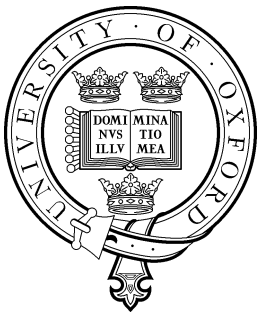
Computing Science

PROCEEDINGS OF THE OXFORD UNIVERSITY COMPUTING LABORATORY PROGRAMMING RESEARCH GROUP STUDENT CONFERENCE '06

Program Chair: Yong Xie
Organizational Chair: Mila Katzarova
Steering Committee Chair: Tom Melham, Duncan Coutts

CS-RR-06-05



Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Forward

The annual student conference of the Programming Research Group (PRG) at the Oxford University Computing Laboratory has been a forum for the presentation of some of the DPhil students' work that they have recently been doing. It is a pleasure to be able to take part in continuing this tradition.

The 16 abstracts in this volume were presented at the PRG student conference '06, held on October 13, 2006 in the Griffiths Room at Keble College. In addition to the students' presentations, the conference also featured an invited talk by Prof. Tom Melham (Director of Graduate Studies), as well as brief announcements of two posters.

The abstracts for the conference presentation were selected by the conference committee from among 19 submissions. The decision for the selection were based on the quality of the work, the effort the students have put in as perceived by the reviewers, and also the students' preferences for oral or poster presentation.

From among many outstanding presentations, Gwyn Skone won the Best Presentation Award; Brian Harrington and Heiko Helble were the runner-ups. The Best Poster Award was given to Elnar Hajiyev.

The organization of the conference reflects the efforts of a large number of people. I would also like to thank members of the conference committee and reviewers for the enormous amount of work they devoted to reviewing the submissions; Prof. Tom Melham and Duncan Coutts for the valuable guidance and help especially at the beginning; Mila Katzarova for her considerable work in organizing the event with me; and our sponsors the Computing Laboratory, and the part-time postgraduate studies in the software engineering program.

Finally, I hope that in reading over these proceedings you will share the sense that we had on the conference committee – that the conference, as a whole, serves as a learning process and research experience for the students, as well as a showcase for some of exciting research being carried out in the Computing Laboratory at Oxford University.

October 2006

Yong Xie
Program Chair
PRG Student Conference'06

Organization

PRG Student Conference'06 is organized by the Oxford University Computing Laboratory Programming Research Group.

Conference Committee

Program Chair:	Yong Xie
Organizational Chair:	Mila Katzarova
Steering Committee Chair:	Tom Melham (Honorary), Duncan Coutts

Referees

Phil Armstrong	Tom Melham	Blanca Rodriguez
Duncan Coutts	Luke Ong	Andrew Simpson
Ivan Flechais	Vasile Palade	

Sponsors

Oxford University Computing Laboratory
Software Engineering Program
Part-Time Postgraduate Studies

Table of Contents

Session 1: Security
Chair: Yong Xie

Trusted Delegation for Grid Computing 1
Andrew Cooper

On the Specification of Secure Transport Layers 3
Chris Dilloway

Efficient Group Authentication Protocol Based on Human Interaction 5
L. H. Nguyen and A. W. Roscoe

Session 2: Programming Tools and Bioinformatics
Chair: Yong Xie

Generated Program Monitors with Minimal Memory Overheads 7
Julian Tibble

Keeping Track of Partial Matches in Trace Monitoring 9
Pavel Avgustinov

Modular Interfaces for Program Components 11
Neil Ongkingco

Protein Structure Computation 13
Gwyn Skone

Session 3: Formal Methods
Chair: Mila Katzarova

Object-Oriented Specification and Verification 15
Eric Kerfoot

Game Semantics of the Safe Lambda-Calculus 17
William Blum

Towards A Unified Model For Workflow 19
Peter Y.H. Wong

Brief Announcement for Posters
Chair: Mila Katzarova

CodeQuest: Querying Source Code with Datalog 21
Elnar Hajiyev, Mathieu Verbaere, Oege de Moor, Kris de Volder

Verification and Optimisation of Cardiac Cellular Models 23
Jonathan Cooper

Session 4: Artificial Intelligence, Algorithms, Applications, etc
Chair: Mila Katzarova

Target Trajectory Prediction for an UAV Helicopter 25
Heiko Helble

An Example-Based Improvement to Statistical Machine Translation 27
James Smith

SaskNet: Automatically Creating a Large Scale Semantic Knowledge Network 29
Brian Harrington

Peer Production of Security Information 31
Max Loubser

Understanding Tractable Constraints 33
Zoltán Miklós

Extensible Non-Cooperative Games 35
Yong Xie

Author Index 37

Trusted Delegation for Grid Computing

Andrew Cooper

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
Andrew.Cooper@comlab.ox.ac.uk

1 Introduction to Trusted Computing

Trusted computing offers potentially great enhancements to the security of commodity systems by implementing a number of trusted operations in a hardware device known as the trusted platform module (TPM). The three main trusted computing functions are as follows:

- *Authenticated boot*: Authenticated boot aims to detect Trojan horses, viruses and other malicious code using mechanisms that resist software attack.
- *Protected storage*: Protected storage allows encryption keys to be stored securely within the TPM where they can only be used by authorised entities.
- *Attestation*: Attestation allows remote entities to determine that a platform is running in a trusted state.

2 A Compartmented Grid Architecture

We have developed a novel architecture for grid computations (refer to Figure 1) that provides strong security functions backed by trusted computing mechanisms. Virtualisation is used to create strong isolated compartments. Grid jobs are run within virtual machines where they are protected from attacks both by rival jobs and the host platform.

The job security manager performs a range of fundamental functions to protect both the data involved in processing grid jobs and the user credentials.

- The *delegation service* provides security policy enforcement for the use of credentials to facilitate restricted delegation.
- The *data migration service* ensures data is integrity and confidentiality protected from the grid services compartment that performs the data transfer.
- The *credential storage service* provides secure storage for credentials.
- The *data storage service* provides tamper-proof transparent encrypted storage for grid jobs so they are protected from compromise throughout the service provider network.
- The *digital rights management service* enforces limited access to data using security policy defined by data owners. For example, access to data can be time-limited using a consistent policy applied to all replicas throughout a distributed computation.

3 Existing Delegation Approaches

Delegation grants an entity the right to impersonate a user's identity to perform privileged operations on their behalf. Proxy certificates are commonly used to delegate user credentials within the grid.

With many copies of the delegated credentials throughout a distributed job, security is only as strong as the weakest link. A mis-configuration or unpatched component at any point

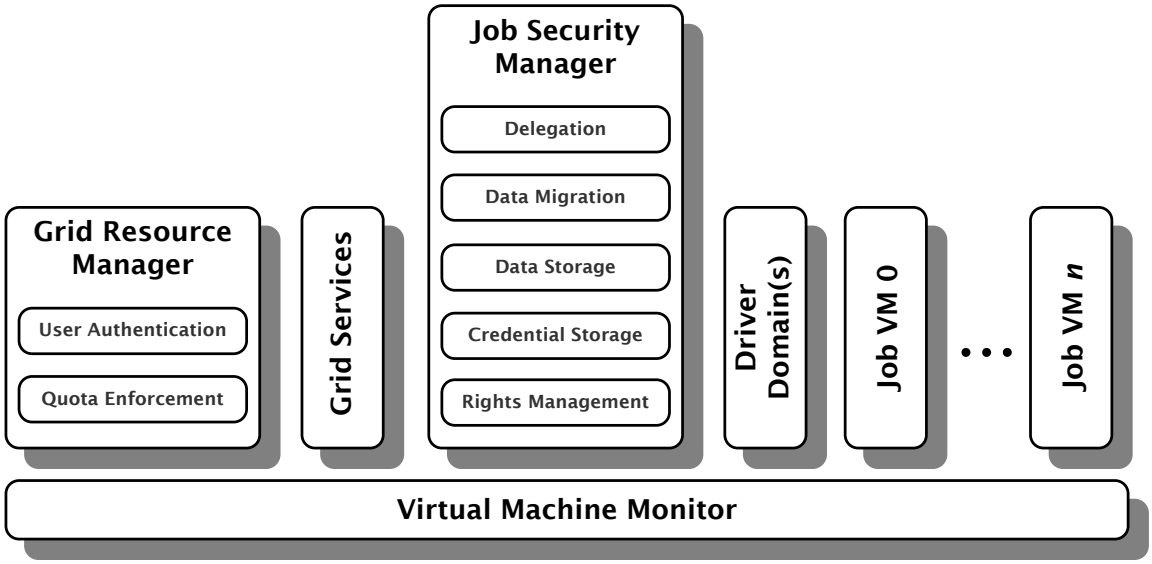


Fig. 1. The Proposed Compartmented Grid Platform Architecture

in the distributed computation could result in a successful attack. In a grid spanning thousands of administrative domains, a single untrustworthy organisation or rogue administrator could bypass the security enforcement by stealing a user's credentials.

Existing grid security architectures such as GSI place trust in the receiving system to enforce policy constraints. There is no reliable way to determine if the delegatee implements the desired constraints, or whether its behaviour has been maliciously altered to subvert the security controls.

4 Trusted Delegation

The credential storage service stores the private key associated with delegation credentials within trusted computing protected storage where it is protected in hardware.

The delegation service restricts access to delegation credentials according to a security policy specified by the job owner. Local policy is defined dynamically according to run-time job requirements. For example, consider when a grid job needs to stage in additional experimental data. In existing approaches, the grid services would hold a long-lived privileged credential that could be used for file staging. Under our approach, grid services are authorised to perform only the minimal set of file transfer operations authorised by the grid job for a limited time-window.

One of the most powerful elements in this architecture is the provision of delegation policy restrictions that cannot be altered by a malicious or run-away grid job. Global policy restrictions always over-ride any local policy defined on the grid node, and persist for the lifetime of the distributed computation. Although restrictions can be defined using current approaches, the policy enforcement is performed by the receiving host. If this host does not understand the restricted policy, or processes it incorrectly, the operation will be allowed to take place. Under the approach presented here, delegation will not occur until a trusted computing attestation challenge has been performed against a security manager on the receiving host to ensure that it correctly implements the policy restrictions.

On the Specification of Secure Transport Layers

Chris Dilloway

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
Christopher.Dilloway@comlab.ox.ac.uk

An increasingly popular technique to simplify the design of a security protocol is to rely on a transport layer protocol to provide some protection against intruders. By depending on a lower level protocol in this way, a security protocol can be designed in a more transparent way. For example, rather than building an authentication mechanism into a protocol, the designer could specify that the messages should be sent over a bilateral TLS connection.

