Centrum voor Wiskunde en Informatica

Static Analyses for State Space Reduction

Gustavo Quiros Michael Weber

SENVA / VeriGEM meeting
2006-04-03

Context

- NIPS: A tiny Virtual Machine for State Space Generation
  - 1 Primary Developer, 2 Students, 0 Case studies, 3 Tools using NIPS
- PROMELA : A non-trivial language for Modelling
- Other compilers: C Code for Microcontrollers, Jackal (Java-like DSM)

Motivation

State Space Reductions:

- Symmetry Reductions
  - Requires detectable symmetry (cache coherence protocols, etc.), not always applicable
- Partial Order Reductions
  - Often intricately entangled with state space generator
  - Conditions not always easy to check (cycle proviso in distributed setting)
- ...
- Path Reduction [Yorav & Grumberg]

Motivation

State Space Reductions:

- Symmetry Reductions
  - Requires detectable symmetry (cache coherence protocols, etc.), not always applicable
- Partial Order Reductions
  - Often intricately entangled with state space generator
  - Conditions not always easy to check (cycle proviso in distributed setting)
- ...
- Path Reduction [Yorav & Grumberg]

Key Idea

- Merge sequences of transitions (steps) that cannot influence the value of a temporal logic specification.
- Determine these steps statically.

Advantages

"Language independence": optimizations done on intermediate format

Modularity: no changes to compiler/interpreter/state space generator

No runtime checks: requires no adjustments to model-checking algorithm

Outline

Path Reduction

(path compression, [Yorav & Grumberg 2004]):

- Based on static program analysis.
- Identifies program locations which *may influence* 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.

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
  - Global Variables
  - Rendezvous Channels
  - Async. Channels
- Priorities
- Dynamic creation of processes, channels

Switching to Unstructured Language

- High-level statement: sequence of byte-code instructions followed by `STEP` instruction.
- Merging: rewrite `STEP` instruction in between.
  - No creation of an intermediate state.
  - Next visible state presents the combined effect of both statements.
  - Easily generalized for longer program paths.

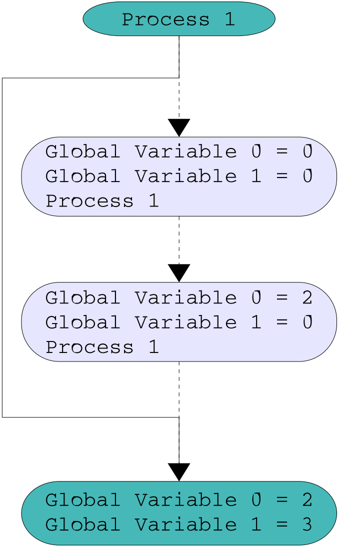
NIPS VM Byte-code

```
int x, y;

init {
  x = 2;
  y = 3;
}
```

```
         GLOBSZ   8
         LDC      0
         STVA     G       4       0
         LDC      0
         STVA     G       4       4
         STEP     N       0
         LJMP     P_init

P_init:  LDC      2
         TRUNC    -31
         STVA     G       4       0
         STEP     N       0
         LDC      3
         TRUNC    -31
         STVA     G       4       4
         STEP     T       0
```



Two Phases

Transformation of the NIPS program: preserves the model's semantics modulo the property to check, but alters the transition system.

Transition-system construction: delegated to the (unmodified) NIPS virtual machine.

Influential Instructions

For process type p and specification S:

Inf_S^p: set of instructions possibly influencing S

- Access of global variables $v_G \in Variables(S)$
- Access of local variables $v_L \in Variables(S)$

External Instructions

For process type p:

Ext^p: set of instructions of p with external effect, i.e. possibly affecting the environment of p

- Global variable access
- Channel operations
- Calls to procedures with instructions $i \in Ext^p$

$Int^p = \{i \mid i \notin Ext^p\}$

Breaking Points

For process type p:

BP^p: nodes $n \in N$ of the control-flow graph (N, X, n_0^p) of p, such that

- n is root n_0^p of the control-flow graph, or
- n is labeled with external instruction $i(n) \in Ext^p$, or
- n is labeled with influential instruction $i(n) \in Inf^p$, or
- n is labeled with call to procedure containing breaking points

