Proceedings of the

Third
York Doctoral Symposium on Computing

Department of Computer Science, University of York, UK
4th November 2010

Editor:
Alvaro Miyazawa

Scope of the Symposium

The aim of the *York Doctoral Symposium* (YDS) is to establish a platform for dissemination and exchange of up-to-date scientific information on theoretical, generic and applied areas of computing by giving doctoral students an opportunity for presenting their ongoing research in computer science and scientific computing. In particular, the symposium intends to bring together young researchers who are active in this wide field and interested in an interdisciplinary exchange of ideas and experience. The symposium also strives to promote research and development for the improvement of interdisciplinary applications of computing.

Topics include, but are not restricted to:

- Artificial Intelligence
- Bioinformatics and Mathematics
- Computer Architectures
- Computer Vision
- Database Systems
- Distributed Systems
- Enterprise, System and Organisational Architectures
- High Integrity Systems Engineering
- Human-Computer Interaction
- Management and Information Systems
- Natural Language Processing
- Non-Standard Computation
- Programming Languages and Systems
- Real-Time Systems
- Signal Processing and Patter Recognition
- Theoretical Computer Science

Organization

YDS'2010 is organized and funded by the Department of Computer Science, University of York, UK.

Executive Commitee

Conference Chair:	Malihe Tabatabaie (University of York, UK)
	Jennifer Owen (University of York, UK)
Programme Chair:	Alvaro Miyazawa (University of York, UK)

Organising Commitee

Committee Members:	Malihe Tabatabaie (University of York, UK)
	Jennifer Owen (University of York, UK)
	James Williams (University of York, UK)
	Ahmad Shahid (University of York, UK)
	Teodor Ghetiu (University of York, UK)
	Chris Poskitt (University of York, UK)
	Jason Reich (University of York, UK)
	Asmiza Abdul San (University of York, UK)

Acknowledgements

The YDS2010 organisers would like to thank Zoe Stephenson for the excellent logo and poster design, and Alan Burns for enabling the YDS conferences.

Programme Commitee

Programme Chair:	Alvaro Miyazawa (University of York, UK)
Committee Members:	Michael Banks (University of York, UK)
	Alan Burns (University of York, UK)
	Burcu Can (University of York, UK)
	Clement Creusot (University of York, UK)
	Andre Freire (University of York, UK)
	Alan Frisch (University of York, UK)
	Edwin Hancock (University of York, UK)
	Oleg Lisagor (University of York, UK)
	John McDermid (University of York, UK)
	James McLaughlin (University of York, UK)
	Becky Naylor (University of York, UK)
	Adam Nellis (University of York, UK)
	Simon OKeefe (University of York, UK)
	Jennifer Owen (University of York, UK)
	Richard Paige (University of York, UK)
	Helen Petrie (University of York, UK)
	Fiona Polack (University of York, UK)
	Chris Poskitt (University of York, UK)
	Richard Ribeiro (University of York, UK)
	Colin Runciman (University of York, UK)
	Malihe Tabatabaie (University of York, UK)
	Jim Woodcock (University of York, UK)

Referees

Waleed Alsanie (University of York, UK)
Michael Banks (University of York, UK)
Burcu Can (University of York, UK)
Andre Freire (University of York, UK)
Alan Frisch (University of York, UK)
Daniel Kudenko (University of York, UK)
Oleg Lisagor (University of York, UK)
James McLaughlin (University of York, UK)
Becky Naylor (University of York, UK)
Adam Nellis (University of York, UK)
Simon OKeefe (University of York, UK)
Jennifer Owen (University of York, UK)
Chris Poskitt (University of York, UK)
Andrew Rae (University of York, UK)
Richard Ribeiro (University of York, UK)
Ahmad Shahid (University of York, UK)
Malihe Tabatabaie (University of York, UK)

Sponsoring Institutions

Department of Computer Science, University of York, UK
Microsoft Research,
Graduate Training Unit, University of York, UK

Table of Contents

Bodies of Information e-Health and its Philosophical Implications

Luciano Floridi[1,2,3]

[1] Research Chair in Philosophy of Information and GPI, University of Hertfordshire
[2] Faculty of Philosophy and IEG, University of Oxford
[3] UNESCO Chair in Information and Computer Ethics.
Address for correspondence: Department of Philosophy, University of Hertfordshire, de Havilland Campus, Hatfield, Hertfordshire AL10 9AB, UK
l.floridi@herts.ac.uk

Abstract. The first part of the talk will introduce an interpretation of the information turn as a fourth revolution. We are not immobile, at the centre of the universe (Copernican revolution); we are not unnaturally detached and diverse from the rest of the animal world (Darwinian revolution); we are not Cartesian subjects entirely transparent to ourselves (Freudian). We are now coming to see that we are not disconnected entities, but rather informational organisms, sharing with biological agents and engineered artefacts a global environment ultimately made of information, the infosphere (Turing revolution). In the second part, the previous framework will be used to understand the development of e-Health and its ethical issues. The fourth revolution is increasingly affecting our views about human nature, its fragility and resilience, its health (including mental health) and how we may shape it and make it flourish. We shall see how human bodies may be interpreted informationally and what this will mean, in the future, in terms of their well-being.

Generating Formal Model Transformation Specification Using a Template-based Approach

Asmiza A. Sani, Fiona Polack, and Richard Paige

Department of Computer Science,
University of York, Heslington, York. YO10 5DD, UK
asmiza,fiona,paige@cs.york.ac.uk

Abstract. Model transformation is a key activity in Model-Driven Engineering (MDE). Transformations map between models, in different languages and/or at different levels of abstraction. Model transformation introduces challenges for *specification, verification and validation.* Normally, MDE development requires planning which includes metamodel and transformation design and a customized testing. This paper describes ongoing work on a unified approach to specifying and verification of model transformations using a *template-based* mapping to a formal specification language. A small example shows how diagrammatic models mapped to Alloy so that the Alloy Analyzer can check transformation properties.

Keywords: Model transformation, verification, template-based transformation, formal methods

1 Introduction

A system can be described using a model. Models describe structural and behavioral details, from a variety of perspectives. A model is formulated in a language (e.g. a Domain Specific Modeling Language (DSML)) that represents the required system domain features.

Model Driven Engineering (MDE) is a model-centric approach to software development. In MDE, every artifact is treated as a model. Whereas models in conventional software engineering are guiding documents, in MDE, the models are the first-class artifacts [18]. In MDE, a model conforms to a metamodel that defines the syntax and semantic of model elements to represent the concepts and features of a particular domain [4]. The metamodel and its model instances are thus bound by syntactic and semantic mappings.

A model transformation takes a model as an input (source) and produces another model as output (target). Figure 1 depicts concepts of model transformation. The transformation is specified (defined) at the metamodel level, and can be executed by a transformation engine on models that conform to the source metamodel. An example of MDE-style model transformation is found in OMG's Model Driven Architecture (MDA) [15]: Figure 2 shows examples of different MDA model viewpoints and concepts; model transformation is used as the driver for evolving these models to a (partial) system implementation.

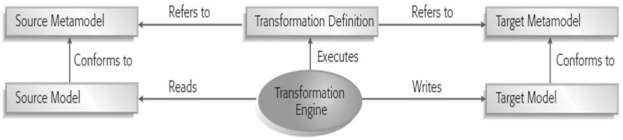

Fig. 1. Concepts of model transformation [6].

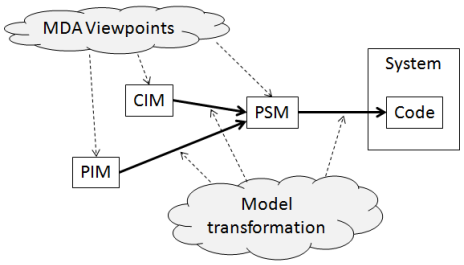

Fig. 2. Fundamental MDA concepts: *Viewpoints* represent levels of abstraction: PIM is the *Platform Independent Model*; CIM is the *Computation Independent Model*; PSM is the *Platform Specific Model*. Based on [15]

Some challenges of MDE model transformation are specifying transformations precisely and abstractly [8]; analyzing model transformation [7]; and verification to ensure correct implementation. This paper outlines work-in-progress on a template-based approach that formalizes (in Alloy) model transformation specifications, to support verification of properties through use of existing formal tools (Alloy Analyzer). The paper focuses on a simple example of a transformation specification from a UML class diagram to a relational database.

2 Verifying model transformations

A transformation engine executes a transformation specification written in a transformation language. Of the various transformation languages and tools, Query/Views/Transformation (QVT) is the OMG's de facto standard [16], whilst ETL [12] and ATL [10] are widely-used hybrid languages. MOLA [11] and SiTRa [1] are examples of imperative model transformation languages, whilst Tefkat [13] uses a declarative approach. (See [12] for a review of model transformation languages and tools.) Checking model transformations conventionally requires custom methods. Formal verification methods can be used to ensure the well-formedness of model transformation specifications, and to check that the specification produces a correct transformation. A formal specification gives a mathematical statement of the required properties of a system, whilst omitting

details of how the properties are to be achieved [19]. Available approaches to formal verification uses Constructive Type Theory [17]; OCL invariants [14, 5]; and coloured Petri nets[13]. However, none provides wide-ranging, tool-supported verification for model transformation.

3 Proposed approach

MDE does not traditionally use formal methods. In considering formal verification of model transformation, it is necessary to protect developers from exposure to too much formalism. To produce a practical formal approach to verifying transformation, Alloy [9] is selected. Alloy is a lightweight formal specification language derived from Z [19], with similarities to object-oriented structure and OCL. These features can be exploited by formally expressing the structure and behaviour of models to be transformed. Alloy is declarative, and can be used in a relational or a state-based way, so Alloy expressions should be capable of representing most model transformation approaches. The Alloy Analyzer tool automatically analyses specifications and reports using a graphical notation, reducing the need for users to have mathematical knowledge.

Alloy specifications comprise *atoms* and *relations*, both of which have properties defined by *fact* and *predicate* expressions. The facts and predicates are used to generate instances. The Alloy Analyzer tool uses *assert* expressions and the *check* command to determine whether an instance property holds; if it does not, then the tool generates a *counter-example*.

Anastasakis et al. in [3] shows how Alloy can be use to verify model transformation but to automatically produce an Alloy specification, the first stage is to map Alloy to a diagrammatic notation (eg. transML [8]), that represents transformation structural and behavioral conditions. Metamodels and diagrammatic transformation notation are converted to Alloy using automated template-instantiation (based on the GeFoRME approach [2]). Figure 3 is an overview of the technique. At this stage, a catalogue of templates is under development, based on examples of transformation specifications represented in UML-like syntax. Templates exist for both structural and behavioural transformation patterns, and the analysis assertions needed for verification.

3.1 Application example

The approach is illustrated using a fragment of a transformation specification that converts a UML Class diagram to a relational database schema. Figure 4 shows the metamodels. The transformation specification here comprises two transformation rules: *Class to Table* (C2T) and and *Attribute to Column* (A2C). Figure 5 shows the transformation rules applied to the metamodels.

Figure 6 shows three templates used to generate an Alloy specification from UML (T1 to T3). Template T4 generates an Alloy representation of a transformation specification, as a mapping from source elements to target elements.

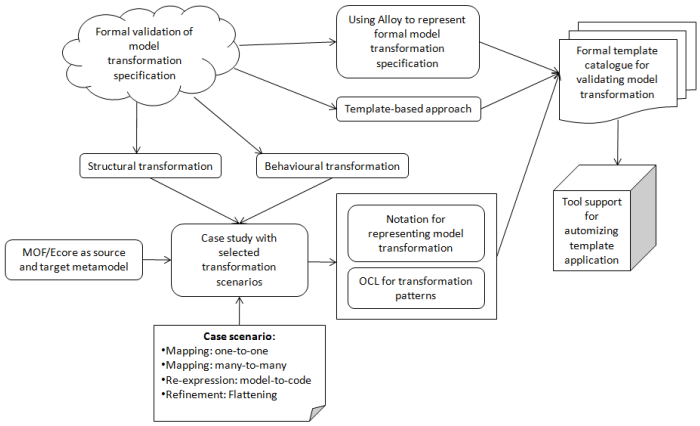

Fig. 3. Overview of the technique for template-based Alloy analysis of transformation specifications

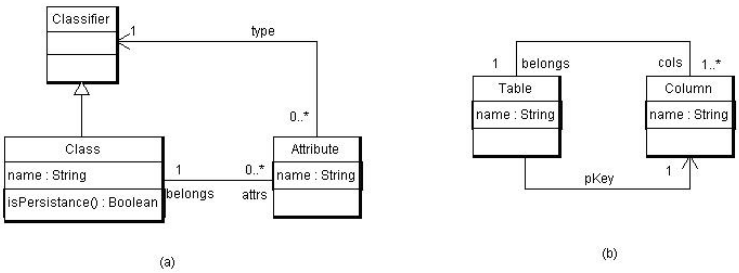

Fig. 4. Metamodel fragments of (a) Source: UML Class source metamodel (b) Target: database table metamodel

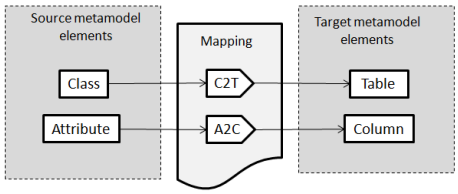

Fig. 5. Transformation rules to transform a Class to a Table

Currently, the mappings are written directly in Alloy. An extract of the instantiation is shown in Figure 7. The complete Alloy transformation specification for the example in Figure 5, generated from the diagrams in Figure 4 and Alloy mapping rules, is given in the Appendix.

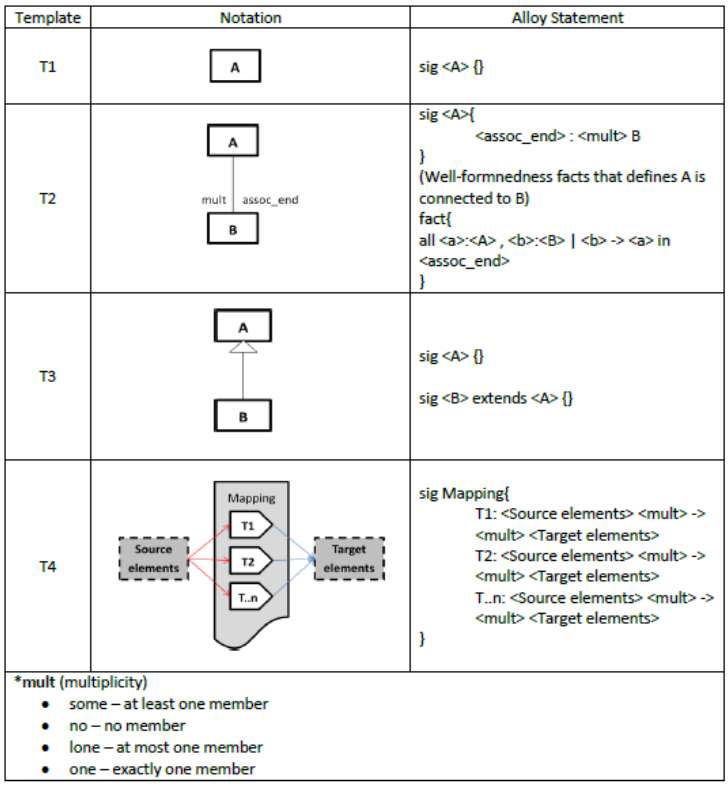

Fig. 6. Some templates to generate Alloy representations of UML fragments and transformation rules: placeholders for names to be inserted from the metamodel are in angle brackets

4 Conclusion and future work

The example used in this paper shows how a formal model transformation specification in Alloy is generated by instantiating templates via diagrammatic notations representing a small set of structural patterns. Currently, the well-formedness rules for the transformation rules and assertions expressions are manually defined to enable validation of transformation using the Alloy Analyzer tools: research is in progress on formalizing transformation rules in existing

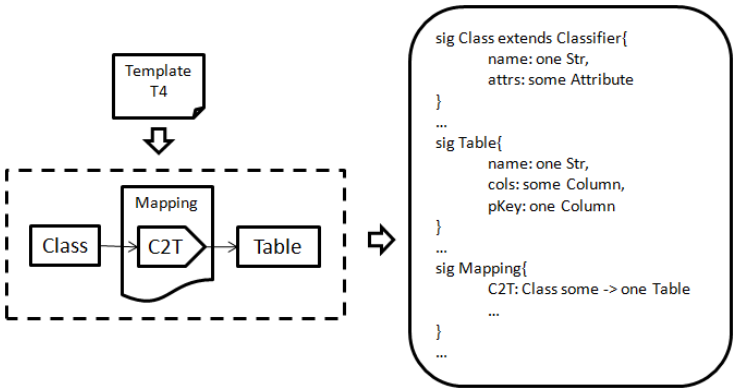

Fig. 7. How to apply the template to generate Alloy specification

transformation languages. Analysis of the execution result of the specification is conducted manually by visually tracing each instance to detect any abnormal occurrence.

To achieve the goal of unifying specification and verification of transformation, including formalization of OCL/EOL for behavioral conditions and identifying test patterns for model transformation which then represented by Alloy's *assert* and *check* expression fragments enables automatic verification of the specification. Ultimately, a collection of patterns for specifying and checking concepts of model transformation, supported by tools, will aid planning and design of a correct and reliable transformation.

References

1. David Akehurst, Behzad Bordbar, Michael Evans, Gareth Howells, and Klaus McDonald-Maier. SiTra: Simple transformations in java. In *Model Driven Engineering Languages and Systems*, volume 4199 of *LNCS*, pages 351–364. Springer, 2006.
2. Nuno Amálio. *Generative frameworks for rigorous model-driven development*. PhD thesis, Dept. Computer Science, Univ. of York, 2007.
3. Kyriakos Anastasakis, Behzad Bordbar, and Jochen M. Kauster. Analysis of Model Transformations via Alloy. In *Models in Software Engineering*, LNCS. Springer, 2007.
4. Jean Bézivin, Mikael Barbero, and Frédéric Jouault. On the applicability scope of Model Driven Engineering. In *Model-Based Methodologies for Pervasive and Embedded Software*, pages 3–7. IEEE, 2007.
5. Jordi Cabot, Robert Claris, Esther Guerra, and Juan de Lara. Verification and Validation of Declarative Model-to-Model Transformations through Invariants. *The Journal of Systems and Software*, 2009.
6. Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM System Journal*, 45(3), 2006.

7. Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *Future of Software Engineering*, pages 37–54. IEEE, 2007.
8. Esther Guerra, Juan de Lara, Dimitrios S. Kolovos, Richard F. Paige, and Osmar dos Santos. transML: A family of languages to model model transformations. In *MoDELS/UML*, LNCS. Springer, 2010. to appear.
9. Daniel Jackson. Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 2002.
10. Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, 2008.
11. Audris Kalnins, Edgars Celms, and Agris Sostaks. Model transformation approach based on MOLA. In *MoDELS/UML'05 Workshop: Model Transformations in Practice Workshop*, 2005. http://sosym.dcs.kcl.ac.uk/events/mtip05/programme.html.
12. Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack. The Epsilon Transformation Language. In *Theory and Practice of Model Transformations*, volume 5063 of *LNCS*, pages 46–60. Springer, 2008.
13. Michael Lawley and Jim Steel. Practical declarative model transformation with Tefkat. In *Model Driven Engineering Languages and Systems*, volume 3844 of *LNCS*. Springer, 2006.
14. László Lengyel, Tihamér Levendovszky, Gergely Mezei, TamásVajk, and Hassan Charaf. Practical Uses of Validated Model Transformation. In *Computer as a Tool*, pages 2200–2207. IEEE, 2007.
15. Joaquin Miller and Jishnu Mukerji eds. MDA Guide Version 1.0.1. Technical Report omg/2003-06-01, OMG, June 2003.
16. OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, 2008.
17. Iman Poernomo. Proofs-as-Model-Transformations. In *Theory and Practice of Model Transformations*, volume 5063 of *LNCS*, pages 214–228. Springer, 2008.
18. Douglas C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, 2006.
19. Michael Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 1992.

Appendix: Alloy transformation and verification of Figure 4 and 5

```
one sig Classifier{}
sig Str{}
sig Class extends Classifier{
        name: one Str,
        attrs: some Attribute
}
sig Attribute{
        name: one Str,
        cbelongs: one Class
}
//Well-formedness rule: Attributes of Class connected by attr
    relation c
```

```
fact{
        all a:Attribute , c:Class | c -> a in attrs
        all a:Attribute , c:Class | a -> c in cbelongs }
//target: relational database metamodel
sig Table{
        name: one Str,
        cols: some Column,
        pKey: one Column }
sig Column{
        name: one Str,
        tbelongs: one Table }
//Well-formedness rule: Column of Table connected by cols and
    pKey relation
fact{
        all t:Table , col:Column | t -> col in cols
        all col:Column , pk:Table | pk -> col in pKey }
//transformation
sig Mapping{
        C2T: Class some -> one Table,
        A2C: Attribute some -> one Column }
//transformation well-formedness rules C2T and A2C
fact C2T{
        all c:Class, t:Table, m:Mapping | c -> t in m.C2T =>
            c.name = t.name
        Class <: attrs in (Class) one -> one (Attribute) }
fact T2PKColumn{
        all t:Table, c:Column, m:Mapping | t -> c in m.
            T2PKColumn }
fact A2C{
all a:Attribute, c:Column, m:Mapping | a -> c in m.A2C => a.
    name = c.name }
//assertions for validating transformation
assert SameNameC2T{
all c:Class, t:Table| c.name = t.name }
assert SameNameA2C{
all a:Attribute, c:Column|a.name = c.name }
assert SameNumAttributeColumn{
all a:Attribute, c:Column|#a =#c }

//check
check SameNameC2T for 2
check SameNameA2C for 1
check SameNumAttributeColumn for 2
run {}
```

Evaluating YETI using Open Source Software

Matthew Patrick and Manuel Oriol

University of York, UK
{matthew.patrick,manuel.oriol}@cs.york.ac.uk
http://www.cs.york.ac.uk

Abstract. Search-based testing can be difficult for commercial applications of software because of the many potential points of failure. Research into test data generation often focusses on small programs that are not representative of the larger systems used in industry. This research introduces a new categorisation of software that can be used to evaluate testing techniques against a variety of software behaviour. These categories are used to evaluate YETI, a random robustness testing tool, with open source software. The experiments are limited to one class for each category, but still a number of faults are found for each one.

Keywords: Random testing; Software categorisation; YETI

1 Introduction

Random search is one of the most straightforward and inexpensive testing strategies [5]. It covers the whole input-space, so is useful in providing an indication of the general characteristics of software [7]. It is sometimes seen as inferior toother strategies because it does not take into account the syntactic or semantic structure of a program [6]. Random testing may need to produce many test cases in order to find a fault, but it produces each one with very little computational expense. Random testing is able to reveal more faults per unit time than a directed search [8]. The key to success with random testing is the availability of an oracle that can quickly verify the results of executing each test case, without the need for human involvement [5]. Unfortunately, it can be just as difficult to produce an oracle that works correctly as to develop the software without any bugs.

The York Extendible Testing Infrastructure (YETI) addresses this problem by considering run-time errors, such as incorrect casting or division by zero, in the absence of assertion violations or contracts. This forms a robustness check that can test software even when there is no explicit oracle. YETI can produce a significant amount of test data in a short space of time, making up to 10^6 calls per minute [11]. YETI can be used to test software written in a number of different programming languages, including Java. This research will evaluate YETI in testing different categories of Java software.

2 Related Work

Many tools have been developed to test software robustness with random input data. JCrasher [2], Jartege [10], Eclat [12] and AutoTest [1] are some examples. YETI is at least 3 orders of magnitude faster than these tools [11] and as mentioned earlier, is programming language independent. Research has also been conducted, guiding test data with specific vulnerable points, or the paths that are taken during its execution [4][3]. These techniques promise to be more likely to find faults, but they require further computation. It would be an interesting experiment to compare their effectivness against YETI.

3 Categories of Software

Software may be categorised by size, programming paradigm or the purpose for which it has been designed. It is easy to make distinctions on structural or developmental issues, but this reveals little about the expected behaviour of the software. Functional categories, such as those used by the Microsoft Asset Inventory Service [9], are difficult to distinguish for testing because they share common behaviour. The ideal form of categorisation would be one that is easily calculable, but also reveals useful information on the behaviour of software.

To aid the evaluation of YETI, we present five categories of software behaviour: functional, open, progressive, user and timed. These categories are not meant to represent the complete behaviour of a software system, but rather to provide insight into the variety of behaviour that has to be tested. For example, a web-based application that involves interaction with users, may also make progressive updates to a database, but both behaviours will need to be tested:

Functional behaviour involves significant numerical calculation and/or algorithmic operations. Examples range from core data structures like *java.Integer* up to complex libraries such as the *Bouncy Castle* cryptography suite. This is the kind of behaviour that is often targeted in traditional testing research.

Open behaviour makes use of features that are shared between multiple parties, with potentially different goals. These systems may also feature a heterogeneity of languages, platforms and architectures.

Progressive behaviour maintains and develops a data source over time. It must always leave the data source in a legal state. The data source may potentially be infinite, or only bounded by the available memory. Typically, this is addressed by dividing it into manageable sections, for example a byte stream.

User behaviour may be command-line, graphical or even web-based. Runtime errors can occur when an unsolicited GUI event is triggered. It is difficult to test the many paths through software featuring user behaviour because of the many ways that the components of its interface may be manipulated.

Timed behaviour includes a real-time performance requirement and typically some form of concurrency. Failures in concurrent systems may result in unexpected deadlock and race conditions. These can be difficult to avoid and expensive to detect because they occur as an interaction between classes.

4 Experiments

The software used in the following experiments is taken from industry, because most programs used to evaluate testing tools are too small-scale to represent a particular category. Relevant failures are those that have not occurred before whilst testing. Hundreds of thousands of random test cases were produced for each class, in an attempt to identify relevant failures. Each test run lasted ten minutes and is presented here with the number of relevant failures and the time it took to find them. Unfortunately, there was only enough time to represent each category with one class. This affects the conclusions that can be made.

Functional

JBullet[1] is a port of the Bullet Physics Library[2] for soft and rigid body dynamics. The original (unported) version is has been used for big budget films such as 2012 and Hancock, as well as games such as Grand Theft Auto IV and Madagascar Kartz. The first port into Java was made in January 2008 and since then there have been 10 further releases, each adding new features and correcting bugs. This experiment used the latest available version of JBullet, released in June 2009 and based on the 2.70-beta1 version of the Bullet Physics Library.

The class used in this experiment is a graphics demo provided with JBullet, *com.bulletphysics.demos.movingconcave*. Figure 1 shows two relevant failures raised whilst testing the class as a result of null pointer exceptions, both of which were found very quickly by YETI. It is possible that more errors would have been found if assertions had been included in the code to ensure accurate calculations. Without the use of assertions, it is difficult to test for incorrect calculations or delays that affect the worst case execution time for particular inputs.

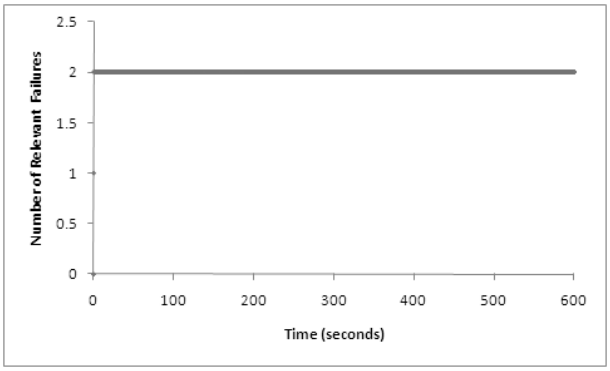

Fig. 1. Functional System Faults

[1] http://jbullet.advel.cz/
[2] http://bulletphysics.org/

Open

Jigsaw[3] is a web-server platform produced by a small team of developers from the World Wide Web Consortium (W3C). It was initially started to experiment with new technologies and, although it lacks some of the features of commercial web servers, it still leads the way in terms of web-protocols and interfaces. We experimented with Jigsaw 2.2.6, which is the latest available version. The Jigsaw development team place an emphasis on readable code and clear documentation and in spite of its experimental status, Jigsaw has been shown to be more robust than the average web-server [13].

Open systems face dangers from attackers outside the system and the possibility of conflict between different modules inside the system. In this experiment, we consider the *jigsaw.ssi.SSIFrame* class. This class ensures that HTML documents are parsed correctly at the server side. It is important that this software possesses fault tolerance and is able to resist tampering. Therefore, it was a surprise to find that it contained at least 19 run-time errors (see figure 2). The errors were caused by null pointer and array index out of bounds exceptions. Although they might be masked by the interaction of classes, these errors could still cause problems under certain circumstances.

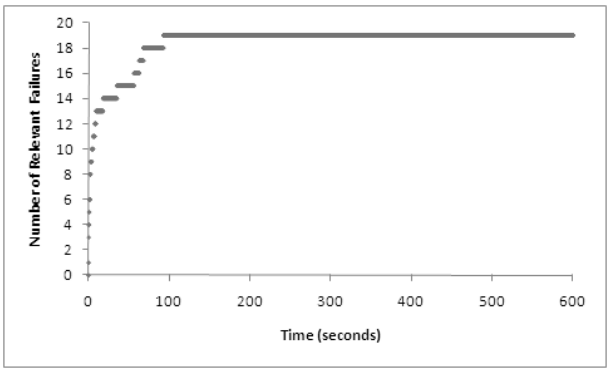

Fig. 2. Open System Faults

[3] http://www.w3.org/Jigsaw/

Progressive

Apache Derby[4] is relational database management system, implemented in Java. The software was first released in 1996 and has been developed by a number of companies before being made open source in 2004. It supports a high level of SQL standards-compliance, but still maintains a small footprint of just 2MB, making it easily testable. For this experiment we used version 10.6.1.0, which is the latest available version of the software.

We tested the *org.apache.derby.impl.sql.SQLParser* class. This class is used to parse queries written in SQL before they are applied to the database. Figure 3 shows that YETI revealed 47 relevant failures, found in two stages. The first 40 errors were found very quickly, but it took longer for the remaining seven to be detected. The errors found in this case were caused by null pointer exceptions.

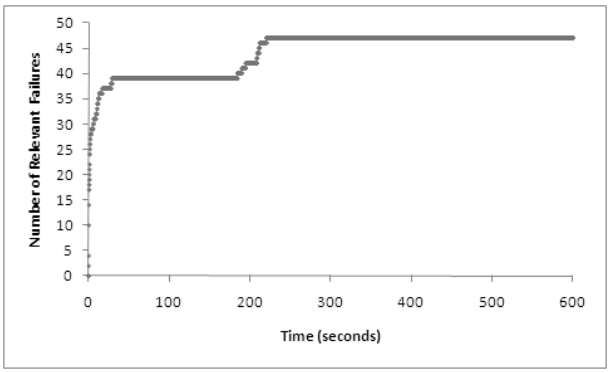

Fig. 3. Progressive System Faults

User

Swing[5] is the primary Java toolkit for graphical user interfaces. It features over thirty standard interface components and has a highly partitioned architecture, making it possible to plug-in new components with ease. The Swing API was first integrated into the core set of Java classes in December 1998. Since then, its performance has been increased for Java SE 6 in 2006. The *javax.swing.JTree* class is used to display hierarchical data. It generated more run-time errors than any other in these experiments (52 in total). The failures were revealed gradually throughout the course of the experiment (see figure 4). It is likely that even more failures would have been found if this experiment were left to run.

[4] http://db.apache.org/derby/
[5] http://java.sun.com/javase/technologies/desktop/

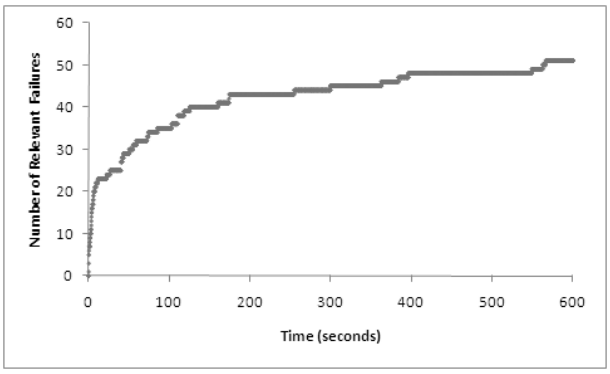

Fig. 4. User System Faults

Timed

Javolution[6] is a real-time library aimed at making Java applications faster and more predictable. It has been used by various other projects, such as Math-Eclipse[7], a symbolic maths engine, and Resoa[8], a cloud computing project. It was first released in October 2004 and has been through 19 major software releases before version 5.5, used in this experiment. We investigated the *javolution.util.FastMap* class, providing a real-time safe hash map (see figure 5). YETI does not perform any kind of timing analysis. Therefore, it is possible that some failures were missed involving missed deadlines or interaction between multiple threads. It would be interesting to evaluate whether the run-time errors predicted by YETI are good predictors of timing related failures.

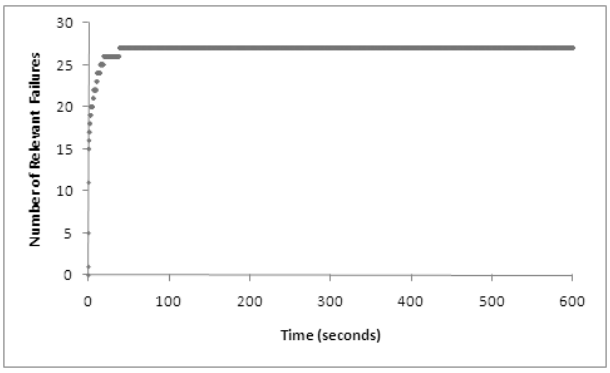

Fig. 5. Timed System Faults

[6] http://javolution.org/
[7] http://sourceforge.net/projects/matheclipse/
[8] http://www.resoa.org/

5 Limitations and Future Work

The biggest limitation to this research is the decision to represent each category with just one class. It is difficult to make a thorough analysis, because fully representative classes are hard to find. Some classes are overly specialised andothers too general to demonstrate the challenges particular to each category. For the results to be useful in determining the relevant software fault characteristics, it would be necessary to analyse a number of classes for each category.

Another limitation is that, without a test oracle, YETI is only able to observe the occurrence of run time exceptions and not whether the software actually produced the correct results. This is a limitation of the testing technique and cannot be resolved without the inclusion of an explicit specification or assertions in the code. One of the main reason for using YETI is that it is able to analyse software for which there is no oracle or for which the specification is inaccurate. An experiment into the relationship between run-time errors and incorrect results may be worthwhile.

6 Conclusions

In conclusion, YETI has been shown capable of finding faults across a wide range of software, even without the use of an explicit oracle. Random robustness tests struggle to find failures that only occur for a small subset of the input domain, but the speed with which YETI produces test cases allows it to find many failures with relatively little computational expense. We have shown that YETI is useful for testing a wide variety of software, particularly in the absence of a test oracle.

References

1. Ciupa, I., Pretschner, A., Leitner, A., Oriol, M. and Meyer B., On the predictability of random tests for object-oriented software, in International Conference On Software Testing, Verification And Validation (2008)
2. Csallner, C. and Smaragdakis, Y., Jcrasher: an automatic robustness tester for java, Software: Practice and Experience, vol. 34, no. 11, 1025–1050 (2004).
3. Ganesh, V., Leek, T. and M. Rinard, Taint-based directed whitebox fuzzing. In Proceedings of the 2009 IEEE 31st International Conference on Software Engineering (2009)
4. Godefroid, P. Levin, M. and Molnar, D., Automated whitebox fuzz testing, in Network and IT Security Conference (2008)
5. Ince, D., The automatic generation of test data. The Computer Journal. 30:1, 63–69 (1987)
6. Myers, G., Badgett, T., Thomas, T. and Sandler, C., The Art of Software Testing, Wiley, New York (2004)
7. Hamlet, R., Maciniak, J., Random Testing, Encyclopedia of Software Engineering, Wiley, New York, 970–978 (1994)

8. Mankefors, S., Torkar, R. and Boklund, A., New quality estimations in random testing, Proceedings of the 14th International Symposium on Software Reliability Engineering, 468–478 (2003)
9. Microsoft Corporation, Software Categories, Microsoft TechNet, http://technet.microsoft.com/en-us/library/bb852143.aspx (2010)
10. Oriat, C., Jartege, a tool for random generation of unit tests for java classes, Centre National de la Recherche Scientifique, Institut National Polytechnique de Grenoble, Universite Joseph Fourier Grenoble I, Tech. Rep. RR-1069-I (2004)
11. Oriol, M., Testing .NET Code with YETI, Proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems (2010)
12. Pacheco, C. and Ernst, M., Eclat, Automatic generation and classification of test inputs, in ECOOP 2005 Object-Oriented Programming, 19th European Conference (2005)
13. Server Watch, Jigsaw: The W3Cs Open Source Web Server, http://www.serverwatch.com/sreviews/article.php/1376671/Jigsaw-The-W3Cs-Open-Source-Web-Server.htm (2002)

Calculated Secure Processes

Michael J. Banks and Jeremy L. Jacob

Department of Computer Science, University of York
{Michael.Banks,Jeremy.Jacob}@cs.york.ac.uk

Abstract. This paper introduces a versatile operator for modifying CSP processes to satisfy particular information flow security requirements. We present and justify an algebraic semantics for this operator, which allows us to derive secure processes from (potentially) insecure processes in a calculational style. Moreover, the operator simplifies the task of verifying the security of processes.
Keywords: computer security, information flow, formal software development, CSP, semantics, confidentiality properties, secure refinement.

