GC open problems +

experiences with Z/Eves,

an ACL2-like prover

Leo Freitas, University of York

ACL2 Seminar @ Austin, Texas

25 March 2008
Outline

- **Purpose of this visit**

- **Verification Grand Challenge: context, motivation, goals**
  - pilot projects: past and current
  - open problems: come and join us :-) 

- **Z/Eves, and ACL2**
  - mechanising Z mathematical toolkit and UTP
  - “waterfall” and proof strategy comparison

- **Conclusions**
  - tools, theory, experiments
  - future collaboration
Hoare’s Verification Grand Challenge (GC6)

A mature scientific discipline should set its own agenda and pursue ideals of purity, generality, and accuracy far beyond current needs

what should we do?

- achieve a significant body of verified programs
- precise external specifications
- complete internal specifications
- machine-checked proofs of correctness
- a collection of verified programs, e.g.,:
  - 6,000,000 lines (e.g., Cousot’s Airbus A380 FCSW)
  - replacing existing unverified ones (e.g., UNIX-POSIX)
What is the philosophy?

- Karl Popper's — falsifiability principle (i.e., know your boundaries)
  expose it to the scenarios it is most likely to fail

- George Polya's — proof paradox (i.e., be general; just enough)
  more general theorems can be easier to prove

- Irwin Lakato's — praise to refutation (i.e., failure lessons)
  we learn as much, if not more, from refutations as with proofs

- Tony Hoare's — success benchmark (i.e., idea becomes widespread)
  reuse is the sign of successful progress
Goals of the Repository

1. accelerate development of verification technology
2. provide focus for verification community
3. provide open access
4. collect challenging applications
5. identify key metrics
6. enumerate challenge problems
7. standardise formats
8. define quality standards
9. curate gathered information

http://vsr.sourceforge.net/gc6index.htm [York]
http://qpq.csl.sri.com [SRI]
Purpose of this visit

Proof and engineering with ACL2

- learn ACL2 + understand how to use it as a Z back engine
- attempt to mechanise parts of the Z/Eves math. toolkit
- understand the engineering behind linking Z to ACL2 (e.g., quantifiers + set rep. + set comprehension + ???)
- discuss Z/Eves (or ACL2) proof-style as attractive to students

Grand challenge projects

- raise awareness to the GC work
- research collaboration with UT @ Austin
- exchange ideas on data curation (e.g., ACL2's :doc)
- For those not familiar with Z, Schemas 101 notes...
Pilot projects

- Mondex
- Verified File Store
- Free RTOS
- Radio Spectrum Auctions
- Tokeneer ID Station
- Cardiac Pacemaker
- Operating System Kernels
- Xenon High Assurance Hypervisor

* and others: TCP/IP, pointer-rich C-code analysis, Apache, etc.

* over 80 publications (20 journal, 60 conference) [Jun/08]
Mondex — [2006]

- electronic purse hosted on a smart card
- developed to high-assurance standard ITSEC Level E6
- consortium led by NatWest, a UK high-street bank
- purses interact using communications device
- strong guarantees needed that transactions are secure
- in spite of power failures and mischievous attacks
- electronic cash can't be counterfeited
- transactions completely distributed: no centralised control
- all security measures locally implemented
- no real-time external audit logging or monitoring
Mondex abstract world and its properties

\[ [\text{NAME, CLEAR}] \]

\( \text{AbPurse} \equiv [\text{balance, lost : N}] \)

\( \text{AbWorld} \equiv [\text{abAP : NAME} \mapsto \text{AbPurse}] \)

Security properties

\( \text{NoValueCreation} \equiv [\Delta \text{AbWorld} | \text{totalBal}_{\text{abAP}'} \leq \text{totalBal}_{\text{abAP}}] \)

\( \text{AllValueAccounted} \equiv [\Delta \text{AbWorld} | \text{totalBal}_{\text{abAP}'} + \text{totalLost}_{\text{abAP}'} = \text{totalBal}_{\text{abAP}} + \text{totalLost}_{\text{abAP}}] \)

\( \text{Authentic} \equiv [\Delta \text{AbWorld}; \text{name? : NAME} | \text{name?} \in \text{dom} \text{abAP}] \)

see PRG (p.13, 21) for couple of other properties...
The original (1996) verification