Example:

$$bn \ldots n \ldots nbn \ldots$$

Steps

For process type p:

St^p: set of paths $\pi = n_0 \ldots n_k$ in the control-flow graph (N, X, n_0^p) of p, such that

- $\forall 0 \leq j \leq k : i(n_j) = \mathtt{STEP} \iff j = k$
- $n_0 = n_0^p$ implies $\pi \in St^p$
- $\pi' = n'_0 \ldots n'_\ell \in St^p$ and $(n'_\ell, n_0) \in X$ implies $\pi \in St^p$

Example:

$$\underbrace{n \ldots n\ \mathtt{STEP}}_{\pi \in St^p}\ \underbrace{n \ldots n\ \mathtt{STEP}}_{\pi' \in St^p}\ n \ldots$$

Breaking Steps

For process type p:

BSt^p: set of paths $\pi \in St^p$, such that

- $\pi = n_0 \ldots n_k$ and $\exists 0 \leq j \leq k : n_j \in BP^p$

Example:

$$\underbrace{bn \ldots n\ \texttt{STEP}}_{\pi \in BSt^p}\ \underbrace{n \ldots b \ldots nbn\ \texttt{STEP}}_{\pi' \in BSt^p}\ \underbrace{n \ldots n\ \texttt{STEP}}_{\pi'' \in St^p}\ n \ldots$$

Elementary Paths

For process type p:

EP^p: set of maximal paths $\pi = \pi_0 \ldots \pi_\ell$ in the control-flow graph of p, such that $\pi_j \in BSt^p \iff j = 0$ and $\pi_j \in St^p$.

Elementary Steps

ES^p: set of nodes $n_k \in N$, such that $\pi_0 = n_0 \ldots n_k$ and $i(n_k) = \texttt{STEP}$ (for each path $\pi = \pi_0 \ldots \in EP^p$).

Example:

$$\underbrace{\underbrace{bn \ldots n\ \texttt{STEP}}_{\pi_0 \in BSt^p}}_{\pi \in EP^p}\ \underbrace{\underbrace{n \ldots nbn\ \texttt{STEP}}_{\pi'_0 \in BSt^p}\ \underbrace{n \ldots n\ \texttt{STEP}}_{\pi'_1 \in St^p}}_{\pi' \in EP^p}\ n \ldots nb \ldots$$

Hideable Steps

For process type p:

HS^p: set of nodes $\{n \mid i(n) = \texttt{STEP}\} \setminus ES^p$

For $n \in HS^p$: rewrite $i(n)$ to `STEP I`.

Example:

$$\underbrace{bn \ldots n \underbrace{\texttt{STEP}}_{n \in ES^p}}_{\pi \in EP^p} \underbrace{n \ldots nbn \underbrace{\texttt{STEP}}_{n \in ES^p} n \ldots n \underbrace{\texttt{STEP I}}_{n \in HS^p}}_{\pi' \in EP^p} n \ldots$$

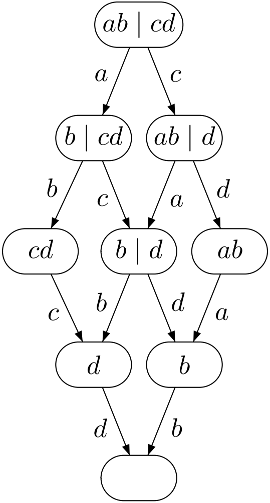
Complications

- Loops without visible `STEP`s
  Ensure every loop contains at least one, preferably in a strategic spot.

- Blocking operations
  `STEP I` dynamic semantics: blocked invisible steps become visible.