1 Introduction

In the broadest sense, a formal specification of a system defines a set of *correctness properties*. For an implementation of a system to satisfy its specification, it must satisfy all the correctness properties present in the specification. Perhaps the most obvious examples of correctness properties are concerned with functionality, but other non-functional requirements of software that can be precisely articulated — such as performance, reliability and resource usage — may also be interpreted as correctness properties.

Confidentiality properties are a class of non-functional correctness properties of software systems that are related to information security. Informally, a confidentiality property codifies an upper bound on what information a user of the system can deduce about the system's behaviour from its local observation of that behaviour. A system's specification may include confidentiality properties to stipulate that an implementation of the system must not (directly or indirectly) leak secret information to untrusted users.

This paper describes an approach for extending the CSP process algebra [1,2] to assist in the construction of software systems that are guaranteed to uphold specified confidentiality properties. Our main contribution is an operator over CSP processes that is parametrised by an encoding of confidentiality properties. This operator can be applied to processes in two ways. First, it may be used to verify that a process satisfies a specified confidentiality property. Secondly, the operator may be applied to any process S to *calculate* a process S' that is guaranteed to satisfy a confidentiality property.

This paper is structured as follows. Section 2 gives an overview of CSP and information flow security and outlines how CSP processes can be analysed for information leaks. Section 3 defines the algebraic semantics of the operator and Section 4 details a worked example in applying the operator to a simple process.

In Section 5, we identify techniques for verifying processes against confidentiality properties. We examine how our approach relates to the wider field of information flow security in Section 6 and summarise our work in Section 7.

2 CSP and Confidentiality

CSP (*Communicating Sequential Processes*) is a formal notation for modelling the patterns of behaviour of concurrent and distributed systems [1,2].[1] In CSP, a process interacts with its environment by engaging in *events* over an *alphabet* (set) of named channels: for example, the process $a \to b \to S$ performs an event on channel 'a' followed by an event on 'b', and then behaves according to S.

CSP features a rich algebra of operators for giving structure to processes, whose semantics are formally defined by a denotational model. In the *traces* model, a trace records the history of a process as a sequence of events. The semantics of a process S in the traces model is characterised by the prefix-closed set of all traces that S may perform: for example, if $S = a \to b \to \mathit{Stop}$, then $\mathit{traces}\,[\![S]\!] = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle\}$.

Consider the following recursive CSP process M and its associated trace set:

$$M \triangleq (h \to l \to M) \sqcap (k \to \mathit{Stop})$$

$$\mathit{traces}\,[\![M]\!] = \left\{ \begin{array}{l} \langle\rangle, \langle h\rangle, \langle h, l\rangle, \langle k\rangle, \langle h, l, h\rangle, \langle h, l, h, l\rangle, \langle h, l, k\rangle, \\ \langle h, l, h, l, h\rangle, \langle h, l, h, l, h, l\rangle, \langle h, l, h, l, k\rangle, \ldots \end{array} \right\}$$

M non-deterministically chooses either to perform a 'h' event followed by a 'l' event and then behave as M again, or to perform a 'k' event and then refuse (forever) to perform any events (Stop).

We model a user of a process (representing a system) as an agent which can observe some — but not necessarily all — of the channels of the process. We define a user's *window* to be the subset of a process's alphabet that contains all events visible to that user. Hence, a user's observation of a trace tr is the *projection* of tr through the user's window w, denoted by $tr \upharpoonright w$. [2]

Suppose that M's environment consists of a "low-level" (untrusted) user $\mathcal{L}$ with window $L = \{l\}$. If $\mathcal{L}$ knows the structure of M then, given an observation ϕ of M viewed through L, $\mathcal{L}$ can deduce all traces of M that are consistent with ϕ. We capture this intuition by defining an *inference function* on processes: [3,4]

$$\mathsf{infer}(M, L, \phi) \triangleq \{tr \mid tr \in \mathit{traces}\,[\![M]\!] \wedge tr \upharpoonright L = \phi\} \tag{1}$$

The traces in $\mathsf{infer}(M, L, \phi)$ are *indistinguishable* from $\mathcal{L}$'s perspective: if this set contains more than one trace, then $\mathcal{L}$ cannot determine which of these traces describes the actual behaviour of M. Nevertheless, $\mathcal{L}$ may still be able to identify common features of all traces in this inference set and so deduce sensitive or valuable information about the actual behaviour of M.

We may insist that M satisfies the following security requirement:

[1] Readers unfamiliar with CSP notation may wish to consult the Appendix.

[2] $tr \upharpoonright A$ is the trace given by removing from tr all events that are absent from set A. For example, $\langle a, b, a, c, a, b\rangle \upharpoonright \{b, c\} = \langle b, c, b\rangle$.

"The occurrence of 'h' events should be kept secret from $\mathcal{L}$."

While $\mathcal{L}$ cannot observe 'h' events directly (since $h \notin L$), if $\mathcal{L}$ makes an observation $\langle l \rangle$ of M, it could deduce that a 'h' event must have occurred in M's execution by calculating the inference set associated with $\langle l \rangle$:

$$\mathsf{infer}(M, L, \langle l \rangle) = \{\langle h, l \rangle, \langle h, l, h \rangle, \langle h, l, k \rangle\} \tag{2}$$

Since all traces in $\mathsf{infer}(M, L, \langle l \rangle)$ feature an initial 'h' event, we conclude that M does not satisfy our security requirement. (Of course, M satisfies other requirements: for example, $\mathcal{L}$ cannot establish that 'k' has occurred, because for each $\mathcal{L}$ observation ϕ of M, $\mathsf{infer}(M, L, \phi)$ contains traces without a 'k' event.)

3 Securing Processes by Extension

In this section, we describe a systematic approach for extending processes to satisfy given security requirements. We write $<P, Q>(S)$ — where S, P and Q are CSP processes — to denote a process S' that behaves like S but, whenever S' can perform an activity that conforms to a behaviour of P, then S' may (at the environment's discretion) instead perform an alternative activity conforming to Q (instead of P) and then proceed to behave as S as if it had performed P.

We select the P and Q processes to express a confidentiality property. For example, we could encode the requirement that $\mathcal{L}$ cannot deduce the occurrence of 'h' events followed by 'l' events in M as a $<P, Q>$ pair by writing:

$$<h \rightarrow l \rightarrow \mathit{Skip}, l \rightarrow \mathit{Skip}>$$

Here, P specifies that an occurrence of a 'h' event immediately prior to an 'l' event is classed as confidential. Q specifies that the absence of the 'h' event represents a plausible non-confidential "cover story" for the confidential behaviour. When applied to a process, the $<P, Q>$ operator masks the occurrence of P behaviours from $\mathcal{L}$'s perspective, by inserting alternative Q behaviours that $\mathcal{L}$ cannot distinguish from P behaviours. Hence, it is essential that the projections of P and Q through $\mathcal{L}$'s window are identical. Formally:

$$\{tr \upharpoonright L \mid tr \in \mathit{traces}\,[\![P]\!]\} = \{tr \upharpoonright L \mid tr \in \mathit{traces}\,[\![Q]\!]\} \tag{3}$$

We define the semantics of $<P, Q>$ by giving a collection of algebraic laws formulated in terms of the standard CSP operators. Hence, the application of $<P, Q>$ to a process S is carried out by recursively applying these laws according to the structure of S. Since $<P, Q>$ is defined in this way, a denotational semantics for $<P, Q>$ can be calculated straightforwardly.[3]

We insist that P, Q and S are divergence-free[4] processes, to simplify our reasoning and to make certain that applying $<P, Q>$ to S can never introduce

[3] Due to space constraints, we do not provide the denotational semantics here.

[4] A process is said to *diverge* if it can perform *"an infinite sequence of internal actions"* (without interacting with its environment) [2].

divergence into S. Furthermore, we require that P and Q always terminate by performing *Skip* as their final action. This condition ensures that applying $<P, Q>$ to a terminating process produces a terminating process.

In what follows, we require a notation for describing the behaviour of a process S after having performed an activity modelled by process P. The predicate $S \,\mathsf{app}\, P$ is satisfied if and only if S features a trace prefixed by a complete trace of P (with the final '✓' generated by the *Skip* in P omitted):

$$S \,\mathsf{app}\, P \;\triangleq\; \exists\, tr \in traces\,[\![S]\!] \bullet tr \frown \langle ✓ \rangle \in traces\,[\![P]\!] \tag{4}$$

We write $S\#P$ to denote the process that behaves as S after executing any P trace that is a trace of S. Of course, $S\#P$ is well-defined only if $S \,\mathsf{app}\, P$ holds.

$S\#P$ is defined in terms of the CSP "after" operator — where S/tr represents all possible behaviours of S after behaving as tr [2, §1.3.4] — as follows:[5]

$$S\#P \;\triangleq\; \bigsqcap\{S/tr \mid tr \frown \langle ✓ \rangle \in traces\,[\![P]\!] \wedge tr \in traces\,[\![S]\!]\} \tag{5}$$

3.1 Distribution through Choice

As we saw in Section 2, $S \sqcap T$ is a process that chooses non-deterministically to behave either as S or as T. Likewise, the process $S \,\Box\, T$ can behave as S or as T, but allows the environment to select between them.

The semantics of $<P, Q>$ applied to processes that offer a choice of multiple initial events is given by distributing $<P, Q>$ across each branch of the choice:

Law 1

$$<P, Q>(S \sqcap T) \;\triangleq\; <P, Q>(S) \sqcap <P, Q>(T) \tag{6}$$
$$<P, Q>(S \,\Box\, T) \;\triangleq\; <P, Q>(S) \,\Box\, <P, Q>(T) \tag{7}$$

3.2 *Stop* and *Skip*

Two fundamental processes in CSP are *Stop* (deadlock) and *Skip* (termination). Applying $<P, Q>$ to *Stop* or *Skip* has no effect, because *Stop* can never perform any P activity. Likewise, since *Skip* can only ever extend the trace with the (invisible) signal event ✓, *Skip* cannot perform any P activity.

Law 2 *Stop and Skip are unaffected by applying* $<P, Q>$.

[5] Notation: $\bigsqcap\{S_1, \ldots, S_n\}$ may be read as $S_1 \sqcap \ldots \sqcap S_n$.

3.3 Prefixing

We now consider the semantics of $<P, Q>(a \rightarrow S)$, where 'a' is any event.

If P cannot perform any events — that is, if $P = \mathit{Skip}$ — the semantics of $<P, Q>$ are not defined by the laws below. We consider such a choice for P improper, because it does not represent any confidential activity.

Law 3 states that if $a \rightarrow S$ cannot behave as P (as when 'a' is not an initial event of P), then we can safely move the $<P, Q>$ operator to S.

Law 3 *Provided that* $\neg\,(a \rightarrow S)\,\mathsf{app}\,P$*:*

$$<P, Q>(a \rightarrow S) \triangleq a \rightarrow <P, Q>(S) \tag{8}$$

We need a prefix law to handle cases where $a \rightarrow S$ may perform an activity encoded by P (i.e. $(a \rightarrow S)\,\mathsf{app}\,P$ holds). In such cases, we require that $a \rightarrow S$ is extended with the behaviours of Q to ensure that $\mathcal{L}$ cannot determine that a P behaviour has occurred. When P and Q do not share the same initial events, the operator should extend $a \rightarrow S$ to offer the environment the choice of performing Q instead of P. This effect is codified by Law 4.

Law 4 *Provided that* $S\,\mathsf{app}\,X$ *holds:*

$$<a \rightarrow X, b \rightarrow Y>(a \rightarrow S) \triangleq \begin{pmatrix} a \rightarrow <a \rightarrow X, b \rightarrow Y>(S) \\ \Box\ b \rightarrow Y \fatsemi <a \rightarrow X, b \rightarrow Y>(S \# X) \end{pmatrix} \tag{9}$$

Law 4 allows $S' = <a \rightarrow X, b \rightarrow Y>(a \rightarrow S)$ to behave as $b \rightarrow Y$ whenever it can behave as $a \rightarrow X$. After behaving as $b \rightarrow Y$, S' reverts to behaving as it would have done *after* performing as $a \rightarrow X$. Law 4 employs deterministic choice between $a \rightarrow X$ and $b \rightarrow Y$ to ensure that S' must always offer $b \rightarrow Y$ to the environment whenever it can offer $a \rightarrow X$.

Up to now, we have considered cases where confidential activities are masked by adding appropriate cover story activities. Alternatively, a cover story may take the form of the *absence* of a confidential activity by setting $Q = \mathit{Skip}$. The semantics of $<P, \mathit{Skip}>(a \rightarrow S)$ are not given by Law 4. Hence, we introduce a new law to accommodate cases where $Q = \mathit{Skip}$.

Law 5 *Provided that* $S\,\mathsf{app}\,X$ *holds:*

$$<a \rightarrow X, \mathit{Skip}>(a \rightarrow S) \triangleq \begin{pmatrix} a \rightarrow <a \rightarrow X, \mathit{Skip}>(S) \\ \Box <a \rightarrow X, \mathit{Skip}>(S \# X) \end{pmatrix} \tag{10}$$

Assuming that $\mathcal{L}$ cannot distinguish $a \rightarrow X$ from Skip, this law generates a process that masks the occurrence of $a \rightarrow X$ from $\mathcal{L}$'s perspective.

3.4 Operator Disposal

If we can prove that S never behaves according to P (i.e. S never performs a confidential activity at any point in its execution), we can be sure that it is unnecessary to introduce the cover story Q within S. If this condition holds, then we can safely remove the $<P, Q>$ operator from S.

Law 6 *When* $\forall\, s \bullet \neg\, (S/s)\,\mathsf{app}\, P$ *holds,* $<P, Q>(S)$ *is equal to* S*.*

This law can be justified by appealing to the repeated application of Law 3 to the expansion of S, in the context of the other laws.

4 Worked Example

Together, these laws are sufficient for applying the $<P, Q>$ operator to simple CSP processes. Let $M' = <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(M)$. We derive the CSP process equal to M' — using the laws of $<P, Q>$ and CSP — as follows:

$$
\begin{array}{ll}
& <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(M) \\
= & \quad \{\ \text{definition of } M\ \} \\
& <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>((h \to l \to M) \sqcap (k \to \mathit{Stop})) \\
= & \quad \{\ \text{distribute operator through internal choice (Law 1)}\ \} \\
& \left(\begin{array}{l} <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(h \to l \to M) \\ \sqcap\ <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(k \to \mathit{Stop}) \end{array}\right) \\
= & \quad \{\ k \to \mathit{Stop} \text{ cannot behave as } h \to l \to \mathit{Skip}\ \text{(Law 3)}\ \} \\
& \left(\begin{array}{l} <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(h \to l \to M) \\ \sqcap\ k \to <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(\mathit{Stop}) \end{array}\right) \\
= & \quad \{\ \text{applying to } \mathit{Stop} \text{ has no effect (Law 2)}\ \} \\
& <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(h \to l \to M) \sqcap k \to \mathit{Stop} \\
= & \quad \{\ h \to l \to M \text{ is confidential (Law 4)}\ \} \\
& \left(\begin{array}{l} h \to <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(l \to M) \\ \Box\ l \to \mathit{Skip} \fatsemi <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>((l \to M) \# (l \to \mathit{Skip})) \end{array}\right) \\
& \sqcap\ k \to \mathit{Stop} \\
= & \quad \{\ \text{carry prefix (Law 3); unfold } S\#P\ \text{(Equation 5)}\ \} \\
& \left(\begin{array}{l} h \to l \to <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(M) \\ \Box\ l \to \mathit{Skip} \fatsemi <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(\bigsqcap\{M\}) \end{array}\right) \sqcap k \to \mathit{Stop} \\
= & \quad \{\ \text{simplify}\ \} \\
& \left(\begin{array}{l} h \to l \to <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(M) \\ \Box\ l \to <h \to l \to \mathit{Skip}, l \to \mathit{Skip}>(M) \end{array}\right) \sqcap k \to \mathit{Stop} \\
= & \quad \{\ \text{fold } M' \text{ (twice)}\ \} \\
& (h \to l \to M' \ \Box\ l \to M') \sqcap k \to \mathit{Stop}
\end{array}
$$

The inference set associated with the $\mathcal{L}$ observation $\langle l \rangle$ of M' is:

$$\mathsf{infer}(M', L, \langle l \rangle) = \{\langle l \rangle, \langle l, k \rangle, \langle h, l \rangle, \langle h, l, h \rangle, \langle h, l, k \rangle\}$$

Observe that $\mathsf{infer}(M, L, \langle l \rangle) \subseteq \mathsf{infer}(M', L, \langle l \rangle)$ (see Equation 2). Hence, if $\mathcal{L}$ observes $\langle l \rangle$, it cannot deduce any more about the behaviour of M' as it could

about the behaviour of M. More importantly, $\mathsf{infer}(M, L, \langle l \rangle)$ features traces free of 'h' events, which implies that $\mathcal{L}$ cannot establish that M' has performed 'h', in keeping with our security requirement. In the next section, we present theorems for proving that this requirement is upheld for *all* of $\mathcal{L}$'s observations of M'.

5 Verification and Refinement

We claim that $<P, Q>(S)$ must satisfy the confidentiality property encoded by P and Q by appealing to the following theorem.