It is common to see protocol simplifications like this. It is, however, rare to see an explanation of the protection the protocol designer expects an SSL/TLS connection to provide. It is rarer still to see a justification that this protection is required, and that it really is provided. It would often be useful to see an explanation of the implications that the choice of transport protocol has on the messages in the security protocol, but this is also frequently omitted.

In 2003, Philippa Broadfoot and Gavin Lowe proposed a layered approach to designing and analysing security transactions [1]. With their approach, a security transaction is layered over a lower-level secure transport protocol. The properties of the transport protocol, once specified formally, can be modelled abstractly, and the analysis of the security protocol can be completed with a much simpler model.

When we describe secure channels, such as those established by TLS, we do not wish to concern ourselves with the design of the transport protocol, we just wish to think in terms of the messages that the agents using them send and receive. We model a network with two layers: the *application layer*, in which agents send and receive messages, and the *secure transport layer*, in which the messages are sent from one location to the other. We abstract away from the secure transport layer, and just study the relationship between *send* and *receive* events in the application layer.

In our model, the intruder can send and receive messages, as the honest agents do; he can also perform three dishonest activities.

Fake The intruder can assume another agent's identity, and use it to send fake messages.
Hijack The intruder can take messages sent by honest agents, and claim them as his own, or another agent's.
Redirect The intruder can redirect messages sent by the honest agents.

The network is shown in Figure 2. The intruder can also overhear everything sent on the network.

There are several desirable properties that a secure channel might guarantee to the agents communicating over it, for example:

Message authentication When an agent B receives a message purportedly from an agent A, then previously A sent that message to B.
Stream integrity The stream of messages received by an agent B must have been sent by a *single* sender, to B, in the order in which B receives them.
Stream authentication The stream of messages received by B supposedly from an agent A was sent by A, to B, in the same order.

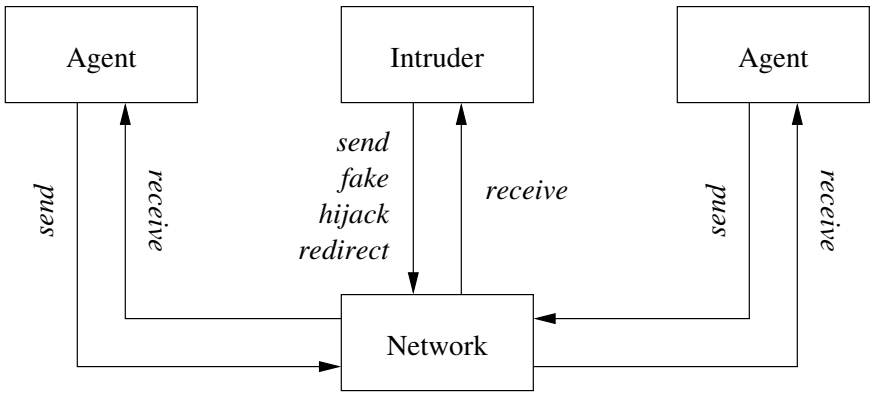


Fig. 1. An abstract network

Confidentiality The contents of the messages sent from A to B cannot be deduced by listening to the network traffic between A and B.

We are creating a hierarchy of specifications of these and other, similar, properties. We are specifying these properties formally, by placing restrictions on the (dishonest) activities the intruder can perform. We are proving equivalences between the channels, and looking for transport layer protocols which implement these abstract channels. We are also analysing security protocols which use channels capable of providing these properties.

There has been some previous work in specifying properties that a channel might provide (eg [1–3]), but we can find no published attempt to examine networks in which several types of secure channel can exist at once. In reality, an agent may be communicating with many other agents over different channels, each of which might provide a different level of security.

We believe that having a hierarchy of channel types can help those designing and analysing protocols. With an exact understanding of the protection that each channel provides, a protocol designer can identify the minimum protection required, and choose the channel type appropriately – the cost of running the protocol can be minimized by choosing the weakest channel necessary. With a knowledge of which properties of the channels chosen by the protocol designer are integral to the security of the protocol, alternatives can be used without introducing new attacks.

References

1. Philippa Broadfoot and Gavin Lowe. On Distributed Security Transations that use Secure Transport Protocols. In *16th IEEE Computer Security Foundations Workshop*, 2003
2. Sadie Creese, Michael Goldsmith, Richard Harrison, Bill Roscoe, Paul Whittaker and Irfan Zakiuddin. Exploting Empirical Engagement in Authentication Protocol Design. *Lecture notes in computer science*, 3450:119–133, 2005.
3. Steve Schneider. Security Properties and CSP. In *1996 IEEE Symposium on Security and Privacy.*

Efficient Group Authentication Protocol Based on Human Interaction

L. H. Nguyen and A. W. Roscoe

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
Long.Nguyen@comlab.ox.ac.uk

1 Introduction

Imagine that a group of people want to transfer data between them securely using their mobile phones or PDAs. Unfortunately none of them knows the authenticated public key of any of the others' equipment, and in any case there is no PKI which encompasses them all. However, if each person tells the others by *human conversation* a public key for his machine, they can then use the Needham-Schroeder-Lowe protocol over the insecure high bandwidth Dolev-Yao network, denoted as $\longrightarrow_N$, to establish secure and authenticated communication. They will have bypassed the intruder for the crucial step of exchanging electronic identities but this idea would require a serious amount of effort on the part of the humans, and therefore what we shall demonstrate is that high quality security can be obtained using this same idea of human activity, captured as low bandwidth *empirical channel*, termed as $\longrightarrow_E$ that is non-spoofable, but with a greatly reduced amount of human work.

In [1] the authors formed a secure network for multi-party scenarios. However, their scheme is shown in [2] to be vulnerable to a combinatorial attack. This protocol introduced, implicitly, the first of two principles which underlie our new protocol:

P1 Make all parties who are intended to be part of a protocol run empirically agree a digest of a complete description of the run.

In [2], Roscoe fixed the protocol by introducing a leader I, who has control over the final *digest* value. In the following description, A [1] is a typical node, $\forall\, A$ means that a message is sent or received by all n parties in the group $\mathbf{G}$ who attempt to achieve a secure link between their PDAs. *Hash* is a cryptographic hash function, and *Digest* is a digest function producing as many bits as we expect the humans to compare. $init(I, A)$ is *true* if $I = A$, and *false* otherwise.

1. $\forall\, A \longrightarrow_N \forall\, A' : (A, INFO_A)$
2a. $I \longrightarrow_N \forall\, S : longhash(N_I)$
2b. $\forall\, S \longrightarrow_E I : committed$
3. $I \longrightarrow_N \forall\, S : N_I$
4. $\forall\, A \longrightarrow_E \forall\, A' :$ Users compare: $Digest(N_I,$ all Messages 1) and $init(I, A)$

Roscoe's scheme, termed as HCBK standing for *Hash Commitment Before Knowledge*, relies crucially upon I being trustworthy: what are we to do if there is no node that is uniformly trusted or it is hard to select one, but we still want a *local PKI* which authenticates the

[1] Each node A has some public information $INFO_A$ that it wants to have authenticated to other members of the group: this might include (i) name and address, (ii) its uncertificated public key or Diffie-Hellman token g^{x_A}, (iii) contextual information to help identify it, such as its location.

$INFO_A$s of all trustworthy nodes? What we want to achieve instead, is that a successful run of the protocol correctly authenticates all the trustworthy parties to each other irrespective of what the others may have done. In order to do this we identify the second principle, derived from the design of HCBK:

P2 If a party A knows it is committed to a specific digest value h, and furthermore has invented a piece of fresh information f, known to A but nobody else, and it is one of the antecedents of h. Then none of the information that A has been sent could have been designed to force its computation of h to be a specific value.

2 New Protocol

Our new protocol is designed so that all nodes rely on **P2**. Therefore, every node will play the same role as the leader in HCBK, and that means each node A now needs some fresh and unpredictable value hk_A.

1. $\forall A \longrightarrow_N \forall A' : A, INFO_A, Hash(hk_A)$
2. $\forall A \longrightarrow_N \forall A' : hk_A$
3. $\forall A \longrightarrow_E \forall A' :$ Users compare $Digest_{hk^*}(\{INFO_A \mid A \in \mathbf{G}\})$ where hk^* is the XOR of all the hk_A's for $A \in \mathbf{G}$

The protocol is secure because neither any one nor any proper subset of $\mathbf{G}$ can determine the final digest value until all the hashkeys are revealed in Message 2, and therefore the intruder cannot constructively manipulate the cryptographic hashes in Messages 1 for his own purposes. This is the effect of **P2**. The best the intruder can do is to commit them blind, having no better than a $1/H$ chance of success, where H is the number of distinct digest values. Readers can read more about how to design a good $Digest$[2] function and the security analysis in the full paper [2]. We call this the *Symmetrised HCBK* scheme (SHCBK).

SHCBK protocol is tolerant of corrupt members but requires expensive computation, n cryptographic hashes. A possible extension is to allow the number of leaders k to vary between 1 and n. This will make the scheme cheaper in computation, k hashes, and it is also not vulnerable to single point of failure as the adversary has to corrupt all the leaders in order to have control over the digest value. We term this extended version as *Hybrid Hash Commitment Before Knowledge*.

3 Conclusion

In this paper, we have proposed a new protocol that resists combinatorial attack and does not require the assumption of trustworthiness of the leader in HCBK. We have exposed some clear principles which underlie our protocol. The work, [2], has been published at FCS-ARSPA'06 workshop in August 2006.

References

1. S. Creese, M. Goldsmith, A. W. Roscoe, and I. Zakiuddin. The attacker in ubiquitous computing environments: Formalising the threat model. *Workshop on Formal Aspects in Security and Trust*.
2. L. H. Nguyen and A. W. Roscoe. Efficient group authentication protocol based on human interaction. Proceeding of *Workshop on Foundation of Computer Security and Automated Reasoning Protocol Security Analysis*, pp. 9-31, Seattle August 2006.

[2] Essentially for all pairs of distinct values $INFOS_1$ and $INFOS_2$, the probability of $Digest_{hk}(INFOS_1)$=$Digest_{hk}(INFOS_2)$ as hk varies, is never significantly different from $1/H$.

Generated Program Monitors with Minimal Memory Overheads

Julian Tibble

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
Julian.Tibble@comlab.ox.ac.uk

A wealth of recent research involves generating program monitors from declarative specifications (*e.g.* [2, 3, 5]). Doing this efficiently has proved challenging, and available implementations often produce monitors that require infeasible amounts of memory. These requirements can be substantially reduced — the analysis and optimisation outlined below does exactly that.[1]

Consider the problem of verifying, at runtime, that the following *safe enumeration* property holds:

> After an enumeration is created, the data-source upon which it is based may not be modified while the enumeration is in use — that is, until the last call to its nextElement() method.

