WHAT’S GOING ON IN THE LOGIC AND COMPUTATION GROUP?

Michael Fisher

Department of Computer Science, University of Liverpool, UK
Aims:

- to introduce members of the Logic and Computation research group to you;
- to give you a flavour of some of the work going on within the group; and
- to encourage you to collaborate with members of the group.

If in doubt, see

http://www.csc.liv.ac.uk/research/logics
Who are we?

Lecturing Staff:

Profs  Michael Fisher, Frank Wolter
SLects  Clare Dixon, Ullrich Hustadt
Lects  Boris Konev, Alexei Lisitsa

Grant Malcolm, Vladimir Sazonov

RAs  Andrey Bovykin, Louise Dennis, Dmitry Tishkovsky

PhDs  Celia Casado, Anthony Hepple, Richard Molyneux

Daniel Pokrywczynski, Dirk Walther, Matt Webster
(+ 2 more due to arrive)

Group created: Jan 2001
What do we do?

Computational processes:
- simple and unambiguous specification
- analysis of properties, via logical routes
- exploration of new computational concepts

→ Logic

Logical descriptions:
- is it true? → automated proof
- is it correct? → formal verification
- is it implementable? → direct execution

→ Computation
Lecturing Staff Research Interests

**Michael:** formal verification; programming languages; temporal logics; agent-based and ubiquitous systems

**Frank:** modal/temporal logics; knowledge representation; expressive description logics; spatial reasoning

**Clare:** modal/temporal logics; security; proof methods

**Ullrich:** modal/temporal/description logics; proof methods

**Boris:** temporal logics; proof complexity; security

**Alexei:** formal verification; supercompilation; temporal logic

**Grant:** algebraic specification; computer virology

**Vladimir:** sequential functionals; semi-structured databases
Brief Selection of Current Work

There follows a number of examples of the current work being carried out in the group.

All areas would welcome collaboration with experts in complexity and algorithms — i.e. you!

Since my understanding of all these areas is limited, I encourage you to contact the people involved directly.

Note also that we are probably the world’s foremost research centre for temporal logics, modal logics and combinations thereof — theory, reasoning methods, applications.
Conservative Extensions of Ontologies (1)

Involved: Frank Wolter

Ontologies \((\mathcal{T}, \text{Sig})\) consist of a logical theory \(\mathcal{T}\) over a signature, \(\text{Sig}\). \(\text{Sig}\) is the vocabulary used to describe the domain of interest and \(\mathcal{T}\) specifies the meaning of the symbols in \(\text{Sig}\).

In applications such as the semantic web, medical informatics, and bio-informatics, ontologies are not static. In particular, such applications frequently require

- extensions of ontologies,
- refinements of ontologies, and
- merging of ontologies.
All of these changes can be captured by the concept of a conservative extension.

Let $\mathcal{T}$ be an ontology and $\mathcal{T}'$ be a set of axioms. Then $\mathcal{T} \cup \mathcal{T}'$ is called a conservative extension of $\mathcal{T}$ if, for every $\varphi$ over the signature $\text{Sig}$,

$$\text{if } \mathcal{T} \cup \mathcal{T}' \models \varphi \text{ then } \mathcal{T} \models \varphi$$

Example [merging]: let $\mathcal{T}_1$ and $\mathcal{T}_2$ be ontologies — is $\mathcal{T}_1 \cup \mathcal{T}_2$ a conservative extension of both $\mathcal{T}_1$ and $\mathcal{T}_2$?

Key work is on:

*assessing the decidability and complexity of finding conservative extensions within expressive description logics.*
Theory of Web-like Databases (1)

Involved: Vladimir Sazonov; Alexei Lisitsa; Richard Molyneux

Standard (relational) databases (DBs) correspond to sets, but *semistructured* or *Web-like* DBs (WDBs) require more.

