The Verified Software Repository

Jim Woodcock

Marktoberdorf Lecture 1

6 August 2008

My lectures

1. *The Verified Software Repository*
2. *Mechanising an OS Kernel Poof*
3. *Formal Refinement of OS Kernels*
4. *A Unifying Theory of Undefinedness*

objectives:

- *provide an overview of experimental research in the Grand Challenge in Verified Software*

Summary

- *The Verification Grand Challenge*
- *The POSIX-compliant file-store*
- *Flash memory*
- *Conclusions*
- *Next lecture*

The Verification Grand Challenge

- Tony Hoare, Jay Misra (also GC6)

  automatically verified software: a grand scientific challenge for computing

- NSF, EPSRC meetings
- Zurich, Macao, Toronto conferences

  vstte.inf.ethz.ch

- workshops: ICECCS 2008, FM08, SBMF 2008, ABZ 2008, …

Publications

- JACM 2003, FACJ 2006, IEEE Computer 2006, Comm CSI 2007
- special journal issues
  - VSTTE: **FACJ 19(4)**, **JOT 2007**
  - LNCS 4171, 5295
  - Mondex: **FACJ 20(1)**
  - ICECCS: **SCP 2008**
- research roadmap **qpq.csl.sri.com**

Mission statement

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

Objectives

to achieve a significant body of verified programs that have:

- *precise external specifications*
- *complete internal specifications*
- *machine-checked proofs of correctness*
- *w.r.t. a sound theory of programming*

Deliverables

1. **comprehensive theory of programming**
   - practical features for reliable programs
2. **coherent toolset**
   - automating and scaling up the theory
3. **collection of verified programs**
   - replacing existing unverified ones
   - continuing to evolve as verified code

A VERIFIED SOFTWARE REPOSITORY

Repository contents

- ***representative classes of software***
- *principal application areas*
- *principal programming paradigms*
- *so at the end of the project:*
  - *no one will be able to say*
  - ***"you can't verify this kind of software"***
- *because we'll have done it!*

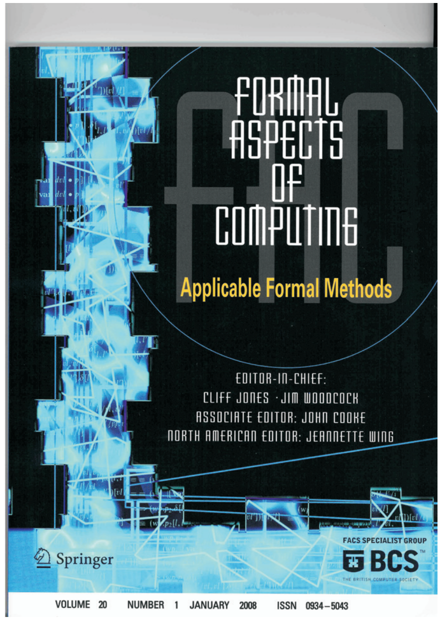
Pilot projects

- **Joshi & Holzmann**
  - a pilot project should be of sufficient

1. **complexity** that traditional methods are inadequate to establish correctness
2. **simplicity** that specification, design and verification could be completed by a dedicated team within 2–3 years
3. **importance** that successful completion would have an impact beyond the verification community

Grand Challenge pilot projects

- Mondex smart-card
- Pacemaker (McMaster)
  - ICSE 2009, SCC
- Tokeneer (Praxis)
- free RTOS
- Hypervisor (Microsoft)
- POSIX-compliant file-store
- Verified OS kernels



Mondex

- *electronic purse hosted on a smart card*
- ***ITSEC Level E6***
- *consortium led by NatWest*
- *purses interact using comms device*
- *strong guarantees for secure transactions*
- *power failures, mischievous attacks*
- *electronic cash mustn't be counterfeited*

The problem

- transactions completely distributed: no centralised control
- all security measures locally implemented
- no real-time external audit logging
- abstract security policy specification: 20pp
- concrete specification: 60pp
- refinement proof: 300pp
- **hand-written proofs**

The challenge

- sanitised version of Mondex documentation publicly available
- originally no question of mechanisation:

“mechanising such a large proof cost-effectively is beyond the state of the art”

- challenge: investigate automation that can now be achieved the correctness proofs

The Mondex experiment

- *one-year project*
- *original project ('96/'97) conducted in Z*
- *Alloy, ASM, Event-B, OCL, PerfectDeveloper, π-calculus, Raise, Z*
- *almost all found the same residual errors*
- *almost all required the same effort*
  - *total elapsed time*
  - *number of proof steps*
  - *level of automation*

Mondex in Z/Eves

- two months of work
- total number of proof steps = 4,544
- 3,041 trivial commands
- 1,039 intermediate steps
- 464 creative steps
- **the time machine**

The POSIX-compliant file-store

- *Pnueli's challenge:* ***verify the Linux Kernel***
- *Joshi & Holzmann's restriction*

> *verify small POSIX subset for flash-memory*
>
> *with strict fault-tolerance requirements*
>
> *for NASA missions*

- ***no corruption from power failure***
- ***technology: NAND-Flash Memory***

Mars Exploration Rovers Project

- **Reeves & Neilson, Spirit Flash Anomaly**
- **Spirit:** launched 10/6/03, landed 4/1/04
- $820M initial 90-day mission
- NASA website 5–11/1/04: 1.7B hits, 34.6 TB
- **21/1/04: contact lost with Spirit**
- storm, empty message, missed comms
- **no science for 10 days**
- most serious anomaly in four-year mission

What went wrong?

- fault caused by flash memory subsystem
- DOS representation of deleted files:
  - unused and deleted entries are different
  - deleted file entries represented in RAM
  - space is never released
- DOS Library configuration error
  - dynamic memory allocation
- Mem Library configuration errors
  - suspension for failed memory request

Consequences

- ***errors allowed all free memory to be consumed***
- *initiating task then suspended*
- *flight computer and FSW re-initialised*
- *cycle then repeated*
- *Spirit disabled*
- ***solution:*** *hardware reset, patch, re-install*

Lessons learnt

- ***extensive and comprehensive testing***
- *performed by multiple organisations*
- *operations team walked through launch, cruise, entry, descent, and landing*
- *anomaly was never seen during any test*
- *tests didn’t produce diversity or volume of data products*
- *didn’t reproduce every turn, manoeuvre and communication window performed in flight*

Initial subset of POSIX

- *no support for file permissions, hard or symbolic links, pipes, sockets*
- `create`, `open`, `close`, `read`, `write`, `truncate`, `ftruncate`, `stat`, `fstat`, `mkdir`, `rmdir`, `rename`, `opendir`, `readdir`, `rewinddir`, `closedir`, `format`, `mount`, `unmount`
- *not initially concerned with encryption, directory listing, regular expressions, …*