This property can be checked using a *tracematch* [1]. Each tracematch consists of a set of *variables*, an *alphabet* defining a set of events relevant to the property being monitored, a *regular expression* over that alphabet, and a block of code called the *body*. For the property shown above, a suitable alphabet has three symbols, defined in terms of the variables *ds* and *e*:

create	an enumeration *e* is created from a data-source *ds*
update	the data-source *ds* is modified
next	the nextElement() method is called on *e*

A violation of the safe enumeration property is then specified by the regular expression "*create next*∗ *update*+ *next*". The body of the tracematch would be run if a violation occured and could print an error message.

When a tracematch is compiled its regular expression is translated to a finite state automaton. Normally, matching using such an automaton is straightforward, but the presence of variables in a tracematch alphabet complicates matters because information about variable bindings for each state must be kept.

The source of memory overheads in program monitoring systems is therefore twofold. Firstly, matching inevitably involves recording the variable bindings for each current potential match. This requires memory, and the number of such bindings is unbounded. Secondly, these bindings reference objects in the program being monitored. Such references can needlessly prevent objects from being garbage-collected.

Some programming languages provide *weak references* which do not prevent the referenced object from being garbage-collected. Blindly using them everywhere, however, will result in an incorrect implementation where data non-deterministically disappears.

For each automaton state, least-fixpoint calculations are used to sort the variables into categories, depending on whether or not a partial-match in that state can be completed without needing to examine the binding for that variable, and whether or not that variable appears in the tracematch body. During the remainder of compilation, code to update

[1] This research was carried out in conjunction with members of the *abc* [1] group.

variable bindings is generated and specialised depending on what categories the variable appears in — sometimes normal references are needed; sometimes weak references are used, and if they are then the action taken when the referenced object is garbage-collected also varies.

When the safe-enumeration property was verified with a sample of program monitoring systems (including tracematches with memory-optimisation turned off), memory usage quickly rose until either memory was exhausted and the program crashed, progress slowed so much that the experiment had to be aborted after ten hours, or termination eventually occured with around 200MB of memory in use.

With the optimisation turned on, the program and tracematch implementation combined barely used more than 1MB, with no upward trend.

References

1. abc. The AspectBench Compiler. `http://aspectbench.org`.
2. Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondřej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding Trace Matching with Free Variables to AspectJ. In *OOPSLA '05*, pages 345–364. ACM Press, 2005.
3. Rémi Douence, Pascal Fradet, and Mario Südholt. Trace-based aspects. In *AOSD '04*, pages 141–150. Addison-Wesley, 2004.
4. Michael Martin, Benjamin Livshits, and Monica S. Lam. Finding application errors using PQL: a program query language. In *OOPSLA '05*, pages 365–383. ACM Press, 2005.
5. Robert Walker and Kevin Viggers. Implementing protocols via declarative event patterns. In *FSE-12*, pages 159–169. ACM Press, 2004.

Keeping Track of Partial Matches in Trace Monitoring

Pavel Avgustinov

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
pavel.avgustinov@comlab.ox.ac.uk

1 Trace Monitoring

Conceptually, a trace monitor is an entity that observes program execution. A (usually declarative) specification of traces of interest is used to trigger additional code when a match to the actual program history occurs. This is currently a very active research area with many proposed approaches (*e.g.* [2, 3]).

It has been observed that one feature that adds a significant amount of expressiveness is *free variables*, *i.e.* the ability to bind contextduring program execution and access it when a full match is achieved. Then, however, a partial match must store a tuple of bindings for the free variables, and with certain base programs the sets of such tuples become huge. Still, whenever a relevant event occurs, we need to update the matching state. If implemented naively, this would entail an iteration over the entire partial match set.

2 Tracematches

Our discussion will focus on tracematches, a trace monitoring feature introduced in [1]. A tracematch consists of three parts: a set of symbol declarations, a regular expression over these symbols, and a body. For concreteness, let us examine an implementation of the *Observer* pattern:

```
1 pointcut updateSubject(s) : /* ... pick out subject updates ... */;
2
3 tracematch(Subject s, Observer o) {
4     sym create after returning(e) : call(Observer+.new(..)) && args(s);
5     sym update after: updateSubject(s);
6
7     create update+
8     { o.notify(s); }
9 }
```

The tracematch pattern (line 7) directly specifies the concern as a regular expression: We see a *create* event (observer creation), followed by one or more *update* events. Each time this matches program history, we notify the observer. Note how the symbols bind the free variables of the tracematch, and the body uses them.

The tracematch pattern in this case is transformed into the simple NFA shown in Figure 1. 1 is the initial state, 3 the final. Conceptually, we can think of partial matches (which are tuples of bound variables) travelling along the transitions of this NFA; each partial match is associated with the automaton state it has reached, and when a partial match enters state 3, the tracematch body is executed using that partial match's recorded bindings. A complete semantics of tracematch matching is beyond the scope of this document, but full details can be found in [1].

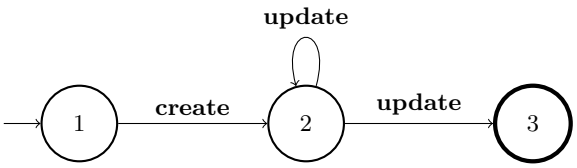


Fig. 1. Non-deterministic Finite State Automaton for the Observer tracematch.

3 Matching State Updates and Indexing

A large source of runtime overhead is the need to iterate over all partial matches whenever a relevant event occurs, either discarding them if they become invalidated or updating their state. A naive (and patently correct) implementation would do this literally: Just walk over the entire set. Note, however, that if the event that occurred binds one of the tracematch formal variables, then we only need to iterate over those partial matches that have compatible bindings, since the event is irrelevant for all other matches.

Let us consider in more detail how we need to update the matching state upon observing a further update event, which binds the tracematch formal s to some runtime value S1. We iterate all partial matches on state 2, and propagate them along the two transitions labelled with update that emerge from that state, adding the binding $s \mapsto S1$ to each partial match we propagate. Now, if s was already bound to S1, this results in no significant change. If, however, a given partial match bound it to a different value, the two incompatible bindings cause the propagated constraint to turn to **false**. Finally, we would execute the tracematch body for each non-**false** partial match on state 3.

This observation allowed us to devise a data structure that enables such selective traversal of the match set. In our running example, we organise the partial matches on state 2 as a map from values of s to sets of matches with compatible bindings. Thus, when we see an *update* event, we look up the corresponding value of s and only iterate over the partial matches that are directly affected.

More generally, this optimisation, which we call *indexing*, can be applied with more than one index variable — the data structure is parameterised by its *index list*, *i.e.* the list of tracematch formals it uses as keys. An empty index list corresponds to the naive case of only storing a simple set; an indexing data structure with indices $(x : xs)$ is a map from values of x to indexing data structures with indices (xs).

Indexing has proved very effective in the case of tracematches. In order to avoid introducing any space leaks into the system, we made careful use of weak references, and we believe that whenever possible, our scheme iterates only the absolutely necessary partial matches. We present an argument that the resulting data structure is correct, i.e. doesn't leak and has the same behaviour as the naive implementation.

References

1. Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondřej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding Trace Matching with Free Variables to AspectJ. In *OOPSLA '05*, pages 345–364. ACM Press, 2005.
2. Rémi Douence, Pascal Fradet, and Mario Südholt. Trace-based aspects. In *AOSD '04*, pages 141–150. Addison-Wesley, 2004.
3. Michael Martin, Benjamin Livshits, and Monica S. Lam. Finding application errors using PQL: a program query language. In *OOPSLA '05*, pages 365–383. ACM Press, 2005.

Modular Interfaces for Program Components

Neil Ongkingco

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
neil.ongkingco@comlab.ox.ac.uk

1 Introduction

Programming languages have provided an ever expanding set of encapsulation constructs to make the development of large programs possible. The introduction of procedures and classes have created a means to decouple program components, leading to increased reuse and maintainability [4]. Software complexity, however, has now increased to the point that classes fall short of encapsulating entire program components. Aspect-oriented programming [2], which inherently cuts through class boundaries, only compounds this shortcoming. A new structure for multi-class components and their aspects is needed to again provide encapsulation.

2 Open modules

Open modules being at a super-package level presents an interesting possibility: that the modules themselves be used to describe entire program components. For this paper a we limit the language in consideration to Java and AspectJ, and define a program component to be a set of classes, perhaps across multiple packages, that together perform a certain function. A particular program component can be explicitly encapsulated by a module, instead of the current approach of implicit encapsulation by packages and documentation. Currently, open modules already specify the interface to the external aspects: which particular joinpoints are advisable. As an example that would require minimum background information, we consider the possibility of extending this interface to the set of API methods of a component. Currently such API methods are merely documented and are not enforced by compile-time checks. A simple addition, as shown in 6.1 shows how the interface methods of a component can be specified in a module.

Listing 6.1. Component Interface

```
1  module M1 {
2      class APIClass1, APIClass2, Internal1, Internal2;
3
4      export : public void APIClass1.apimethod1();
5      export : public void APIClass1.apimethod2();
6      export : public * APIClass2.*(..); //exports all public methods of APIClass2
7      open M2; //open inclusion (affects M2)
8      constrain M3; //constrained inclusion (affects M3)
9  }
10 module M2 {
11     class M2APIClass, M2InternalClass;
12     export : public * M2APIClass.*(..); //visible to all classes (open)
13 }
14 module M3 {
```

```
15      class M3APIClass, M3InternalClass;
16      export: public * M3APIClass.*(..); //visible to class  members of M1,
17                                         // but not external  classes  (constrained)
18  }
```

These interface definitions will also be affected by module inclusion. Constrained inclusion will hide any exported methods in the included module, while open inclusion will retain them. In the example above, the methods of `M2APIClass` will still be accessible to classes outside the module, as `M2` was included using open inclusion. The methods of `M3APIClass`, however, will no longer be accessible to classes outside the module, as its containing module `M3` was included using constrained inclusion. The exported methods of `M3APIClass`, however, will still be accessible to the class members of the including module `M1`.

The introduction of these component method interfaces allows for compile-time checks on what would normally be just documented policies on component use that are enforced through other means. It also makes multi-package components more encapsulated, as one would normally be forced to declare a method public even if a package in a component is merely intended for internal component use.

