UNIVERSITY OF TWENTE.

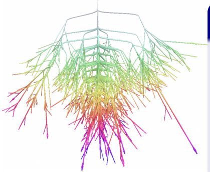
Formal Methods & Tools.

Multi-Core Model Checking

Alfons Laarman

November 14, 2013

State Space Explosion

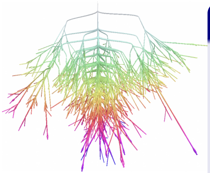


(LaQuSo project)

An exponential problem

- system data
- system components
- property size

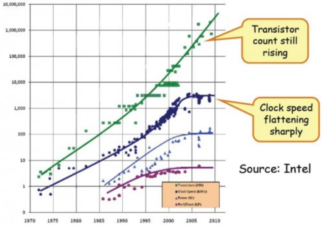
State Space Explosion



(LaQuSo project)

An exponential problem

- system data
- system components
- property size

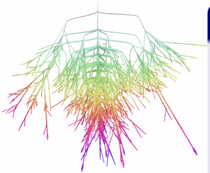


Source: Intel

Approach

- multi-core model checking

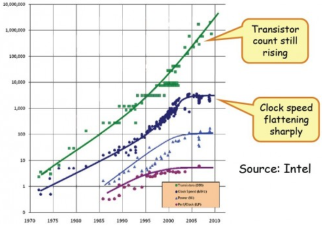
State Space Explosion



(LaQuSo project)

An exponential problem

- system data
- system components
- property size



Source: Intel

Approach

- multi-core model checking
- Confluence / partial-order reduction
- Symbolic techniques (BDD-based and SAT-based)
- On-the-fly techniques
- Compression techniques

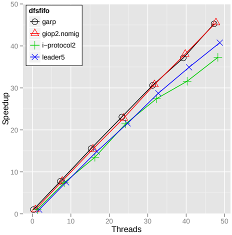
Multi-Core Model Checking

Research questions

- Can model checking scale (linearly, ideally) on modern multi-cores?

Speedup:
$S_P = T_{seq}/T_P$

Ideal: $S_P = P$
Linear:
$S_P = P/c$



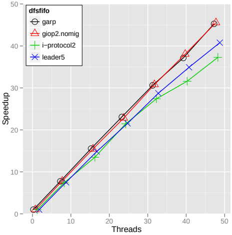
Multi-Core Model Checking

Research questions

- Can model checking scale (linearly, ideally) on modern multi-cores?
  - Formalisms: plain, timed, stochastic, etc
  - Properties: Reachability, LTL, CTL, etc

Speedup:
$S_P = T_{seq}/T_P$

Ideal: $S_P = P$
Linear:
$S_P = P/c$



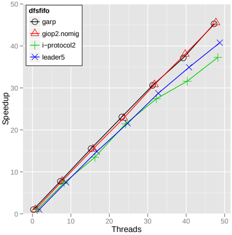
Multi-Core Model Checking

Research questions

- Can model checking scale (linearly, ideally) on modern multi-cores?
  - Formalisms: plain, timed, stochastic, etc
  - Properties: Reachability, LTL, CTL, etc
- Are our parallel solutions compatible with other techniques?

Speedup:
$S_P = T_{seq}/T_P$

Ideal: $S_P = P$
Linear:
$S_P = P/c$



$+$

- Partial-order reduction (POR)
- Symbolic exploration
- On-the-fly techniques
- Compression techniques

Challenges

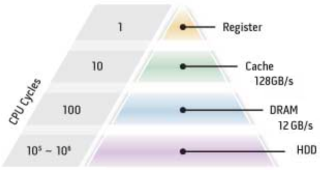
Difficulties of parallelism

- Correctness of data structures and algorithms

Challenges

Difficulties of parallelism

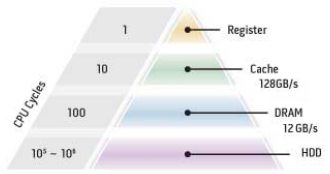
- Correctness of data structures and algorithms
- Steep memory hierarchies