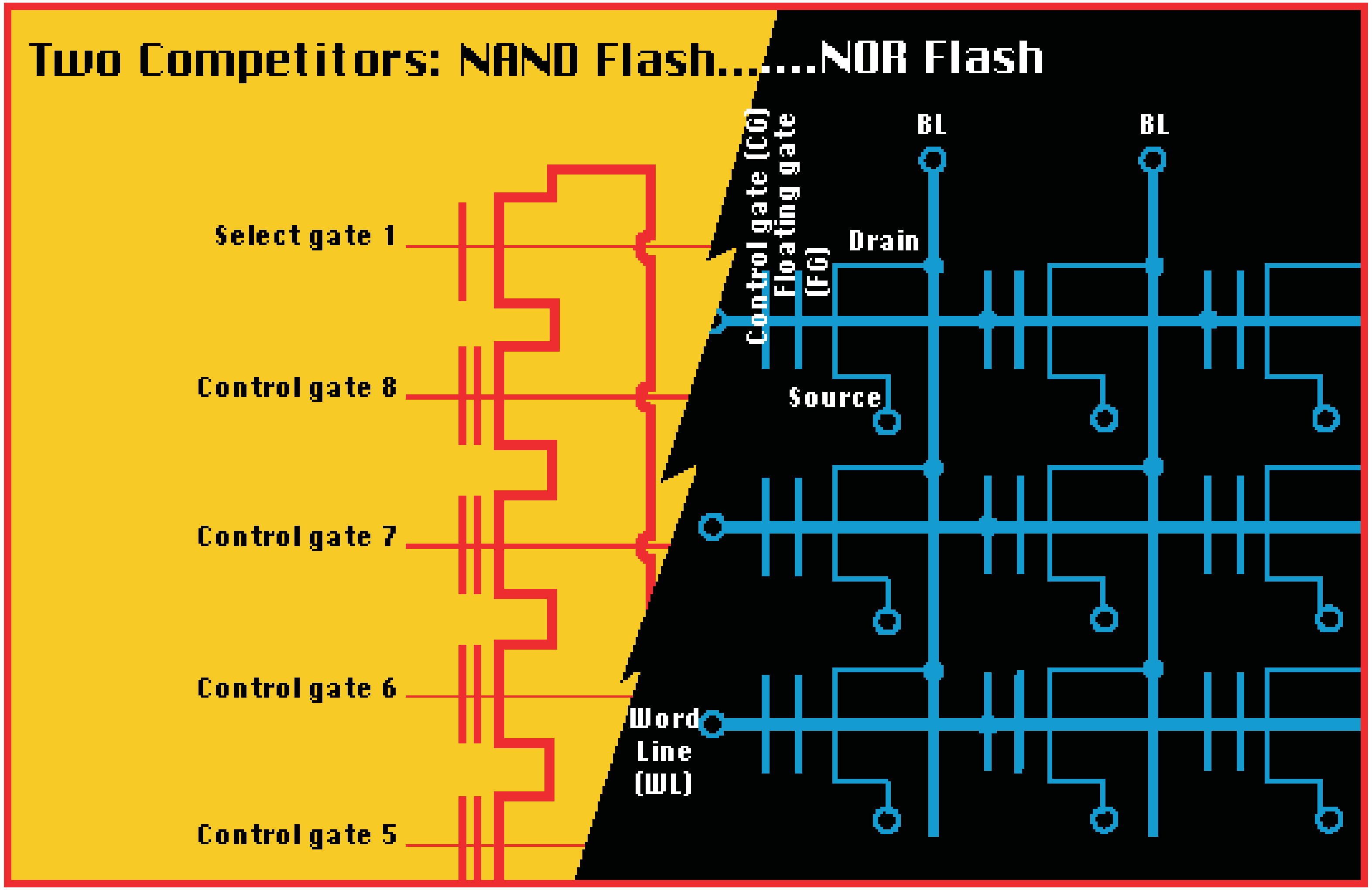
Flash memory

- ***electrically erased, programmable ROM***
- *nonvolatile and relatively dense*
- ***large erase blocks, limited erases***
- *sophisticated algorithms, data structures*
- *write operations can only **clear** bits*
- ***set** bits by erasing regions (2–500kB)*
- *two main technologies: NOR & NAND*

Motivation

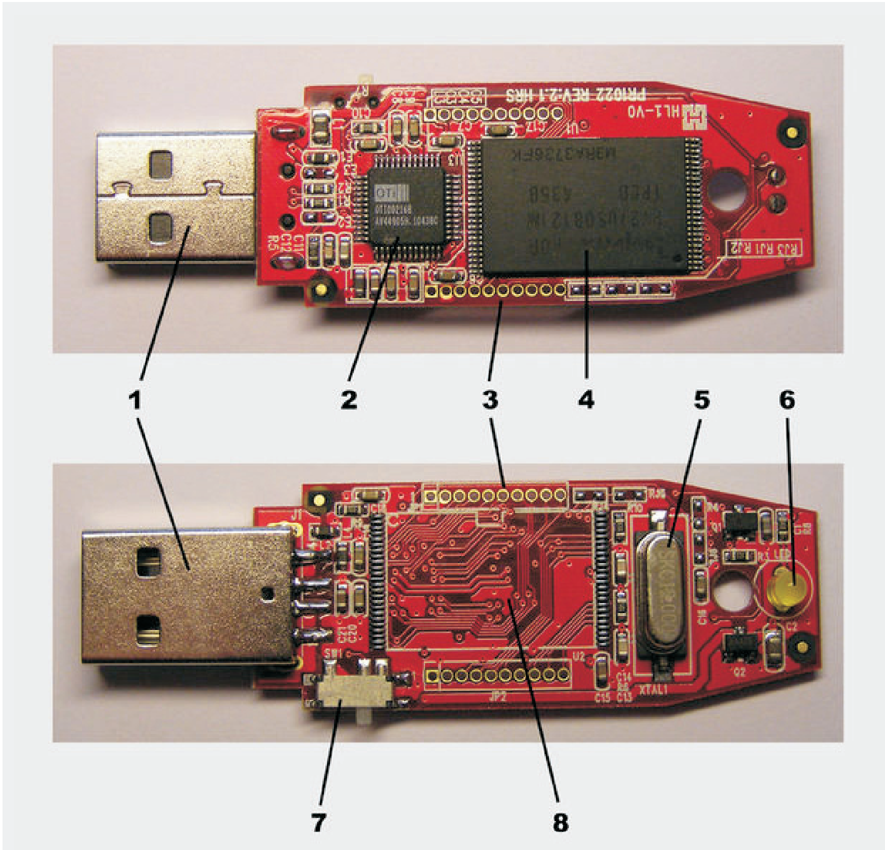
- GC flash file system mini-challenge
- major technology, major business
- sales turnover, worldwide
  - 2005: US$10.2B
  - 2007: US$15.2B
  - **2009: US$20.9B**

NAND versus NOR Flash II



A NAND-flash USB stick

1. USB connector
2. Controller
3. Test points
4. Flash memory chip
5. Crystal oscillator
6. LED
7. Write-protect switch
8. Second chip connector