This is but one language feature that can be placed in the module construct. Open modules may also contain aspects, and the *extent* of these aspects (i.e. the set of joinpoints they can advise) can be limited to only the members of a module. Aspects themselves are envisioned to be part of program components, and the module construct encapsulates its effects. This allows for the integration of aspect-containing program components with minimal unintended effects. Features for handling aspect composition, precedence encapsulation and open classes among others can also be located in modules as well, as they are at the right level for specifying effects on multi-class entities.

References

1. Jonathan Aldrich. Open Modules: Modular Reasoning about Advice. In *Proceedings of the European Conference on Object-Oriented Programming*, volume 3586 of *LNCS*, pages 144–168. Springer, 2005.
2. G. Kiczales. Aspect-oriented programming. *ACM Comput. Surv.*, 28(4es):154, 1996.
3. Neil Ongkingco, Pavel Avgustinov, Julian Tibble, Laurie Hendren, Oege de Moor, and Ganesh Sittampalam. Adding open modules to aspectj. In *Proceedings of the 5th international conference on Aspect-oriented software development*, pages 39–50, New York, NY, USA, 2006. ACM Press.
4. D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.
5. The AspectJ team. The AspectJ home page. http://eclipse.org/aspectj/, 2006.

Protein Structure Computation

Gwyn Skone

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
gwyn.skone@comlab.ox.ac.uk

The sequencing of the human genome and subsequent expansion of understanding of the processes of the human body have brought great increases in the scope of therapeutic drug applications. In the past year, reports of possible treatments for diseases such as Alzheimer's, Huntington's, and multiple sclerosis have been seen; the common theme to these afflictions (and many others) is that they are caused or sustained by certain proteins. These molecules are of central importance to biochemistry. This presentation seeks to demonstrate how computer science can be and has been of benefit to research in this field.

Firstly, an introduction to the fundamental principles of protein structure and behaviour is given. Amino acids and peptide chains are described, and the four hierarchical abstractions of geometric arrangement explained [1]. With this foundation prepared, the basic problems in computational biochemistry (folding, alignment and annotation, and docking) are outlined [2, 3] and major existing techniques for their investigation reviewed [4].

Applications of solutions to the above problems are varied, but we focus on one of much interest worldwide at this time: that of rational drug development. This topic is described, with illustrations of real and current projects of both desktop and globally distributed scales [5, 6]. In particular, the use of the Fast Fourier Transform (FFT) for convolution-based correlation scoring is noted [7].

An implementation of a docking program employing the established FFT algorithm is presented, showing the uses and limitations of the method. The program offers Java classes for representing proteins and other molecules, along with an interface for docking algorithms. This permits several algorithms to be applied in turn to a pair of molecules, each progressively improving the quality of docking. Finally, the current status of the project is given, with the details of an optimisation stage to refine the initial results from an FFT execution, and proposed routes of investigation from that point onwards.

References

1. Branden, C., & Tooze, J.:Introduction to Protein Structure. 2 ed. (1999). Garland. 0-815-32305-0.
2. Richards, F.M.: Areas, Volumes, Packing, and Protein Structure. Ann. Rev. Biophys. Bioeng., **6** (1977), 151–176. Annual Reviews.
3. Sinha, N., & Smith-Gill, S.J.: Protein Structure to Function via Dynamics. Protein Pept. Lett., **9** (2002) 367–377. Bentham.
4. Connolly, M.L.: Molecular Surfaces: A Review. (1996) http://www.netsci.org/Science/Compchem/feature14.html
5. Richards, W.G.: Virtual screening using grid computing - the screensaver project. Nature Reviews (Drug Discovery), **1** (2002) 551–555. Nature Publishing.
6. Venkatachalam, C.M., Jiang, X., Oldfield, T., & Waldman, M.: LigandFit: a novel method for the shape-directed rapid docking of ligands to protein active sites. J. Mol. Graph. Model. **21** (2003) 289–307. Elsevier Science.
7. Katchalski-Katzir, E., Shariv, I., Eisenstein, M., Friesem, A.A., Aflalo, C., & Vakser, I.A.: Molecular Surface Recognition: Determination of geometric fit between proteins and their ligands by

correlation techniques. Proc. Natl. Acad. Sci. USA. **89** (1992) 2195–2199. National Academy of Sciences.

Object-Oriented Specification and Verification

Eric Kerfoot

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
eric.kerfoot@comlab.ox.ac.uk

Designing and implementing object-oriented systems correctly requires a method of reasoning about the design and some means of verifying that the implementation achieves these design goals. This paper proposes a specification language, Tesla-Z, which addresses the lack of formal methods that accurately describe the interrelationships between objects, describe method semantics correctly, and accurately model the implications of aliasing. An implementation language, called Tesla, has been designed to facilitate the automatic verification of code specified with Tesla-Z.

Current methods inadequately describe the complex behaviour that exists between objects, such that seamingly correct specifications produce incorrect implementations, thus successful proofs can be completed on incorrect code. Their implementation languages may also make verification more difficult due to complex semantics that were not optimized for this task. Thus the goals of Tesla-Z are to accurately model objects and provide a comprehensive model to verify against, and the goals of Tesla are to provide an implementation language that facilitates implementation of that specification and automatic proofs of correctness.

This is achieved by ensuring both Tesla-Z and Tesla share a significant portion of their semantic descriptions which is defined using Z [5]. Since Tesla-Z builds on Z this simplifies the task of specification, and later in verifying the implementation of that specification. Tesla itself is designed with constructs and data structures that are analogous to those in Tesla-Z. Implementing a specification is thus simplified and often a more direct translation, which combined with the shared semantic description aids in verification. To facilitate the development of tools to aid in specification and verification, the Tesla compiler uses a system of plugins that allow CASE tools to access the compilation process.

The semantic descriptions for types, references, and other components are shared between Tesla-Z and Tesla, such that there are shared analogous structural and logical constructs in both. The final step of implementing a specification is thus a straightforward process of mapping one Tesla-Z construct, such as an attribute or method, to one in Tesla. Tesla-Z is able to reason about the concepts found in Tesla, such as its reference semantics and the consequences of aliasing that this entails. Methods are also used in the specification that have true method semantics, as opposed to operation schemas as in Object-Z [4] whose semantics are very different.

Tesla itself includes data structures and expressions that are analogues of the set theory components of the specification. This makes the implementation step much more straightforward and easier to verify. A class attribute that is defined using set theory constructs in Tesla-Z can be directly implemented using an analogous class in Tesla. The semantics of these classes is guaranteed by their specifications to match that of these constructs.

Verification of code is the main objective that research into Tesla will be geared towards, which is the task of proving the correctness of code that implements a given specification. Programs written in languages such as Java or C# have proven to be extremely difficult to verify with specifications and automated theorem provers, an example of which is Spec# [3] that integrates specification into C#. This is due to the nature of their semantics and language constructs. These were never designed to produce verifiable code and so do not have

clear and concise formal semantic descriptions. Their complex semantics make requirements on correctness proofs that are often too great to overcome. Research has been undertaken into partial verification (ESC/Java [1]) which is purposely incomplete and unsound in exchange for feasibility, or into verifying subsets of Java (JavaCard [2]). By simplifying the semantics, practical complete proofs become possible.

The Tesla project is designed to provide an implementation language with a clear, concise formally defined semantics that can be used with the specification language to facilitate automated verification. Tesla itself includes expressions and statements that can more easily be assigned formal meanings. This ensure that proving properties about the language itself can be undertaken with much less requirement on the part of the theorem prover.

To realize this, much research has been put into Tesla's semantic specification. This description uses Z to specify the static and dynamic semantics of the language, and so can be easily used with Tesla-Z to prove correctness in a set theoretic framework. This description has been specifically designed to facilitate reasoning about references, so that aliasing, object invariants polymorphism, and frame properties can be specified in Tesla-Z, and the implementation of such specifications verified.

The Tesla compiler has been implemented in a modular fashion to allow plugins to be loaded at compile time. These plugins use blocks of text in Tesla source files, called metadata, to provide facilities normally implemented as separate CASE tools. Doing so eliminates the need to rewrite parsers and other components these tools normally require. Metadata can contain the Tesla-Z specification for Tesla definitions, and so plugins will be created to perform specification analysis code verification at compile time. Other possible areas of research will use metadata and plugins to investigate ways of integrating formal methods with Tesla, such as automated code generation, concurrency analysis, algorithm analysis, and code optimization.

References

1. Cormac Flanagan, K. Rustan M. Leino, Mark Lillibridge, Greg Nelson, James B. Saxe, and Raymie Stata. *Extended Static Checking for Java*. In Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation (PLDI2002), volume 37, pages 234245, June 2002.
2. Marieke Huisman Nstor Catao. *Formal Specification and Static Checking Of Gemplus Electronic Purse Using ESC/Java*. In ME 2002: Formal Methods - Getting IT Right : International Symposium of Formal Methods Europe, Copenhagen, Denmark, July 22-24, 2002, volume 2391 of LNCS. Springer, January 2002.
3. K. Rustan M. Leino Mike Barnett and Wolfram Schulte. *The Spec# Programming System: An Overview*. 3362:4969, 2005.
4. Graeme Smith. *The Object Z Specification Language*. Springer Verlag, 1999.
5. J. M. Spivey. *The Z Notation: A Reference Manual*. International Series in Computer Science. Prentice Hall, 2 edition, 1992.

Game Semantics of the Safe Lambda-Calculus

William Blum

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
William.Blum@comlab.ox.ac.uk

1 The safety condition

The *safety condition* has been introduced in [5] as a syntactic restriction for higher-order recursion schemes (grammars) that constrains the order of the variables occurring in the grammar equations. The authors of [5] were able to prove that the Monadic Second Order (MSO) theory of the term tree generated by a safe recursion scheme of any order is decidable.[1]

When transposed to the λ-calculus [3], the safety condition gives rise to the *Safe λ-calculus*, a strict sub-language of the λ-calculus. A first version appeared in the technical report [2]. We propose a simpler and more general version where term types are not required to be homogeneous [4]. A noteworthy feature of the Safe λ-calculus is that no variable capture can occur when performing substitution and therefore it is unnecessary to rename variables when computing β-reductions.

Little is known about the Safe λ-calculus and there are many problems that have yet to be studied concerning its computational power, the complexity classes that it characterises, its interpretation under the Curry-Howard isomorphism, and its game-semantic characterisation [1]. Our contribution concerns the last problem.