- seriously security critical
- Logica (and Oxford) used Z for development process
- formal models of system and abstract security policy
- hand proofs that system design possesses security properties
- abstract security policy specification about 20 pages of Z
- concrete specification (n-step protocol) about 60 pages
- verification suitable for external evaluation
  - about 200 pages of refinement proof *
  - 100 pages of derivation of refinement rules

* biggest Z spec of its day; recently (2009) superseded by iPhex
Mondex concrete world and its protocol

- **abstract level**: transfer of money from one purse to another
- **concrete level**: n-phased message exchange protocol
  - single transfer involves many messages
  - no control over concrete message (e.g., could have duplicates)
  - separate transactions distinguished via (increasing) numbers
- **status flags indicated at which point each purse is**
  - `eaFrom` — expecting any payer
  - `eaTo` — expecting any payee
  - `epr` — expecting payment request
  - `epv` — expecting payment value
  - `epa` — expecting payment acknowledgement

```latex
STATUS ::= eaFrom | eaTo | epr | epv | epa
```
Mondex concrete world and its protocol

\[ \text{CounterPartyDetails} \equiv [\text{name : NAME}; \text{value, nextSeqNo : } \mathbb{N}] \]
\[ \text{TransferDetails} \equiv [\text{from, to : NAME}; \text{value : } \mathbb{N}] \]
\[ \text{PayDetails} \equiv [\text{TransferDetails}; \text{fromSeqNo, toSeqNo : } \mathbb{N} | \text{from} \neq \text{to}] \]
\[ \text{MESSAGE} ::= \text{startFrom}⟨⟨\text{CounterPartyDetails}⟩⟩ \text{ unprotected} \]
\[ | \text{startTo}⟨⟨\text{CounterPartyDetails}⟩⟩ \]
\[ | \text{readExceptionLog} \]
\[ | \text{req}⟨⟨\text{PayDetails}⟩⟩ \text{ assumed crypto.} \]
\[ | \text{val}⟨⟨\text{PayDetails}⟩⟩ \]
\[ | \text{ack}⟨⟨\text{PayDetails}⟩⟩ \]
\[ | \text{exceptionLogResult}⟨⟨\text{NAME} × \text{PayDetails}⟩⟩ \]
\[ | \text{exceptionLogClear}⟨⟨\text{NAME} × \text{CLEAR}⟩⟩ \]
\[ | \perp \text{ forged message} \]

- \textbf{complete valid transaction is made of:}

  \[ \text{startFrom} \rightarrow \text{startTo} \rightarrow \text{req} \rightarrow \text{val} \rightarrow \text{ack} \]
Mondex concrete world and its protocol

\[\text{ConPurse}\]
\begin{align*}
\text{balance} & : \mathbb{N} \\
\text{exLog} & : \mathbb{P} \text{ PayDetails} \\
\text{name} & : \text{name} \\
\text{nextSeqNo} & : \mathbb{N} \\
\text{pdAuth} & : \text{PayDetails} \\
\text{status} & : \text{STATUS}
\end{align*}

\[
\forall \text{pd} : \text{exLog} \bullet \text{name} \in \{ \text{pd.from, pd.to} \}
\]

\[
\text{status} = \text{epr} \Rightarrow \text{name} = \text{pdAuth.from} \\
\land \text{pdAuth.value} \leq \text{balance} \\
\land \text{pdAuth.fromSeqNo < nextSeqNo}
\]

\[
\text{status} = \text{epv} \Rightarrow \text{pdAuth.toSeqNo < nextSeqNo}
\]

\[
\text{status} = \text{epa} \Rightarrow \text{pdAuth.fromSeqNo < nextSeqNo}
\]
The players

- Alloy (MIT)
- Event-B (Southampton)
- OCL (Bremen)
- PerfectDeveloper (Escher)
- $\pi$-calculus (Newcastle)
- Raise (Macao/DTU)
- Z (York)

agreed to work for one year, without funding

...separately and silently:

- a group in Augsburg began work using KIV and ASMs
Two distinct approaches

- **Archaeologists**
  - make as few changes as possible to original documentation
  - shouldn’t change models just to make verification easier
  - otherwise, how would we know that results had anything to do with the original specification?