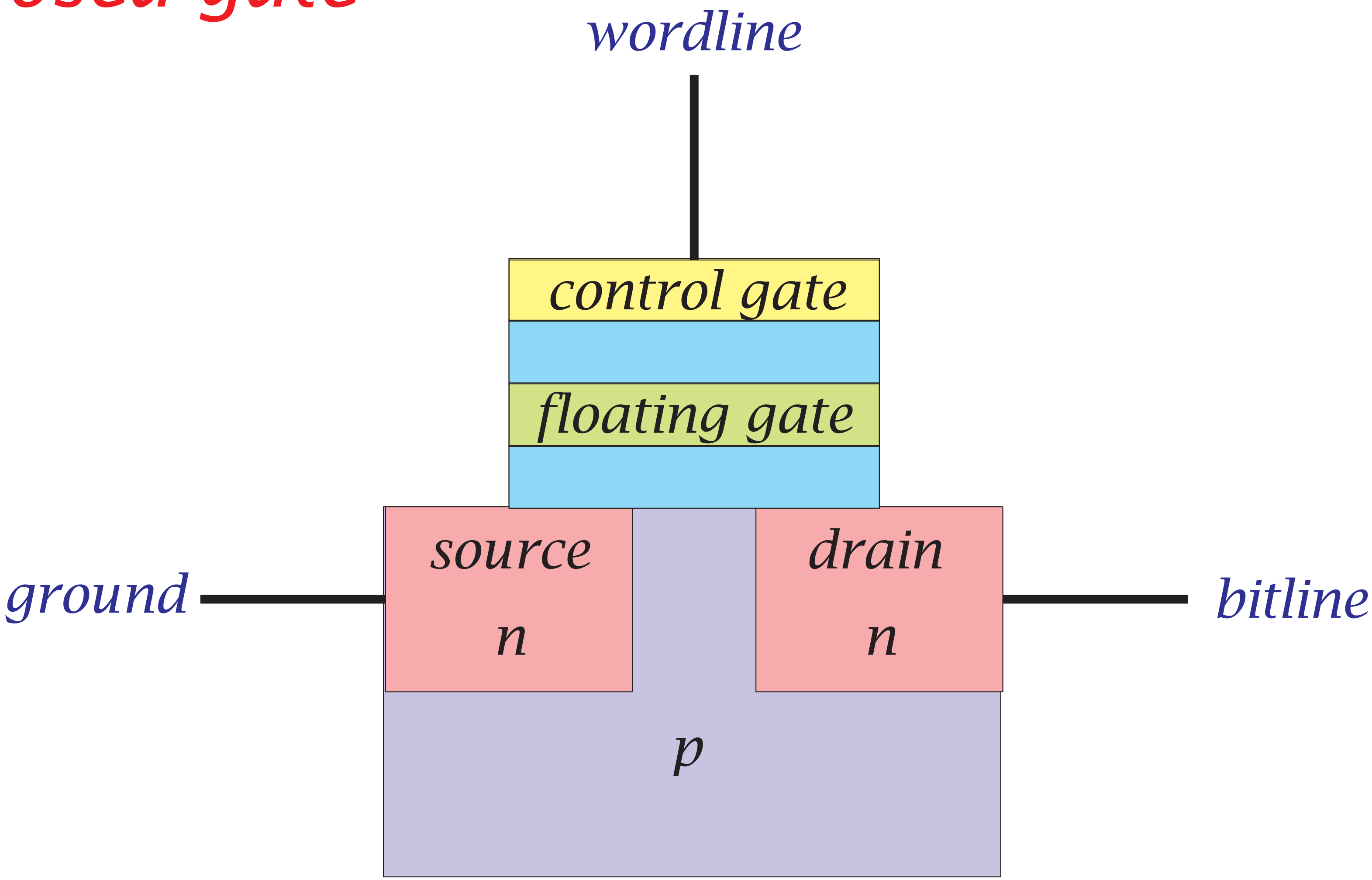
Writing a zero

- **memory cell: transistor with extra gate**
- source, drain, control, **floating** gates
- thin oxide layers isolate floating gate
- grounded source and control at programming voltage: electrons tunnel through oxide to floating gate
- extra negative charge in floating gate raises cell's turn-on threshold by increasing negative potential opposing voltage

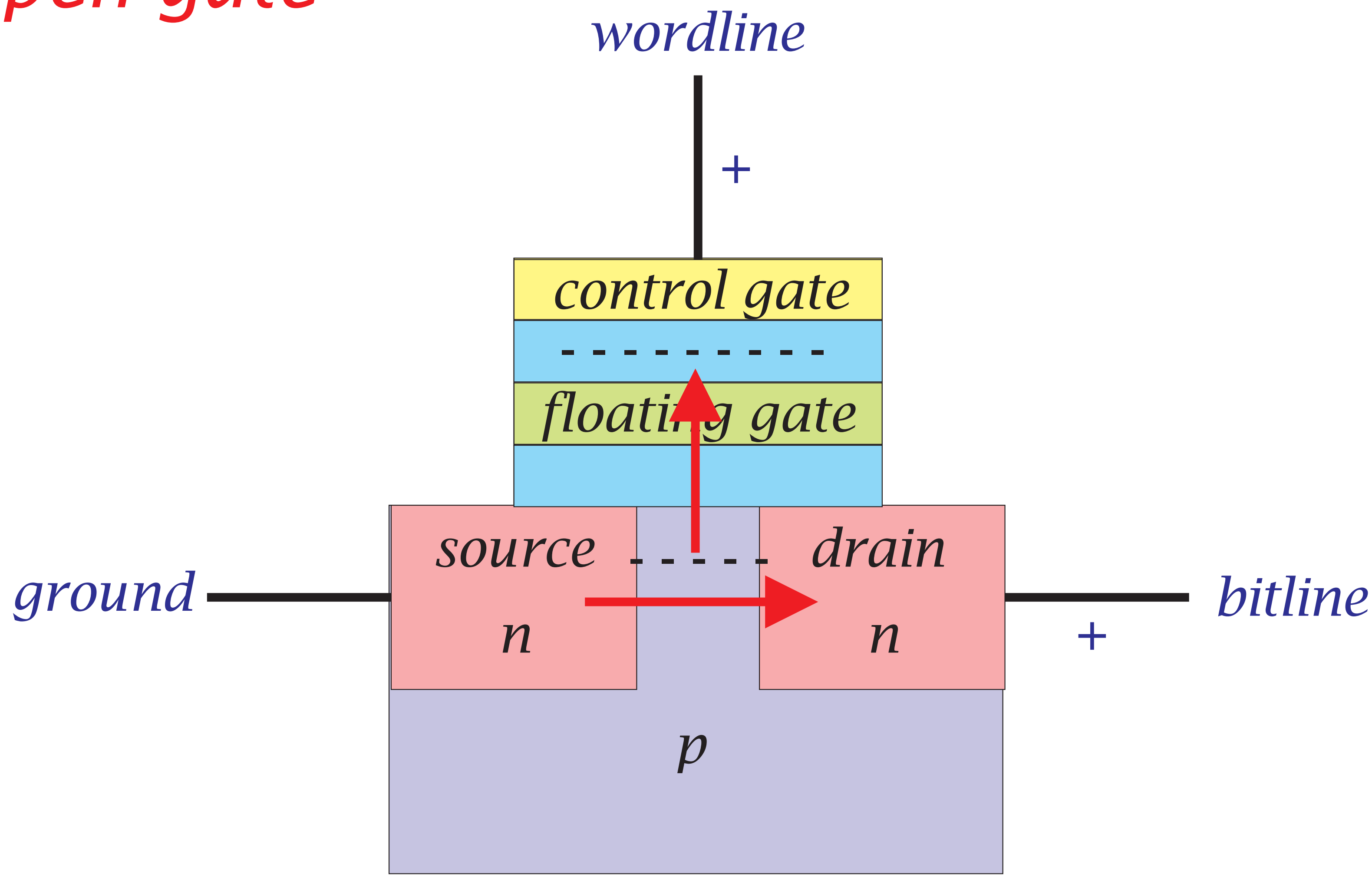
Erasing a cell

- *grounded control gate and source at programming voltage: removes electrons from floating gate reduces turn-on threshold*
- ***turns cell back to a one***
- *erasing doesn't happen as* ***quick as a flash***
- *takes* ***a l o n g time***
- *high voltage involved requires high current*
- *erase one group of cells at a time*

Closed gate


wordline
control gate
floating gate
source
n
drain
n
ground
bitline
p

Open gate


wordline
+
control gate
floating gate
source
n
drain
n
p
ground
bitline
+

Handling failure

- *flash prone to unrecoverable block failure*
  - ***workload-related ageing***
- *wear-levelling algorithms reduce failure rate*
- *need to model persistent failure*
- *spare page space assists with error recovery*
- *fault tolerance must be organised by host*
- *present a fault-free view to higher levels*

Accomplishments

- ***abstract specification*** *of POSIX in Z/Eves*
  - *mechanised Morgan & Sufrin Unix i/f*
  - *mechanised specification of Posix 1003.21 real-time distributed systems communication interface*
  - *IEEE Posix Working Group interface requirements*
- ***concrete implementation*** *in Z hashmaps lifted from JML*

- ***abstract file mappings, B^+-trees***
- ***mechanised model of flash memory***
  - *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*
- *benchtop prototype*

Conclusions

- *pilot project shows that the verification community is willing to undertake competitive and collaborative projects*
- *…and that there is some value in doing this*
- *we need to expand to work with different tools, techniques, paradigms*
- *we need comparisons of results*
- *we need to curate key parts of experiments*

Next lecture

- *pilot project: verifying OS kernels*
- *Mechanising an OS Kernel Proof*

Index

Mechanising an OS Kernel Proof

Jim Woodcock

Marktoberdorf Lecture 2

7 August 2008

Summary

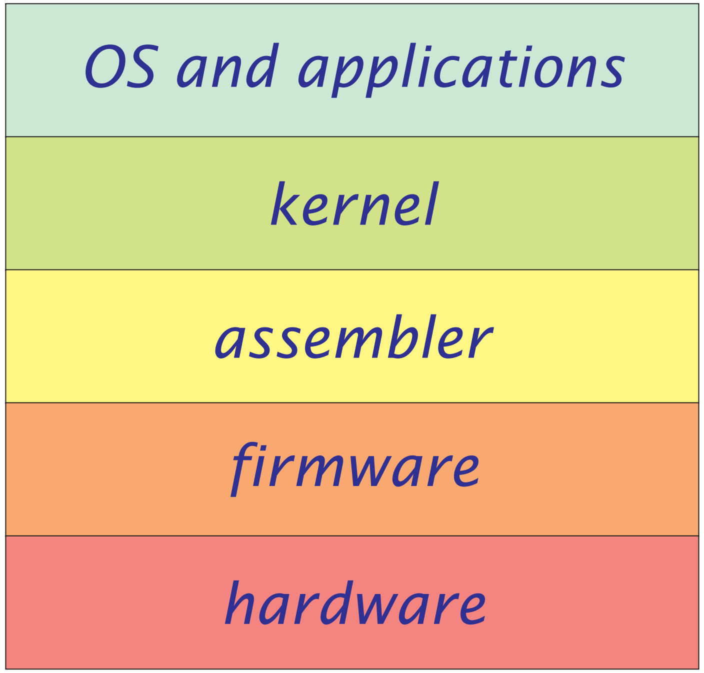
- A verified simple OS kernel
- Organisation
- Specification
- Proof statistics
- Conclusions