Challenges

Difficulties of parallelism

- Correctness of data structures and algorithms
- Steep memory hierarchies
- Cache coherence protocol

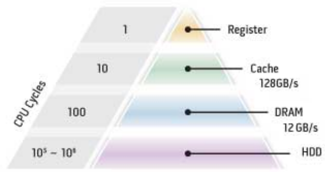



```
#define B (1024*1024*1024)

int main (void) {
    int result = 0;
    for (int i = 0; i < B; i++)
        result++;
    return result;
}
```

Challenges

Difficulties of parallelism

- Correctness of data structures and algorithms
- Steep memory hierarchies
- Cache coherence protocol



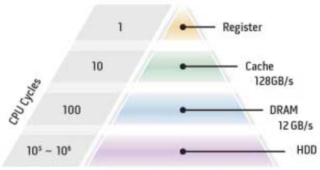

```
#define B (1024*1024*1024)

int main (void) {
    int result = 0;
    for (int i = 0; i < B; i++)
        result++;
    return result;
}
```

```
#define P 16

static void count (void *arg) {
    int *counter = (int *) arg;
    for (int i = 0; i < B / P; i++) ( *counter)++;
}
int main (void) {
    pthread_t thread[P];
    int counters[P] = 0;

    for (int i = 0; i < P; i++)
        pthread_create (&thread[i], NULL, count, &counters[i]);

    int result = 0;
    for (int i = 0; i < P; i++) {
        pthread_join (thread[i], NULL);
        result += counters[i];
    }
    return result;
}
```

Challenges

Difficulties of parallelism

- Correctness of data structures and algorithms
- Steep memory hierarchies
- Cache coherence protocol



$T = 27$

```
#define B (1024*1024*1024)

int main (void) {
    int result = 0;
    for (int i = 0; i < B; i++)
        result++;
    return result;
}
```

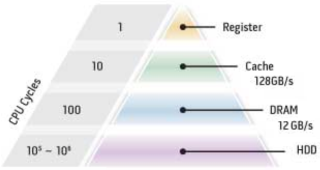
$T_{16} = 32$

```
#define P 16

static void count (void *arg) {
    int *counter = (int *) arg;
    for (int i = 0; i < B / P; i++) ( *counter)++;
}
int main (void) {
    pthread_t thread[P];
    int counters[P] = 0;

    for (int i = 0; i < P; i++)
        pthread_create (&thread[i], NULL, count, &counters[i]);

    int result = 0;
    for (int i = 0; i < P; i++) {
        pthread_join (thread[i], NULL);
        result += counters[i];
    }
    return result;
}
```

Challenges

Difficulties of parallelism

- Correctness of data structures and algorithms
- Steep memory hierarchies
- Cache coherence protocol (false sharing)



$T = 27$

```
#define B (1024*1024*1024)

int main (void) {
    int result = 0;
    for (int i = 0; i < B; i++)
        result++;
    return result;
}
```

$T_{16} = 32$

$T_{16} = 1.8$

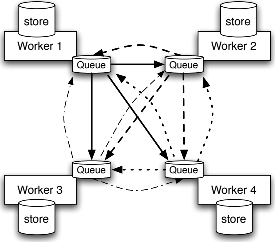
```
#define P 16

static void count (void *arg) {
    int *counter = (int *) arg;
    for (int i = 0; i < B / P; i++) ( *counter)++;
}
int main (void) {
    pthread_t thread[P];
    int __attribute__ ((aligned(64))) counters[P] = 0;

    for (int i = 0; i < P; i++)
        pthread_create (&thread[i], NULL, count, &counters[i]);

    int result = 0;
    for (int i = 0; i < P; i++) {
        pthread_join (thread[i], NULL);
        result += counters[i];
    }
    return result;
}
```

(Explicit-state) reachability

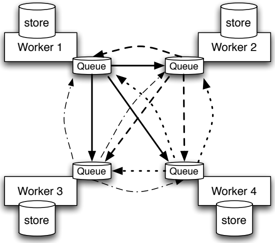
Problem:
find all reachable states from $s_0 \in S$ using a next-state function: $post(S) \rightarrow 2^S$