- **Technologists**
  - use different proof technology now available
  - these new technology don’t work for Z
  - two choices:
    - translate existing models into new languages
    - create new models better suited to new tools
Z (York)

- **Leo Freitas and Jim Woodcock**
- **Z/Eves theorem prover**
- **only archaeologists**
- **mechanise all proofs, faithful to original formalisation**
- **two changes about (wrong) implicit finiteness assumptions**
- **progress: succeeded in mechanising (95% of) the project**
- **taken over two months to complete proof**
- **about 30 man-days using Z/Eves + few more days to writing up**
Results: Z/Eves effort

- informal hand proofs were useful and particularly thorough
- about 318 (115) verification conditions (VCs) of different complexity
- average of eight proof steps per VC (total of 4544 steps)
- complete precondition table not calculated originally
  - biggest proof effort (66% of 4544 steps)
  - proofs were massively (71%) reused
  - exposed missing invariants and finiteness problems
- built-in automation: 67% (3041) steps require little interaction
- other parts abstracted into general lemmas with some effort
- 23% (1049) intermediate steps require internal knowledge of Z/Eves
- 10% (464) creative steps require domain knowledge (∃ witnesses)
Results: Z/Eves effort

- **general theories needed about language constructs**
  - one-point-mu specification pattern
  - extended theory of functional overhiding
  - general finiteness proof strategy for schema bindings
  - minor bugs in the Z Standard

- **missing properties in intermediate design**
  - operations involving non-authentic purses are permitted
  - payment details must be finite and were not properly defined

- but Z/Eves theorem prover hasn’t changed in ten years

- original project could have mechanised the proof

- motivation and expertise lacking, not proof technology?

- Let’s build a time machine!
## Results: time travelling experiment

<table>
<thead>
<tr>
<th>Machine</th>
<th>Tardis</th>
<th>Zaratustra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month/Year</td>
<td>June 1995</td>
<td>March 2006</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel ATx586</td>
<td>Pentium 4 dual core</td>
</tr>
<tr>
<td>Speed</td>
<td>160MHz</td>
<td>2.2GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>80MB</td>
<td>2GB</td>
</tr>
<tr>
<td>HD</td>
<td>1.6GB</td>
<td>120GB</td>
</tr>
<tr>
<td>VM</td>
<td>160-190MB</td>
<td>3-4GB</td>
</tr>
<tr>
<td>OS</td>
<td>Win NT 4</td>
<td>Win XP 2002 Tablet SP2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification (Ch.3–7)</td>
<td>2:25:18</td>
<td>06:56</td>
<td>21 ×</td>
</tr>
<tr>
<td>Preconditions (Ch.8)</td>
<td>24:01:10</td>
<td>2:51:08</td>
<td>8 ×</td>
</tr>
<tr>
<td>Refinement (Ch.14–29)</td>
<td>power cut!</td>
<td>35:26</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification (Ch.3–7)</td>
<td>1:42:30</td>
<td>02:57</td>
<td>35 ×</td>
</tr>
<tr>
<td>Preconditions (Ch.8)</td>
<td>30:21:31</td>
<td>49:59</td>
<td>35 ×</td>
</tr>
<tr>
<td>Refinement (Ch.14–29)</td>
<td>1:00:00</td>
<td>10:29</td>
<td>6 ×</td>
</tr>
</tbody>
</table>
Mondex: current status

- Mondex case study shows that verification community is willing to undertake competitive and collaborative projects...
- ... and that there is some value in doing this

**collection of 8 papers in FACJ 20.1, Jan 2008**

- next:
  - other teams continue to work
  - different paradigms
  - detailed comparison of results
  - curation of key parts of experiments
Verified file store — [2007–]

- Joshi & Holzmann (JPL) — Spin
  - original problem, serious development effort
- Woodcock & Freitas (York) — Z/Eves
  - specifications, refinements, analyses in Z
- Butler (Southampton) — Event-B + Rodin
  - Event-B, Rodin toolset, hierarchical file system
- Oliveira (Minho) — VDM-tooset
  - hardware models in VDM
- Jackson (MIT) — Alloy
  - model & analysis of flash file system
Project roadmap