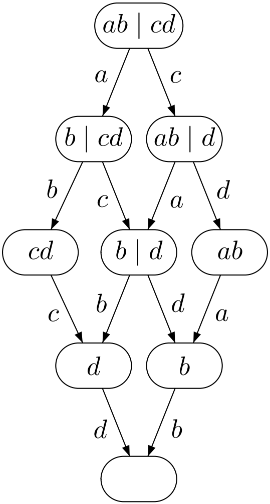
Path Reduction vs. POR

Full Transition System

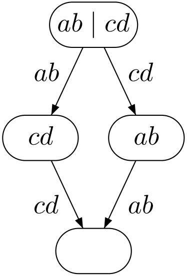


Path Reduction vs. POR

Full Transition System

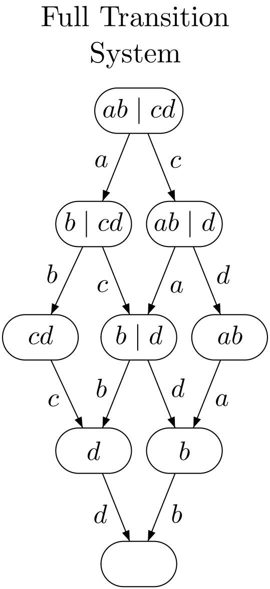


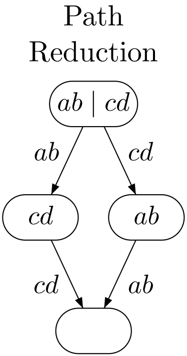
Path Reduction



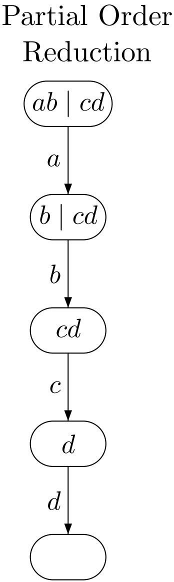
- Static
- Preserves branching structure

Path Reduction vs. POR



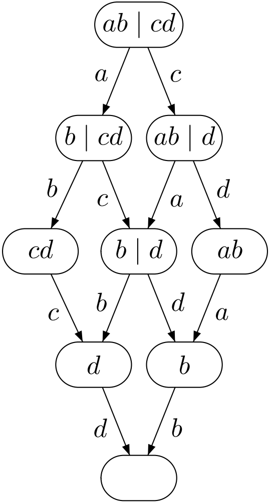


- Static
- Preserves branching structure

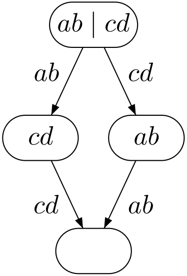


Path Reduction vs. POR

Full Transition System

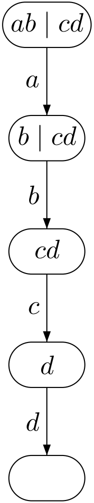


Path Reduction

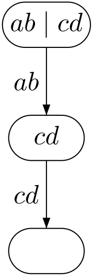


- Static
- Preserves branching structure

Partial Order Reduction



PR+POR



Other Static Reductions

- Dead Variable Reduction (DVR)
- Step Confluence Reduction (SCR)

```
        [...]
        STEP N
label0: [...]
        STEP N
        LJMP label0
```

Confluent steps

```
        [...]
label1: STEP N
label0: [...]
        LJMP label1
```

After SCR

Highly dependent on model

Reducing Sequential Models

Single process: translating C programs for micro controllers

- No state space explosion
- Complex sequential control-flow
- Large State spaces due to local non-determinism: high branching factor
- Partial Order Reduction completely ineffective