Theorem 1 ($\mathcal{L}$ cannot deduce P). *Given any L-observation ϕ of $<P, Q>(S)$, $\mathcal{L}$ can never establish (with certainty) that a P activity has taken place.*

This theorem is justified from the semantics of $<P, Q>$ by establishing that the process $<P, Q>(S)$ always permits a Q activity to be performed in place of a P activity. Since we insist that P and Q yield the same observations to $\mathcal{L}$ (Equation 3) — and that after performing Q, $<P, Q>(S)$ behaves as it would after performing P (Law 4) — it must be the case that if $\mathcal{L}$'s inference set contains a trace $s \frown p \frown s'$ (where $p \frown \langle \checkmark \rangle \in \mathit{traces}\, [\![P]\!]$), then it also contains a trace $s \frown q \frown s'$ (where $q \frown \langle \checkmark \rangle \in \mathit{traces}\, [\![Q]\!]$). It follows that $\mathcal{L}$ can never deduce from its observation of S that a confidential P activity has been performed.

$<P, Q>$ is an idempotent operator. (The proof follows by applying structural induction to each of the laws presented in Section 3.) It is therefore necessary to apply $<P, Q>$ to a process just once, to obtain a process that respects Theorem 1. We can appeal to this result to characterise secure processes algebraically.

Theorem 2 (Verifying Security). *Given a process S such that $<P, Q>(S) = S$, S must satisfy the confidentiality property encoded by $<P, Q>$.*

We now discuss the relationship between the $<P, Q>$ operator and refinement. It has long been known that refining a secure process may result in an insecure process [4,5]. This problem arises because non-determinism may serve two purposes in process specifications: to avoid describing "don't-care" implementation details (*underspecification*) and to ensure the behaviour of the process is *unpredictable* from $\mathcal{L}$'s perspective. It follows that naïvely resolving non-determinism within a process S (by refinement) may result in the removal of cover story behaviours from S. In turn, this may introduce new sources of information flow to $\mathcal{L}$ — that were absent from S — that allow $\mathcal{L}$ to deduce the presence of confidential activity in the refined process.

The standard CSP refinement relation is failures-divergences refinement [2], which corresponds to the resolution of (finite) non-determinism in processes: $S \sqcap T$ is refined by S (and T).

The process M' (see Section 4) can be refined to:

$$M'_0 \triangleq (h \rightarrow l \rightarrow M'_0 \,\Box\, l \rightarrow k \rightarrow \mathit{Stop}) \qquad \text{or} \qquad M'_1 \triangleq k \rightarrow \mathit{Stop}$$

Both of these processes satisfy our security requirement, since $\mathcal{L}$ cannot deduce the occurrence of a 'h' event by observing either of these processes through its window. It is perhaps tempting to claim that all refinements of a process $<P, Q>(S)$ satisfy the confidentiality property encoded by $<P, Q>$, but this is not so. Consider the process:

$$M_2' \triangleq (h \rightarrow l \rightarrow k \rightarrow \mathit{Stop}) \,\Box\, l \rightarrow (h \rightarrow l \rightarrow k \rightarrow \mathit{Stop} \,\Box\, l \rightarrow k \rightarrow \mathit{Stop})$$

M_2' is a refinement of M'. However, if $\mathcal{L}$ makes the observation $\langle l, k \rangle$ of M_2', then it can deduce that 'h' must have occurred, since the inference set associated with that observation contains only the trace $\langle h, l, k \rangle$. This result indicates that it may be necessary to re-apply the $<P, Q>$ operator after performing refinement.

6 Related Work

The canonical notion of information flow security is *noninterference*, which characterises the absence of any information flow about a high-level user's interactions with a system to low-level users [6]. While approaches for verifying that systems satisfy noninterference-like properties have been studied extensively, they are not widely used in practice, because it is often necessary to allow users to exchange data. Our confidentiality properties are weaker than noninterference, but are better suited for capturing practical security requirements than noninterference, since they permit non-secret information to flow to low-level users.

Our formalisation of confidentiality properties is loosely based on work by Mantel [7], who devised a "schema" condition for verifying that a system does not leak confidential information, expressed in terms of a set of confidential traces and a mapping from confidential traces to cover story traces. We specialise this approach by directly encoding confidential and cover story activities as processes. While the $<P, Q>$ operator is less general than Mantel's schema, it enables us to manipulate a CSP process using CSP itself and thereby allows us to provide an algebraic characterisation of whether a process is secure in Theorem 2.

Our method of verifying that a process satisfies a confidentiality property is related to Roscoe et al.'s *low-level determinism* test for CSP processes [8]. This test identifies whether a low-level user's observations of a process S can be encoded as a deterministic process: if so, the actions of other users cannot influence the observations of S made by $\mathcal{L}$, which implies that S satisfies noninterference. In contrast, we insist only that $\mathcal{L}$'s observations cannot be perturbed by the occurrence of confidential activities.

Finally, and more generally, there is a connection between our operator and *aspect-oriented programming* (AOP). AOP encourages software developers to implement the secondary *aspects* of a program (such as logging or user authorisation checks) separately from its core functionality [9]. A variant of our operator could potentially be used to support the construction of systems in an AOP style: for instance, the first argument of the operator would specify the "join points" in a program at which an aspect should be triggered, while the second argument would specify an aspect's behaviour.

7 Conclusion

In this paper, we have defined an algebraic operator for rewriting CSP processes to ensure they uphold particular security requirements. Applying this operator to a process guarantees that the resulting process satisfies the confidentiality property encoded by the operator. This suggests that the operator holds promise for constructing software that is "secure by design".

A potential drawback of using the operator is that adding cover stories to a process may violate functional requirements on the process's behaviour (such as safety properties). This difficulty may be mitigated by carefully selecting the cover stories associated with a confidential activity. However, it may be impossible to avoid this difficulty altogether when developing systems to satisfy both functionality and confidentiality requirements, since these requirements place opposing constraints on the information flow from a system to its users [4]. Indeed, there is a trade-off between functionality and confidentiality requirements: if the customer specifies a strong confidentiality property (such as noninterference) for a system, then it may be necessary to weaken the functional requirements on the system's design (and vice versa).

We believe that the application of the operator to CSP models exhibiting finite behaviour can be (partially) automated, with the assistance of model-checking tools (such as FDR [10]) to determine the points in a process's execution where it can perform confidential activities. However, the problem of automatically analysing a larger CSP model in this way may be intractable or undecidable, especially in the presence of state variables or unbounded nondeterminism. Furthermore, we have left the semantics of the operator undefined for processes expressed using the full algebra of CSP. An important topic for future work is to identify laws for applying the operator to concurrent processes.

On a final note, taking a formal approach to security engineering can help us to gain confidence that a computer system does not leak secret information to low-level users, but it is unwise to assume that any system implementation is secure in all circumstances. In particular, we have not addressed potential sources of information leakage within a system's implementation (such as its responsiveness or power consumption) that are not modelled by its specification.

Acknowledgements. Michael Banks is supported by an EPSRC DTA studentship. Thanks to the anonymous referees for their helpful comments and suggestions for making the paper more accessible to the YDS audience.

References

1. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International Series in Computer Science. Prentice Hall, Inc. (1985)
2. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall PTR, Upper Saddle River, NJ, USA (1997)
3. Jacob, J.L.: Security specifications. In: Proceedings of the 1988 IEEE Symposium on Security and Privacy. (1988) 14–23

4. Jacob, J.L.: On the derivation of secure components. In: Proceedings of the 1989 IEEE Symposium on Security and Privacy, IEEE Computer Society (1989) 242–247
5. Roscoe, A.W.: CSP and determinism in security modelling. In: Proceedings of the 1995 IEEE Symposium on Security and Privacy, IEEE Computer Society (1995) 114–127
6. Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proceedings of the 1982 IEEE Symposium on Security and Privacy, IEEE Computer Society (April 1982) 11–20
7. Mantel, H.: A Uniform Framework for the Formal Specification and Verification of Information Flow Security. PhD thesis, Universität Saarbrücken (July 2003)
8. Roscoe, A.W., Woodcock, J.C.P., Wulf, L.: Non-interference through determinism. In: ESORICS '94: Proceedings of the Third European Symposium on Research in Computer Security. Volume 875 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (1994) 33–53
9. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: ECOOP'97 Object-Oriented Programming. Volume 1241 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (1997) 220–242
10. FSE: FDR2 User Manual. Formal Systems (Europe) Ltd. (June 2005)
11. Davies, J.: Using CSP. In: Refinement Techniques in Software Engineering. Volume 3167 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2006) 64–122

A CSP Notation

This appendix describes some of the fundamental CSP operators informally. (For a more comprehensive description, the reader is directed to the various introductory tutorials on CSP that are available, such as Davies' tutorial [11].)

Prefix The process $a \rightarrow S$ waits until the environment is ready to accept an 'a' event, then performs an 'a' event and subsequently behaves as S.

Non-deterministic choice The process $S \sqcap T$ non-deterministically chooses to behave either as S or as T. The environment cannot influence how the non-determinism is resolved.

Deterministic choice The process $S \Box T$ offers the environment a choice between S and T, wherever possible. For instance, $a \rightarrow S \Box b \rightarrow S$ accepts either an 'a' or a 'b' event, before behaving as S.

Sequence The process $S \fatsemi T$ behaves as S until reaching *Skip* (Section 3.2), whereupon the process continues to behave as T.

In addition, CSP features a variety of operators for composing processes in parallel, hiding the occurrence of internal events from the environment, conditional choice (if-then-else) and iteration.

Selection of Seed Examples in ILP

Tarek Abudawood and Peter Flach

Intelligent Systems Laboratory, University of Bristol, UK
Dawood@cs.bris.ac.uk and Peter.Flach@bristol.ac.uk

Abstract. In this paper we investigate different methods of selecting seed examples and their effect on learning first order rules in ILP. Based on large experimental results we concluded that they have significant impact on accuracies, the number of rules and the speed of induced models.

1 Introduction

Inductive logic programming (ILP), is concerned with inducing first-order clausal models from examples and background knowledge [7]. ILP systems, such as PROGOL [6] and ALEPH [11], learn to generalise rules of the form $Head \leftarrow Body$[1] from positive and negative examples (training examples) such that they can be used to predict future unseen examples. PROGOL and ALEPH learners follow a seed-example-driven approach to learn a theory that explains the positive examples (sufficiency condition) and not the negative examples (necessity condition). A learner typically starts by taking a single training example at a time, called a seed example, use it generates a single rule based on this example an ensure that it generalises to other existing examples. The learner evaluates the quality of a considered rule using an evaluation measure such as *compression* that favours pure rules that only cover positive examples and penalises long rules that contains many literals.

ILP systems follows the separate and conquer (SAC) paradigm implementing the covering algorithm where examples that have been covered by a newly accepted rule are removed from the set of training examples. The learning proceed by picking up a new seed example from the remaining training examples and the same procedure is repeated until no training example is left or no further rules can be induced for the remaining ones. PROGOL, as a case in point, takes the seed examples one by one sequentially and blindly as they are presented in the training files.

We observed that changing the order of the seed examples, on several problems, has an impact on the learning task and that was the main motivation of our work. We argue that ordering the seed examples can improve the behaviour of such systems. Hence, if there is a way to enhance the selection mechanism, we might be able to improve the quality of induced rules produced by such example-driven induction systems. Such improvement might be exemplified by having more accurate models, having more compact models and/or generating faster models.

The key problem is finding a criteria to order or rank the seed examples upon it before the learning is undertaken. During the learning phase, a learner should be able to

[1] *Head* is a single literal and *Body* is a conjunction of literals.

pick up seed examples in a selected order. All what is known about these examples is contained in the background knowledge including the class label. Therefore, one may think of using statistical methods that look into the distribution of elements appear in the background knowledge of a given domain w.r.t. each class and use them to compute scores or probabilities over the examples. These scores or probabilities may be used to order examples accordingly as in probabilistic models such as Naive Bayes (NB) classifier [5] or LexRank (LR) ranker [4]. However, it is not feasible to apply such methods in an ILP context directly due to the complex relational structure that imposes dependencies between the structural predicates and that invalidate the independencey assumption that holds for NB and LR. After all, the idea is not to have a ranker that predicts a total ordering over examples but to selectively pass the examples to an existing ILP classifier in a way that improve its performance.

In this paper we present a new approach *Selecting Seed Examples* (SSE) that feed training examples to PROGOL selectively. SSE simply evaluates the difficulty of all examples, order them according to their difficulties and learn their classification rules w.r.t. to that order. SSE is done in three stages which will be discussed in details in Section 2.

As to the authors' knowledge there are no published research investigated the effect of seed examples' bias in ILP. The reason why this issue has not received much attention by the machine learning community might be due to the naive assumption that they have no, or little effect on the learning process. This paper will show that a simple ordering can results in a significance effect on the learning process and; therefore, the learning outcome.

Details of our approach are given in Section 2. Results of our approach compared to PROGOL's default method and a random method over twenty datasets are shown in Section 3. Section 4 summaries and concludes our work in this paper.

2 Selecting Seed Examples

SSE tries order examples according to their difficulties in a pre-learning process. This is done in three different stages. Given mode declarations (language bias), background knowledge, type definitions and the training examples, PROGOL learns some rules, called basic rules *BR*, in the usual way. This is the first stage of our approach. We use the *BR* to order the examples based on the frequencies of examples covered by each rule in the second stage. The underlying idea is that the examples are presented to the PROGOL again in the third and last stage but this time in a selected order determined by the second stage and used to induce the final set of rules *R* that represents the output model of the learning system. The ordering in the second stage reflects how hard or easy to cover the examples during the learning process. There are three more variations of the methodology used in the second stage as will be seen later.

In SSE we are introducing two main selection mechanisms: ascending and descending, called Ascend and Descend respectively. The descending mechanism is done by ordering[2] the examples from the most difficult example to cover, where none of the basic

[2] Note that ordering the negative examples is not necessary as the negative examples are generally not selected as seed examples in seed-example-driven ILP systems.

rules or few number of basic rules learned in stage 1 cover them, to the least difficult example, where examples are covered by many basic rules learned in stage 1. On the other hand the ascending mechanism is done by ordering the examples in the opposite way.

Generally speaking we are interested in comparing the results of learning from ordered examples and learning from PROGOL's default approach, where the seed examples are selected regarding to their sequence in the training file blindly. Therefore, in the default approach the examples are learned in the common way by introducing mode declarations, background knowledge, type definitions and the training examples to PROGOL directly without any pre-ordering mechanism. A similar experiment with a random ordering of the examples has been conducted as well, called Random.

While Default and Random are done in a single and straight forward stage, both ordering mechanisms, Ascend and Descend, take three stages to be performed. There are four different variations of SSE approach: ordering on the bases of rules, weighted rules, partial rules and weighted partial rules. Since we have two ordering mechanisms, Ascend and Descend, with these four variations of the SSE approach we will end up with eight methods as can be seen in Table 1 in addition to PROGOL and Random.

Method	Abbr.	Method	Abbr.
Descending Rules	DR	Descending Weighted Rules	DWR
Ascending Rules	AR	Ascending Weighted Rules	AWR
Descending Partial Rules	DPR	Descending Weighted Partial Rules	DWPR
Ascending Partial Rules	APR	Ascending Weighted Partial Rules	AWPR
PROGOL's default	PROGOL	PROGOL's Random	Random

Table 1. Table shows the eight methods of SSE approach, which are used to order the examples, in addition to PROGOL's default and Random, that can be seen at the last, and their abbreviations.

It deserves mention that the procedure applied to the Random method can be seen in Stage 1 of SEE approach. The importance of this stage for SSE is to produce the basic rules [3] *BR*, Definition 1, that will be used in Stage 2 to achieve the ordering. Stage 1 remains the same for all the eight methods. In the following subsections we will explain each one of the four variations of SSE approach with respect to the ascending and descending mechanisms in more details.

Definition 1 (Basic Rule). *A basic rule br_i is the ith rule that appear in the learned model of Stage 1 in SSE approach and used for the purpose of evaluating and ordering the seed examples.*

2.1 SSE with Rules

This is the simplest and most general SSE method among all the eight. There are three basic stages that can be seen in Algorithm 1. As mentioned earlier, the main aim of Stage 1 is to produce the basic rules model *BR* to be used as a criteria of measuring

[3] Basic rules: because these rules do not form the final model in the ordering mechanisms but they are importance to order the examples in the second stage only.

the difficulties of the seed examples but not to be considered in the final classification model.

Definition 2 (Difficulty of example). *Let X be a set of all training examples, $x_i \in X$ is an example, E is the total number of training examples, BR is a set of rules and cover is a boolean coverage function defined in Equation 2, then a difficulty of the i-th example $dif(x_i)$ can be measured by summing the total number of rules $br_j \in BR$ that cover x_i as follows:*

$$dif(x_i) = \sum_{j=1}^{|BR|} cover(br_j, x_i) \tag{1}$$

$$cover(br_j, x_i) = \begin{cases} 1 \ ; \ br_j \ covers \ x_i \\ 0 \ ; \ otherwise \end{cases} \tag{2}$$

The difference between Descend and Ascend comes in Stage 2 where the examples are ordered according to their difficulties in a descending or ascending order respectively. The difficulties are measured according to the coverage of all the basic rules on each example. The difficulty of an example is defined in 2. The higher the number of rules that cover an example the less difficult the example is and the vice versa. In Ascend the examples are ordered from the least difficult examples, which are covered by the most number of basic rules, to the most difficult ones, which are covered by the smallest number of basic rules. On the other hand, the ordering of the examples is done in the opposite way for Descend method. Stage 3 in SSE with Rules algorithm presents the seed examples to PROGOL for the second time but this time they will be learned according to the ordering made in the second stage, descending or ascending.

2.2 SSE with Partial Rules

In the previous method, the first stage was used to generate the basic rules. The basic rules were used later in the second stage to perform the ordering for Ascend and Descend. It is not hard to imagine that if the number of basic rules were quite small,e.g. $|BR| < 5$, the impact of Ascend and Descend can not be noticed especially with the existence of ties, when many examples covered by exactly the same number of rules. Aiming at widening our experimental scale and obtaining a larger amount of basic rules, another method, SSE with partial rules, has been applied in order to derive a better selection of seed examples.

The idea of partial rules inspired by [10] where rules are broken down into smaller parts called partial rules and assigned some weights with a neural network algorithm called *Winnow* during the learning phase. When an unseen example is encountered the partial rules that cover that particular example and their weights are used to predict its class. Since sometimes it is hard to classify examples with rules under the existence of noise and missing background knowledge or even in the case of rule clashes[4] [1], the

[4] Rule clash happens when more than one rule belonging to different classes cover same example or same subset of examples.

aim of using partial rules is to allow extra flexibility by relaxing the most interesting rules [1, 10].