- **five strands of work**
  - **POSIX**
    - ✓ application programmer’s interface
  - **CICS**
    - ✓ transaction processing API
  - **INTEL**
    - * file system architecture
  - **NVMHCI**
    - * non-volatile memory host-controller interface
    - * device drivers for flash memory
  - **ONFI**
    - ✓ Open NAND Flash Interface
York accomplishments

- **abstract specification** of Posix interface in Z/Eves
  - mechanisation of Morgan & Sufrin’s Unix filing store (S)
  - mechanisation of Patrick Place’s specification of Posix 1003.21 Real-time distributed systems communication (1/2 S)
  - IEEE Posix Working Group interface requirements
- **concrete implementation** in Z hashmaps lifted from JML
- **abstract file maps, B⁺-tree implementation** (e.g., Z-to-VDM)
- **mechanised model of flash memory standard (ONFI)**
  - pages, blocks, logical units, targets, devices
  - memory addressing, defect marking
  - mandatory command set: reset, read, write, protect, page program, change read/write column, block erase
Introduction to CICS — IBM Hursley’s Reports

- **IBM’s transaction processing system providing:**
  - high availability, integrity, and performance
  - reliability, and scalability

- **Most important of services are spread across sets of APIs:**
  - continuous operation and parallel execution from multiple users
  - connectivity with database management systems
  - built-in facilities for ensuring data integrity
  - failure recovery and transaction back out

- **They have been formally specified in Z**

- **Original work won a Queen’s Award for design excellence!**
CICS within formalisation of POSIX file-stores

- part of next GC pilot project suggested by NASA’s JPL
- POSIX documentation guideline
  - standards with formal specification
  - Morgan & Sufrin’s UNIX file store
- thinking of CICS as a transaction processing interface to a file store
- cherry-picking CICS APIs that are related to file systems
  - mechanisation of the APIs using Z/Eves

MSc project for whole APIs: to be used in the GC POSIX pilot project

TODO: refinement proof of their link with lower level specs.
CICS File Control API — experimental (student) results

- 223 paragraphs, 84 operation (split in their cases)
- 73 precondition proofs (11 still to proof)
- 118 rewriting rules and theorems
- all discharged with 1213 proof commands
  - 56% no creativity, but just button clicks
  - 26% some knowledge of which rewrite rule to choose/click
  - 18% more creative steps (i.e., existential instantiations)
- 5 months work by an MSc student - no previous experience!
- quite nice learning curve when compared with other (Z) provers
- one SCP journal paper on the main results
What have we learned?

General theorem proving laws

• reuse of laws from Mondex and POSIX hashmaps

• reuse of (general) laws $\Rightarrow$ greater automation
  – finiteness: sets and schema bindings
  – free types: injectivity of constructors
  – schema calculus: surgical expansion

Strengthen the open source $\mathbf{Z/X}$!

• Canadian government negotiation for release of EVES’ licence

• looking for potential backend engines for $\mathbf{Z/X}$
On-going work

- **device driver correctness (NVMHCI)**
  - C programs
  - discovering assertions using **Daikon**
  - inspiring fresh work using ILP (inductive logic programming)
- **reclaim algorithms**
- **handling failure**
  - flash devices prone to unrecoverable block failure
  - fault tolerance must be organised by host
    - **workload-related aging**
  - wear-levelling algorithms minimise failure rate
Operating system kernels — [2008–]

- Pnueli’s original pilot project suggestion:
  - the Linux kernel

- possible departure points
  - Practical File System Design (Dominic Gianpaolo)
  - Formal Refinement for Operating System Kernels (Iain Craig)

Related Grand Challenge pilot projects

- free RTOS (open-source, but no specification)
  - widely used real-time operating system kernel
  - pointer-rich implementation

- Xenon hypervisor (high-assurance open-source separation kernel)
  - modelling/verification of abstract security policy using Circus
  - modelling/verification of critical concrete C-code parts
A verified simple OS kernel