2 The correspondence theorem

The difficulty in giving a game-semantic account of Safety lies in the fact that it is a syntactic restriction whereas Game Semantics is by essence a syntax-independent semantics. The solution consists in finding a particular syntactical representation of terms on which the plays of the game denotation can be represented. To achieve this, we use ideas recently introduced in [6]: a term is canonically represented by the abstract syntax tree of its η-long normal form, referred as the *computation tree*. A computation is described by a justified sequence of nodes of the computation tree respecting some formation rules and called a *traversal*. Traversals permit us to model β-reductions without altering the structure of the computation tree via substitution. We prove the following result:

Theorem 1 (Correspondence theorem) *The set of traversals of the computation tree is isomorphic to the set of uncovered plays of the game denotation of the term.*

In other words, traversals are just representations of the uncovering of plays of the strategy denoting the term. By defining an appropriate *reduction* operation which eliminates traversal nodes that are "internal" to the computation, we obtain an isomorphism between the strategy denotation of a term and the set of reductions of traversals of its computation tree.

[1] In fact it has been shown in [6] that it is also true for unsafe recursion schemes.

3 Game-semantic characterisation

We introduce the notions of *incrementally-justified strategies* and *incrementally-bound computation trees*. Using the Correspondence Theorem, we show that for β-normal terms the computation tree of a term is incrementally-bound if and only if its strategy denotation is incrementally-justified. We have the following theorem:

Theorem 2 (Game-semantic characterisation of safety) *Safe simply-typed terms in β-normal form have incrementally-bound computation trees. Reciprocally, a closed term in η-long normal form with an incrementally-bound computation trees is safe.*

Since pointers in the plays of *incrementally-justified strategies* are by definition uniquely reconstructible, we obtain the following corollary:

Corollary 1 *The pointers in the game semantics of safe simply-typed terms can be recovered uniquely from the underlying sequences of moves.*

4 Extension to Safe Idealized Algol

We define Safe IA to be the Safe λ-calculus augmented with the constants of Idealized Algol (IA) [8] as well as a family of combinators Y_A for every type A. We show that terms of the Safe PCF [7] fragment are denoted by incrementally-justified strategies and we give the key elements for a possible extension to full Safe IA.

References

1. Samson Abramsky and Guy McCusker. Game semantics. In *Logic and Computation: Proceedings of the 1997 Marktoberdorf Summer School*. Springer-Verlag, 1998. Lecture notes.
2. Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. Safety is not a restriction at level 2 for string languages. Technical report, University of Oxford, 2004.
3. Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.
4. William Blum. Transfer thesis. University of Oxford, August 2006.
5. T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS*, pages 205–222. Springer, 2002. LNCS Vol. 2303.
6. C.-H. Luke Ong. On model-checking trees generating by higher-order recursion schemes. In *Proceedings of LICS*. Computer Society Press, 2006.
7. Gordon D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):225–255, 1977.
8. John C. Reynolds. The essence of algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. IFIP, North-Holland, Amsterdam, 1981.

Towards A Unified Model For Workflow

Peter Y.H. Wong

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
Peter.Wong@comlab.ox.ac.uk

1 Introduction

The emergence of workflow management systems offers supports for composing, coordinating and monitoring the execution of human tasks and component services. One challenge is to provide a formal semantics and the facility to model, analyse and reason about workflow processes at both *orchestration* and *choreography* levels. This paper gives an overview of the formalisation of van der Aalst et al.'s workflow patterns [4] using the process algebra CSP [3], and of a case study based on a realistic business process. The modelling example in this paper captures both workflow orchestration and choreography. The complete formalisation of workflow patterns can be found in a longer paper [6]; the complete formalisation of the workflow choreography model will appear in a future publication.

Workflow patterns, introduced by van der Aalst et al., are the "gold standard" for benchmarks of workflow languages [4]. These patterns range from simple constructs to complex routing primitives; their scope is limited to static control flow. We model each control flow pattern in CSP. A basic workflow activity is defined as the CSP process $P(a, X)$.

$$P(a, X) = init.a \rightarrow work.a \rightarrow ||| \, b : X \bullet init.b \rightarrow SKIP$$

2 Example

This case study examines a business process of reserving and booking airline tickets, defined in an XML choreography description language called Web Service Choreography Interface (WSCI) [5]. This example includes three participants: a traveler, a travel agent and an airline reservation system, This section gives an overview of the orchestration of the traveler, and describes our approach to modelling the choreography of the complete system.

Traveler's Interface - In this interface the traveler can order a trip by setting up an itinerary for airline tickets ($ORDER$). She may then change her itinerary many times ($CHITIN$) or cancel the itinerary ($CAITIN$). Thereafter she can reserve the seats ($RTICKET$). After the reservation she can proceed with the booking ($BOOK$), or the reservation may be cancelled due to expiry ($TIME$) respectively. After she has booked her ticket, the travel agent and the airline will send her the tickets ($TICKET$) and statement ($STATE$), The CSP process $TRAVEL$ models the complete orchestration of the traveler. Details of synchronous sets and subprocesses are omitted.

$$\begin{aligned} TRAVEL = (ORDER \parallel (CHITIN \triangle SCOPE)) \parallel (CAITIN \Box (RTICKET \parallel \\ (TIME \Box ((CANRES \Box (BOOK \parallel (TICKET ||| STATE))))))) \end{aligned}$$

Modelling Choreography - Apart from traveler process, the choreography of the complete airline tickets reservation business process also includes the airline reservation system

(*AIRLINE*) and the travel agent (*AGENT*). Based on the WSCI global model of the ticket reservation system, the collaboration of the three participants is modelled by the CSP process *MODEL* where the set *CONNECT* is the set of pairs of WSCI operations modelled as a set of pairs of CSP events. The complete choreography, defined as the process *GLOBAL*, is modelled by composing the three participant processes and the process *MODEL* in parallel. Definitions of subprocesses are omitted.

$$
\begin{aligned}
&MODEL = \Box(a, b) : CONNECT \bullet init.a \rightarrow init.b \rightarrow MODEL \\
&INTER = AIRLINE \parallel (TRAVEL \parallel AGENT) \\
&GLOBAL = (START \parallel (((INTER \parallel MODEL) \triangle (END \,\Box\, succ \rightarrow SKIP)) \\
&\qquad\qquad \parallel (ABORT \,|||\, SUCC))) \fatsemi GLOBAL
\end{aligned}
$$

Since the process *GLOBAL* describes precisely the complete dynamic control flow of the choreography, by using refinement [3], we can formally make assertions about properties which the business process must satisfy. Such assertion can be expressed as the following *failures* refinement where *SPEC* defines the properties to be satisfied and *HIDE* is the set of all hidden events. The detail of *SPEC* is omitted.

$$SPEC \sqsubseteq_F GLOBAL \setminus HIDE$$

Assertions made using refinements can be formally verified using the CSP model checker FDR [1]. Note the CSP model given in this paper is simplified to illustrate our approach to modelling choreography. We have excluded the application of constraints by parallel compositions and the reduction of the model's state.

3 Future Work

Future work will include investigating other choreography description languages such as WS-CDL [2] and developing a "generic" unified model for workflow orchestration and choreography. We will also extend our present CSP model with a formal exception and compensation handling mechanism. Furthermore, we will extend our model with dataflow semantics, hence unifying the semantics of workflow processes in both business and scientific domains, including the clinical domain.

References

1. Formal Systems (Europe) Ltd. *FDR2 User Manual*, 1998. `www.fsel.com`.
2. N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. *Web Services Choreography Description Language 1.0*, 2005. W3C Candidate Recommendation.
3. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1998.
4. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, July 2003.
5. W3C. *Web Service Choreography Interface 1.0*, 2002. `www.w3.org/TR/wsci/`.
6. P. Y. H. Wong and J. Gibbons. A Process Algebraic Approach to Workflow Verification, 2006. Submitted for publication.

CodeQuest: Querying Source Code with Datalog

Elnar Hajiyev[1], Mathieu Verbaere[1], Oege de Moor[1], Kris de Volder[2]

[1] Computing Laboratory
University of Oxford
United Kingdom
{elnar.hajiyev,mathieu.verbaere,oege}@comlab.ox.ac.uk

[2] Software Practices Lab
University of British Columbia
Vancouver, Canada
kdvolder@cs.ubc.ca

1 Introduction

A *code query* helps to identify locations of interest in the source of a program. It is often desirable to ask semantical questions that cannot be solved by purely text-based pattern matching. Such semantic code queries are important for program understanding, checking coding style conventions, fault detection, refactoring, and aspect weaving.

There have been numerous proposals for specialised query languages. These range from complex pattern languages (as in AspectJ [6]) to Prolog as part of LogicAJ [3]. IDE tools also exhibit a wide range of query technologies, ranging from a generalisation of AspectJ's pointcuts in the CME [7] to a full logic programming language in JQuery [4].

In the software maintenance community, there is a long tradition of using storing information about the source in a database, and then querying that information via database queries. Jarzabek [5] presents an SQL-like query language with special primitives for code queries, implemented on top of a Prolog system. ASTLog is a query language for inspecting abstract syntax trees [1].

Thus far there has not been a rigorous assessment of the relative merits of these different technologies, for instance scalability. This poster represents the first steps towards such an assessment, as well as a synthesis of the best ideas of the works cited above. To wit, the contributions of the research reported here are:

- The combination of earlier strands of work, leading to the use of Datalog as the query language, with a traditional database system as its backend.
- The construction of an optimising compiler from Datalog to SQL, both targeting built-in recursive queries and a custom implementation of recursion via stored procedures.
- The collection of a number of benchmarks to compare query engines.
- A comparison of four query engine technologies with respect to these benchmarks.

2 Example Code Queries

Presented queries use Prolog notation as suggested by many of the works cited above. The first query is checking a common style rule – there are no declarations of non-final public fields. When such fields occur, we want to return both the field F and the enclosing type T:

$$q_1(T, F) :- type(T), child(T, F), field(F), modifier(F, public), \mathbf{not}(modifier(F, final)) .$$

In the second query we wish to determine all methods M that write a field of a particular type, which is T or any of its subtypes:

$$q_2(M, T) :- method(M), writes(M, F), type(F, FT), subtypestar(T, FT) .$$

Here the main relation of interest is $subtypestar(T, FT)$, which relates a type T to its subtypes FT. It is defined recursively:

$$subtypestar(T, T) \ :- type(T) .$$
$$subtypestar(T, FT) :- subtype(T, V), subtypestar(V, FT) .$$

3 Datalog

While the descriptions in the previous section are certainly attractive, the use of logic programming has a number of problems. First, all facts must be kept in memory. So it does not meet the scalability criterion. Furthermore, to achieve the desired efficiency, it is necessary to pollute the elegant specifications with mode annotations and the use of the cut operator to control backtracking. Fortunately it turns out that typical code queries do not require the full power of Prolog. In particular, it is rare for code queries to build up intermediate data structures such as lists. We can therefore restrict our attention to the *Datalog* language [2], which is essentially Prolog without data structures: it manipulates relations only. This subset is expressive enough for code queries, and it is furthermore possible to provide a scalable implementation on top of existing database systems.

