</td>
</tr>
<tr>
<td>2</td>
<td>97.8</td>
<td>99.3</td>
<td>99.5</td>
</tr>
<tr>
<td>3</td>
<td>59.1</td>
<td>90.4</td>
<td>92.9</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>98.4</td>
<td>99.6</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>99.2</td>
<td>99.2</td>
</tr>
</tbody>
</table>

It is interesting to note that in two occasions (out of five), the third daemon performed better than the first one. The first daemon performs the standard EDSM algorithm (i.e. always traversing the highest scoring merge), while the third daemon traverses a path which starts with a merge which is
incompatible with the first two merges. This indicates that when introducing search diversification through pairwise incompatible merges we are able to considerable improve on what EDSM and S-EDSM can do.

For a complete listing of results the reader is referred to [5].

4 Conclusion and Future Perspectives

Search diversification is clearly an important technique to exploit in DFA learning. Instead of focusing solely on improving the heuristic by which merges are ordered, we are also exploiting a heuristic which orders different merge sequences in the DFA lattice. This heuristic is based on pairwise incompatibility of merges. Further research in this area is currently being carried out.

Another area which is currently being researched is that of developing algorithms which perform well when noise is present in the training set. There is currently no algorithm which is capable of learning a 50 state DFA from a training set of 5000 strings with 10% noise.

This paper has presented an overview of the research currently being carried out by the Grammatical Inference Research Group (GIRG) at the CSAI department. Our aim is that of designing better DFA learning algorithms which further push the boundaries of DFA learning.

References

Author Index

Abela, Charlie, 52
Cachia, Ernest, 1, 2, 9
Claessen, Koen, 22
D’Emanuele, Karlston, 11
Inguanez, Frankie, 1
Micallef, Mark, 2
Montebello, Matthew, 20, 52
Pace, Gordon J., 11, 22, 30
Rosner, Michael, 45
Scerri, Simon, 52
Spina, Sandro, 61
Staff, Chris, 41
Vella, Christine, 30
Vella, Mark, 9
For the price of a cup of coffee a day...

...There are so many things you can do for the future of your kids!

The aim of the Ministry for Investment, Industry and Information Technology is that of ensuring that all Maltese citizens take an active role in today’s developments, in a society that is increasingly reliant on the use of information and communications technologies.

We want that your children will also enjoy access to hardware, software and advantageous packages for broadband Internet.

The HelloIT Programme offers computer hardware and peripherals, software, related training and Internet service connection packages at reduced and affordable prices!

Ministry for Investment, Industry and Information Technology

helloit
“Try Something new Today”

The following companies have joined forces with the Ministry and are offering equipment and training at slashed prices.

For more information log on to www.miti.gov.mt or phone 21 255495
Ixor is a fast-growing UK software and services company delivering innovative electronic payment and financial transaction systems.

(1) Java Architects / Developers
You will have the skill and the experience to engineer distributed transactional systems in Java / J2EE. REF: JDA0905

(2) Quality Assurance and Technical Operations Engineer
You will have a strong understanding of Java / J2EE plus knowledge of networking and Unix. REF: TOE0905

(3) Online Fraud Prevention Agents
You will be experienced in online credit-card fraud detection and prevention. Gambling industry experience an advantage. REF: OFPM0905

(4) Customer Support Representatives
You will have excellent English writing skills and be able to understand customer problems and help resolve them. Ability to speak Cantonese an advantage. REF: CSR0905

- Attractive premises in Ta' Xbiex
- Training provided
- Excellent packages

For further information visit http://www.ixaris.com/jobs.html

To apply e-mail your CV to jobs@ixaris.com by Monday 3rd October 2005 giving details of relevant experience and stating which of the positions you are applying for, using the code. Applications will be dealt with in the strictest confidence.

www.ixaris.com