

25MC

New Challenges in Model Checking

Gerard J. Holzmann

(Joint work with Rajeev Joshi and Alex Groce)



NASA/JPL Laboratory
for Reliable Software

Jet Propulsion Laboratory
California Institute of Technology



Some challenges in software verification

- 1975-1985 – Specification
 - What is the best specification formalism for verification?
 - Process algebras, analytical models, FSMs, Petri Nets, ...

Premordial version of Promela from 1980 (the input language for pan)

```

PROCESS (Ph0)
  IF F1?fork -> F0?fork; F1!fork; F0!fork
  :: F0?fork -> F1?fork; F0!fork; F1!fork
  FI
END

PROCESS (Ph1)
  IF F0?fork -> F1?fork; F0!fork; F1!fork
  :: F1?fork -> F0?fork; F1!fork; F0!fork
  FI
END

PROCESS (F0)
  IF true -> Ph0!fork; Ph0?fork; Ph1!fork; Ph1?fork
  :: true -> Ph1!fork; Ph1?fork; Ph0!fork
  FI
END

PROCESS (F1)
  IF true -> Ph1!fork; Ph1?fork; Ph0!fork
  :: true -> Ph0!fork; Ph0?fork; Ph1!fork
  FI
END

```

dining philosophers

```

$ prc Ph0 Ph1 F0 F1 # compile model
$ pan Ph0 Ph1 F0 F1 # run the verifier
May 16 23:46 1981 Page 1

```

```

MESSAGE      F0      F1      Ph0      Ph1
fork
fork          ----->
fork!         ----->>
fork!         ----->>>
fork?                +
fork?                +
=====

```

Specification
originally interpreted as sugared
algebraic expressions – soon replaced
with a direct encoding as automata

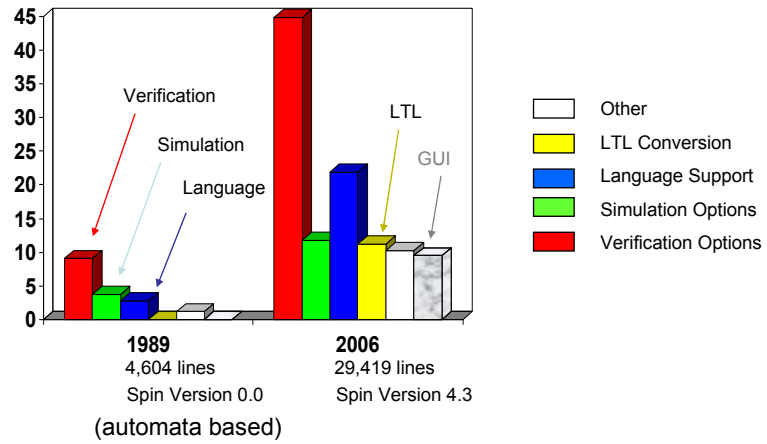
Verification

3

Some challenges in software verification

- 1975-1985 – Specification
 - What is the best specification formalism for verification?
- 1985-1995 – Algorithms
 - How do we *optimize* the verification steps?
 - BDDs, Partial Orders, Bitstate hashing, etc.

Algorithms... efficiency... optimization...



25MC GJH

5

Some challenges in software verification

- 1975-1985 – Specification
 - What is the best specification formalism for verification?
- 1985-1995 – Algorithms
 - How do we *optimize* the verification steps?
- 1995-2005 – Model Extraction
 - How can we extract models directly from code?
- 2005-2015

25MC GJH

6

Modex / FeaVer

<http://cm.bell-labs.com/cm/cs/what/modex/>

```
int
main(void)
{ int lower, upper, step;
  float fahr, celsius;

  lower = 0;
  upper = 300;
  step = 20;

  fahr = lower;
  while (fahr <= upper) {
    celsius = (5.0/9.0) * (
      printf("%4.0f %6.1f\n",
        fahr = fahr + step;
    )
  }
```

mechanical
conversion
from ANSI-C
into a Spin 4.0 Model
(abstractions optional)

```
c_state "float fahr" "Local main"
c_state "float celsius" "Local main"

active proctype main()
{ int lower;
  int upper;
  int step;

  c_code { Pmain->lower=0; };
  c_code { Pmain->upper=300; };
  c_code { Pmain->step=20; };

  c_code { Pmain->fahr=Pmain->lower; };
  do
  :: c_expr { (Pmain->fahr <= Pmain->upper) };
  c_code { Pmain->celsius =
    ((5.0/9.0)*(Pmain->fahr-32.0)); };
  c_code { Printf("%4.0f %6.1f\n",
    Pmain->fahr, Pmain->celsius); };
  c_code { Pmain->fahr = (Pmain->fahr+Pmain->step); };
  :: else -> break
  od
}
```

