Centrum voor Wiskunde en Informatica

DivSPIN

A SPIN-compatible Distributed Model Checker

M. Leucker   Michael Weber   V. Foreijt   J. Barnat

ParaDiSe Parallel & Distributed Systems Laboratory

SENVA Meeting
November 2005
Grenoble

DiVinE & SPIN

SPIN

Reverse-Engineered SPIN

Reverse-Engineered SPIN: NIPS

Reverse-Engineered SPIN: NIPS

Never Implement Promela Semantics (again)

Verification Scenario

State space too large for exhaustive sequential verification

- Distributed Model Checking to the rescue!

Obstacles:

- Possibility to reuse already developed model?
- Available hardware? (Cluster, GRID, . . .)
  Maintenance?
- Complex software dependencies:
  MPI, Java runtime, libraries, compiler(?), . . .
  in non-conflicting versions!

Outline

DivSPIN Goals

In a nutshell:

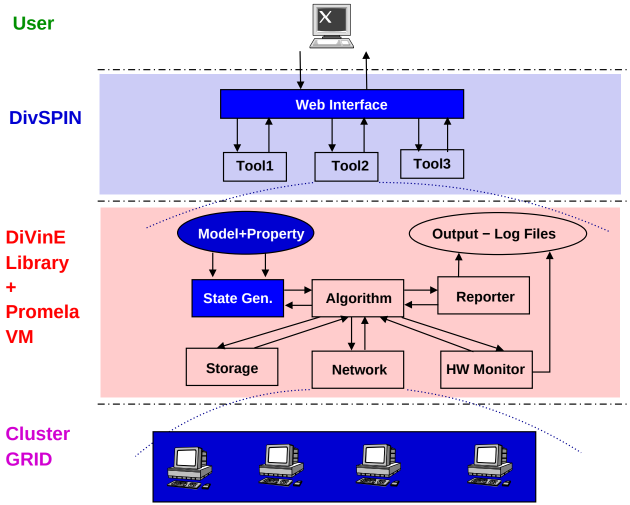
> "Provide distributed model checking tool with extremely low access barrier which is attractive to large user base of sequential model checking tools."

Effortless trial use

- Always up-to-date
- No installation
- No software dependency hell
- No expensive hardware to buy and maintain

ParaDiSe Parallel & Distributed Systems Laboratory

DivSPIN Overview



Cluster

Parsecs Cluster at RWTH

26 nodes, each:

- 2x Intel Pentium 500MHz
- 512MB RAM
- 100MBit switched Ethernet

* Dramatization

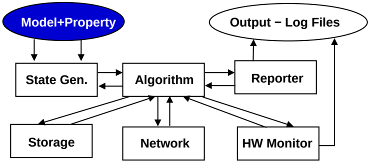
ParaDiSe Parallel & Distributed Systems Laboratory

TUM

CWI

Modelling Language: Promela

```
chan ch = [1] of {int};

proctype P (x) {
  do
  :: ch!x;
  od
}
proctype Q (x) {
  do
  :: ch?x;
  :: break;
  od
}
```

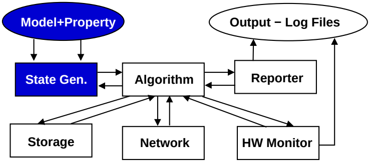


- Non-determinism
- Concurrency
- Synchronisation
- Priorities
- Extensions possible (clocks, heap, . . .)

Only basic data types (integers, records)