A verified simple OS kernel

- **pilot project for grand challenge in verified software**
- Iain Craig
  - OS kernel domain expert, modeller
- Leo Freitas & Jim Woodcock
  - verifiers, Z/Eves
- exploratory phase: verified domain models
- **no changes just to make proofs easier**

Operating system kernels

- **central to most OSs**
- manages hw/sw resources
- lowest abstraction layer for memory, processors and I/O devices
- provides inter-process communication and system calls



Kernel features

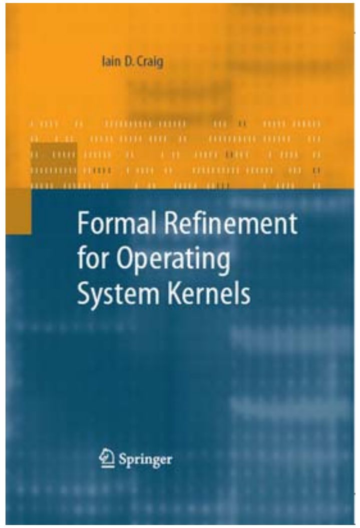
- low-level scheduling of processes
- inter-process communication
- process synchronisation
- context switching
- manipulation of process control blocks
- interrupt handling
- process creation and destruction
- process suspension and resumption

Kernel development

- ***complex programming task***
- *critical component*
  - *correct functionality*
  - *good performance*
- *can't use many simplifying abstractions*
- *typical for embedded and real-type systems*

Craig's OS kernel project

Iain D. Craig,
Formal Refinement for OS Kernels,
Springer, 2007



- **objectives: to demonstrate**
  - feasibility of top-down kernel development
    - specification, refinement, code, Z notation, GCL, **hand-written proofs**
    - ideal for VSR experiments

Craig's first OS microkernel

- *process table*
- *priority queue scheduler*
- *global semaphore table*
- *synchronous message passing system*
- *sleep mechanism*
- *initialisation and interface routines for system calls*

Craig's second OS microkernel

- ***separation kernel (Rushby)***

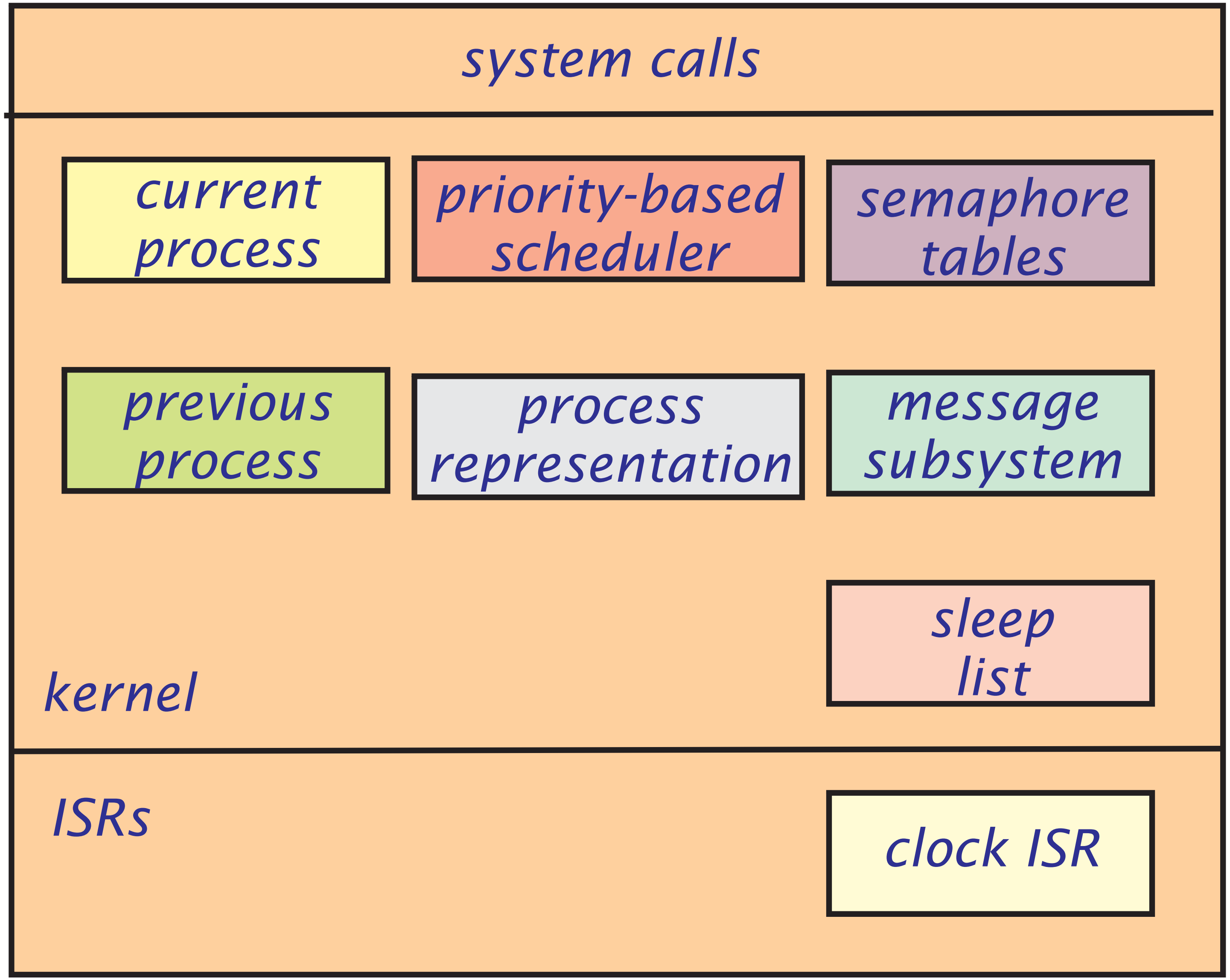
OS kernel pilot project objectives

- **to investigate**
  - tractability of mechanising all models
  - tractability of formalising all proofs
  - **feasibility of a tool chain**
    - Z/Eves
      - ⟶ ZRC-Refine/Gabriel
        - ⟶ Spec#-Boogie/PL
  - curation of results in VSR

Location

- ✓ A verified simple OS kernel
- ⇝ Organisation
- Specification
- Proof statistics
- Conclusions

Organisation



Process representation

- **process table PTAB**
- **used** set of allocated process identifiers
- set of mappings refined to one-dimensional arrays (vectors)
- mappings keyed by process identifiers

Uniform process representation

- **priority:** orders scheduler's ready queue
- -ve high, +ve low, default 0
- **state:** documents process and triggers context switch
- **stack:** address of process's stack-top (for context switching)
- **incoming msg:** latest synchronous message
- **waking time:** woken processes return to scheduler's ready queue

Scheduler

- ***ready*** *queue: highest priority at head*
  - *priority queue specified as separate module refined to chain*
  - *complex, but removes need to allocate additional storage inside kernel*
- *idle process identifier*
  - *explicit process: infinite loop with no body*
- *currently executing process identifier*
- *previously executing process identifier*

Semaphores

- counting semaphores for synchronisation
- kernel maintains single semaphore table
- table size is compile-time constant
- **operations on semaphores**
  - initialise
  - allocate semaphore
  - free semaphore
  - signal (V operation)
  - wait (P operation)

Semaphores

- semaphore table refined to bit maps
- semaphore contains counter and FIFO queue
- queue defined as separate type
- FIFO refined to chain: *next* map in *PTAB*
- chaining very useful
  - each semaphore has separate FIFO queue
  - vector: allocate space per semaphore
  - here: space is allocated once, but each FIFO can be arbitrarily long