Similarly, we would like to allow extra flexibility in our SSE technique by dividing the rules into smaller parts that hopefully will give a better estimation of how difficult the examples are. Our definition of a partial rule can be seen in Definition 3.

Definition 3 (Partial rule). *A partial rule pr of a rule r is a rule that inherits the same head (consequent) literal as the original rule and all or subset of its body (antecedent) literals that collectively form a first-order feature which is defined in 4.*

Definition 4 (First-order feature). *A first-order feature*[5] *is a conjunction of zero or more structural literals and at least one functional literal*[6] *and meets the following four conditions: a structural literal consumes one of the variables already introduced; a functional literal consumes the variables introduced by structural literals or the head literal; a variable is introduced in a feature is consumed by a structural literal, a functional literal, or the head literal; and a variable in the head literal must be consumed within the feature.*

Definition 5 (Structural Literal). *A literal* $l : p(A,B)$ *is said to be structural* $\iff$ *there is a component/object relationship between A and B. Otherwise, the literal is functional.*

Noticeably, the number of examples covered by a rule will always $\leq$ the number of examples covered by a part of that rule as a part of a rule is more general than the rule itself. Thus, partial rules may help in revealing more information about the difficulties of the examples. As in SSE with rules, examples covered by many partial rules are less difficult then the ones covered by no or fewer number of partial rules. Hence, with partial rules we will get even more evidence of their difficulties.

Not every rule can be boiled down into parts when applying the partial rule algorithm as a consequence of having singleton features. However, generally this method will generate much more basic partial rules *BPR* that may give a better estimation of the difficulty of seed examples and; therefore, it may help to acquire better ordering of seed examples.

The only modification in learning with partial rules in comparison with the previous algorithm, SSE with Rules, is the use of *CollectPartialRules* function instead of *CollectRules*. The rest of the algorithm remains the same but applied to partial rules instead of rules this time.

2.3 Learning with Weighted Rules and Weighted Partial Rules

In the previous two methods, the difficulty of an example was measured with regard to the number of basic rules or basic partial rules that cover that particular example. However, it can be noticed that the basic rules or partial rules themselves carry some extra information about how general or specific they are and such information is ignored

[5] Our definition of the first-order feature is an adaptation of the Definition 6 in [3]

[6] A functional literal can also be called a property

in the previous two techniques. It makes more sense to use such kind of information to gain further improvement.

In order to enhance our methods and incorporate such extra information that may improve the sensitivity of measuring the difficulty of each example, two new methods that assign weights for the basic rules as well as the basic partial rules are introduced. Since there is no difference between the process of assigning the weights for basic rules and basic partial rules, we will just explain the weighted basic rules and the reader is referred to Section 2.2 where the difference between the two was discussed.

Instead of computing the difficulty of an example x_i as the number of basic rules that cover it, Equation 1, here we additionally multiply the boolean coverage function by the weight of basic rules as defined in 6. This is done under the assumption that a weight of a rule reflects its specificity (how specific it is) and can be used to evaluate the difficulties of the examples accordingly.

Definition 6 (Weighted difficulty of example). *Let X be a set of all examples, $x_i \in X$ is an example, E is the total number of training examples, BR is a set of basic rules, cover is a boolean coverage function defined in Equation 2, and W is a set of weights such that $\forall br_j \in BR$ there exists $\left\{ w_j \in W | w_j = \frac{\sum_{x_i}^{x_E} cover(br_j, x_i)}{E} \right\}$. Then a weighted difficulty of the i-th example $wdif(x_i)$ can be computed as follows:*

$$wdif(x_i) = \sum_{j=1}^{|BR|} cover(br_j, x_i).w_j \tag{3}$$

As can be seen from Definition 6, every basic rule br_j is weighted according to its coverage on all the training examples, the weight are normalised and thus $1 \geq w_j \geq 0$. A value close to one means that a basic rule covers many examples and hence it is a very general rule. A value close to 0 means that a basic rule covers small number of examples and hence it is a very specific one. When computing a difficulty of an example in equation 3, the weights are taken into account this time.

To explain the difference between SSE with Rules and SSE with Weighted Rules method let us have a look at Table 2 and consider a case of having five examples and three rules. The difficulties of the five examples with regards to the basic rules and weighted basic rules methods can be seen in column 5 and 6 respectively. The final ordering of examples for both methods in descending and ascending order is given in the last four columns. As can be noticed from Table 2, br_1 is generalised over the most number of examples compared to the other two rules, as it covers 3 examples, and hence it is regarded as the least specific one. Both br_2 and br_3 cover 2 examples. With SSE with Rules, x_2 and x_4 have the same difficulty values even though x_4 is covered by the least specific rule, r_1. On the other hand, SSE with Weighted Rules method allows a better assessment of the example difficulties. It decreases the difficulty value of x_2 indicating that it is more difficult to cover than x_2 with regard to the weights assigned to each partial rule.

Selecting seed examples with weighted rules and weighted partial rules algorithms use a new modified version of $CalculatingDifficulty()$ function that implements the new difficulty function shown in Equation 3.

	br_1	br_2	br_3	dif	$wdif$	dif		$wdif$	
						Ascend	*Descend*	*Ascend*	*Descend*
x_1	0	0	1	1	0.4	3	3	5	1
x_2	0	1	1	2	0.8	1	5	2	4
x_3	1	0	0	1	0.6	4	2	3	3
x_4	1	1	0	2	1.0	2	4	1	5
x_5	1	0	0	1	0.6	5	1	4	2
w_j	0.6	0.4	0.4						

Table 2. A simple demonstration of calculating the difficulties with SSE with Rules and SSE with Weighted Rules methods over a set of five examples. Columns 2-4 show the boolean coverage of rules over examples and their weights at the last row. The last four columns show the ordering of the examples (indicated by indices) in Ascend and Descend methods with and without rule weights (column 5 and 6 respectively).

3 Empirical Evaluation

The ten methods discussed earlier and shown in Table 1 have been applied to twenty binary datasets both relational and propositional[7]. The examples for each dataset are 10-fold-cross validated in order to average the results over 10 separate runs. Each fold is generated randomly for all the methods except PROGOL. In addition, a further randomisation has been done within each fold itself. As for the eight SSE methods, a randomisation is also done when ordering examples ascending and descending in the second stage when ties are encountered.

Friedman test has been used for evaluating the performance of each method against all the others. It is a non-parametric test equivalent to the repeated-measures ANOVA [2]. Given a particular performance aspect, the Friedman test ranks the wins and losses of each method, averages them over each data set, and compares these average ranks. In our case the performance is measured according to three aspects: accuracy, number of rules and speed resulted from using each one of the 10 methods on PROGOL.

The post-hoc Nemenyi test was used to compare the absolute difference between the average rank of two methods against the critical difference CD. If difference between the average rank of two methods $\geq CD$ then there is a significant difference between the two methods in favour of the one with the highest average rank; otherwise, the null hypothesis can not be rejected. Moreover, Bonferroni-Dunn test has been applied to test the performance of each method against the default performance of PROGOL as a control method. A graphical illustration of the Nemenyi and Bonferroni-Dunn post-hoc tests have been drawn for each one of the three performance aspects. For more details about Friedman test the reader is referred to [2].

The post-hoc test results of applying Friedman test on 20 datasets w.r.t. accuracies, number of rules and running time are shown in Figure 1, 2 and 3 respectively.

Accuracy: Generally speaking the performance of all the 10 methods are quite close when looking at Figure 1. The p-value calculated by Friedman test reveals no significance difference at $p < 0.10$ between all the involved methods regarding the accuracy. However the reader can see that PROGOL's default approach comes last due to acquiring a rank > 6 in 14 out of 20 datasets and accordingly it gained the worst average rank. We draw the post-hoc Nemenyi test here just for the clarification purpose. As Friedman test showed that the null hypotheses can not be rejected at $p < 0.10$, no method is better

[7] The propositional data sets have been transformed into prolog format in the obvious way.

than any other, it is illustrated by thick line connects all the methods together in Figure 1. However, the post-hoc Bonferroni-Dunn test shows that there is a significant differences at $p < 0.10$ in favour of Random, DWR, AR, DR and DPR against PROGOL having PROGOL as a control method.

Compactness: All the four ascending methods, DR, DWR, DPR and DWPR, ranked the top four methods as they produce the lowest number of rules indicating the highest compactness. All the four methods are significantly better when compared to PROGOL in Bonferroni-Dunn test and almost all,, except of DWR, in Nemenyi test. The ascending mechanism does not seem to be effective with regards to this aspect and often produces high number of rules contrasting, as expected, the descending mechanism. Figure 2 illustrates the compactness results.

Speed: The only significant difference w.r.t the speed of the learning methods has been reported between DR and AWR favouring the former using Nemenyi test while AWR has been reported to be significantly worst than PROGOL as illustrated by Figure3.

General Remarks: It can be seen that PROGOL by default performs badly amongst all 10 methods presented as it always gets low rank and consequently plotted on the right hand side of the three critical difference figures. We have seen that by just randomising the seed examples, better results can be achieved. This suggests that when someone is to apply an ILP system that induce rules based on individual seed examples, they should ensure that seed examples are, at least, picked up randomly rather than sequentially. Another way is to randomise the examples once at the beginning of the learning and then go through them sequentially during the learning phase.

Descending methods generally generates the most compact models. This means that the learning algorithm starts with the most difficult seed examples, at the top of the training file. Those examples are assumed to be hard and hence, inducing rules for them is more difficult than inducing rules for the examples that come later in the training process. This consequently seems to force the learner to explore the hypotheses space deeply to find good hypothesis that characterises such hard examples. Following SAC strategy, examples which are covered by a newly induced rule are removed from the training file. Since the few induced rules at the beginning are deeply searched and characterise the class of interest (the positive class) well, it tends to cover many easy examples (placed at the bottom of the training process) and thus few number of rules are induced at the beginning resulting in compact models.

In contrast, ascending methods, not surprisingly, seem to give the opposite effect. They seem to produce the largest models. Descend does the reverse by learning the most easy examples first and induce some specific rules for them. These rules are not good enough to be generalised over the most difficult examples toward the end of the training file when such hard examples are encountered. As a results, the learner will have to induce more rules for these hard examples encountered later during the learning phase and consequently that results in generating larger models and more specific rules.

The speed of models appear to be correlated with their compactness, the higher the number of rules the higher the running time and vice versa. This is because learning few rules usually requires less time than learning a high number of rules due to the fact that PROGOL starts a new search for a rule every time a new seed example is selected and ignores the hypotheses considered previously following the covering algorithm.

4 Conclusion

We have demonstrated the effect of selecting the seed examples for PROGOL and showed that simple selection methods can affect the behaviour and consequently the performance of such seed-example-driven ILP systems. Running PROGOL in the default mode generally does not yield good performance on all the chosen aspects (accuracy, compactness and speed of learning a model) when compared to the other methods. Randomising the seed examples appears to be helpful to get some moderate improvement of PROGOL performance at a very low cost. As for our new SSE methods, descending methods (DR, DWR, DPR, and DWPR), usually get the highest performance especially w.r.t compactness and speed aspects. Ascending methods, in contrast, generally get bad performance as can be expected and we have explained the reason why this is happening at the end of Section 3. Our aim is fulfilled by highlighting the impact of selection of seed examples on such seed-example-driven learning. We believe that the use of advanced statistical methods will bring better insights and higher improvements, and thus more work has to be done in this direction to determine the best way of presenting the examples to the learners.

References

1. Kamal Ali, Stefanos Manganaris, and Ramakrishnan Srikant. Partial classification using association rules. In *Knowledge Discovery and Data Mining*, pages 115–118, 1997.
2. Janez Demšar. Statistical Comparisons of Classifiers Over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
3. Peter Flach and Nicolas Lachiche. 1BC: A First-Order Bayesian Classifier. In *Inductive Logic Programming*, volume 1634/1999 of *Lecture Notes in Computer Science*, pages 92–103. Springer Berlin, January 1999.
4. Peter Flach and Edson Matsubara. A simple lexicographic ranker and probability estimator. In *European Conference on Machine Learning*, volume 4701 of *Lecture Notes in Computer Science*, pages 575–582. Springer, September 2007.
5. Tom Mitchell. *MACHINE LEARNING*. McGRAW-Hill, 1997.
6. Stephen Muggleton. Inverse Entailment and Progol. In *Proceedings of the Sixth International Workshop on Inductive Logic programming*, volume 13, pages 245–286. Springer, 1995.
7. Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
8. D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI Repository of machine learning databases, 1998.
9. Mehran Sahami, Susan Dumais, David Heckerman, , and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.
10. Sukree Sinthupinyo, Cholwich Nattee, Masayuki Numao, Takashi Okada, and Boonserm Kijsirikul. Improving Multiclass ILP by Combining Partial Rules with Winnow Algorithm: Results on Classification of Dopamine Antagonist Molecules. In *New Frontiers in Artificial Intelligence*, volume 1634/1999 of *Lecture Notes in Computer Science*, pages 527–537. Springer Berlin, July 2007.
11. Ashwin Srinivasan. The Aleph Manual. Technical report, University of Oxford, 2001.

Algorithm 1 Selecting Seed Examples with Rules

procedure LEARN($X_{Train}, X_{Test}, BK, MD$)
  Start[Stage1] // Learning basic model: basic rules
    $X_{TrainRand} = RandomOrdering(X_{Train})$
    $LearnResults = PROGOL\_Learn(X_{TrainRand}, BK, MD)$// Learning model
    $BR = CollectRules(LearnResults)$// Extract rules from the learning model
  End
  Start[Stage2] // Selecting Seed Examples
    $DIF = CalculateDifficulty(BR, X_{Train})$ // DIF is a $|X_{Train}| \times |BR|$ binary matrix
    $X_{Des} = OrderExamplesDescending(DIF, X_{Train})$ // X_{Des} is X_{Train} with new order
    $X_{Asc} = OrderExamplesAscending(DIF, X_{Train})$ // X_{Asc} is X_{Train} with new order
  End
  Start[Stage3]
    $TestResults_{Des} = PROGOL\_LearnTest(X_{Des}, X_Test, BK, MD)$// Learning and Testing on Descend
    $TestResults_{Asc} = PROGOL\_LearnTest(X_{Asc}, X_Test, BK, MD)$// Learning and Testing on Ascend
    $Output TestResults_{Des}$
    $Output TestResults_{Asc}$
  End
end procedure

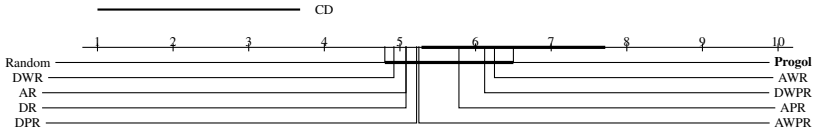

Fig. 1. Post-hoc tests for comparing the accuracies of the 10 methods shown on Table 1.

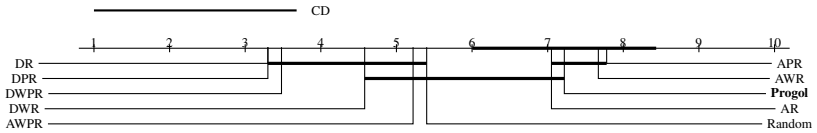

Fig. 2. Post-hoc tests for comparing the number of rules produced by the 10 methods shown on Table 1.

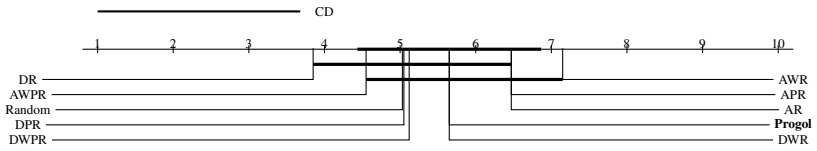

Fig. 3. Post-hoc tests for comparing the speed of models of the 10 methods shown on Table 1.

Visually Guided Control in Concurrent Decision Making Problems

Jose Nunez-Varela*

School of Computer Science, University of Birmingham,
B15 2TT, Birmingham, UK
j.i.nunez@cs.bham.ac.uk

Abstract. The goal of this project is to develop principled ways for the coordination of a robot's oculomotor system and the rest of its body motor systems. We propose the development of a coordination mechanism within a concurrent decision making framework in order to control physical and perceptual actions in parallel. The problem is to decide which physical actions to perform next and where should the robot's gaze be directed in order to gain relevant information about the environment. These decisions are made based on the task being executed, the current uncertainty of the environment and the expected reward that actions might receive.

Keywords: Gaze control, concurrent decision making processes, Markov decision processes, partial observability, learning

1 Introduction

The oculomotor system in a robot is a resource with the main function of gaining relevant information about the environment, so that the robot is able to make better decisions. However, the oculomotor system must be *shared* amongst the different actions the robot is performing. Furthermore, the resource is *limited*, defined by the cameras' field of view, and *noisy*, adding uncertainty to the problem. Therefore, a coordination mechanism must exist in order to control where to direct the cameras (gaze control) whilst other physical actions are taking place.

The proposed coordination framework is being implemented and will be tested using the iCub simulator[8] (Figure 1). A humanoid is best suitable for our purposes since it has multiple degrees of freedom (DoF) and an oculomotor system[1]. The problem then is to decide what the robot should do next in terms of physical and perceptual actions, so that the task being performed is accomplished successfully. Physical actions are produced by the different motor systems in the robot, e.g. arms, hands, torso or legs, whereas perceptual actions are camera movements. As shown in Figure 1, a simple task could be to pick up objects from the table and put them inside the containers.

* This work is funded by CONACYT-Mexico.

[1] In theory, our system could be adapted to work with non-humanoid robots, as long as they provide several DoF and an oculomotor system.

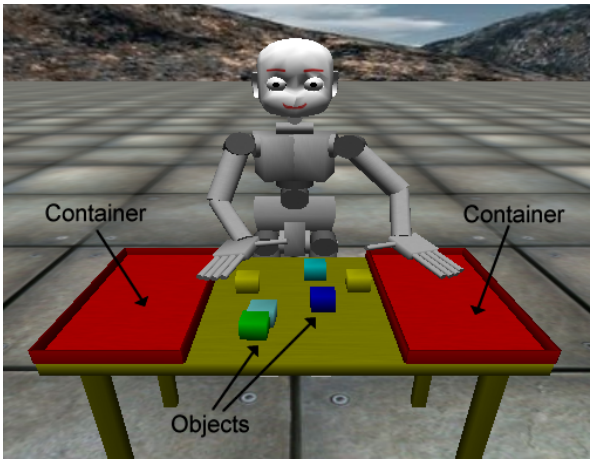

Fig. 1. Snapshot of the iCub Simulator.