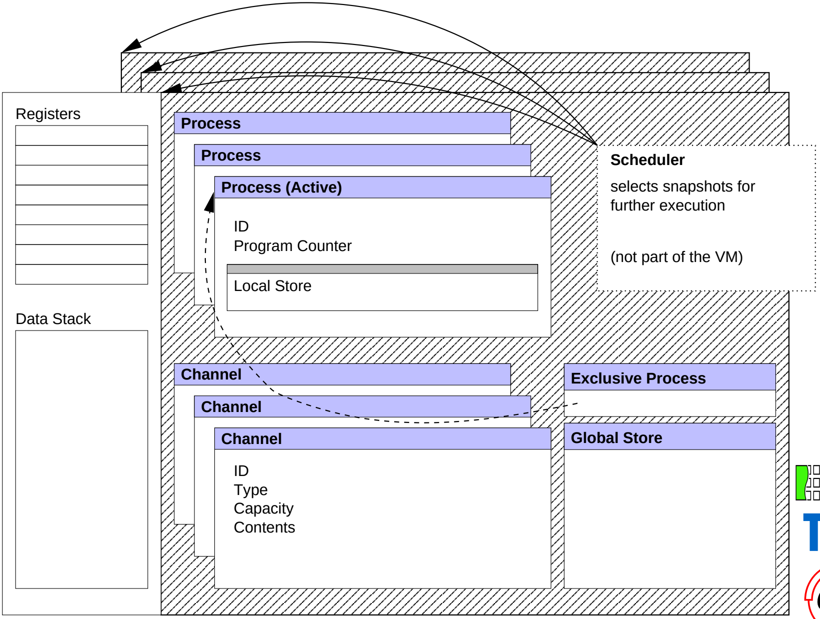
NIPS VM Design

- SPIN not very accessible (to distributed settings, anyway)
- Complete formal specification from scratch [WeberSchürmans05]



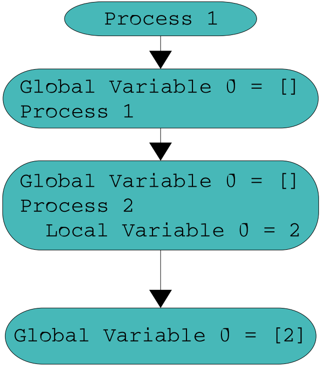
- Generally: least effort for compiler & VM implementor
- RISC-like instruction set
- Observable & unobservable steps in the language
- States: binary objects, directly manipulated by VM
- Performance: comparable to SPIN
- Well suited for embedding (unlike SPIN . . .)

NIPS VM Design



NIPS Example

```
chan c = [1] of {int}

proctype send (int x) {
  c!x
}
init { run send(2) }
```

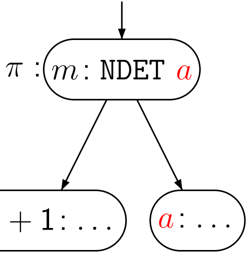



```
         GLOBSZ   2
         LDC      -31
         CHNEW    1        1
         STVA     G        2u       0
         STEP     N        0
         LJMP     P_init

P_send:  LDVA     G        2u       0
         TOP      r0
         CHFREE
         NEXZ
         PUSH     r0
         CHADD
         PUSH     r0
         LDVA     L        4        0
         CHSETO   0
         STEP     T        0

P_init:  LDC      -31
         LDC      2
         LRUN     4        1        P_send
         NEXZ
         STEP     T        0
```

Interesting Instructions

- NDET a

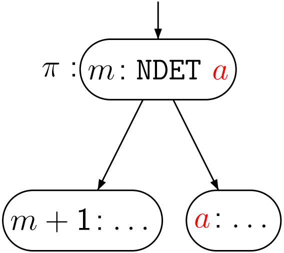
Explicit notion of non-determinism



Interesting Instructions

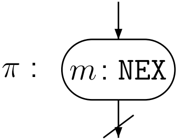
- NDET a

Explicit notion of non-determinism



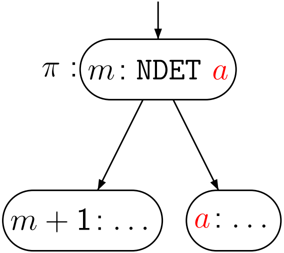
- NEX

Aborts process execution



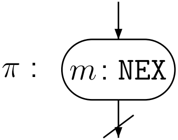
Interesting Instructions

- NDET a



Explicit notion of non-determinism

- NEX



Aborts process execution

- STEP M'

Finish multiple internal steps

$$\Gamma[M] \xrightarrow{\pi}_{step} \Gamma'[M']$$

NIPS VM Implementation

- $\approx$ 5kLOC of C code (including everything)
- State size: order of tens of bytes,
  within few bytes of SPIN's states
- No fancy optimizations yet
- Performance: comparable to SPIN
- Well suited for embedding (unlike SPIN ...)