25MC GJH

7

Some challenges in software verification

- 1975-1985 – Specification
 - What is the best specification formalism for verification?
- 1985-1995 – Algorithms
 - How do we *optimize* the verification steps?
- 1995-2005 – Model Extraction
 - How can we extract models directly from code?
- 2005-2015? – Software Construction
 - How can we structure software to facilitate more powerful types of direct model checking?

25MC GJH

8

Our challenge project:

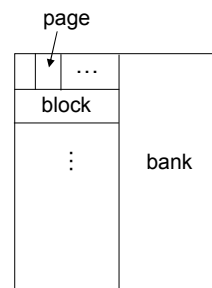
- design, build, and verify a *reliable* file system module for non-volatile memory (flash memory) for use on NASA spacecraft
 - harsh environment: radiation, bit-flips, hardware failure, sudden loss of power, sudden resets/reboots
 - strong requirements: never lose data, never corrupt data, preserve transaction atomicity, low overhead, good performance
 - NASA/JPL missions have had enough trouble with flash memory software that an alternative looks attractive
 - e.g., the MER Sol-18 anomaly (costing several days to remedy), and trouble with flash memory on the Deep Impact mission (two different flash file system packages)

25MC GJH

9

Some relevant features of NAND flash

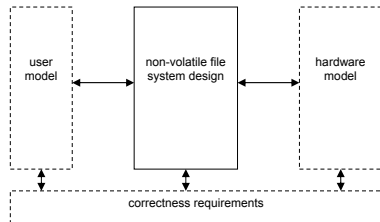
- Can only write a page *once*
 - writing flips selected bits from 1 to 0
 - If you write a new version of a page, the old version may still be around
 - No data can be stored in a fixed location
- Can only erase pages in *multiples of 32*
 - erase sets all bits in page to 1
- Blocks of 32 pages can become un-writable (*Bad*) at any time
- Can only *erase a block a limited number of times* before the block goes bad
 - Need wear-leveling algorithms
- Page read/write/erase cycles are slow
 - 2000 μ sec for a block erase
 - \sim 200 μ sec for a full page read or write
 - 10 μ sec to read a page header
- Any operation can fail:
 - block erases can fail
 - page writes can fail
 - page reads can fail with uncorrectable EDAC errors



25MC GJH

10

The challenge is to design and build the module *in such a way* that it is strongly verifiable



Context for Module Verification

```

active proctype flash_disk()
{
  byte b, p;
  bool v;
  do
  :: flash?readpage(b,p,_) ->
  ...
  :: flash?writepage(b,p,v) ->
  assert(b < NBL);
  assert(p < PPB);
  assert(blocks[b].meta[p] == free);
  if
  :: blocks[b].bad -> user!error
  :: else ->
  if
  :: blocks[b].meta[p] = v;
  user!success
  :: blocks[b].bad = true ->
  user!error
  fi
  fi
  :: flash?eraseblock(b,_,_) ->
  ...
  od
}

```

25MC GJH

11

Our attempt (so far)

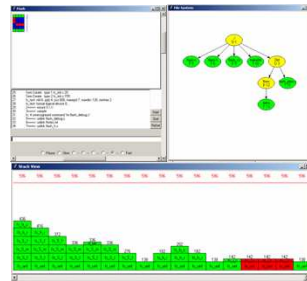
About 6,300 lines of ANSI-C code written to conform to a small set of self-imposed coding rules that should make a lot of things easier:

simulation
visualization

static analysis
strong testing
derivation of strict stack
and memory bounds

logic model checking (Spin)
theorem proving (ACL2)

The scary part: this code may actually fly on a real space mission



25MC GJH

12



NASA/JPL Laboratory
for Reliable Software

Gerard Holzmann
Rajeev Joshi
Alex Groce
Klaus Havelund