The problem is formulated in a hierarchical concurrent decision making framework based on Markov decision processes (MDPs) [9] and reinforcement learning [12]. MDPs allow us to formally model a decision making problem in terms of states, actions, transitions and rewards. An MDP can later be solved via reinforcement learning, which is attractive because learning can take place in an unsupervised manner. The robot then learns a *policy*, i.e. a mapping between states to actions. This policy tells the robot what to do depending the state it is in.

Real-world problems pose more difficulties not directly addressed by MDPs. Real environments are normally partially observable, i.e. uncertainty about the current state of the environment is present; this can be modelled using *partially observable* MDPs (POMDPs) [6]. Actions might have different duration of time, whereas MDPs assume single-step actions; this can be modelled via *semi*-MDPs (SMDPs) [9]. MDPs execute one action at a time, however depending on the DoF available parallel actions can be performed (For instance, a humanoid can use both arms to pick up objects at the same time, while looking at a certain point in space). Our framework tries to take into account these properties.

Section 2 presents a summary of related work, then Section 3 describes our proposed framework. Section 4 explains the experiments to be carried out, and Section 5 provides some final remarks. It is important to note that this is an ongoing project and results are still not available.

2 Related Work

Current research on visual perception has focused on active vision and attentional systems [3, 4, 14], where image processing techniques help to identify regions of interest so that the robot can move the camera to get different images

of one object. On this respect our main interest is not on the image processing algorithms, but on incorporating visual perception as part of the decision making process.

Sprague's visually guided control [11] presents a way to coordinate physical and perceptual actions. The agent first learns a set of behaviours modelled as completely observable MDPs using reinforcement learning. During the execution of the policy learnt, partial observability is added to the problem. The true state of the environment is maintained via a set of visual routines (perceptual actions). These routines process visual input and transform it into state values. Only one visual routine can be executed at a time, thus the problem is to decide which visual routine to select every time step. The goal is to keep the values of the state variables updated, since these variables are used to decide which physical action to perform next. The disadvantages with this framework are that physical and perceptual actions are executed sequentially, instead of concurrently, actions are single-step and assumes a single-motor system.

Rohanimanesh presents a concurrent decision making framework for the coordination of actions [10]. He proposes two models: the *concurrent action model* (CAM) and the *coarticulation* model. In both models it is possible to execute in parallel a set of single-step actions and temporally extended actions. These extended actions are modelled as *options* [13]. Options are high-level actions where it is possible to "look" inside them, since each option is composed of simpler actions. The models are based on SMDPs [9], the difference with normal MDPs is that SMPDs allow the use of durative actions, instead of just single-step actions. The disadvantages are that these models assume a completely observable environment and no perceptual actions are taken into account.

Concurrent MDPs [7] are capable of modelling concurrent planning problems, where the time actions take to complete is part of the cost of executing such actions. The generalised SMDP [16] models problems with multiple asynchronous events and actions. As with Rohanimanesh's models, these do not take into account partial observability and perceptual actions.

3 Coordinating Gaze and Actions

We propose a hierarchical concurrent decision making framework for the coordination of physical and perceptual actions.

3.1 How the System Works

Figure 2 illustrates the interaction between components in our system. The system works in the following way:

– The robot first learns to perform a particular task keeping its policy in memory. Learning occurs in a completely observable environment, thus no perceptual actions are learnt.

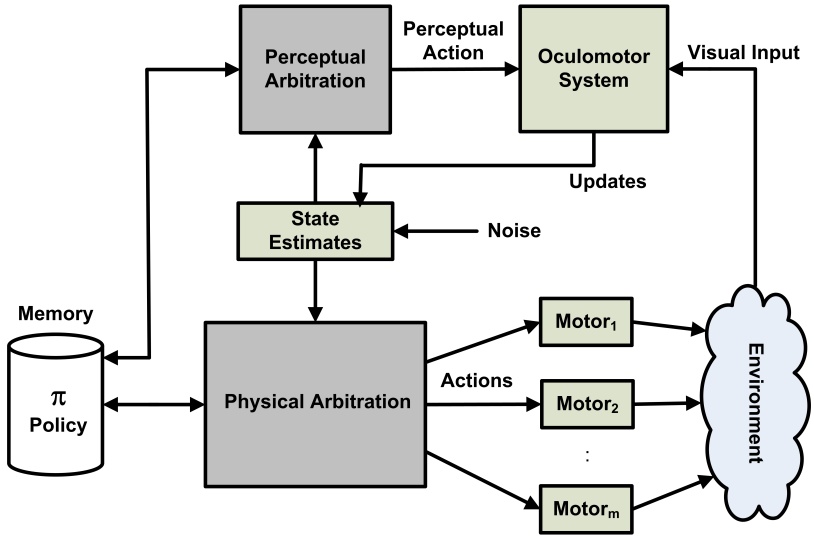

Fig. 2. Interaction of the system's components.

- Physical arbitration is in charge of selecting the physical actions that should be performed every decision epoch. This selection is based on the policy and the belief about the state of the environment.
- The selected actions are sent to the corresponding motor system (e.g. left arm or right arm) and are executed in parallel. Actions will likely change the environment.
- At the same time, perceptual arbitration selects amongst the perceptual actions currently available. Each perceptual action defines a location at which the robot's gaze can be fixated. The idea is to choose the perceptual action that maximises the value of the physical action with the highest risk of losing reward.
- The perceptual action makes the oculomotor system move the cameras towards a specific part of the environment. Here we consider a stereoscopic visual system.
- The oculomotor system extracts relevant information from the visual input to update the current state of the environment.
- The current state of the environment is an estimate of the true state, and is degraded due to noise and the cummulative error in the estimate.

3.2 Modelling the Task

The task is modelled using a variation of the CAM formulation[2]. The model is formalised as a tuple $\langle \mathcal{S}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{T} \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{O}$ is the set of options (temporally extended actions), $\mathcal{P} : \mathcal{S} \times \mathfrak{p}(\mathcal{O}) \times \mathcal{S} \times \mathbb{N} \rightarrow [0, 1]$ is the

[2] For a detailed explanation of the model please refer to chapter 3 of [10]

transition probability distribution, where $\mathfrak{p}(\mathcal{O})$ is the power set of options and $\mathbb{N}$ is the set of natural numbers, $\mathcal{R} : \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ is the reward function, and $\mathcal{T}$ is the concurrent termination mechanism. The power set allows us to select a set of options every decision epoch, which we call a *multi-option*. The set of natural numbers is used to define the time it takes to execute each option.

Because CAM is based on SMDPs, it is possible to use SMDP Q-learning [1] for learning. A policy is defined as the mapping from states to multi-options: $\pi : \mathcal{S} \to \mathfrak{p}(\mathcal{O})$. A multi-option contains options that are executed in parallel and are mutually exclusive.

3.3 Learning Phase

Learning takes place at two different levels of abstraction:

- At the higher level, the robot learns the task modelled by CAM using SMDP Q-learning.
- At the lower level, the robot learns the policy for each option used at the higher level.

Each option is modelled as $\mathcal{O} = \langle \mathcal{M}, \mathcal{I}, \beta \rangle$, where $\mathcal{M}$ is an MDP, $\mathcal{I}$ is the initiation set $\mathcal{I} \subseteq \mathcal{S}$, denoting the set of states in which the option can be initiated, and $\beta : \mathcal{S} \to [0, 1]$ represents a termination condition.

An MDP $\mathcal{M}$ is formalised as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of single-step actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function represented as $\mathcal{P}(s, a, s')$ being the probability of transition from state s to state s' after action a is executed, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function written as $\mathcal{R}(s, a)$ being the reward of taking action a in state s.

Q-learning [15], a reinforcement learning algorithm, is used for learning each option. A policy at this level is defined as the mapping from states to single actions.

3.4 Physical Action Selection

Having a policy, and assuming the state of the environment is always known, the robot just needs to follow such policy by checking the current state of the environment and executing the best action for that particular state. However, adding partial observability to the problem means that the current state of the environment is not completely known. Thus, the robot needs to decide what to do based on an estimate of the true state of the environment.

The problem is to create a selection mechanism that takes into account the belief of being in a particular state. Furthermore, this action selection must be done at both levels, first the robot needs to select a multi-option, and then, select single-step actions for each option.

For both levels, action selection can be done based on the Q-MDP algorithm used to find approximate solutions to POMDPs [2, 5]. First, the robot selects a multi-option according to:

$$\mathbf{o}_E = \arg\max_{\mathbf{o}} \sum_s bel(s) Q(s, \mathbf{o}) \tag{1}$$

Where $\mathbf{o}_E$ represents the multi-option with the highest expected reward, according to the belief $bel(s)$ of being in state s, and $Q(s, \mathbf{o})$ is the Q-value previously learnt. A multi-option is represented using bold face (e.g. $\mathbf{o}$).

Once a multi-option is chosen, we need to select the single-step actions for each active option:

$$a^i_E = \arg\max_{a} \sum_{s^i} bel(s^i) Q(s^i, a) \tag{2}$$

Because each option i have a different duration it is possible that one option in a multi-option finishes before the others. Therefore, every time an option terminates the control passes to the higher level. However, the action selection at this moment will have to take into consideration the remaining unfinished options. It is possible to resume the execution of an option if it is modelled as a Markov option[3].

3.5 Perceptual Coordination

The selection of good physical actions depends almost entirely on having the correct state information. Thus, it is essential to have a good mechanism for the selection of perceptual actions. We follow an approach similar to [11].

As explained above, perceptual actions are not learnt but planned. Since there is only one oculomotor system, it must be shared amongst the different actions currently being executed. The problem is of deciding where to direct the cameras in order to gather relevant information about the environment.

Similarly to the selection of physical actions, perceptual actions can be chosen at both levels. At the higher level, perceptual coordination is given by:

$$p_H = \max_{p} \sum_{\sigma} \left[P(\sigma \mid bel, p) \max_{\mathbf{o}} \sum_s bel^{\sigma,p}(s) Q(s, \mathbf{o}) \right] \tag{3}$$

In words it means to pick a perceptual action that maximises the expected return of the multi-option that is going to lose more reward if it is not given access to perception. $P(\sigma \mid bel, p)$ is the probability of having observation σ given the current belief and taking perceptual action p. And $bel^{\sigma,p}(s)$ is the belief that results assuming perceptual action p is taken and observation σ occurs. At this level, perceptual actions can be more general, e.g. *look at the table* instead of looking at a specific object.

At the lower level, perceptual coordination is defined with a similar equation but now taking into account the single-step action for each option being executed:

[3] The Markov property means the present state determines the future states, i.e. the past is not taken into account.

$$p_L = \max_{p} \sum_{\sigma^i} \left[P(\sigma^i \mid bel, p) \min_{i} \max_{a^i} \sum_{s^i} bel^{\sigma^i, p}(s^i) Q(s^i, a^i) \right] \tag{4}$$

Here the robot will pick the perceptual action p_L, such that the value of the "worst" action any option i picks is maximised.

By considering camera movements as perceptual actions, it is necessary to define the different locations at which the robot might want to look at. Furthermore, these locations should be generated during execution time, since for each instance of the task objects have different locations. This is also one of the reasons why perceptual actions are not learnt.

Because our main concern is testing the coordination mechanism, we will make the assumption that the object locations are known to the robot. Thanks to the simulator we can know the locations of the objects and we can provide this information to the robot. Thus, there is no need at this moment for the robot to process visual input. Our main goal is for the robot to fixate on the object that is essential at any given moment in order to increase its certainty about features of the object (e.g. its location, orientation, category, etc.). This information will be modelled as probability distributions.

4 Experiments

For now, experiments are being carried out using the iCub simulator [8]. The advantage of simulation is that it is possible to have higher control of the robot and the environment (Figure 1 shows a snapshot of the simulator).

The first task we are interested in solving is simple but enough to test our coordination framework. The goal is to pick all the objects from the table and then put them inside either one of the containers. By not restricting the container in which the objects should be placed, we avoid possible collisions with the arms and the need to switch the object from one hand to the other. Nevertheless, our aim is to model also this latter problem and more complicated tasks (e.g. the colour of the object could determine in which container should be placed).

There are several properties to evaluate in our model. The most important one is the coordination mechanism, in particular the decision of which perceptual action to perform. To evaluate our model we need to compare it with other decision mechanisms, e.g. random selection of perceptual actions and scheduling algorithms, such as round robin.

Concurrency is another property to be evaluated. In theory, concurrent actions seem a better choice for a robot with multiple DoF. This also depends on the tasks, some tasks are sequential in nature. For evaluation we can compare our model with its sequential counterpart.

5 Conclusions

The problem of how to coordinate gaze and actions in a robot is posed in terms of a concurrent decision making framework, where camera movements are a

consequence of the current uncertainty about the environment and the reward expected to be recieved by performing some actions. Here, we have defined the problem and proposed a framework to solve such problem.

The implementation of such framework is been realised at the moment and experiments will soon be carried out. We are confident that the results to be obtained will provide a more efficient way for the coordination of physical and perceptual actions.

References

[1] Bradtke, S., Duff, M.: Reinforcement learning methods for continuous-time Markov decision problems. Advances in Neural Information Processing Systems pp. 393–400 (1995)
[2] Cassandra, A.: Exact and approximate algorithms for partially observable Markov decision processes. Ph.D. thesis, Brown University (1998)
[3] Frintrop, S.: VOCUS: A visual attention system for object detection and goal-directed search. Springer-Verlag New York Inc (2006)
[4] Frintrop, S., Jensfelt, P.: Attentional landmarks and active gaze control for visual SLAM. IEEE Transactions on Robotics 24(5), 1054–1065 (2008)
[5] Hauskrecht, M., Meuleau, N., Kaelbling, L., Dean, T., Boutilier, C.: Hierarchical solution of Markov decision processes using macro-actions. In: Proc. 14th Annual Conference on Uncertainty in AI (UAI-98). pp. 220–229 (1998)
[6] Kaelbling, L., Littman, M., Cassandra, A.: Planning and acting in partially observable stochastic domains. Artificial Intelligence 101(1-2), 99–134 (1998)
[7] Mausam: Stochastic planning with concurrent, durative actions. Ph.D. thesis, University of Washington (2007)
[8] Metta, G., Sandini, G., Vernon, D., Natale, L., Nori, F.: The iCub humanoid robot: an open platform for research in embodied cognition. In: Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems. pp. 50–56. ACM (2008)
[9] Puterman, M.: Markov decision processes: discrete stochastic dynamic programming. Wiley Series in Probability and Mathematical Statistics (1994)
[10] Rohanimanesh, K.: Concurrent decision making in Markov decision processes. Ph.D. thesis, University of Massachusetts Amherst (2006)
[11] Sprague, N.: Learning to coordinate visual behaviors. Ph.D. thesis, The University of Rochester (2004)
[12] Sutton, R., Barto, A.: Introduction to reinforcement learning. MIT Press Cambridge, MA, USA (1998)
[13] Sutton, R., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial intelligence 112(1), 181–211 (1999)
[14] Vogel, J., Murphy, K.: A non-myopic approach to visual search. In: Computer and Robot Vision, 2007. CRV'07. Fourth Canadian Conference on. pp. 227–234 (2007)
[15] Watkins, C., Dayan, P.: Q-learning. Machine learning 8(3), 279–292 (1992)
[16] Younes, H.: Verification and planning for stochastic processes with asynchronous events. Ph.D. thesis, Carnegie Mellon University (2005)

Learnt Novelty with Overlapping Neural Cell Assemblies

Kailash Nadh

School of Engineering and Information Sciences, Middlesex University,
London, United Kingdom
k.nadh@mdx.ac.uk
http://kailashnadh.name/research

Abstract. Cell Assemblies (CAs) are central to many higher order cognitive processes such as perception, recognition, and recollection. Concepts are encoded as neuronal CAs in mammalian cortical areas, where CA formation and learning happen based on the Hebbian CA theory. Multiple CAs may share a common subset of neurons that produce dynamics that distinct, localist, orthogonal CAs do not exhibit, such as inherent support for generalisation and learnt novelty. This paper discusses implicit learning of novel behaviour in overlapping CAs, with a simulated agent based on a biologically realistic neural CA model.

Keywords: Artificial Intelligence, Neural Networks, Cell Assemblies

1 Introduction and Background

The CA [1] is the neural basis of the fundamental cognitive process of associative memory, and is the basis of many higher order phenomena [2], [3], [4], [5], [6], [7]. It is a reverberating circuit of spatially distributed groups of neurons that have high mutual synaptic strength [8]. Formation of CAs account for long term memory and their reverberating behaviour accounts for short term memory. CAs are learnt by a Hebbian learning rule, whereby modifications in the synaptic transmission efficacy are driven by correlations in the firing activity of pre-synaptic and post-synaptic neurons [9]. That is, repeated co-activation of neurons by a stimulus causes an increase in their mutual synaptic strength leading to the formation of a CA that is bound to the stimulus.

A CA may be activated when a subset of its neurons fire. The high mutual synaptic strength of its constituent neurons may cause it to reverberate as its neurons undergo cascades of activation, even after the stimulus that triggered it is removed. If the group of firing neurons happen to belong to more than one CA, activity may spread to other CAs. Prolonged co-activation may cause different CAs to merge, and learnt lateral inhibition may cause certain parts of a CA to overpower other parts, eventually forming separate CAs as a result of competitive learning. Such behaviour of overlapping CAs exhibit many interesting dynamics that disjoint CAs do not, such as the inherent ability to learn novel concepts and behaviour.

This paper discusses the nature of such overlapping CAs with a simulated agent that acquires novel behaviour via implicit, unsupervised learning. The agent is capable of moving in a virtual environment in three directions. The agent capable of animation is chosen so as to better visualise learnt novelty.

2 Network Architecture

The model discussed in this paper uses a neural network with simulated CAs based on *fatiguing Leaking Integrate and Fire* (fLIF) neurons that share many common characteristics with biological neurons [10]. fLIF neurons are an extension of the LIF (Leaky Integrate and Fire) model [11], [12]. fLIF neurons collect activation from pre-synaptic neurons and fire on surpassing a threshold, that is, they integrate and fire [13]. On firing, a neuron loses its activation level. Otherwise, the activation leaks gradually, resembling the behaviour of biological neurons.