Consider the WWW. It can be seen as a set of relations between the current node and other (reachable) nodes. e.g.

\[ t = \{a_1 : v_1, \ldots, a_n : v_n\} \]

describes a fork

\[ t \xrightarrow{a_1} v_1, \ldots, t \xrightarrow{a_n} v_n \]

- labels on edges carry elementary information
- labelled edges are analogs to *clickable links* in WWW
However, nodes in WWW can link to themselves. For example, $x = \{a : x\}$ generates an $a$ labelled loop $x \xrightarrow{a} x \text{ or } a \bigcirc$

Moreover, different nodes (set names) may be treated as informationally equivalent. This leads to abstract hyperset theory as a database approach.

A query language $\Delta$ for WDBs is being developed and investigated theoretically. The key results are on:

characterising the expressive power of various versions of query language $\Delta$ over such WDBs in terms of complexity theory (PTIME, LOGSPACE).
Involved: Grant Malcolm; Matt Webster

Imagine that a typical virus (embedded in assembly language) is of the form

set A = 1
set B = 2
if \((A>0) \& (B>0)\) then ‘BAD THING’

Virus checkers check for such instruction sequences.

Metamorphic viruses modify the code sequence to disguise the virus, e.g. by inserting irrelevant instructions.

This ensures the code is *syntactically* different to that hunted by virus checkers, but remains *semantically* equivalent.
Example:

goto L1
L2: if (A>0) & (B>0) then "BAD THING"
L1: set C = 0
    set B = 1
    set A = 1
    increment B
    goto L2

So: why not build an algebraic theory to describe properties of instruction sequences and then show above is equivalent to the earlier virus.

Algebraic tools, such as OBJ, are now being used to detect such metamorphic viruses.
You need to check that no intruder can

a) discover a private key they are not allowed to, or
b) discover personal information of others, etc.

Logical abstraction of knowledge allows us to represent the information within different components during the computation.

\[ K_A K_B \text{content} \land K_A \neg K_C \text{content} \]

\( K_X \phi \) means that agent \( X \) knows \( \phi \).
However, the agent’s knowledge changes over time, which naturally leads us to use temporal logics of knowledge.

\[ K_A K_B \text{content} \land K_A \neg K_C \text{content} \land \lozenge K_C \text{content} \]

Although such logics have a high complexity (PSPACE, at least), we have developed automated tools for proving properties → resolution, tableau

Current work concerns

- application of techniques to a wider class of security protocols,
- improvement in tool efficiency, and
- refinement of logic (and associated proof methods).
Translation-based theorem proving (1)

Involved: Ullrich Hustadt

Problem: - required is a calculus for $\text{Logic}_1$
- available is a calculus for $\text{Logic}_2$

Solution: - translate problems from $\text{Logic}_1$ into $\text{Logic}_2$
- then use the calculus for $\text{Logic}_2$

Approach requires

\[ \Pi: \text{Logic}_1 \longrightarrow \text{Logic}_2 \]

such that

\[ \models_{\text{Logic}_1} \varphi \text{ iff } \models_{\text{Logic}_2} \psi \rightarrow \Pi(\varphi) \]

defines $\text{Logic}_1$

and $\Pi$ is effective and captures $\text{Logic}_1$ as well as possible.
Translation-based theorem proving (2)

Using the translation approach, we can preserve

- the structure of formulae of Logic$_1$, and
- the complexity of the satisfiability problem in Logic$_1$

Logic$_1$:
Combination of modal/description logics
Examples: $\mathcal{SHIQ}$, OWL-DL, K, KD, KD45, S5, \ldots

Logic$_2$:
Fragment of first-order logic
Examples: DL$^*$, Maslov’s class K, fluted logic,\ldots

Calculus for Logic$_2$:
Ordered resolution with selection or
Basic superposition + clause form transformation

$\rightarrow$ effective and efficient decision procedures for Logic$_1$
Involved: Alexei Lisitsa

Let $S$ be a parameterized system (a protocol), about which we would like to establish some safety property, $P$.

We write a program $\varphi_S$ simulating execution of $S$ for $n$ steps, where $n$ is an input parameter.