- pilot project for the grand challenge in verified software
- Iain Craig (Northampton)
  - operating system kernel domain expert
  - modeller
- Leo Freitas & Jim Woodcock (York)
  - verifiers
- exploratory phase
- verified domain models

see Iain’s book...
Operating system kernels

- **kernel**: central component of most operating systems
- manages hardware/software resource interface: lowest-level abstraction layer for memory, processors and I/O devices
- provides facilities to applications processes through inter-process communication mechanisms and system calls
Kernel features

- low-level scheduling of processes (dispatching)
- inter-process communication
- process synchronisation
- context switching
- manipulation of process control blocks
- interrupt handling
- process creation and destruction
- process suspension and resumption
- targeting embedded devices
Kernel development

- very difficult and complex programming task
- critical component
  - correct functionality and good performance
- can’t use many abstractions that simplify programming
- typical for embedded and real-type systems
OS kernel pilot project


- **objectives:** demonstrate
  - feasibility of top-down OS kernels development
    * formal specification, refinement, implementation
    * Z notation, Dijkstra’s guarded-command language
    * hand-written proofs

- **pilot project objectives:** investigate
  - tractability of mechanising all models
  - tractability of formalising all proofs
  - feasibility of a tool chain
    * Z/Eves $\rightarrow$ ZRC-Refine/Gabriel $\rightarrow$ Spec#-Boogie/PL
  - curation of results in verified software repository
Z/Eves and ACL2 waterfalls

- prop. calc. + congruence closure + linear arithm.
- types + type prescription (grules) + proof context
- forward and backward chaining + compound recognisers
- unconditional + conditional rewrite rules
- normalisation + subsumption + eq. substitution

Differences

- expansion is up to the user + left-to-right reasoning
- not much heuristics over what to do with certain terms
- great handling of schema calculus and quantifiers
- axiomatised (non-computable) set-theoretical abstractions
- limited / restricted induction schemes
Z/Eves architecture

- front-end for Spivey’s Z for Lisp (EVES) FOL back engine
  - ZF set theory to quantifier-free FOL
  - tailored for the schema calculus
- typechecking + domain checking + interactive proof
- axiomatic, generic, abstract type definitions (\LaTeX+ XML)
- Python, Emacs, and shell interfaces: Win, Linux, Mac, Sparc
- fast and easy-to-use (e.g., student projects ex.)
- support for modules for specification and proofs
- questionably sound, yet rigourously built
- lack of exposure due to irrelevant secrecy from its contractor
Proving theorems with Z/Eves

- can handle pretty large (industrial-scale) specifications
- abstract (general) + concrete (finely guided) proof scripts
- highly automated handling of Z toolkit expressions
- nicely spread proof effort
- 70% blind (easy); 15% aware (medium); 15% creative (hard)
- easy to learn: students with no previous experience take sizeable experiments in around four months
- largest specs known: Mondex
  - 200 definitions, operations, schemas
  - 320 lemmas, theorems, automation rules
  - 4700 proof steps for the 320 proofs
  - runs all just over 2 hours
Z/Eves case studies: 2002—2009

- 2003: Specialised automata theory (225/205/?)
- 2004: Refinement calculus automation (86/91/?)
- 2005: Circus model checker + op. semantics (200/150/?)
- programmable link with Java and CZT (2Kloc)
- 2006: Mondex smartcards (200/220/4700)
- IBM CICS file (216/120/?) + task (218/121/1416) APIs
- 2007: Unix file stores (200/100/1200) + part of POSIX XNFF Std (150/100/1000)
- hashmap + B^+-tree + JML interfaces + journaling flash FS +
- ONFI flash hardware interface (58/64/?) + NVCHMI device drivers (2 Kloc)
- 2008: OS kernels embedded systems (?) + Xenon high-assurance hypervisor (?)
- 2009: Tokeneer ID station
- 2002-2009: set-theoretical col. of lemmas (-/118/?)
Free RTOS

- open source real-time mini-kernel (RTOS scheduler)
- WITTENSTEIN high integrity systems
- designed for real-time performance with limited resources
- accessible, efficient, and popular
  - widely used in many applications
  - rapid growth as a widely adopted open-source solution
  - > 5,000 downloads per month
  - ranked 250/170,000 on SourceForge
  - (officially) ported to 17 architectures
- 2,242 lines of code
Kernel features

- pre-emptive, co-operative, hybrid configuration options
- supports both tasks and co-routines
- unlimited tasks and priorities
- queues, binary semaphores, counting semaphores, recursive semaphores, mutexes for communication and synchronisation between tasks, or between tasks and interrupts
Requirements

R1  Host applications shall be permitted to be structured as a set of autonomous prioritised tasks.