A state $s \in S$ is a (fixed) K-sized vector: $\langle v_1, \ldots, v_K \rangle$

Static partitioning or shared hash table



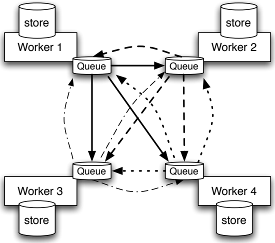
Static partitioning or shared hash table

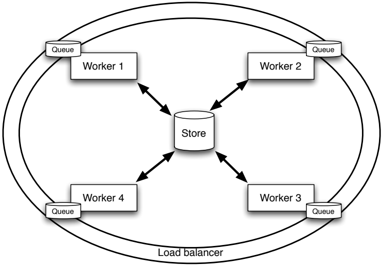


Static partitioning

- X On-the-fly (BFS)
- ± Scalability (communication on queues)

Static partitioning or shared hash table





Static partitioning

- X On-the-fly (BFS)
- ± Scalability (communication on queues)

Shared hash table

- ✓ On-the-fly: (pseudo) DFS & BFS
- ? Scalability

Lockless Hash Table: Design

Laarman, van de Pol, Weber [fmcad10]

Main bottlenecks

- State store: concurrent access
- Graph traversal: Random memory access (bandwidth)

Lockless Hash Table: Design

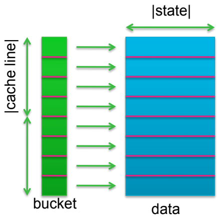
LAARMAN, VAN DE POL, WEBER [FMCAD10]

Main bottlenecks

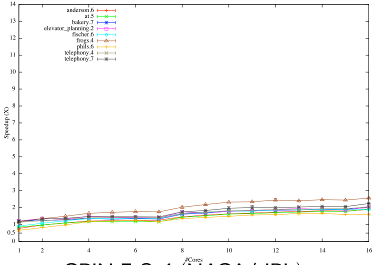
- State store: concurrent access
- Graph traversal: Random memory access (bandwidth)

Design

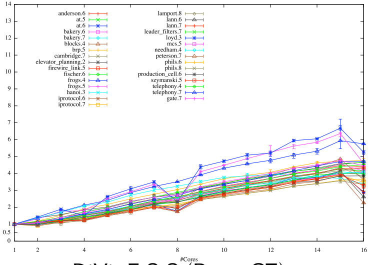
- Hash memoization
- Walking the Line
- In-situ locking



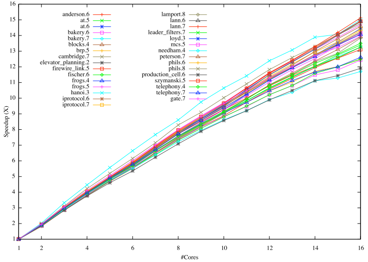
Experiments from 2010 (BEEM database)



SPIN 5.2.4 (NASA/JPL)



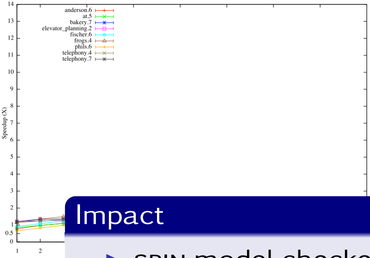
DiVinE 2.2 (Brno,CZ)

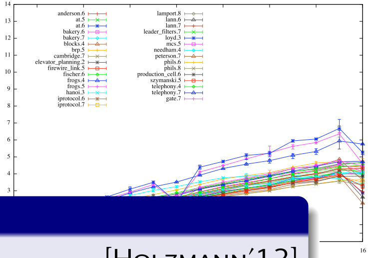


LTSmin

(shared hash table)

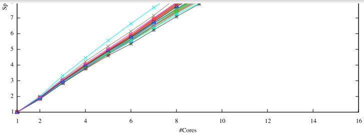
Experiments from 2010 (BEEM database)