Synchronous message passing

- **protocol**
  - receiving process enters psreceiving state
  - suspends and waits for sending process
  - on reception, enters psready state
  - joins scheduler's ready queue
- **if sending process's receiver is not in psreceiving state, sender must retry**

System calls

- create process
- terminate
  - finalisation for all non-initial processes
  - kills currently active process
- get process identifier
- send/receive a synchronous message
- allocate/deallocate semaphore
- wait/signal semaphore
- sleep

System call protocol

- disable interrupts
- perform operation
- re-enable interrupts
- ensures operation is indivisible
- most system calls execute quickly

Sleeping processes

- **high-level specification refined to chain**
- clock implemented as Interrupt Service Routine (ISR) or interrupt handler
- on every hardware clock interrupt, clock ISR increments tick
- processor ticks, say, every 10μs
- keep track of time since boot with (hour,minute,second) counters
- Intel IA32 has 18.4MHz clock
- inevitable clock drift

Low-level operations required

- **enable and disable interrupts**
- **return from interrupt (IRET)**
- **context switch:**
  - registers stored on top of process stack
  - no permanent store needed in PTAB
  - arbitrary number of registers in PTAB
- **half-context switch:** sets up initial process's registers on creation

Context switching

- ***scheduler raises context switch interrupt***
- *handled by ISR:*
  - *pushes outgoing process's registers onto its stack*
  - *pops incoming process's registers from the stack (from previous interrupt)*
  - *executes IRET and incoming process is switched in*
- *clearly impossible for initial process*
- *stack must be set up with dummy values*

Location

- ✓ *A verified simple OS kernel*
- ✓ *Organisation*
- ⇝ *Specification*
- *Proof statistics*
- *Conclusions*

Specification

$$\begin{array}{|l} minpid, maxpid : \mathbb{N} \\ \hline \langle\langle \textit{rule dPIDNotEmpty} \rangle\rangle \\ minpid \leq maxpid \end{array}$$

$$PID == minpid \mathrel{..} maxpid$$

$$\begin{array}{|l} nullpid : \mathbb{N} \\ \hline \langle\langle \textit{disabled rule dNullPID} \rangle\rangle \\ \forall p : PID \bullet p < nullpid \end{array}$$

$$GPID == \{nullpid\} \cup PID$$

The process state

$PSTATE ::= pstate$

PSTATE ::= *psterm* | *psrunning* | *psready* | *pswaitsema* | *pssleeping* | *pssending* | *psreceiving*

- *psterm:* terminated state
- *psrunning:* currently executing process
- *psready:* ready to execute, but not executing
- *pswaitsema:* waiting on a semaphore
- *pssleeping:* waiting for timer expire
- *pssending:* sending (maybe waiting)
- *psreceiving:* ready to receive

Process priorities

$$minprio, maxprio : \mathbb{Z}$$

$$\langle\langle \textit{ rule dPRIONotEmpty } \rangle\rangle$$

$$maxprio \leq minprio$$

$$PRIO == maxprio \mathrel{..} minprio$$

Messages

$[\ MSG\ ]$

$$
\begin{array}{|l}
nullmsg : MSG \\
msgsrc : MSG \rightarrow PID \\
msgdest : MSG \rightarrow PID \\
msgsize : MSG \rightarrow \mathbb{N}
\end{array}
$$

Addresses & time

$[\, WORD \,]$

$$
\begin{array}{|l}
nulladdr, maxaddr : \mathbb{N} \\
\hline
\langle\langle \textit{disabled rule dNullAddr} \rangle\rangle \\
nulladdr = 0 \\
\langle\langle \textit{disabled rule dMaxAddr} \rangle\rangle \\
nulladdr < maxaddr
\end{array}
$$

$$ADDR == nulladdr \mathrel{..} maxaddr$$

$$TIME == \mathbb{N}$$

System error flags

$$SYSERR ::= sysok \mid pdinuse \mid unusedpd \mid ptabfull \mid$$
$$schedqfull \mid schedqempty \mid alreadyasleep \mid$$
$$toomanysleepers \mid notallocsema \mid nofreesemas \mid$$
$$procalreadyhasmsg \mid destinationnotrcving \mid$$
$$badmsgdestination \mid nomsg$$

$$SysOk \mathrel{\widehat{=}} [\, serr! : SYSERR \mid serr! = sysok \,]$$
$$PDInUse \mathrel{\widehat{=}} [\, serr! : SYSERR \mid serr! = pdinuse \,]$$
$$PTABFull \mathrel{\widehat{=}} [\, serr! : SYSERR \mid serr! = ptabfull \,]$$

The process table

PTAB

$$used, free : \mathbb{P}\, PID;\ \ prio : PID \rightarrow\!\!\!\!\!+\ PRIO$$

$$state : PID \rightarrow\!\!\!\!\!+\ PSTATE$$

$$stacktop : PID \rightarrow\!\!\!\!\!+\ ADDR$$

$$smsg : PID \rightarrow\!\!\!\!\!+\ MSG$$

$$wakingtime : PID \rightarrow\!\!\!\!\!+\ TIME$$

$$used \in \mathbb{F}\, PID \land free = PID \setminus used$$

$$used = \operatorname{dom} prio = \operatorname{dom} state = \operatorname{dom} smsg$$

$$= \operatorname{dom} wakingtime = \operatorname{dom} stacktop$$

Process table initialisation

PTABInit

$$PTAB'$$

$$used' = \emptyset$$

Process table operation classification

$$UsedPID \mathrel{\widehat{=}} [\, \Xi PTAB;\ p? : PID \mid p? \in used \,]$$

$$GotFreePIDs \mathrel{\widehat{=}} [\, \Xi PTAB \mid used \subset PID \,]$$

AllocPID

$$\Delta PTAB;\ p! : PID$$

$$p! \notin used \land used' = used \cup \{p!\} \,]$$

Setting the process priority

SetProcPrio

$\Delta PTAB$

$p? : PID$

$pr? : PRIO$

$$prio' = prio \oplus \{(p? \mapsto pr?)\}$$

Updating the process state

SetProcState

$$\Delta PTAB$$
$$p? : PID$$
$$st? : PSTATE$$

$$state' = state \oplus \{(p? \mapsto st?)\}$$

Setting the process waiting time

SetProcWaitingTime

$\Delta PTAB$

$p? : PID$

$t? : TIME$

$$wakingtime' = wakingtime \oplus \{(p? \mapsto t?)\}$$

Adding basic process information

AddPDESC

$SetProcPrio$

$SetProcState$

$SetProcWaitingTime[\, t? := 0 \,]$

$$smsg' = smsg \oplus \{(p? \mapsto nullmsg)\}$$

Adding the process description

$$
\begin{aligned}
& AddPD \\
& \quad \widehat{=} \;\; ((GotFreePIDs \land AllocPID) \fatsemi \\
& \qquad\qquad (\lnot\; UsedPID[\, p!/p? \,] \\
& \qquad\qquad\;\; \land\; AddPDESC[\, p!/p? \,] \\
& \qquad\qquad\;\; \land\; SysOk) \\
& \qquad\quad \lor\; PDInUse) \\
& \qquad \lor\; PTABFull
\end{aligned}
$$

Proof statistics

Proofs	
lemmas	42
frules	8
grules	12
precondition	75
theorems	63
Total	**200**

Commands		
trivial	3,704	61%
intermed.	1,578	26%
complex	782	13%
Total	**6,064**	

Location

- ✓ A verified simple OS kernel
- ✓ Organisation
- ✓ Specification
- ✓ Proof statistics
- $\rightsquigarrow$ Conclusions

Conclusions

- **OS kernels: GC pilot project**
- *free RTOS, Microsoft hypervisor, POSIX-compliant flash file store, Mondex*
- *substantial body of modelling in Z*
  - *specs, refinements, implementations*
- *larger than Mondex, but much commonality*
- *objectives: mechanised domain models & refinement proofs*
- **emphasis on tool chain**

Index

Unifying Theories of Undefinedness

Jim Woodcock
Marktoberdorf
8 August 2008

Summary

- The problem
- Semantical systems
- Semantics
- Some particular semantical systems
- Guards
- Example: VDM & Z
- Conclusions

The problem

- Alloy, ASM, B, Perfect, Raise, VDM, Z, ...
- common first-order logic, but differences in treatment of “undefined” values
  - Alloy: relational image, scalars as singleton sets
  - Raise: left-to-right evaluation (McCarthy)
  - VDM: 3-valued logic (Kleene)
  - Z: semi-classical logic
- **how are these related?**