R2  The highest priority task that is able to execute (not blocked or suspended) shall be scheduled.

R3  Execution time shall be shared between tasks of equal priority.

R4  The time taken for swapping from a lower priority task to a higher priority task shall be deterministic.

R5  Tasks shall be permitted to voluntarily yield.

R6  Tasks shall be able to create and delete other tasks.

R7  Tasks shall be able to change their priority or the priority of any other task in the system.

R15 All inter-process comm. shall use the same queue primitive.
Radio Spectrum Auctions

- contact: Robert Leese (Smith Institute)
- radio spectrum: economically valuable resource
- combinatorial auctions package bidding
- often accompanied by secondary markets
- underlying theory still being developed
- computational challenges of winner/price determination
- growing collection of case studies
Auction process

- **primary stage:** clock auction for price discovery
- **supplementary stage:** sealed bids
- implemented through web interface
- auctioneer sets prices in each category
- bidders respond with demand
- prices raised where demand exceeds supply
- 12 rounds of bidding per day
- auction rules ensure bidding activity levels
- limited scope for extensions to bidding windows
**Tokeneer ID Station**

- **contact: Rod Chapman (Praxis)**

- *Common Criteria EAL5 is achievable cost-effectively*

- *... by Praxis Correctness by Construction (CbyC)*

- *measured productivity and defect rates under controlled conditions*

- **what is Tokeneer?**
  - provides protection to secure information held on a network of workstations in physically secure enclave
  - demonstrates use of smart cards and biometrics
### Project statistics

<table>
<thead>
<tr>
<th></th>
<th>source lines</th>
<th>contracts &amp; annotations</th>
<th>loc/day coding</th>
<th>LOC/day entire project</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core (SPARK)</strong></td>
<td>9,939</td>
<td>16,564</td>
<td>203</td>
<td>38</td>
</tr>
<tr>
<td><strong>Support (Ada95)</strong></td>
<td>3,697</td>
<td>2,240</td>
<td>182</td>
<td>88</td>
</tr>
</tbody>
</table>

- **Total effort:** 260 man days
- **Team:** 3 people part-time
- **Total schedule:** 9 months elapsed
**NSA Conclusions**

- *CbyC produces high-quality, low-defect, cost-effective software*
- *conforms to Common Criteria EAL5+*
- *can be applied by others (student interns)*
- *results of independent testing and reliability analysis*
  - minor defects (in user-documentation only): two
  - major or critical defects: zero
- *results published in ISSSE 2006 conference*
- *entire project archive available under BSD-like licence*
Mini-challenges with Tokeneer

- more proofs
  - functional spec refines security policy
  - design refines functional spec
  - remaining security properties, in Z and in SPARK

- increased automation

- theorem prover tag-team show-down \((PS: \text{qnt-free})\)
  - prove all SPARK VCs with Simplifier vs Yices vs ACL2?

- new refinement from functional specification

- sparkinfo@praxis-his.com

- www.praxis-his.com

- www.sparkada.com
Cardiac pacemaker

- **contact:** Alan Wassyng (McMaster)
- **Boston Scientific mini-challenge**
- **specification** defines functions & operating characteristics, identifies system environmental performance parameters, and characterizes anticipated uses
- **the challenge**
  - anything up to complete version of the pacemaker software
- **certification framework to simulate licensing**
  - to explore licensing evidence and standards
  - objective basis for comparison between putative solutions
- **sqrl.mcmaster.ca/pacemaker.htm**
Conclusions

- GC work instigates experiments and usage of tools
- variety of experiments to strengthen tool support
- community effort is necessary for success
- invite ACL2-interested community to collaborate / participate
- good opportunity for papers on tools and experimental work


http://qpq.csl.sri.com [SRI]