The activation A of a neuron i at time t is:

$$A_{i_t} = \frac{A_{i_{t-1}}}{\delta} + \sum_{j \in V_i} w_{ij} s_j \tag{1}$$

The current total activation is the remnant activation from the last time step divided by decay factor δ, plus incoming activation. This new activation is the sum of the active inputs s_j of all neurons $j \in V_i, V_i$ being the set of all neurons connected to i that fired at $t-1$, weighted by the connection from neuron j to i. The neuron fires when the accumulated activation A exceeds a threshold θ. Firing is a binary event determined by a Heaviside function, and activation of w_{ij} is sent to all neurons j to which the firing neuron i has a connection. Fatiguing causes the threshold to be dynamic, $\theta_{t+1} = \theta_t + F_t$. F_t is positive (F_+) if the neuron fires at t and negative (F_-) if it does not. An increase in threshold causes the total amount of activation required for a neuron to fire to increase. This makes it difficult for a neuron to fire after many steps of frequent firing. Similarly, the threshold of a neuron decreases with each step it does not fire.

The simulated agent is driven by a network of 4000 fLIF neurons with distance biased connections [10]. The network is a rectangular array of neurons with distance organised toroidally. This topology makes it likely for a neuron to have excitatory connections to neighbouring neurons, and less likely to far away ones. Inhibitory connections in the network are set up randomly. The connectivity rule for excitatory neurons is given by Equation 2. There exists a connection between neuron i and j of a network only if $C_{ij} = 1$.

$$\begin{aligned} C_{ij} &= 1, \text{if } r < (1/(d * v)) \\ C_{ij} &= 0, \text{if not} \end{aligned} \tag{2}$$

where r is a random number between 0 and 1, d is the neuronal distance and v is the connection probability. This indicates that connections in a network are

influenced by distance between neurons and the connection probability factor. Distance $d = 5$ throughout all the simulations, as it has been observed to work well. Inspired by the biological neural topology, long distance intra-network connections are also present, connected by long distance axons with many synapses [15].

3 Learning Cell Assemblies

Learning in the network is driven by a correlatory Hebbian learning rule [1],[16], by which synaptic connection weights are modified based on the following equations:

$$\Delta^{+} w_{ij} = (1 - w_{ij}) * \lambda \tag{3}$$

$$\Delta^{-} w_{ij} = w_{ij} * -\lambda \tag{4}$$

w_{ij} is the synaptic weight from neuron i to j and λ is the learning rate. During each cycle, weights change based on the state of pre-synaptic and post-synaptic neurons. If both neurons co-fire, the weights increase as per the Hebbian rule (Equation 3). If only the pre-synaptic neuron fires, weights decrease as per the anti-Hebbian rule (Equation 4). Thus w_{ij} changes and approximates k, the likelihood of j firing if i fires.

Each neuron in the network is either excitatory or inhibitory, where all synapses leaving the neuron are either excitatory or inhibitory [17]. A similar learning rule applies to inhibitory neurons that makes the synaptic weight approximate $k - 1$, where k is the likelihood that the post-synaptic neuron fires when the pre-synaptic neuron fires.

Thus, the recruitment of neurons to different CAs may be due to repeated co-presentation of similar or ambiguous stimuli that increase their mutual synaptic strength. This suggests that events that tend to co-occur should somehow belong together. Every time these events occur in conjunction, they drive certain subgroups of neurons to fire in correlation, resulting in the association of the respective events [8]. Shared bit patterns in the Hopfield model are examples of a computational CA model [18].

Prior models based on disjoint CAs have been used to encode spatial maps, sequential, and many to many associations [10], but they show a relatively high degree of deterministic behaviour that arises from the limited ability to acquire novelty by learning due to the disjoint topology of learnt CAs. Furthermore, biological CAs in the brain are known to be of overlapping nature [14].

4 The Simulated Agent

The agent is able to perform three actions, move up, move right and move left, driven by three CAs. The CAs are labelled **UP**, **RIGHT** and **LEFT** respectively. Each action is executed based on the activation level of its corresponding

CA. For instance, if **UP** has 20% of its neurons active, the agent moves up 20% of a unit distance.

Since all three CAs are in the same network, they have excitatory and inhibitory connections with each other. Training runs for 600 cycles [1], where patterns corresponding to **UP**, **RIGHT** and **LEFT** are presented for 200 cycles in succession, so that their respective CAs are learnt. The test phase lasts for 300 cycles, where random neurons of each of the three CAs are excited for 100 cycles each. That is, at cycle 0, **UP** is excited; at cycle 100, **RIGHT** is excited; at cycle 200, **LEFT** is excited. During the test phase, the agent executes movements based on the activity levels of CAs corresponding to the three actions.

The test is run in two separate modes; one with disjoint, orthogonal CAs, and one with overlapping CAs. The agent's movements in each mode is recorded so as to compare the behaviour emerging from orthogonal and overlapping CAs. The simulation was run on ten different network configurations and the results were found to be consistent. The results from one such run are presented below.

4.1 Behaviour from Orthogonal CAs

The patterns for each of the three CAs are presented for 200 cycles each in succession. The CAs are disjoint and do not share neurons. Each CA consists of 33% of the neurons of the entire network. The test was repeated on 10 different network configurations and the paths taken by the agent were found to be consistent 100% of the time. Figure 1 shows the path taken by the agent in the 300 cycles of test on one such run.

Since the first CA to be excited is **UP**, at the 0th cycle, the agent starts moving **UP**. Similarly, at the 100th cycle, the agent starts moving **RIGHT**, and at the 200th cycle, the agent starts moving **LEFT**. Each of these actions continue for 100 cycles. The disjoint nature of the CAs mean that the absence of shared neurons cause an active CA to suppress other CAs via learnt inhibitory connections. When the **UP** CA is active, the agent is solely moving **UP** without any deviations as other CAs are completely inactive. This is indicated by the linearity of the movements plotted in the figure. The dotted line marks the centre of the environment, at the base of which the agent starts.

4.2 Behaviour from Overlapping CAs

In the mode with overlapping CAs, the patterns share a 25% overlap of their constituent neurons with each other, where **UP** and **RIGHT** overlap, and **RIGHT** and **LEFT** overlap. That is, **UP** and **RIGHT** share 25% of each other's neurons, and **RIGHT** and **LEFT** share 25% of their neurons. **UP** and **LEFT** do not physically overlap, but over the course of learning, they form shared sub-CAs via their overlaps with **RIGHT**.

[1] One cycle is 10 milliseconds in simulated time, when the neural network is updated based on the Hebbian learning rule (Section 3).

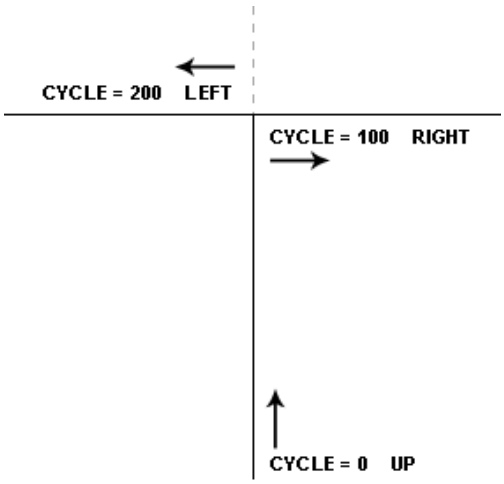

Fig. 1. Path taken by the agent with orthogonal CAs

The patterns for each of the three CAs are presented for 200 cycles each in succession in the training phase, so that their corresponding CAs are learnt. In the test phase, each of the three CAs are partially excited for 100 cycles one after the other.

Figure 2 shows the path taken by the agent over the course of the test. The path is visibly erratic compared to that in Figure 1, and the agent seems to have moved **UP**, **RIGHT**, and **LEFT** at the same time at relatively the same rate. The largest deviations in the path are at cycles 0, 100 and 200 respectively, when the corresponding CAs receive external excitation. Over the next few cycles, the movement settles into novel pseudo-stable states which seem to be combinations of all three CAs.

5 Discussion and Conclusion

The simulated agent with overlapping CAs has been shown to develop novel concepts from the three base CAs it was trained with. The resulting behaviour is dynamic, compared to the relatively deterministic behaviour from orthogonal CAs. The novel pseudo-stable CAs formed from the overlaps change over the course of time, via implicit learning.

The CA model used in the simulation is based on fLIF neurons that have biological plausible characteristics such as distanced biased topology, and excitatory and inhibitory properties. CAs in the network are learnt gradually over a course of many cycles via external stimuli.

The agent driven by overlapping CAs manages to learn implicit sub-actions dynamically from varying combinations of the base actions **UP**, **RIGHT** and **LEFT**, as indicated by the nature of its movement. Figure 2 shows movements

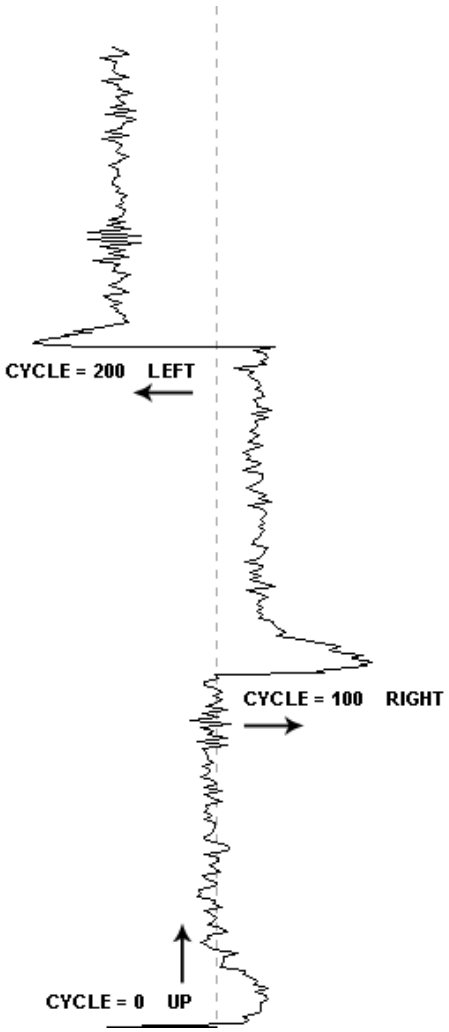

Fig. 2. Path taken by the agent with overlapping CAs

such as **UP-RIGHT**, **UP-LEFT**, **UP-RIGHT-LEFT** and **LEFT-RIGHT**. Even though there is no physical overlap between **UP** and **LEFT**, **UP-LEFT** seems to have emerged from their interactions with **RIGHT**. Also, the strength of these novel pseudo-stable combinations appear to vary dynamically over the course of time, where for instance, **UP-RIGHT** is dominant compared to **UP-LEFT** at certain points.

Overlapping CAs overcome the relatively deterministic behaviour that disjoint CAs (or other similar models) exhibit. They introduce a certain degree of randomness that eventually settle to pseudo-stable sub-states leading to the

emergence of many interesting dynamics, which many other esoteric models seem to lack. Another benefit of overlapping CAs are their inherent support for generalisation [19], where prototypicality emerges from CAs that share common attributes. Also, neurons shared between different CAs have the ability to participate in different kinds of information processing [20]. Overlapping CAs may also help overcome capacity constraints in neural networks, where shared neurons make it possible to have more CAs than the total number of neurons in a network [21].

While the above discussed simulation demonstrates how overlapping CAs can implicitly acquire novelty, other simulations have used overlapping CAs to model different cognitive tasks [22], [23]. It is also suggested that human cortical semantic memories are distributed and overlapping in nature [24]. This would enable unique episodical memories and context-free information to be encoded as a part of a larger semantic network, where overlapping patterns may make possible distributed access to the entire knowledge structure [25].

It seems likely that overlapping CAs, with their inherent ability to learn novelty among many others, are important for implementing biologically realistic neural models. This work is intended at furthering the understanding of dynamics of neural processing.

Acknowledgements. This work was partially supported by EPSRC grant GR/R13975/01.

References

[1] Hebb, D.O.: The organization of behavior. Wiley, New York (1949)
[2] Harris, K.: Neural signatures of cell assembly organization. Nature Reviews Neuroscience 6, 399–407 (2005)
[3] Maurer, A.P., Cowen, S.L., Burke, S.N., Barnes, C.A., McNaughton, B.L.: Organization of hippocampal cell assemblies based on theta phase precession. Hippocampus 16(9), 785–794 (2003)
[4] Pulvermuller, F.: Words in the brain's language. Behavioral and Brain Sciences 22, 253–336 (1999)
[5] Pasupathy, A., Connor, C.: Population coding of shape in area v4. Nature Neuroscience 5:12, 1332–1338 (2002)
[6] Funahashi, S.: Neuronal mechanisms of executive control by the prefrontal cortex. Neuroscience Research 39, 147–165 (2001)
[7] Foster, J., Alexander, G.: Neuron acitivity related to short-term memory. Science 173, 652–654 (1971)
[8] Wennekers, T., Palm, G.: Cell assemblies, associative memory and temporal structure in brain signals. In: R.Miller, editor, Time and the Brain. Conceptual advances in Brain Research, Vol. 2. pp. 251–274. Harwood Academic Publishers (2000)
[9] Gerstner, W., Kistler, W.K.: Mathematical formulations of Hebbian learning. Biological Cybernetics 87(5-6), 404–415 (2002)

[10] Huyck, C.R., Nadh, K.: Multi-associative memory in fLIF cell assemblies. In: A. Howes, D. Peebles, R.C.E. (ed.) 9th International Conference on Cognitive Modeling - ICCM2009. pp. 81–87. Manchester, UK (2009)
[11] Gerstner, W.: Integrate and fire neurons and networks. In: The handbook of brain theory and neural networks, pp. 577–581. The MIT Press (2002)
[12] Maas, W. and Bishop, C. M.: Pulsed Neural Networks. MIT Press (2001)
[13] McCulloch, W. S. and Pitts, W.: A logical calculus of the ideas immanent in nervous activity. In: Bulletin of Mathematical Biophysic, 5:115–133 (1943)
[14] Sakurai, Y.: The search for cell assemblies in the working brain. Behavioural Brain Research 91, 1–13 (1998)
[15] Churchland, P., Sejnowski, T.: The computational brain, MIT Press (1992)
[16] Huyck, C.R.: Overlapping cell assemblies from correlators. Neurocomputing 56, 435–9 (2004)
[17] Eccles, J.C.: Chemical transmission and dale's principle. Progress in Brain Research 68, 3–13 (1986)
[18] Hopfield, J.J.: Neurons with graded response have collective computation properties like those of two-state neurons. In: Proceedings of the National Academy of Sciences. vol. 81, pp. 3088–3092 (1984)
[19] Nadh, K., Huyck, C.R.: A pong playing agent modelled with massively overlapping cell assemblies. (Manuscript submitted for publication).
[20] Sakurai, Y.: How do cell assemblies encode information in the brain? Neuroscience & Biobehavioral Reviews 23(6), 785 – 796 (1999)
[21] Wickelgren, W.A.: Webs, cell assemblies, and chunking in neural nets. Canadian Journal of Experimental Psychology 53(1), 118–131 (1999)
[22] Garagnani, M., Wennekers, T., Pulvermuller, F.: Recruitment and consolidation of cell assemblies for words by way of hebbian learning and competition in a multi-layer neural network. Cognitive Computation 1(2) (2009)
[23] Huyck, C.R.: Creating hierarchical categories using cell assemblies. In: Connection Science. vol. 19:1, pp. 1–24 (2007)
[24] Karalyn, P., Peter J. Nestor, T.T.R.: Where do you know what you know? the representation of semantic knowledge in the human brain. Neuroscience 8(12), 976–987 (2007)
[25] Eichenbaum, H.: How does the brain organize memories? Science 277(5324), 330–332 (1997)

Morpheme segmentation in a non-parametric Bayesian framework

Santa Basnet

Department of Computer Science

University of York, Deramore Lane, York, YO10 5GH, UK

santa@cs.york.ac.uk

Abstract. This paper discusses work on morphological segmentation under a non-parametric Bayesian framework. We show that the statistical properties of text can be captured by employing the cache of morpheme frequencies. We analyze this system for English and Nepali text. Standard test dataset for English is readily available as part of the Morpho Challenge experiment. However such a resource does not exist for Nepali. This paper also describes the dataset we have constructed for Nepali.

1 Introduction

Morphology is study of the internal structure of words. Morpheme segmentation is the process of segmenting words into their constituent segments. These segments are stems (or roots) and affixes. Linguistically, word must have at least one stem. The affix present before and after the stem is known as prefix and suffix respectively. A word can have more than one affix. In the case of compound words, more than one stems are present.

Table 1. Morphological structure of English word "unsegmented" and similar Nepali word "नटुक्रयाइएको (indirect form)".

Word	Segments(Morphs)
unsegmented	un(prefix) + segment(stem) + ed(suffix)
नटुक्रयाइएको	न (prefix) + टुक्र (stem) + आइ (suffix) + एको (suffix)

We consider the process of adding affixes to the stem as the primary way to induce word variations. In this paper, we present an unsupervised approach to morpheme induction utilizing the frequency of morphemes derived from the corpus. It is a well established notion that words in a corpus follow a power law distribution [18]. Dirichlet Process (DP) is the distribution over distributions. The same distribution can be achieved by Chinese Restaurant Process (CRP) [17]. CRP provides the mathematical framework of producing power law distribution.

This paper is organized as follows. Section 2 reviews the related work on unsupervised morphology learning. Section 3 describes the learning model and the data structure of the system. Section 4 describes training and testing phase with evaluation of results. Finally Section 5 provides discussion on the results and further extension. Section 6 concludes the paper.

2 Related Work on Unsupervised Morphology

There has been lot of work done in the area of unsupervised learning of morphology. Earlier work [1, 2, 3], used statistical properties such as successor and predecessor variety count of words in a corpus to indicate the morpheme boundaries. Additionally, the transitional probabilities of word appearing as substring of another word used to detect the morpheme boundaries [4]. Further, the basic RePortS algorithm [4] is extended by employing better morpheme induction and segmentation strategies. In [9], the equivalence sets of letters are generated with the help of small edit distance word pairs and improve the results for German language. In [10, 11], language specific heuristics are added for composite suffix detection, incorrect suffix attachments and orthographic changes during model learning.