The talk

1. unify undefinedness in strict, McCarthy, Kleene, semi-classical, and classical logics
2. show how to use classical or semi-classical logic to prove facts in a monotonic partial logic with guards
3. show how to refute conjectures with tight guards
4. show how to use classical logic to prove facts of a semi-classical system

Collaboration

- ***Saaltink***, *Freitas, and Harwood*
- *Dealing with undefined in classical logic (1997)*
- *Logic and description in Z (1992)*
- *Weak axioms for definite description (1992)*
- *Cliff Jones & John Fitzgerald*

Basic sets and constructors

- booleans: $\mathbf{B} \;\widehat{=}\; \{\, \mathbf{t}, \mathbf{f} \,\}$, universe of values: $\mathbf{U}$
- undefined value: $\bot$ $\quad (\bot \notin \mathbf{B}, \bot \notin \mathbf{U})$
- lifting: $X^{\bot} \;\widehat{=}\; X \cup \{\bot\}$ $\quad (\bot \notin X)$
- repetition: $X^k \;\widehat{=}\; \overbrace{X \times \cdots \times X}^{k\ times}$

  $X^* \;\widehat{=}\; \bigcup \{\, n : \mathbb{N} \bullet X^n \,\}$
- total functions: $X \to Y$
- partial functions: $X \nrightarrow Y$

Ordering

- b *is at least as defined as* a

$$a \sqsubseteq b \;\widehat{=}\; a = b \vee a = \bot$$

- *pointwise extension to tuples*

$$(x_1, \ldots, x_k) \sqsubseteq (x'_1, \ldots, x'_k) \;\widehat{=}\; x_1 \sqsubseteq x'_1 \wedge \cdots \wedge x_k \sqsubseteq x'_k$$

- *pointwise extension to functions*

$$f \sqsubseteq f' \;\widehat{=}\; \operatorname{dom} f = \operatorname{dom} f' \wedge \forall\, x : \operatorname{dom} f \bullet f(x) \sqsubseteq f'(x)$$

Properties of functions

suppose that $f : X \to Y,\ g : X^k \to Y$

$$\textbf{\textit{monotonic}}(f) \mathrel{\widehat{=}} \forall\, x, x' : X \bullet x \sqsubseteq x' \Rightarrow f(x) \sqsubseteq f(x')$$

$$\textbf{\textit{strict}}(g) \mathrel{\widehat{=}}$$

$$\forall\, x_1, \ldots, x_k : X \bullet$$

$$x_1 = \bot \vee \cdots \vee x_k = \bot \Rightarrow g(x_1, \ldots, x_k) = \bot$$

$$\textbf{\textit{definite}}(g) \mathrel{\widehat{=}}$$

$$\forall\, x_1, \ldots, x_k : X \bullet$$

$$g(x_1, \ldots, x_k) = \bot \Rightarrow x_1 = \bot \vee \cdots \vee x_k = \bot$$

$$\textbf{\textit{strict}}(g) \Rightarrow \textbf{\textit{monotonic}}(g)$$

A basic first-order language

- *arity:* $\alpha : (Fun \cup Pred) \to \mathbb{N}$
- *syntax for terms and formulae*

$$v : Var$$

$$f : Fun$$

$$p : Pred$$

$$t : Term ::= v \mid f(t_1, \ldots, t_k) \mid \iota\, x \bullet \Phi$$

$$\Phi : Form ::= true \mid t_1 = t_2 \mid p(t_1, \ldots, t_j) \mid$$

$$\neg\, \Phi \mid \Phi \vee \Phi \mid \forall\, x \bullet \Phi$$

where $k = \alpha(f),\ j = \alpha(p)$

Semantical systems

$$\Sigma \;\widehat{=}\; (\mathcal{F}_\Sigma, \mathcal{P}_\Sigma, =_\Sigma, \neg_\Sigma, \vee_\Sigma, \iota_\Sigma, \forall_\Sigma)$$

$\mathcal{F}_\Sigma \subseteq (\boldsymbol{U}^\perp)^* \nrightarrow \boldsymbol{U}^\perp$ *admissible dens of funcs*

$\mathcal{P}_\Sigma \subseteq (\boldsymbol{U}^\perp)^* \nrightarrow \boldsymbol{B}^\perp$ *admissible dens of preds*

$=_\Sigma \in \boldsymbol{U}^\perp \times \boldsymbol{U}^\perp \to \boldsymbol{B}^\perp$ *meaning of equality*

$\neg_\Sigma \in \boldsymbol{B}^\perp \to \boldsymbol{B}^\perp$ *meaning of negation*

$\vee_\Sigma \in \boldsymbol{B}^\perp \times \boldsymbol{B}^\perp \to \boldsymbol{B}^\perp$ *meaning of disjunction*

Semantical systems

$\iota_{\Sigma} \in (\boldsymbol{U} \nrightarrow \boldsymbol{B}^{\perp}) \to \boldsymbol{U}^{\perp}$ *meaning of def. desc.*

$\forall_{\Sigma} \in (\boldsymbol{U} \nrightarrow \boldsymbol{B}^{\perp}) \to \boldsymbol{B}^{\perp}$ *meaning of univ. quant.*

provided

$$\forall\, f : \boldsymbol{U} \nrightarrow \boldsymbol{B}^{\perp} \bullet f \neq \emptyset \Rightarrow \iota_{\Sigma}(f) \in (\mathrm{dom}\, f)^{\perp}$$

Strict and definite semantical systems

$$\mathbf{strict}(\Sigma) \mathrel{\widehat{=}} (\mathcal{F}'_\Sigma, \mathcal{P}'_\Sigma, =_\Sigma, \neg_\Sigma, \vee_\Sigma, \iota_\Sigma, \forall_\Sigma)$$

where

$$\mathcal{F}'_\Sigma \mathrel{\widehat{=}} \{ f : \mathcal{F}_\Sigma \mid \mathbf{strict}(f) \}$$

$$\mathcal{P}'_\Sigma \mathrel{\widehat{=}} \{ p : \mathcal{P}_\Sigma \mid \mathbf{strict}(p) \}$$

similarly for $\mathbf{definite}(\Sigma)$

$\mathbf{monotonic}(\Sigma)$:

- $=_\Sigma, \neg_\Sigma, \vee_\Sigma, \iota_\Sigma$, and $\forall_\Sigma$ are monotonic
- every member of $\mathcal{F}_\Sigma$ and $\mathcal{P}_\Sigma$ is monotonic

Comparing semantical systems

- *suppose* $A, B : \mathbb{P}\, X$
- *Hoare preorder:*

$$A \sqsubseteq_H B \;\; \widehat{=} \;\; \forall\, a : A \bullet \exists\, b : B \bullet a \sqsubseteq b$$

$$\Sigma \sqsubseteq_H \Sigma' \;\; \widehat{=} \;\; \mathcal{F}_\Sigma \sqsubseteq_H \mathcal{F}_{\Sigma'} \wedge \cdots \wedge \forall_\Sigma \sqsubseteq_H \forall_{\Sigma'}$$

Semantics

M is a model over a semantical system Σ

$$M \;\widehat{=}\; (D_M, P_M, F_M)$$

- $D_M \subseteq \mathbf{U}$ is the domain ($\perp \notin D_M$)
- for every $p \in Pred$,
  $P_M(p) \in ((D_M^{\perp})^{\alpha(p)} \to \mathbf{B}^{\perp}) \cap \mathcal{P}_\Sigma$
- for every $f \in Fun$,
  $F_M(f) \in ((D_M^{\perp})^{\alpha(f)} \to D_M^{\perp}) \cap \mathcal{F}_\Sigma$

$$M \sqsubseteq M' \;\widehat{=}\; D_M = D'_M \wedge F_M \sqsubseteq F'_M \wedge P_M \sqsubseteq P'_M$$

Lemma

suppose that $\Sigma \sqsubseteq_H \Sigma'$ *and* M *is a model over* Σ*, then there's a model* M' *over* Σ' *such that* $M \sqsubseteq M'$

