An Alternative Structure for SPIN?

PJA de Villiers
Department of Computer Science
University of Stellenbosch
Stellenbosch
South Africa

Introduction

- Project goals:

  1. Validation of a microkernel

  2. Learn about validators (implement one specifically to validate the microkernel)

- **Focus of this talk: experience that may have some value for SPIN**

The validation process

1. Generate a state

2. Determine whether it is a new state

3. If not a new state, then

   (a) a loop

   (b) a revisited state

4. Analyse the state

What is different?

- State generation
  - Code generated for abstract machine
- State storage
  - State cache with replacement + compaction
- State analysis
  - CTL for correctness claims
  - Nested subformulae handled as “subproblems”
- Highly modular design

State generation

Measurements (interpreted state generator):

	Philos	Elevator	Scheduler
Trans	2338593	9086599	9908147
Time (secs)	123.52	439.81	450.07
Trans/sec	18933	20660	22015

Measurements (SPIN):

	pftp1	leader	pftp2
Trans	1301620	185032	1223060
Time (secs)	1:08.38	9.29	47.98
Trans/sec	19035	19917	25491

- Abstract machine: validator more understandable, fast enough

- Difficult to isolate the effect of interpretation

State storage

- How costly is state compaction?

Module	% of time
State generator	50.43
Scheduler	10.90
Storage	21.63
Compaction	15.17
Other	1.87

- Substantial reduction in state size. For example, states of a rather complex model (the scheduler of the microkernel) fit in 16 bytes)

Benefits of state compaction. . .

- cache performance is improved
  - fewer state replacements (more states fit into memory)
  - hash function less costly to compute (state is smaller)
- Exhaustive state exploration possible for larger models

Computing compact states...

```
VAR c: (red, green, blue, white, black);
    b: BOOLEAN;
    p: RECORD x, y: 0..149 END;
```

- Just enough space allocated per variable

- Compacted state S is computed as

  $$c + 5 \cdot (b + 2 \cdot (p.x + 150 \cdot p.y))$$
  $$= c + 5 \cdot b + 5 \cdot 2 \cdot p.x + 5 \cdot 2 \cdot 150 \cdot p.y$$

- Each variable z has two associated "shield" factors z_{lo} and z_{hi} ($b_{lo} = 5$ and $b_{hi} = 5 \cdot 2$)

Extracting values from compacted states...

- The value of variable $z = (S \; mod \; z_{hi}) \; div \; z_{lo}$

- When the value of variable z changes to z', the updated state is $S' = S + z_{lo} \cdot (z' - z)$

- The factors z_{lo} and z_{hi} are computed at compile time

- The cost of variable lookup is 2 operations (mod, divide)

- The cost of modifying a variable is 3 operations (add, subtract, multiply)

$\boxed{\textbf{State analysis}}$

Handling nested temporal subformulae...

- Example: $AG(\alpha \Rightarrow AF\beta)$

- Key idea: assume $AF\beta$ is true and postpone evaluation until it is more convenient

- Subset of states computed where further evaluation is needed (states where α is true)

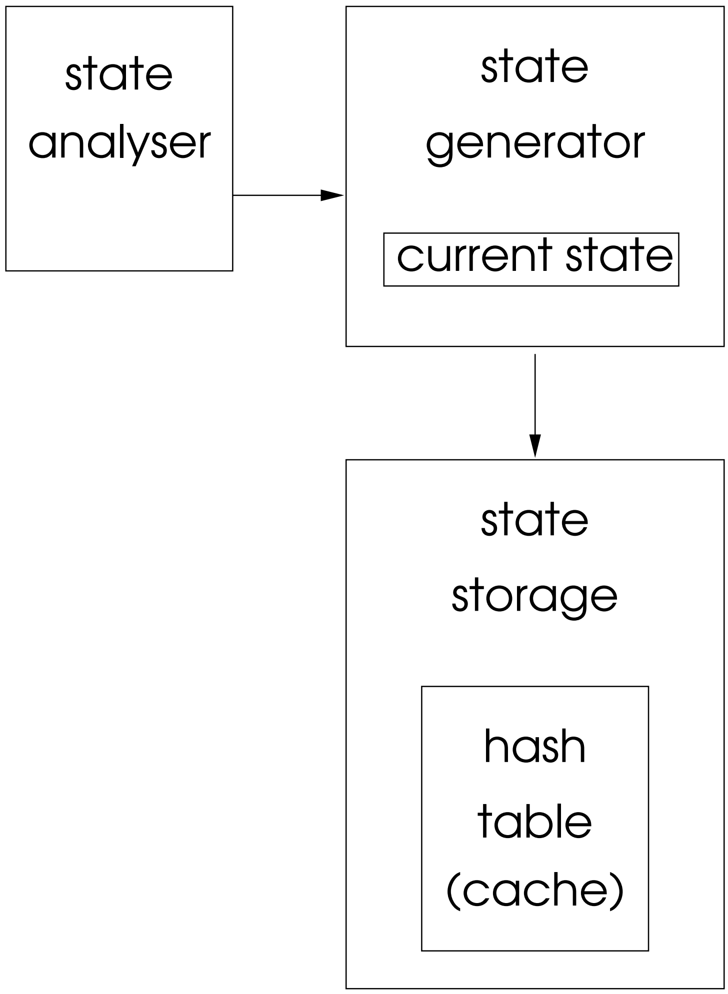
- No storage needed for values of nested subformulae

Avoiding unnecessary work...

$$
\begin{aligned}
EF\alpha &\equiv \alpha \vee EXEF\alpha \\
AF\alpha &\equiv \alpha \vee AXAF\alpha \\
EG\alpha &\equiv \alpha \vee EXEG\alpha \\
AG\alpha &\equiv \alpha \vee AXAG\alpha \\
E(\alpha\ \mathcal{U}\ \beta) &\equiv \beta \vee (\alpha \wedge EXE(\alpha\ \mathcal{U}\ \beta)) \\
A(\alpha\ \mathcal{U}\ \beta) &\equiv \beta \vee (\alpha \wedge AXA(\alpha\ \mathcal{U}\ \beta))
\end{aligned}
$$

- use the cache to avoid unnecessary work
- cache invariant: the current subformula holds for all states in the cache
- requirement: no state is entered in the cache before it is known that the current subformula holds for it

Structure of validator



Module sizes

Module	Lines	Code size
State generator	803	17132
Scheduler	221	3040
Storage	142	1568
Compaction	161	2196
Total	**1327**	**23936**

Conclusions

- Interpretation simplifies the validator and is probably efficient enough

- Compaction does not cost much and makes a state cache very effective

- The main functions of a validator can be implemented in three relatively independent modules without making it grossly inefficient. These modules are:
  - state generation
  - state storage and compaction
  - state analysis