4 Experiments

We compared four different query engines on four Java projects, for the queries presented earlier. The query engines are: JQuery [4]; XSB; MS SQL Server with built-in recursion via CTEs; and MS SQL Server with our own implementation of recursions via stored procedures. The projects we ran the queries on are Jakarta (3 KSLOC), the source of JQuery (38 KSLOC), JFreeChart (93KSLOC) and Eclipse (1311 KSLOC). These projects were chosen because of their wide availability, as well as their size characteristics.

The results of our experiments have clearly shown that CodeQuest approach to query source code strikes the right balance between expressiveness and scalability and outperforms other systems when querying large pieces of source code. Detailed numbers are available on the poster.

5 Conclusions

We have compared different technologies for implementing code query engines. It is clear that for small code bases, a tabled implementation of Prolog performs extremely well. For scalability, however, an implementation of Datalog on top of a database system is much more preferable. In future we plan to further investigate optimisations of recursion - we already do considerably better than the built-in facilities of MS SQL Server. An important problem is also to be able to update the database incrementally. Finally, we plan to integrate this technology with the AspectBench Compiler for AspectJ.

References

1. Crew, Roger F.: ASTLOG: A language for examining abstract syntax trees, USENIX Conference on Domain-Specific Languages (1997) 229–242
2. Gallaire, H. and Minker, J.: Logic and Databases, Plenum Press, New York (1978)
3. Günter Kniesel,Tobias Rho,Stefan Hanenberg: Evolvable Pattern Implementations Need Generic Aspects, ECOOP 2004,Workshop on reflection,AOP & metadata for Software Evolution
4. Doug Janzen and de Volder, Kris: Navigating and querying code without getting lost, 2nd International Conference on Aspect-Oriented Software Development (2003) 178-187
5. Stan Jarzabek: Design of Flexible Static Program Analyzers with PQL, IEEE Transactions on Software Engineering (1998) 197-215
6. Gregor Kiczales, John Lamping, Anurag Menhdekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, John Irwin: Aspect-oriented Programming ECOOP (1997) 220–242
7. Peri Tarr, William Harrison, Harold Ossher: Pervasive Query Support in the Concern Manipulation Environment, IBM Research Division, Thomas J. Watson Research Center (2004) RC23343

Verification and Optimisation of Cardiac Cellular Models

Jonathan Cooper

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
jonathan.cooper@comlab.ox.ac.uk

Abstract

Understanding the human heart is a subject of much interest. Heart problems are a major cause of death in developed countries, and much effort has been put into elucidating the causes of heart disease in the hope of developing cures. Many drugs also have adverse side effects on the heart, which drug companies would like to avoid.

Computer simulations have played an important part in this research [4], however simulations are often very time consuming—some can take days to compute even on high performance computing resources. There is thus considerable interest in optimisation techniques, with a view to making whole-heart simulations tractable. Reliability of heart model simulations is also of great concern, particularly considering clinical applications. Simulation software should be easily testable (against empirical data) and maintainable, which is often not the case with extensively hand-optimised software. Automating and verifying any optimisations is thus crucial.

We present a framework for automatically optimising models of the electrical activity of single myocytes (heart muscle cells). This electrical activity is what causes the heart to beat. The models treat a cell like an electrical circuit, representing various features by ordinary differential equations (ODEs).

CellML [2] is an XML language designed for describing biological cell models from a mathematical modeller's perspective, and is being developed at the University of Auckland. It gives us an abstract format for such models, and provides effectively a *structured mathematical description* of the *ODE system*. It is gaining in popularity, with various simulation environments being able to import CellML models.

From a computer science perspective, CellML looks like a domain specific programming language, and so we are investigating the gains available from exploiting this viewpoint. We describe various static (i.e. compile-time) checks for CellML models (including checking for dimensional consistency), and investigate the possibilities of provably correct optimisations. In particular, we demonstrate that partial evaluation [3] is a promising technique for this purpose, and that it combines well with a lookup table technique, commonly used in cardiac modelling, which we have automated [1].

References

1. J. Cooper, S. McKeever, and A. Garny. On the application of partial evaluation to the optimisation of cardiac electrophysiological simulations. In *PEPM '06: Proceedings of the 2006 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pages 12–20, New York, NY, USA, 2006. ACM Press.

2. W. J. Hedley, M. R. Nelson, D. P. Bullivant, and P. F. Nielsen. A short introduction to CellML. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 359(1783):1073–1089, June 2001.
3. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.
4. D. Noble. Modelling the heart: insights, failures and progress. *BioEssays*, 24:1155–1163, 2002.

Target Trajectory Prediction for an UAV Helicopter

Heiko Helble

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
heiko.helble@comlab.ox.ac.uk

1 Introduction

A trajectory prediction method for dynamic ground targets being visually tracked from a UAV helicopter platform is proposed. This paper addresses the problem of target occlusion during tracking. Experiments have shown that complex target movement patterns which do not exhibit repetitive dynamic properties can not be modelled with a single dynamic model. Therefore, this work uses observation and subsequent *learning* for successful trajectory prediction.

The Oxford Aerial Tracking System (OATS) consists of a miniature helicopter, an automatic flight controller, on-board computing hardware and a camera mounted on a two-axis gimbal. OATS is described in more detail in our previous publication [1].

2 Trajectory Prediction for OATS

2.1 Prerequisites

The Oxford Aerial Tracking System has the aim to autonomously track arbitrary ground objects from within an UAV helicopter. This paper describes a novel technique that allows predicting of the future trajectory of a visually tracked ground object. The motivation for this research is twofold: Firstly, the tracked target can become temporarily occluded due to obstacles that are in the line of sight between robot and target. Secondly, the tracked target might try to hide from the observer, that is, purposely manoeuvre behind obstacles like buildings and trees so that the observer loses its track on the target. After experimenting with the Extended Kalman Filter and other techniques to be considered, it became evident that the lack of a dynamic model for the task at hand disqualifies these attempts from the very start. We rather have to act on the assumption that the target we are trying to track possesses highly non-linear dynamics that do not follow any fixed rules that model its behaviour close enough. The suggested solution to this problem is to observe the behaviour pattern of the target and *learn* its dynamics in the process. Artificial neural networks (ANN) have the ability to be used as an arbitrary function approximation mechanism which learns from observed data.

2.2 Neural Networks for Target Trajectory Prediction

My approach for target trajectory prediction uses two independent feed-forward ANN (featuring 6 input, 10 hidden, and 3 output neurons), one for predicting the x coordinates (ANN_x) and the other one for predicting the y coordinates (ANN_y) of a target moving in the plane. Combining the results of the two ANN permits real-time prediction of the future trajectory of the tracked ground target. In order to gain a set of data points that represent a history of the past six x coordinates, the signal is passed through delay blocks that are

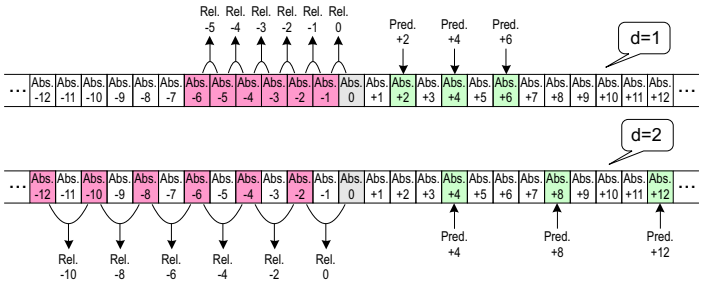


Fig. 1. Timeline of coordinates fed into the ANN for trajectory prediction.

aligned in a linear sequence. After each delay, the signals of the x and y coordinates are tapped and converted into relative signals.

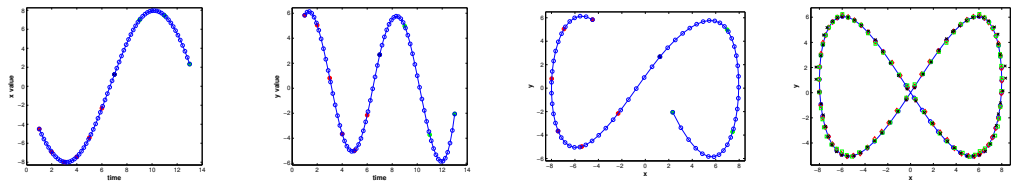
In order to predict further into the future than the six time steps given by $X_{pred(+6)}$, an additional time span of past values has to be considered. We achieve this by using variable delay blocks. Fig. 1 illustrates for two cases ($d = 1$ and $d = 2$) how the variable delay affects the creation of a set of history coordinates to be fed into the ANN. The results shown in Fig. 2 were obtained from real-time experiments with our ANN implementation using a delay value of $d = 5$ time steps. Six relative coordinates that lie in the past, which are gained from seven absolute coordinates reaching back a maximum of 30 time steps are used. In Fig. 2(a) to 2(c) red dots indicate prior, black dots present, and green dots posterior coordinates.

For filling out the gaps between the control coordinates, interpolating cubic splines are employed (shown as a blue line in Fig. 2(a) and Fig. 2(b)). Since each of the two ANN we use gives us a prediction result in one dimension, the required 2-D trajectory can be obtained by combining the results gained by ANN_x and ANN_y (shown in Fig. 2(c)). Fig. 2(d) finally shows a plot of the original tracking data together with the predicted coordinates. The target describes a "figure eight" in the plane.

The presented trajectory prediction method demonstrates how artificial neural networks can be employed for learning target movement patterns that stem from visual object tracking. The results suggest that two feedforward ANN run in parallel are capable of adapting themselves via batch learning well enough so that complex movement patterns can be predicted with high accuracy.

References

1. Helble, H., Cameron, S.: OATS: Oxford Aerial Tracking System. Proceedings of TAROS 2006, Guildford, UK, Sep. 2006, pp. 72-77



(a) ANN_x prediction (b) ANN_y prediction (c) 2D prediction (d) Overlay with orig.

Fig. 2. Target trajectory prediction results for $d = 5$.