Semantics I

$$a[d/v](v') = \begin{cases} d & \textit{if } v' = v \\ a(v') & \textit{otherwise} \end{cases}$$

$$[\![x]\!](\Sigma, M, a) = a(x)$$

$$[\![f(t_1, \ldots, t_k)]\!](\Sigma, M, a) =$$

$$F_M(f)([\![t_1]\!](\Sigma, M, a), \ldots, [\![t_k]\!](\Sigma, M, a))$$

$$[\![\iota\, x \bullet \Phi]\!](\Sigma, M, a) = \iota_\Sigma(h(x, \Phi))$$

$$h(x, \Phi) = \lambda\, d : D_M \bullet [\![\Phi]\!](\Sigma, M, a[d/x])$$

$$[\![\mathit{true}]\!](\Sigma, M, a) = \boldsymbol{t}$$

Semantics II

$$[\![t_1 = t_2]\!](\Sigma, M, a) =$$

$$=_\Sigma ([\![t_1]\!](\Sigma, M, a), [\![t_1]\!](\Sigma, M, a))$$

$$[\![p(t_1, \ldots, t_k)]\!](\Sigma, M, a) =$$

$$P_M(p)([\![t_1]\!](\Sigma, M, a), \ldots, [\![t_k]\!](\Sigma, M, a))$$

$$[\![\neg \Phi]\!](\Sigma, M, a) = \neg_\Sigma([\![\Phi]\!](\Sigma, M, a))$$

$$[\![\Phi \vee \Phi']\!](\Sigma, M, a) = \vee_\Sigma ([\![\Phi]\!](\Sigma, M, a), [\![\Phi']\!](\Sigma, M, a))$$

$$[\![\forall x \bullet \Phi]\!](\Sigma, M, a) = \forall_\Sigma(h(x, \Phi))$$

Lemma

if Σ *is a semantical system,* M *is a model over* Σ*,* a *is an assignment over* M*, and* t *is a term, then* $[\![t]\!](\Sigma, M, a) \in D_M^{\perp}$

Truth

$$\Sigma \models \Phi \quad \widehat{=} \quad [\![\Phi]\!](\Sigma, M, a) = \boldsymbol{t}$$

for every model M *over* Σ

and assignment a *over* M

Theorem

suppose c *is a construct,* Σ *and* Σ' *are semantical systems,* M *is a model over* Σ*,* M' *is a model over* Σ'*,* a *is an assignment over* M*,* a' *is an assignment over* M'*,* $\Sigma \sqsubseteq \Sigma'$*,* $M \sqsubseteq M'$*,* $a \sqsubseteq a'$*, and that either* Σ *or* Σ' *is monotonic*

then

$$[\![c]\!](\Sigma, M, a) \sqsubseteq [\![c]\!](\Sigma', M', a')$$

Some particular semantical systems

- strict system Σ_s
- Kleene system Σ_k
- left-to-right system Σ_{lr}
- semi-classical systems
- classical systems

Strict system Σ_s I

$$\mathcal{F}_s = \{ f : (\boldsymbol{U}^{\perp})^* \nrightarrow \boldsymbol{U}^{\perp} \mid \textbf{\textit{strict}}(f) \}$$

$$\mathcal{P}_s = \{ p : (\boldsymbol{U}^{\perp})^* \nrightarrow \boldsymbol{B}^{\perp} \mid \textbf{\textit{strict}}(p) \}$$

$$=_s (x, y) = \begin{cases} \perp & \textit{if } x = \perp \textit{ or } y = \perp \\ \mathbf{t} & \textit{if } x = y \neq \perp \\ \mathbf{f} & \textit{otherwise} \end{cases}$$

$$\iota_s(f) = \begin{cases} x & \textit{if } \perp \notin \operatorname{ran} f \\ & \textit{and } \operatorname{dom}(f \rhd \{\mathbf{t}\}) = \{x\} \\ \perp & \textit{otherwise} \end{cases}$$

Strict system Σ_s II

$$\forall_s(f) = \begin{cases} \bot & \textit{if } \bot \in \operatorname{ran} f \\ \boldsymbol{t} & \textit{if } \operatorname{ran} f = \{\boldsymbol{t}\} \\ \boldsymbol{f} & \textit{otherwise} \end{cases}$$

$\neg_s$	
$\boldsymbol{t}$	$\boldsymbol{f}$
$\bot$	$\bot$
$\boldsymbol{f}$	$\boldsymbol{t}$

$\vee_s$	$\boldsymbol{t}$	$\bot$	$\boldsymbol{f}$
$\boldsymbol{t}$	$\boldsymbol{t}$	$\bot$	$\boldsymbol{t}$
$\bot$	$\bot$	$\bot$	$\bot$
$\boldsymbol{f}$	$\boldsymbol{t}$	$\bot$	$\boldsymbol{f}$

undefined very contagious

Kleene system Σ_k I

$$\mathcal{F}_k = \{ f : (\boldsymbol{U}^\perp)^* \nrightarrow \boldsymbol{U}^\perp \mid \boldsymbol{monotonic}(f) \}$$

$$\mathcal{P}_k = \{ p : (\boldsymbol{U}^\perp)^* \nrightarrow \boldsymbol{B}^\perp \mid \boldsymbol{monotonic}(p) \}$$

$$=_k \; = \; =_s$$

$$\iota_k = \iota_s$$

Kleene system Σ_k II

$$\forall_k(f) = \begin{cases} \boldsymbol{f} & \textit{if } \boldsymbol{f} \in \operatorname{ran} f \\ \boldsymbol{t} & \textit{if } \operatorname{ran} f = \{\boldsymbol{t}\} \\ \bot & \textit{otherwise} \end{cases}$$

$$\neg_k = \neg_s$$

$\vee_k$	$\boldsymbol{t}$	$\bot$	$\boldsymbol{f}$
$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$
$\bot$	$\boldsymbol{t}$	$\bot$	$\bot$
$\boldsymbol{f}$	$\boldsymbol{t}$	$\bot$	$\boldsymbol{f}$

Left-to-right system Σ_{lr}

$\mathcal{F}_{lr} = \mathcal{F}_k$

$\mathcal{P}_{lr} = \mathcal{P}_k$

$=_{lr} \;=\; =_s$

$\iota_{lr} = \iota_s$

$\forall_{lr} = \forall_s$

$\neg_{lr} = \neg_s$

$\vee_{lr}$	$\boldsymbol{t}$	$\bot$	$\boldsymbol{f}$
$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$
$\bot$	$\bot$	$\bot$	$\bot$
$\boldsymbol{f}$	$\boldsymbol{t}$	$\bot$	$\boldsymbol{f}$

Lemmas

1. Σ_s, Σ_k, and Σ_{lr} are all monotonic

2. $\Sigma_s \sqsubseteq \Sigma_{lr} \sqsubseteq \Sigma_k$

Semi-classical systems

Σ is semi-classical providing

1. $\Sigma_s \sqsubseteq \Sigma$
2. *every* $p \in \mathcal{P}_\Sigma$ *yields only defined results:*

   $\bot \notin \mathrm{ran}\, p$
3. *equality yields only defined results:*

   $\bot \notin \mathrm{ran} =_\Sigma$

terms can fail to denote

but predicates are classical

Theorem

suppose Σ *is a semantical system with* $\Sigma_s \sqsubseteq \Sigma$

there is a semi-classical system Σ' *with* $\Sigma \sqsubseteq_H \Sigma'$

Classical systems

Σ *is classical providing*

1. $\Sigma_s \sqsubseteq \Sigma$
2. *every* $f \in \mathcal{F}_\Sigma$ *is definite*
3. *every* $p \in \mathcal{P}_\Sigma$ *is definite*
4. *for all* $f \in \boldsymbol{U} \nrightarrow \boldsymbol{B}^\perp$, *we have* $\iota_\Sigma(f) \neq \perp$

Theorem

suppose Σ *is a semantical system with* $\Sigma_s \sqsubseteq \Sigma$

there is a classical system Σ' *with* $\Sigma \sqsubseteq_H \Sigma'$

Guards

suppose c is a construct

G is a guard for c in Σ *if, for every model M over Σ, and every assignment a*

1. $[\![G]\!](\Sigma, M, a) \neq \bot$
2. *if* $[\![G]\!](\Sigma, M, a) = \boldsymbol{t}$ *then* $[\![c]\!](\Sigma, M, a) \neq \bot$