Largest example so far: LUNAR Routing Protocol
10M states generated, $\approx$ 6 GB memory
Memory filled in less than two minutes
(AMD Athlon 64 3500+)

Benchmarking against SPIN

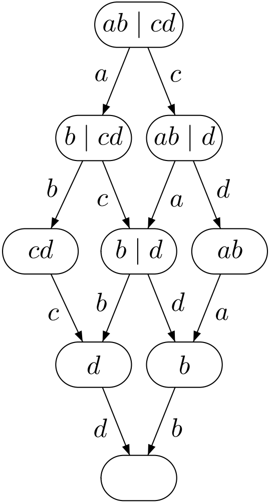
- Breadth-first state space generation (no POR)
- SPIN: generating C, compiled
- VM: compiling to bytecode, interpreted
- AMD Athlon64 3500+ (sequential)

	era(34)	leader(6,12)	ptrsn_N(3)	pftp
Total states				
SPIN	713,817	382,151	45,927	1,275,180
SPIN+Opt	342,082	106,449	25,371	219,167
NIPS	347,012	382,465	51,118	1,378,184
NIPS+PR	2,914	6,241	853	301,603
State-space generation time (seconds)				
Spin+Opt	26.441	0.12	0.085	0.770
NIPS+PR	0.177	0.176	0.012	4.996

ParaDiSe Parallel & Distributed Systems Laboratory

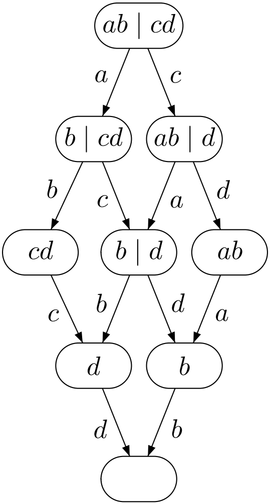
Path Reduction vs. POR

Full Transition
System

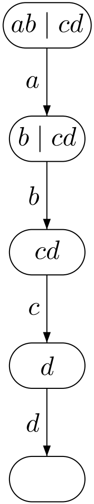


Path Reduction vs. POR

Full Transition System

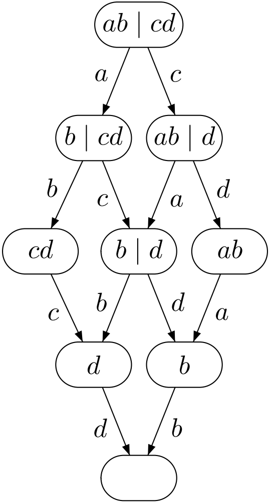


Partial Order Reduction

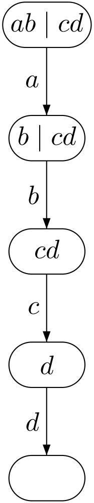


Path Reduction vs. POR

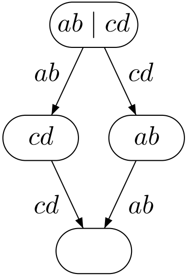
Full Transition System



Partial Order Reduction



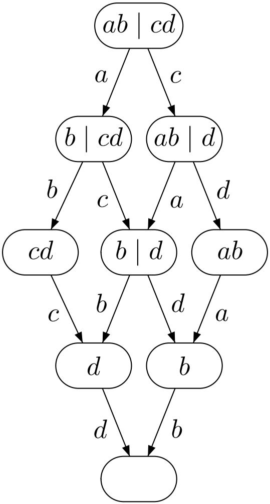
Path Reduction



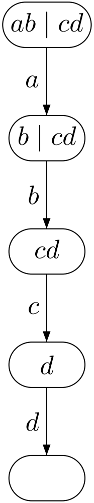
- Static
- Preserves branching structure

Path Reduction vs. POR

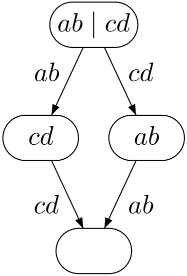
Full Transition System



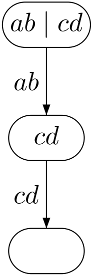
Partial Order Reduction



Path Reduction



PR+POR



- Static
- Preserves branching structure

Path Reduction

(path compression, [Yorav&Grumberg 2004]):
Collapse sequences of steps that cannot influence the value of a temporal logic specification

- Based on static program analysis.
- Identifies program locations which *may affect* the value of the specification.
- Preserves $\mathsf{CTL}^*_{-\mathbf{X}}$.
- Presented as applied to a simple high-level concurrent language (static processes, synchronous communication).