An Example-Based Improvement to Statistical Machine Translation

James Smith

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
james.smith@comlab.ox.ac.uk

1 Background

Machine Translation refers to the use of computers to automate some or all of the process of translation from one human language to another. Much of the research in the past has focused on linguistically motivated rule-based approaches, which require a rigourous understanding of the languages involved in order to create complex rules for translation. Over the last 15 years, interest has shifted considerably towards corpus-based techniques, which use parallel bilingual corpora to automatically derive models for translation between the two languages. Approaches using such techniques can be roughly grouped into two classes: Statistical Machine Translation (SMT) and Example-Based Machine Translation (EBMT). SMT builds a statistical translation model from the corpora which, at it's most basic level, gives the probability of each word in the target language for a given word in the source language. EBMT uses the corpora as a database of translation examples, and tries to use as much information from the suitable examples it finds as possible.

2 EBMT to Enhance SMT

EBMT and SMT work better at translating different types of text. SMT is good is good at translating data at the word or phrase level, but resorts to a language model to resolve ambiguities. EBMT can work very well with enough context, but is terrible with single words or small, common phrases. Our hope is that by combining these two approaches into a hybrid system, we can produce a translation system which outperforms either method individually.

Current SMT systems generally perform much better than EBMT systems. As EBMT is our main area of interest, we have decided to implement the EBMT/SMT hybrid by building an EBMT element on top of an existing SMT system. The EBMT system translates what it can, then hands over to the SMT system to finish off.

3 Implementation

The first stage in our translation process is finding a suitable match in the example database for the input sentence. This consists of two major steps. Firstly the examples are passed through a preliminary filter which is based on the number of N-grams shared between the input and each example. The retrieved examples are then scored using a more rigorous method. The purpose of this scoring is simple: to determine the suitability of the example for translating the input. What this actually means is a difficult question and is arguably the most involved stage of EBMT. Our current algorithm uses a Levenshtein distance between the input and the example. This counts the number of word insertions, deletions, and

substitutions required to convert the first sentence into the second. We expect in the future to further adapt this stage to examine and compare syntactic structure.

Having found the best match, we must use the example to translate the input. This is currently implemented in a greedy longest-match fashion. The system finds the biggest textual match between the input and example and attempts to translate it, then continues with any non-overlapping untranslated segments until it can translate no more. In the future, we hope to incorporate more intelligence into this step, as an example may not always be selected for a long string match. This may involve extracting templates from the text and substituting the appropriate parts of the input into the holes.

The third and final step in traditional EBMT is the recombination of the translated fragments. In our system, however, this is not a concern, as we now pass what we have over to the SMT system. The SMT system uses the translations we have already generated, and then uses its own models to translate any part of the input which the EBMT system could not deal with. The combination of translations is then passed through a language model for minor reordering.

4 Results

Unfortunately, at this stage, the hybrid system performs slightly worse than the SMT system. The evaluation so far has only been done using the Bleu metric, so we are currently doing further analysis to determine where and why the EBMT component is generating inferior translations.

SaskNet: Automatically Creating a Large Scale Semantic Knowledge Network

Brian Harrington

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
Brian.Harrington@comlab.ox.ac.uk

1 Introduction

This presentation will discuss the current state of development of the SaskNet (Spreading Activation based Semantic Knowledge NETwork) project. The goal of the SaskNet project is to develop a system which can automatically extract knowledge from natural language text, and build a large scale semantic network based on that knowledge.

SaskNet translates natural language sentences into fragments of a semantic network. Each of these fragments is taken as an update to the existing knowledge network, and so the fragment (and thus the sentence) is interpreted within the reference frame provided by all the other sentences already processed. This means that the network improves with each sentence it processes, and knowledge gained from previous documents can help with future processing.

2 Motivation

Semantic networks are a valuable part of many research projects in Artificial Intelligence. For this reason, projects like ConceptNet [1] and the Cyc Project [2] have spent years manually creating networks with varying degrees of success. The purpose of the SaskNet project is to use natural language processing tools to automatically generate a semantic network in a matter of hours which would have taken years to create previously.

3 Semantic Analysis

SaskNet uses the Clark & Curran Parser [3] and CCG2Sem [4] to transform input sentences into Discourse Representation Structures (DRS) [5]. In DRS format, the constituent objects of each sentence and the relationships between these objects are identified. Each sentence is treated as an update to an existing discourse, and anaphora such as pronouns are bound to their antecedents. For example, in the sentence *"The woman entered the room. She smiled"* the pronoun *she* in the second sentence is bound to the *woman* object created in the first sentence.

Once the semantic analysis of a sentence has been completed, SaskNet uses the DRS to create a semantic network fragment from each sentence. The network created is a directed hierarchical graph with nodes representing objects and concepts and edges representing semantic relations. A simplified network is shown in figure 1.

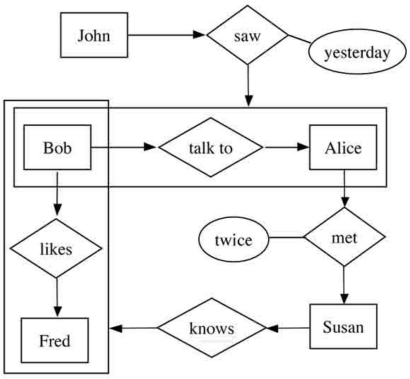


Fig. 1. A simplified Semantic Network created from the sentences "John saw Bob talk to Alice yesterday. Alice met Susan twice. Susan knows Bob likes Fred."

4 Spreading Activation

Spreading activation mimics the way neural activity spreads in the human brain. When a node in a semantic network receives a certain amount of activation it fires, sending activation to all neighbouring nodes. By firing a node and monitoring the spread of activation, SaskNet can determine which nodes are closely semantically related in the network.

SaskNet uses spreading activation to decide which object nodes should be mapped together in the network. In a process known as *Object Resolution*, two nodes are fired and their activation spread is monitored. Similar patterns of activation indicate similar semantic identities. If two nodes have very similar patterns of spreading activation, they likely refer to the same real world entity, and therefore should be collapsed into a single node.

Object resolution is a very important part of SaskNet as it allows information from multiple data sources to be combined into a single cohesive network.

References

1. H Liu and P Singh: Conceptnet a practical commonsense reasoning tool-kit. BT Technology Journal, 22:211 226, Oct 2004.
2. Douglas B. Lenat: Cyc: A large-scale investment in knowledge infrastructure. Communications of the ACM, 38(11):33 38, 1995.
3. Stephen Clark and James R. Curran: Parsing the WSJ using CCG and loglinear models. In Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL 4), pages 10411, Barcelona, Spain, 2004.
4. Johan Bos: Towards wide-coverage semantic interpretation. In Proceedings of Sixth International Workshop on Computational Semantics IWCS-6, pages 423, 2005.
5. Hans Kamp and Uwe Reyle: From Discourse to Logic : Introduction to Modeltheoretic Semantics of Natural Language. Formal Logic and Discourse Representation Theory. Kluwer Academic, Dordrecht, 1993.

Peer Production of Security Information

Max Loubser

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
Max.Loubser@comlab.ox.ac.uk

Usability and social factors are increasingly important considerations in security research and community oriented applications are dominating Internet development. Few of the numerous current and proposed measures against the growing malicious software problem take into account that human behaviour can weaken security.

The end-user machine is an important area to secure; the safety of powerful, broadband connected machines in homes and offices is now as important as the security of databases, servers and routers. The users of these machines often do not understand the security implications of running downloaded applications. It is also these open platforms that give the Internet its generative properties [1].

The large-scale cooperative production - peer production - of information is forcing a re-evaluation of traditional processes of information production [2]. The output of large numbers of people, collaborating and loosely connected with the Internet, is now beginning to rival that of large corporations.

Security information has a number of properties that make it suitable for peer production. Threats are difficult to classify because it is not always clear what should be considered to be bad - what is annoying or dangerous for one person is not necessarily so for another. Transparency in the process of threat assessment is required by the consumers of security information and information personalised to the specific user's preferences is often needed. Constantly updating and documenting ever-changing threats is expensive. Peer producing security information can provide non-partisan, inexpensive and frequently updated information.

Unfortunately inadequately usable protection mechanisms and inadequate user awareness are still major problems in protecting PCs from bad applications. The collaborative nature of the peer production process has the positive side-effect of spreading awareness and facilitating understanding of the scope and effects of bad code. This property of the peer production process can be expanded to make security mechanism more usable.

Usable security research has shown that a user's motivation to pay attention to security is based on an awareness and understanding of the risks involved [3]. If there is no such awareness a user is likely to be lax in her application of security measures.

Social navigation is the process of acquiring awareness and making decisions based on information gathered from others in a social space, either directly or indirectly, through the cues and clues provided by their behaviour. We use social navigation when deciding which pub to go to or when opting to go in a certain direction at the airport. When conducting our activities online, there is a lack of important social information. Social navigation has been employed in research in usable security [4]. In this work visual displays of system actions, incorporating information from a community of users, are used to increase security. Similarly, decisions about which code to allow to run on a PC can be informed in a number of ways by social information. Social information indicates the commonality of problems. Social exchanges create an understanding of the threats and enables an open and transparent process of producing security information. When information is sourced from a user community it is easy to make clear why code is classified as it is (for example 'because seventy percent of people say so').

A process or application for peer production can employ social navigation to embed social information into a security mechanism. The information production process should leverage the community's collective wisdom to enable independent decision-making by the user about which programs to allow to execute. Individual decisions should be aggregated and made available to the users. The unique properties of security information should be considered in design - absolute definitions of good and bad code cannot be known *a priori* and built into a security mechanism. The application of social navigation to end-user security is the primary aim of my work.

The second major aim of my work is of a normative nature - to design for generativity. Non-peer production processes of security information are moving towards support of a less generative Internet by affording the user less freedom and control over applications used on his/her PC. The Trusted Computing Group (TCG) effort [5] is the foremost example of this trend and this effort is backed by major producers of consumer code advice. A social approach stimulates generativity by empowering the user with suitable information for independent decision-making - moving the decision from a trusted third party to the user's desktop. This move of the responsibility for decision-making is aligned with the trend of the research in usable security [6]. I argue that design principles for generativity are almost directly opposed to those of the TCG.

The main contribution of my work is the combination of peer production concepts with social navigation and the application of this combination to security for end user PCs. The work includes the development of a set of design considerations for peer production of security information with specific application to downloadable applications - to deal with the threat they may present through deceptive or badly written code. I argue that a 'neighbourhood watch'-type approach will be necessary for future security of end-user PCs and that this approach will assist in meeting the Grand Challenge [7] in information security while also stimulating a generative Internet.