Let $T_P(\_)$ be a testing program which, given a state $s$ of $S$, returns the result of testing the property $P$ on $s$.

We use a functional programming language (Refal) to implement a program $T_P(\varphi_S(n))$ and a supercompiler (SCP4) to transform a program to a form, from which one can easily establish the required property.

→ typically, just a syntactic check.
Supercompilation is a semantics-based program transformation technique devised by Turchin. Main idea is to carry out symbolic execution of a functional program running on partially defined input. This analysis is used to formulate a simpler program demonstrating the same behaviour.

Verification so far:
- of parameterized cache coherence protocols;
- of Petri nets; and
- of a Java Metalock Algorithm.

When it works it is very efficient (i.e. seconds). In some cases (i.e Futurebus protocol) it works considerably faster than any other methods (known to us)!
Involved: Michael Fisher; [MikeW]

Agents can be programmed in any language: Java; C; XML. However, as the key aspect of agents is to describe not just what and agent does, but why it does it, it is very useful to have higher level programming concepts:

- beliefs — information the agent has;
- goals — motivations the agent has for doing something;
- plans — mechanisms for achieving goals; and
- deliberation mechanisms for deciding between goals/plans.

High-level agent programming languages provide these.
Previous work:

N.B., the *trick* here is to encode beliefs, goals, etc., within the complex state of the JPF2/Spin state machine.
Current work:

\[
\text{AgentSpeak} \quad 3\text{APL} \quad \text{METATEM} \quad \downarrow \quad \downarrow \quad \downarrow
\]

\text{AGENT INTERFACE LAYER}

\text{AJPF}

\text{AJPF} is essentially JPF2 with the theory of AIL \textit{built in}.

\text{AJPF} incorporates slicing, abstraction, etc.
Involved: Clare Dixon; Michael Fisher; Boris Konev

Model checking is the most popular temporal verification technique.

Deductive temporal verification is seen as too complex, with PSPACE-complete decision procedures, versus polynomial model checkers.

But why?

One obvious reason is that, if we specify a system by the states it is in, then we spend a lot of time specifying that the system cannot be in two states at once, etc.
What if we build into the logic an ‘XOR’ property, with exactly one proposition from \( \{s_1, \ldots, s_n\} \) being true at any moment?

Then we can specify simple systems just by

\[
(s_1 \land l_1) \implies \lozenge s_2 \\
(s_1 \land l_2) \implies \lozenge (s_1 \lor s_2) \\
(s_2 \land l_3) \implies \lozenge s_2
\]

without explicitly having

\[
\square (s_1 \oplus s_2) \land \square (l_1 \oplus l_2 \oplus l_3)
\]

For this logic, complexity of proof search is polynomial!
Current Funded Projects [all EPSRC]

- Dynamic Ontologies: a Framework for Service Descriptions
  [expressive description logics: analysis; complexity; tools]
  — *Frank Wolter (PI)*, *Clare Dixon*, *Boris Konev*

- Knowledge Representation and Reasoning about Distances
  [spatial logics: analysis; complexity; tools]
  — *Frank Wolter (PI)*, *Ullrich Hustadt*

- Practical Reasoning Approaches for Web Ontologies and Multi-Agent Systems
  [translation-based methods/tools for complex logics]
  — *Ullrich Hustadt (PI)*

- Model Checking Agent Programming Languages
  [verification methods for agent programming languages]
  — *Michael Fisher (PI)*, *Mike Wooldridge (Agents)*
Aims to:

*develop, apply and exploit high quality formal verification techniques with the aim of increasing industrial, commercial and governmental collaboration at local, regional and international levels.*

Ongoing collaboration with: SciSys; Sony-Ericsson (?).

Promising connections with: BAE Systems; Vectra; Deloitte.
I have provided an overview of some of the activities within the Logic and Computation group.

As you have seen, many involve (or need) work on more efficient algorithms or refined complexity.

We are always happy to discuss our work further, or to collaborate directly.