G is tight *if*

- *whenever* $[\![G]\!](\Sigma, M, a) = \boldsymbol{f}$*, then* $[\![c]\!](\Sigma, M, a) = \bot$

Theorems

1. suppose $\Sigma \sqsubseteq_H \Sigma'$, either Σ or Σ' is monotonic, and G is a guard for Φ in Σ
   if $\Sigma' \models G$ and $\Sigma' \models \Phi$, then $\Sigma \models \Phi$
2. suppose $\Sigma \sqsubseteq_H \Sigma'$, either Σ or Σ' is monotonic, and G is a tight guard for Φ in Σ
   $\Sigma' \models G$ and $\Sigma' \models \Phi$ iff $\Sigma \models \Phi$

Guards for definite systems

- *tight guards for **definite**(Σ_s)*
- *guards for **definite**(Σ_k)*
- *guards for **definite**(Σ_{lr})*

Tight guards for **definite**(Σ_s)

$$G_s(x) = true \quad x \in Var$$

$$G_s(f(t_1, \ldots, t_k)) = G_s(t_1) \wedge \cdots \wedge G_s(t_k) \quad f \in Fun$$

$$G_s(\iota\, x \bullet \Phi) = (\forall\, x \bullet G_s(\Phi)) \wedge (\exists_1\, x \bullet \Phi)$$

$$G_s(t = t') = G_s(t) \wedge G_s(t')$$

$$G_s(p(t_1, \ldots, t_k)) = G_s(t_1) \wedge \cdots \wedge G_s(t_k) \quad p \in Pred$$

$$G_s(\neg\, \Phi) = G_s(\Phi)$$

$$G_s(\Phi \vee \Phi') = G_s(\Phi) \wedge G_s(\Phi')$$

$$G_s(\forall\, x \bullet \Phi) = \forall\, x \bullet G_s(\Phi)$$

Guards for *definite*(Σ_k)

$$G_k(x) = true \quad x \in Var$$

$$G_k(f(t_1, \ldots, t_k)) = G_k(t_1) \wedge \cdots \wedge G_k(t_k) \quad f \in Fun$$

$$G_k(\iota\, x \bullet \Phi) = (\forall\, x \bullet G_k(\Phi)) \wedge (\exists_1\, x \bullet \Phi)$$

$$G_k(t = t') = G_k(t) \wedge G_k(t')$$

$$G_k(\neg\, \Phi) = G_k(\Phi)$$

$$G_k(\Phi \vee \Phi') =$$

$$(G_k(\Phi) \wedge \Phi) \vee (G_k(\Phi') \wedge \Phi') \vee (G_k(\Phi) \wedge G_k(\Phi'))$$

$$G_k(\forall\, x \bullet \Phi) = (\forall\, x \bullet G_k(\Phi)) \vee (\exists\, x \bullet G_k(\Phi) \wedge \neg\, \Phi)$$

Guards for **definite**(Σ_{lr})

$$G_{lr}(x) = true x \in Var$$

$$G_{lr}(f(t_1, \ldots, t_k)) = G_{lr}(t_1) \wedge \cdots \wedge G_{lr}(t_k) f \in Fun$$

$$G_{lr}(\iota\, x \bullet \Phi) = (\forall\, x \bullet G_{lr}(\Phi)) \wedge (\exists_1\, x \bullet \Phi)$$

$$G_{lr}(t = t') = G_{lr}(t) \wedge G_{lr}(t')$$

$$G_{lr}(p(t_1, \ldots, t_k)) = G_{lr}(t_1) \wedge \cdots \wedge G_{lr}(t_k) p \in Pred$$

$$G_{lr}(\neg\, \Phi) = G_{lr}(\Phi)$$

$$G_{lr}(\Phi \vee \Phi') = G_{lr}(\Phi) \wedge (\Phi \vee G_{lr}(\Phi'))$$

$$G_{lr}(\forall\, x \bullet \Phi) = \forall\, x \bullet G_{lr}(\Phi)$$

Theorems

1. *if* c *is a construct, then* $G_s(c)$ *is a tight guard for* c *in* ***definite***(Σ_s)
2. *if* c *is a construct, then* $G_k(c)$ *is a guard for* c *in* ***definite***(Σ_k)*, and a tight guard for* c *in* ***strict***(***definite***(Σ_k))
3. *if* c *is a construct, then* $G_{lr}(c)$ *is a guard for* c *in* ***definite***(Σ_{lr})*, and a tight guard for* c *in* ***strict***(***definite***(Σ_{lr}))

Guards for indefinite systems

- $div.pre(x, y) \Leftrightarrow x \in \mathbb{Z} \land y \in \mathbb{Z} \land y \neq 0$
- π *is a precondition mapping*

1. $\pi \in (Fun \cup Pred) \nrightarrow Pred$
2. *for every* $g \in \operatorname{dom} \pi$ *we have*
   $\alpha(\pi(g)) = \alpha(g)$
3. $\operatorname{dom} \pi \cap \operatorname{ran} \pi = \emptyset$

M respects π

- provided
  1. for any $g \in (Fun \cup Pred) \setminus \mathrm{dom}\,\pi$, $M(g)$ is definite
  2. for any $g \in \mathrm{dom}\,\pi$, any $x_1, \ldots, x_k \in D_M$ (where $k = \alpha(g)$), if $M(\pi(g))(x_1, \ldots, x_k) = \mathbf{t}$, then $M(g)(x_1, \ldots, x_k) \neq \bot$
- $\Sigma \vDash_\pi \Phi$ if $[\![\Phi]\!](\Sigma, M, a) = \mathbf{t}$ for every M over Σ that respects π and every a over M

Guards

suppose c *is a construct*

G is a guard for c in Σ and π *if, for every model* M *over* Σ *that respects* π*, and every total assignment* a

1. $[\![G]\!](\Sigma, M, a) \neq \perp$

2. *if* $[\![G]\!](\Sigma, M, a) = \mathbf{t}$ *then* $[\![c]\!](\Sigma, M, a) \neq \perp$

Theorem

suppose π *is a precondition mapping,* $\Sigma \sqsubseteq_H \Sigma'$,
either Σ *or* Σ' *is monotonic, and* G *is a guard*
for Φ *in* Σ *and* π
if $\Sigma' \vDash_\pi G$ *and* $\Sigma' \vDash_\pi \Phi$, *then* $\Sigma \vDash_\pi \Phi$

Example: VDM & Z

- *VDM uses LPF, a variant of Kleene's logic, which is monotonic*
- $\Sigma_s \sqsubseteq \Sigma_k$
- *we can construct a classical system* Σ' *such that* $\Sigma_k \sqsubseteq \Sigma'$
- *suppose* G *is a guard for* Φ *in* Σ_k
- ***if*** $\Sigma' \models G$ ***and*** $\Sigma' \models \Phi$ ***then*** $\Sigma_k \models \Phi$

Proof for semi-classical systems

semi-classical systems aren't in general monotonic

Ordering on semantical systems

- Σ *necessitates* Σ'
- $\Sigma \preceq \Sigma'$ *iff for every formula* Φ, *if* $\Sigma \models \Phi$, *then* $\Sigma' \models \Phi$
- $\preceq$ *is a preorder*

Theorems

1. *suppose* $\Sigma \sqsubseteq \Sigma'$*, and* Σ *is monotonic, then* $\Sigma \preceq \Sigma'$
2. *suppose* Σ *is a semi-classical, then* $\Sigma_k \preceq \Sigma$
3. *suppose* Σ *is a semi-classical, then* $\Sigma_{lr} \preceq \Sigma$

- *use classical logic to prove facts about Kleene logic (VDM)*
- *these are also facts of the semi-classical system (Z)*

Conclusions

- ***unified treatment of undefinedness***
- *how to use classical logic to prove facts in a monotonic partial logic with guards*
- *exhibited guards for several systems*
- *shown how classical logic can be used soundly to prove semi-classical facts*
- *Z/Eves does this with guards from the left-to-right system*
- ***basis for cross-verification in VSR***

Practical application

- *flash file store requires search trees*
- *VDM specification and refinement:* B^{+}*–trees*
- *syntactic translation into Z*
- *records, free types, sets, lists, mappings*
- *types and separated preconditions*
- *proofs in Z/Eves*
- *left-to-right guards guarantee:*
  - ***every theorem valid in VDM and in Z***

Thanks!

Index