Assumption:
Specification is a formula over program variables.

Path Reduction for NIPS

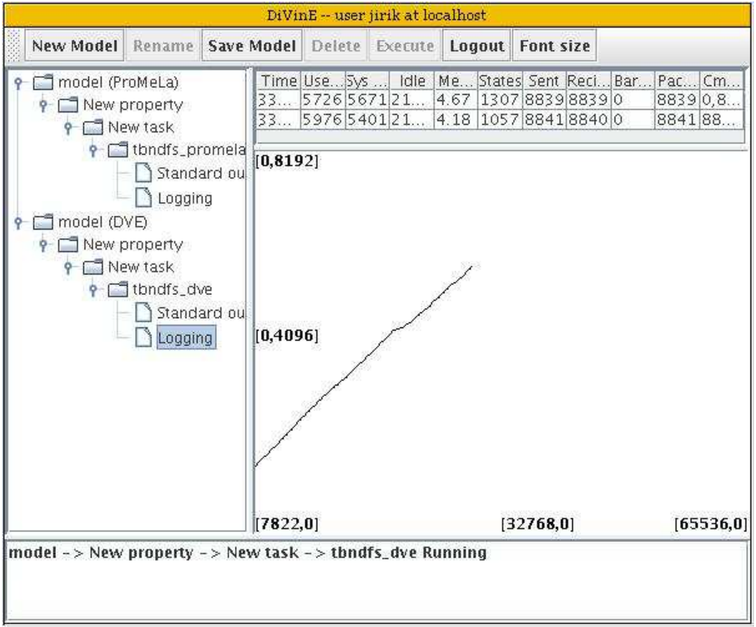
Versus [Y&G 2004]:

- Adapted to
  - Dynamic creation of channels and processes
  - Asynchronous communication
  - Global variables $\mapsto$ informal communication channels.
- Two phases:
  - *Transformation* of the NIPS program: preserves the model's semantics and just alters the transition system.
  - *Transition-system construction*: delegated to the (unmodified) NIPS virtual machine.

Less involved framework (almost too simple...)
Explicit notion of steps, program transformation
Hard work: delegate to VM

VM Todo List

- How to present counter examples?
  Currently: in terms of VM states
  Need clean reflective interface to VM
- How to integrate reduction techniques?
  (Partial Order Reduction, τ-confluence)


DiVinE -- user jirik at localhost
New Model
Rename
Save Model
Delete
Execute
Logout
Font size
model (ProMeLa)
New property
New task
tbndfs_promela
Standard ou
Logging
model (DVE)
New property
New task
tbndfs_dve
Standard ou
Logging
Time
Use...
Sys ...
Idle
Me...
States
Sent
Reci...
Bar...
Pac...
Cm...
33... 5726 5671 21... 4.67 1307 8839 8839 0 8839 0,8...
33... 5976 5401 21... 4.18 1057 8841 8840 0 8841 88...
[0,8192]
[0,4096]
[7822,0]
[32768,0]
[65536,0]
model -> New property -> New task -> tbndfs_dve Running

Extensions: Work In Progress

- More frontends (NTIF, ATMEL instruction set, ...)
- Formalizing VM in ACL2 theorem prover
- State compression
- More (semi-)static analyses (compatible with distributed settings)
  - Dead Variables ($\approx$ 10% reduction) [Y&G 2004] $\Longrightarrow$ Symmetry Reduction
  - ...

Conclusion

DivSPIN goals

- Ready-to-use distributed model checker
- Tap large user base

- Building on DiVinE framework
- Utilizing Promela as modelling language
- State-space generation:
  NIPS virtual machine
  Building block!