The other well known approaches in unsupervised morphology learning are based on the application of the Minimum Description Length (MDL) principle [12]. In [5], MDL prior is used to represent the optimal segments of the corpus. Further, [6] uses the prior distributions of morpheme frequency and morpheme length to measure the goodness of induced morphemes. The work [7], focuses on the stem+suffix languages. The stem forms group called signature and each signature shares a set of possible affixes and uses the MDL techniques for model optimization.

The probable morphologically related word pairs are generated using orthographic similarity in terms of edit distance and semantic similarity in terms of mutual information (MI) measures [8]. The stem-suffix paradigms for a language are induced by grouping all stem+suffix pairs of words [13]. The induced paradigms help to analyze the given word as certain or possible for segmentation. In [14], the paradigm are learned using the syntactic information i.e. PoS of word. In [16], the word families (clusters) of morphologically similar words are identified by building a graph where the nodes are words and the edges are the transformational rules. The community detection method described by Newman [15] is used for the identification of word-families. The given words are segmented with the help of these induced families and the morphological transformation rules.

In [21], the non-parametric Bayesian framework is defined to learn the Probabilistic Context Free Grammar (PCFG) rules through Adapter Grammar. In morpheme segmentation, these rules generate the morphemes and the adapter effectively learns the morphemes of its language. It uses the adapter based on Pitman-Yor process [17] to specify the distributions used in non-parametric Bayesian statistics such as Dirichlet process [17, 19, 20]. In [22], the adapter

grammar framework is used in unsupervised word segmentation for Sesotho. In [25], the adapters based on Chinese Restaurant Process and Pitman Yor Process model the distribution of words and are used for word segmentation.

3 Model Description

We develop following two models for the morphological segmentation.

3.1 Basic Model

This model is extremely simple and assumes that the words are independent units of language. Possible morpheme candidates are generated by splitting the word into all possible two parts. We further make the same independence assumption regarding the generated morpheme segments. These morphemes and morpheme weights are stored in cache. The weight of a morpheme is derived from the frequency of parent words.

The morpheme segments in this model are represented as the menu items in a restaurant as shown in fig. 1. The words are analogous to the customers and can choose the item from the listed menu or can choose new item. We split a word in all possible two parts. If the proposed segment is available in the cache of menu list then the model choose it from listed menu and if not, it will select the segment as new menu item and will add to the menu list. Initially there is no item in the cache of menu list (morphemes).

Let $m_1,...,m_k$ be the morphs present in the cache. Dirichlet Process sampler uses the following relation to define the predictive probability of next morph m_{k+1}:

$$P(m_{k+1} \mid m_1,...,m_k,\alpha,G_0) = \begin{cases} \frac{N_{k+1}}{\alpha+H}, & \text{If } m_{k+1} \in \text{cache}. \\ \frac{\alpha}{\alpha+H}, & \text{If } m_{k+1} \notin \text{cache} \text{ then } \text{cache} := \text{cache} \cup \{m_{k+1}\}.\ m_{k+1} \sim G_0. \end{cases} \quad (1)$$

where, H is count of previously processed morphemes, N is the number of parent words contained by the morph m_{k+1} and G_0 is the base distribution. The concentration parameter $\alpha > 0$ determines the variance in the probabilities of morphs i.e. higher the value of α, higher number of morphs according to the number of input words. This part captures the morphemes frequency behaviour in the corpus which obeys the power law distribution. Equation (1) can be derived using conjugacy of Multinomial-Dirichlet distribution [20]. Hence, the final probability of morpheme segment is given by equation (2).

$$P(m_{k+1} \mid m_1,...,m_k,\alpha,G_0) = \frac{\alpha \times G_0 + N_{k+1}}{\alpha+H} \quad (2)$$

For example, all the possible two parts segments of word "chunked" are: c+hunked, ch+unked, chu+nked, chun+ked, chunk+ed, chunke+d, chunked+$. We calculate the probability of each using equation (2). For example, the probability of "chunk+ed" is:

$$P(w = chunk + ed \mid m_1,...,m_k, \alpha, G_0) = \frac{\alpha \times G_0 + MorphWeight(chunk)}{\alpha + H} \times \frac{\alpha \times G_0 + MorphWeight(ed)}{\alpha + H}$$

Value of α is constant and G_0 is uniform for all morpheme segments. Initially, H=0 and is updated according to the number of words processed during the learning. For a word with n possible splits, we compute the probability of each split $P_1,...,P_n$. For example, for the word "chunked", we get 7 possible split probabilities $P_1,...,P_7$. We then normalize the n probabilities obtained for all splits and calculate the weighted probability relation (3).

$$Weighted\ Probability = Normalized\ Probability \times Frequency\ of\ word \quad (3)$$

where, $Normalized\ Probability = \frac{P_k}{\sum_{i=1}^{n} Pi}$ for all $1 \le k \le n$.

Finally, we update the weighted probability in the cache for each morpheme segment as morpheme weight.

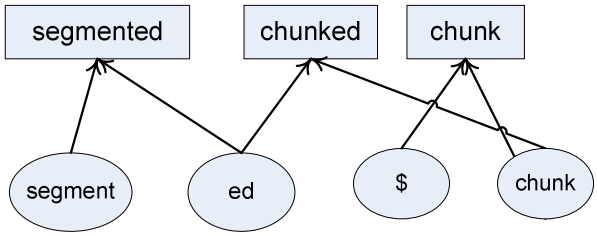

Fig. 1. Cache representation of morphs "$", "chunk", "ed", "un", "segment".

In Fig. 1, the boxes represent the word and ovals represent the morpheme segment. These are the part of cache. In oval, a morpheme "ed" is connected with "segmented" and "chunked". Hence, the weight for "ed" is derived from the words "segmented" and "chunked" and stored in cache. Similarly, the weights of other morphemes are derived.

3.2 Grammar Model

In the basic model, we consider the segments set and all morphemes to be in same cache model. In this section, we extend our model by distinguishing prefix, stem and suffix distribution of a word. We assume that a word can have prefix, stem and suffix in their successive positions and the probability is given by:

$$P(w = s+t \mid M_{prefix}, M_{stem}, M_{suffix}) = \begin{aligned} & P(s \mid M_{prefix}) \times P(t \mid M_{stem}) \\ & + P(s \mid M_{stem}) \times P(t \mid M_{stem}) \\ & + P(s \mid M_{stem}) \times P(t \mid M_{suffix}) \end{aligned} \quad (4)$$

In (4) M_{prefix}, M_{stem} and M_{suffix} are the respective morpheme models for prefixes, stems and suffixes. We built the respective cache for each model. Equation (4) can be divided into 3 parts and rewritten as,

$$P(w = s+t \mid M_{prefix}, M_{stem}, M_{suffix}) = P_1 + P_2 + P_3$$

where,

$$P_1 = P(s \mid M_{prefix}) \times P(t \mid M_{stem}),$$
$$P_2 = P(s \mid M_{stem}) \times P(t \mid M_{stem})$$
and
$$P_3 = P(s \mid M_{stem}) \times P(t \mid M_{suffix})$$

During model learning, we update the models M_{pref} and M_{stem} with probability P_1. Similarly we update the respective models with probability P_2 and P_3.

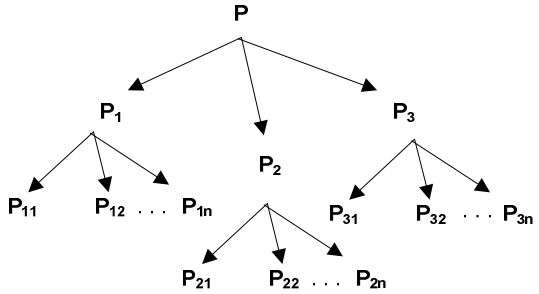

Fig. 2. Model update mechanism of a word having *n* segments set.

For example, in segments "chunk+ed", with probability P_1 "chunk" is considered as prefix and "ed" is considered as stem. Similarly with probability P_2, "chunk" is considered as stem and "ed" is considered as stem too. Finally, with probability P_3, "chunk" is considered as stem and "ed" is considered as suffix. In this way, we update the probabilities of *n* number of two parts segments in each model for a word as shown in fig. 2.

3.3 Segmentation

We have used two techniques to get the final segments of a word. In the first technique, segment with the highest weighted probability (using equation (3)) is selected. This is Maximum a Posteriori (MAP) estimation of segments given the

model. Initially, the weights of all two parts segments for a word are assigned to zero and updated by the weighted probability. After completion of all iterations, we choose the segment set having maximum weight for a word. This method will only give two segments of a word.

In second approach, we align the morpheme sequence using dynamic programming i.e. Viterbi alignment to find the best morpheme sequence according to weighted probability measure. In grammar model, we use the following criteria to validate the candidate morpheme sequence to be aligned.

Let $m_1, m_2, \ldots, m_n$ be the morpheme sequence of a word. Then,

a. At least one of m_i should be in a stem category.
b. Stems can alternate with prefixes and suffixes but m_n can't be prefix and m_1 can't be suffix of words.
c. For $1< i \leq k$, if m_i is a prefix, then m_{i+1} must be a stem or a prefix.
d. For $1 < i \leq k$, if m_i is a suffix, then m_{i-1} must be a stem or a suffix.

The same test strategies are applied in [10] & [11] to validate the segmentation of a word among its all possible segmentations.

4 Evaluation

4.1 Dataset Preparation

In our experiment for English, we used the training and test set provided by the Morpho Challenge [24]. The training set contains 167,377 distinct words from 24,447,034 words. In test set, it contains 532 distinct words.

In case of Nepali, we extracted our corpus from daily news papers as well as the weekly, half monthly and monthly magazines during 2009-10. We pre-processed it to create word-frequency list as input of our system. This corpus contains 819,506 distinct words from 44,856,552 words. We have tested our model on basic Nepali verbs only. The test set is prepared by manual segmentation of 1,500 randomly selected distinct words from the training set. We took out all the verbs as training set from the corpus which contains 34,391 distinct words.

4.2 Evaluation Metric

Evaluation scripts are provided by the Morpho Challenge [24] along with training and test dataset. We used the same evaluation metrics described in Morpho Challenge [24]. The Precision (P), Recall (R) and F-Measure (F) are calculated in terms of morpheme boundary detection. These are calculated using the following formula.

$$P = \frac{H}{H + I} \tag{5}$$

$$R = \frac{H}{H + D} \tag{6}$$

$$F = \frac{2H}{2H + I + D} \tag{7}$$

where, H is the count of correct boundary hits, I the count of boundary markers incorrectly positioned and D the count of boundary markers not placed. These counts are based on comparison with the gold standard. For example, if the word "sprinting" is segmented as "s+print+ing", where '+' is the boundary marker, then the first '+' is counted as an incorrect boundary insertion and the second '+' is counted as the correct boundary hit (comparing with the correct segmentation "sprint+ing"). Similarly, in case of "un+segmented", the first '+' is counted as a correct hit. It also counts the missing boundary in between "segment" and "ed". In our system, we counted all the insertion, deletion and correct boundary hits of the test words and used the above equations (5), (6) and (7) to evaluate Precision (P), Recall (R) and F-Measure (F) respectively.

4.3 Results

We performed the evaluation on both English and Nepali dataset described in 4.1. We applied our both systems described in sections 3 with this dataset. We set the value of α to 1000 for our experiments. We also set the minimum length of a word to 4 for segmentation. The evaluation metric described in 4.2 gives the below results.

Table 2. Evaluation results of the Basic Model.

Language	Precision (P)	Recall (R)	F-Measure (F)
English	56.84%	46.18%	50.96%
Nepali	51.25%	37.19%	43.10%

Table 3. Evaluation results of the Grammar Model assuming two segments only.

Language	Precision (P)	Recall (R)	F-Measure (F)
English	58.65%	47.60%	52.55%
Nepali	53.16%	38.12%	44.40%

Table 4. Evaluation results of the Grammar Model assuming the multiple segments.

Language	Precision (P)	Recall (R)	F-Measure (F)
English	45.81%	22.12%	29.83%
Nepali	53.92%	43.58%	48.20%

5 Discussion & Future Work

The above results are not good as compared to the current unsupervised morphological state of the art for English [27]. The best system in terms of F-Measure achieved 62.31% [23]. One possible explanation comes from the Zipfian nature of words in a corpus. So, the highly frequent words have greater influences thereby degrading the quality of segments and estimates based around identifying best segments. The result given by the grammar model is improved in the MAP estimation where the weights of frequently occurred morphemes are distributed among prefixes, stems and suffixes.

For Nepali there is no benchmark to compare our unsupervised approach. In our experiment the Grammar model shows improvement over the basic model. This is not the case for English where we see degradation in performance (see Table 2, 3 and 4). This is due to the nature of our test set where we have chosen verbs only and with a single stem.

Our model makes use of word frequencies alone as a way to capture the context which results in inefficient performance. For example, it leaves out many words unsegmented i.e. segmented with null suffix. Furthermore, the over segmentation error in best sequence alignment occurs due to the influence of most frequent morphemes. We plan to extend this system by employing the Hierarchical Dirichlet Process (HDP) [26] which should be able to overcome the influence of heavily weighted morphemes.

6 Conclusion

In this paper, we have described the task of morphological analysis by capturing the statistical properties of words. Our result shows that the morpheme behaviour follows the power law and this characteristic can be captured to segment the words into morphemes. However the same behavior can lead to over segmentation and we believe HDP to be able to reduce the bias. In addition to our model, to our knowledge, this paper describes the first results in unsupervised morphological segmentation for Nepali.

References

1. Harris, Z. S.: From Phoneme to Morpheme. In: Language, 31(2):190-222 (1955)
2. Hafer, M. A., Weiss, S. F.: Word segmentation by letter successor varieties, In: Information Storage & Retrieval, 10:371-385 (1974)
3. Dang, M. T., Chaudri, S.: Simple unsupervised morphology analysis algorithm (SUMAA). In proceedings of the PASCAL Challenges Workshop

on Unsupervised Segmentation of Words into Morphemes, Venice, Italy (2006)

4. Keshava, S., Pitler, E.: A simpler, intuitive approach to morpheme induction. In PASCAL Challenge Workshop on Unsupervised Segmentation of Words into Morphemes. (2006)
5. Creutz, M., Lagus, K.: Unsupervised discovery of morphemes. In proceedings of the ACL-02 workshop on Morphological and phonological learning - Volume 6 pages 21-30. (2002)
6. Creutz, M., Lagus, K.: Unsupervised Segmentation of Words Using Prior Distributions of Morph Length and Frequency. In proceeding of the 41st meeting of ACL-03, pages 280-287. (2003)
7. Goldsmith, J.: Unsupervised Learning of the Morphology of a Natural Language. Computational Linguistics, 27:2, pages 153-198. (2001)
8. Baroni, M., Matiasek, J., Trost, H.: Unsupervised discovery of morphologically related words based on orthographic and semantic similarity. In proceedings of the ACL workshop on Morphological and phonological learning Volume 6, pages 48 – 57. (2002)
9. Demberg, V.: A Language-Independent Unsupervised Model for Morphological Segmentation. In proceedings of the 45th Annual Meeting of the ACL, pages 920-927. (2007)
10. Dasgupta, S., Ng, V.: High-Performance, Language-Independent Morphological Segmentation. In proceedings of NAACL HLT 2007, pages 155-163. (2007)
11. Dasgupta, S., Ng, V.: Unsupervised word segmentation for Bangla. In proceedings of ICON, pages 15- 24. (2007)
12. Rissanen, J.: Stochastic Complexity in Statistical Inquiry. Word Scientific Publishing Co, Singapore. (1989)
13. Zeman, D.: Unsupervised Acquiring of Morphological Paradigms from Tokenized Text. In CLEF 2007. LNCS, vol. 5152, pp. 892–899. Springer, Heidelberg. (2007)

14. Can, B., Manandhar, S.: Unsupervised Learning of Morphology by Using Syntactic Categories. In Working Notes for the Cross Language Evaluation Forum (CLEF) Workshop, Corfu, Greece. (2009)
15. Newman, M.: Fast algorithm for detecting community structure in networks. Physical Review. E 69, 0066133. (2004)
16. Bernhard, D.: MorphoNet: Exploring the Use of Community Structure for Unsupervised Morpheme Analysis. Working Notes for the CLEF 2009 Workshop Corfu, Greece. (2009)
17. Wallach, H.M., Jensen, S.T, Dicker, L., Heller, K.A.: An Alternative Prior Process for Nonparametric Bayesian Clustering, Appearing in Proceeding of the 13th International Conference on AI and Statistics(AISTATS), Chia Laguna Resort, Sardinia, Italy. (2010)
18. Zipf, G.K.: Selected Studies of the Principle of Relative Frequency in Language.Cambridge, MA.: Harvard University Press. (1932)
19. Ishwaran, H., James, L.F.: Generalized weighted Chinese restaurant process for species sampling mixture models. Statistica Sinica, 13:1211-1235. (2003)
20. Zhang, X.: A Very Gentle Note on the Construction of Dirichlet Process. The Australian National University, Canberra. (2008)
21. Johnson, M., Griffiths, T.L., Goldwater, S.: Adaptor Grammars: A Framework for Specifying Compositional Nonparametric Bayesian Models. In Advances in neural information processing systems. (Vol. 19). (2007)
22. Johnson, M.: Unsupervised word segmentation for Sesotho using Adaptor. Grammars, Proceedings of the Tenth Meeting of the ACL Special Interest Group on Computational Morphology and Phonology, pages 20-27. (2008)
23. Virpioja, S., Kohonen, O.: Unsupervised Morpheme Discovery with Allomorfessor. In Cross Language Evaluation Forum (CLEF). (2009)
24. Unsupervised segmentation of words into morphemes - Challenge 2005, http://www.cis.hut.fi/morphochallenge2005/evaluation.shtml
25. Goldwater, S.: Nonparametric Bayesian models of lexical acquisition. Ph.D. thesis, Brown University. (2006)

26. Johnson, M., Blunsom, P., Cohn, T., Goldwater, S.: A note on the implementation of Hierarchical Dirichlet Process. Proceedings of the ACL-IJCNLP Conference Short Papers, pages 337-340, Suntec, Singapore. (2009)
27. Unsupervised segmentation of words into morphemes - Challenge 2009, http:// http://www.cis.hut.fi/morphochallenge2009/eng1.shtml

Author Index