References

1. J. Zittrain, "The generative internet," *Harvard Law Review*, vol. 119, May 2006.
2. Y. Benkler, *The Wealth of Networks*. Yale University Press, May 2006.
3. A. Adams and M. A. Sasse, "Users are not the enemy," *Communications of the ACM*, vol. 42, no. 12, pp. 40–46, 1999.
4. P. DiGioia and P. Dourish, "Social navigation as a model for usable security," in *SOUPS '05: Proceedings of the 2005 symposium on Usable privacy and security*. New York, NY, USA: ACM Press, 2005, pp. 101–108.
5. (2006, Feb.) TCG specification architecture overview. [Online]. Available: https://www.trustedcomputinggroup.org/groups/TCG_1_0_Architecture_Overview.pdf
6. M. A. Sasse and I. Flechais, "Usable Security: What is it? How do we get it?" in *Security and Usability: Designing Secure Systems That People can Use*, L. F. Cranor and S. Garfinkel, Eds. O'Reilly, Aug. 2005, pp. 13–30.
7. M. E. Zurko, "User-centered security: Stepping up to the grand challenge," in *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 187–202.

Understanding Tractable Constraints

Zoltán Miklós

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
zoltan.miklos@comlab.ox.ac.uk

1 Introduction

Constraint satisfaction problems (CSPs) are NP-complete in general, therefore it is important to identify tractable subclasses. A possible way to find such subclasses is to restrict the structure of the problem hypergraph. One of the most important structural restrictions is k-bounded hypertreewidth hypergraphs (Gottlob et al. [5]). There are many equivalent problems to CSPs, in particular, the evaluation of boolean conjunctive queries over a relational database [4]. A comparison of structural restriction methods for solving CSPs can be found in [3].

It is already known, that for a larger class, namely for the class of k-bounded generalized hypertreewidth hypergraphs, the CSP is solvable in polynomial time. Our research was motivated by the question of whether we can recognize these hypergraphs effectively.

2 The Acyclic Hypergraph Sandwich Problem

We show that the recognition problem can be reformulated in terms of the Acyclic Hypergraph Sandwich Problem. Using the reformulation, we could identify a special case of the problem with a polynomial time algorithm. If a hypergraph H has bounded dimension, that is, the size of it's largest edge is bounded by a constant d, then we can decide effectively whether it has k-bounded generalized hypertreewidth. We show, that the recognition problem is in this case in the low complexity class LOGCFL.

3 Component decomposition

Eventually, it turned out that the recognition problem for k-bounded generalized hypertreewidth hypergraphs is NP-complete [7]. On the other hand, other recent results show that the generalized hypertreewidth of a hypergraph can only be a constant factor smaller than the hypertreewidth, $hw(H) \leq 3ghw(H) + 1$, see [1]. The question naturally arises: is there a tractable decomposition concept that is more general decomposition method than hypertree decomposition with tractable recognition algorithm?

We give positive answer to this question: we introduce a new decomposition concept, a subclass of generalized hypertree decomposition, with a LOGCFL recognition algorithm, that captures a larger class of hypergraphs than hypertree decomposition. These results help to understand the nature of tractable cases and to identify further islands of tractability.

4 Degrees of acyclicity

Fagin [2] investigated different degrees of acyclicity for hypergraphs. While hypertree decomposition makes it possible to use the effective techniques of the most general acyclicity

concept of α-acyclicity for a larger class of hypergraphs, the more restricted β- and γ-acyclicity concepts possess useful properties. We investigate whether we can extend these concepts as well. So far, we could identify a class of hypergraphs, called β-covers that partially inherit the useful properties of β-acyclicity. We are currently working on a more proper generalization of the concept β-acyclicity.

Using recent results from complexity theory (see [6]), we pinpoint the precise complexity of the decision problems for acyclic hypergraphs : recognizing α-, β-, and γ- acyclic hypergraphs are complete for deterministic logarithmic space.

References

1. Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree-width and related hypergraph invariants. In *Proceedings of the 3rd European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB'05)*, volume AE of *DMTCS Proceedings Series*, pages 5–10, 2005.
2. Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM (JACM)*, 30(3):514–550, Jul 1983.
3. Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artifical Intelligence*, 124(2):243–282, 2000.
4. Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3):431–498, 2001.
5. Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences (JCSS)*, 64(3):579–627, May 2002.
6. Omer Reingold. Undirected st-connectivity in log-space. In *Proceedings of the thirty-seventh annual ACM Symposium on Theory of computing*, pages 376–385, 2005.
7. Thomas Schwentick, Georg Gottlob, and Zoltán Miklós. private communications, 2005.

Extensible Non-Cooperative Games

Yong Xie

Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.
Yong.Xie@comlab.ox.ac.uk

Game theory studies the strategic interactions among players (agents) using a game-theoretical approach, e.g. Nash Equilibrium [2] is a set of strategies, one for each player, such that no player has incentive to unilaterally change his/her action. The area that links game theory with computer science, called *Algorithmic Game Theory* (AGT), has recently received an increasing amount of attention in the theoretical computer science community, and has many potential applications in computer science, such as the inter-domain routing [1]. However, one of the main obstacles that prevent AGT to be widely used in large systems such as the Internet, is its lack of extensibility or scalability:

- *Incomplete Information*: Information about the players, their possible moves, and respective payoffs, could be hidden, especially when the number of players is large and the players are under different administrative domains with restriction in sharing information. But certain abstraction of the hidden information can be obtained.
- *Dynamic Nature*: The players should be able to join or quit the game dynamically and seamlessly, i.e. there should be minimal effect to other players in playing their nash equilibrium strategies.

To resolve these issues, we propose the concept of *Extensible Games*[1] (X-games). Firstly, we introduce two ways of (de)composing games: additive and multiplicative (de)composition, i.e. each player plays a two-player *component game* against each other player, and the his/her choices are the same in each of these games; the utilities are added (weighted sum) and multiplied respectively.

Theorem 1. *Additive-(De)Composition is both sound and complete in preserving Nash Equilibrium, i.e. for each nash equilibrium of any n-player game, there exists a way to additive-decompose the game into $\binom{n}{2}$ 2-player component games such that every player's nash equilibrium strategy in every component game is the same as the one in the n-player game. Multiplicative-(De)Composition is sound in preserving Nash Equilibrium.*

In fact, for the additive and multiplicative compositions to work, it is not necessary for each player's nash equilibrium strategy in the n-player game to be same as the one in *every* 2-player component game. In other words, so long as the aggregate payoffs satisfy the condition for nash equilibrium in n-player game, some 2-player component games can be neglected given that certain conditions are met. We call such negligible 2-player component games *Incorporated Games*, and the conditions *Game Incorporation Conditions*, which can be tested by treating each incorporated game as a black-box. If a player is involved in $(n-2)$ incorporated component games, then he/she is an *Incorporated Player*, i.e. he/she can be added/removed from the game seamlessly without affecting other players' nash equilibrium strategies. A player who is not an incorporated player, is a *Core Player*.

We define the *Extensibility* of a game to be the number of incorporated players in a game, and the *Scope* of a game to be the number of core players in a game. Fixed scope or

[1] So far, we focus on non-cooperative games

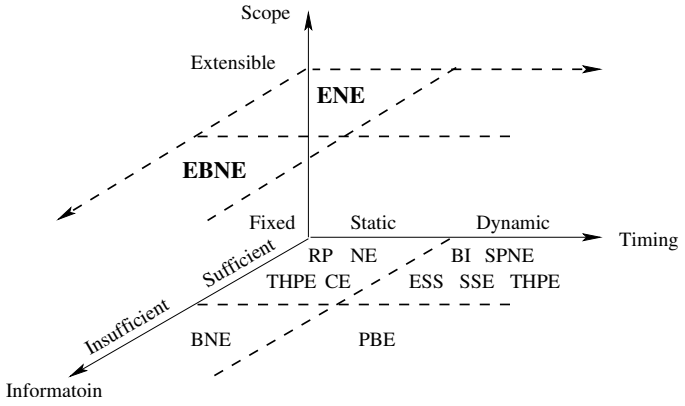


Fig. 1. Illustration of extensible equilibria (shown in bold) together with existing solution concepts in the literature. Games are classified according to timing (static/dynamic), information (sufficient/insufficient) and scope (fixed/extensible).

zero extensibility of a game means all the players in the game are core players. At the other extreme, all but one players are incorporated players – *fully extensible*.

Given a fixed-scope game $G = (P, U)$ where P is the set of players and U is the set of payoff matrices/functions, the corresponding extensible game is expressed as $G' = (P', U', C)$ where $P' \subseteq P$, $U' \subseteq U$, and C is the set of game incorporation conditions.

With the introduction of extensible games and the game incorporation conditions, it becomes inadequate to describe the information of a game using the completeness criteria. Existing game theory literature classify the information of a game to be either complete or incomplete, based on the knowledge of the players, their possible moves, and respective payoffs – lacking any one of the them will lead to a game of incomplete information. However, in extensible games, players with incomplete information ($U' \subseteq U$) could also find their equilibrium strategies so long as the game incorporation conditions (C) are met, and such conditions can be interpreted as useful abstractions of the missing information. We propose the *sufficiency* of information (for players to choose equilibrium strategies) to be a more precise way to describe information of a game than completeness.

The solution concepts for extensible games are extensions of the ones in fixed-scope games by classifying players into incorporated or core players – *Extensible Equilibria*. For example, Nash Equilibrium in a static fixed-scope game with sufficient information will yield *Extensible Nash Equilibrium* (ENE) in the corresponding static extensible game with sufficient information. Figure 1 depicts two sample extensible equilibria solution concepts: Extensible Nash Equilibrium (ENE), and Extensible Bayesian Nash Equilibrium (EBNE).

More details can be found in [3].

References

1. Joan Feigenbaum, Christos Papadimitriou, Rahul Sami, and Scott Shenker. A bgp-based mechanism for lowest-cost routing. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 173–182, New York, NY, USA, 2002. ACM Press.
2. Jr. J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–296, 1951.
3. Yong Xie. Extensible non-cooperative games. Transfer thesis, September 2006.

Author Index

A. W. Roscoe, 5
Andrew Cooper, 1

Brian Harrington, 29

Chris Dilloway, 3

Elnar Hajiyev, 21
Eric Kerfoot, 15

Gwyn Skone, 13

Heiko Helble, 25

James Smith, 27
Jonathan Cooper, 23
Julian Tibble, 7

Kris de Volde, 21

L. H. Nguyen, 5

Mathieu Verbaere, 21
Max Loubser, 31

Neil Ongkingco, 11

Oege de Moor, 21

Pavel Avgustinov, 9
Peter Y.H. Wong, 19

William Blum, 17

Yong Xie, 35

Zoltán Miklós, 33