Sequential Models: Relaxed Rules

- External operations no breaking points anymore:
  $Ext^p := \varnothing$

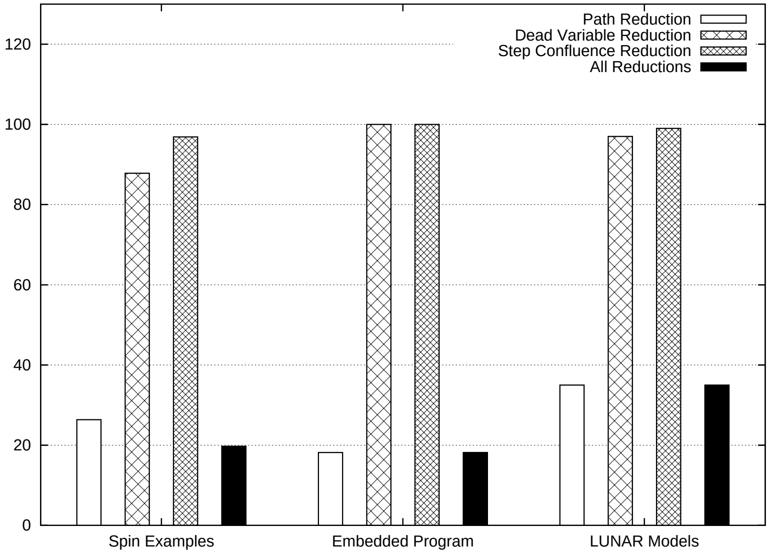
(unless e.g., global variable is influential on specification)

Benchmarks

Model	States Total	States %	Transitions Total	Transitions %	Elapsed Time sec.	Elapsed Time %
4nodes_1down.b	226598	100.00	246664	100.00	3.29	100.00
4nodes_1down+pr.b	65813	29.04	76169	30.88	1.87	56.84
4nodes_1down+dvr.b	224916	99.26	244982	99.32	3.62	110.03
4nodes_1down+scr.b	226507	99.96	246534	99.95	3.31	100.61
4nodes_1down+pr+dvr+scr.b	65714	29.00	76031	30.82	2.09	63.53
4nodes_1up.b	8119319	100.00	8723018	100.00	93.14	100.00
4nodes_1up+pr.b	3265954	40.22	3853245	44.17	66.59	71.49
4nodes_1up+dvr.b	7872574	96.96	8476273	97.17	107.13	115.02
4nodes_1up+scr.b	8119298	100.00	8722988	100.00	92.12	98.90
4nodes_1up+pr+dvr+scr.b	3265933	40.22	3853215	44.17	81.72	87.74
4nodes_simul1.b	1668408	100.00	1779254	100.00	19.43	100.00
4nodes_simul1+pr.b	637818	38.23	734444	41.28	13.72	70.61
4nodes_simul1+dvr.b	1611913	96.61	1722759	96.82	21.94	112.92
4nodes_simul1+scr.b	1668408	100.00	1779254	100.00	19.53	100.51
4nodes_simul1+pr+dvr+scr.b	637786	38.23	734412	41.28	16.27	83.74
mobile1.b	38597	100.00	117957	100.00	0.81	100.00
mobile1+pr.b	11183	28.97	27549	23.36	0.31	38.27
mobile1+dvr.b	23003	59.60	72429	61.40	0.54	66.67
mobile1+scr.b	38597	100.00	117957	100.00	0.81	100.00
mobile1+pr+dvr+scr.b	5267	13.65	13875	11.76	0.18	22.22
mobile2.b	12818	100.00	39227	100.00	0.31	100.00
mobile2+pr.b	3861	30.12	9441	24.07	0.13	41.94
mobile2+dvr.b	7620	59.45	24051	61.31	0.21	67.74
mobile2+scr.b	12818	100.00	39227	100.00	0.31	100.00
mobile2+pr+dvr+scr.b	1841	14.36	4787	12.20	0.09	29.03

Host: core-1 (AMD64 Quad-Opteron 1.4GHz, 16GB RAM, single-threaded)

Benchmarks


Path Reduction
Dead Variable Reduction
Step Confluence Reduction
All Reductions
0
20
40
60
80
100
120
Spin Examples
Embedded Program
LUNAR Models

Path Reduction for NIPS VM

Versus [Y&G 2004]:

- Adapted to
  - Dynamic creation of channels and processes
  - Asynchronous communication
  - Global variables $\mapsto$ informal communication channels.

Less involved framework: program transformation
Hard work: delegate to VM

Conclusions

- Path Reduction
  - Works as post-processor for PROMELA, C microcontroller programs (via intermediate representation)
  - Reduction: $\approx 50 - 60\%$ (on average)
  - Not always translates to corresponding speedup(!)
- Highly model-dependent
- Some of the reductions are not expressible in PROMELA syntax

Backup Slides