Impact

- SPIN model checker .... [HOLZMANN'12]
- GPU model checking ...... [SULEWSKI ET AL '11,12]
- Parallel BDDs .... VAN DIJK, LAARMAN, VAN DE POL [AVOCS12][PDMC12]



LTSmin

(shared hash table)

Reachability

- Scalability comes from limiting bandwidth usage
- Correctness established with model checker

	Explicit state	+ Compression	+ POR	+ On-the-fly
Reachability	✓	?	?	✓

Reachability

- Scalability comes from limiting bandwidth usage
- Correctness established with model checker

	Explicit state	+ Compression	+ POR	+ On-the-fly
Reachability	✓	?	✓	✓

- Partial-order reduction can be computed (state) locally

Reachability

- Scalability comes from limiting bandwidth usage
- Correctness established with model checker

	Explicit state	+ Compression	+ POR	+ On-the-fly
Reachability	✓	X	✓	✓

- Partial-order reduction can be computed (state) locally
- No compression, but states are often very similar due to locality

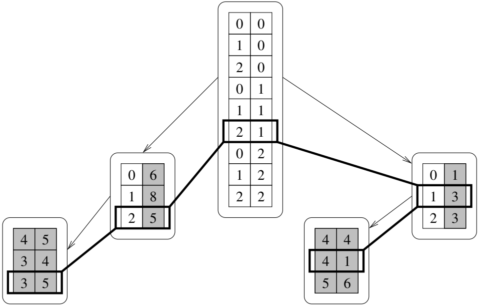
$$\langle 3, 5, 5, 4, 1, 3\rangle \longrightarrow \langle 3, 5, 9, 3, 1, 3\rangle$$

Recursive indexing

[HOLZMANN 97][BLOM ET AL. 08]

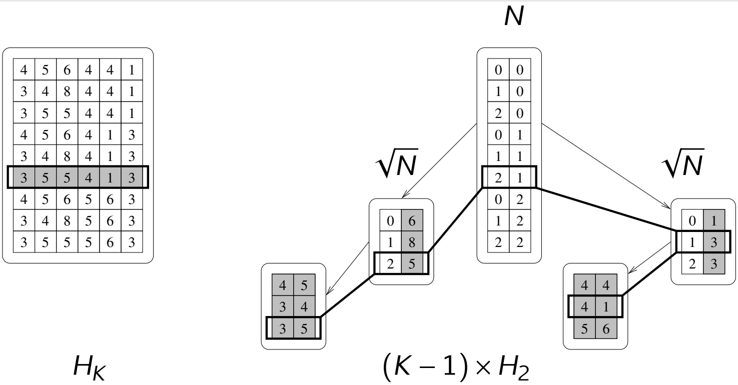
4	5	6	4	4	1
3	4	8	4	4	1
3	5	5	4	4	1
4	5	6	4	1	3
3	4	8	4	1	3
3	5	5	4	1	3
4	5	6	5	6	3
3	4	8	5	6	3
3	5	5	5	6	3

H_K



$(K-1) \times H_2$

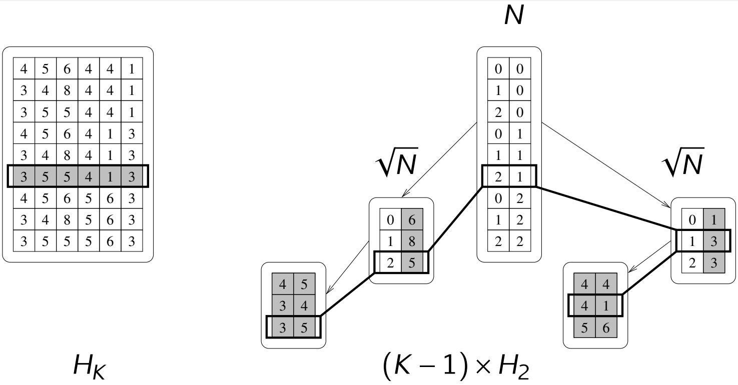
Recursive indexing
[HOLZMANN 97][BLOM ET AL. 08]



✓ Combinatorial $\implies$ balanced tree $(N + 2\sqrt{N} + 4\sqrt[4]{(N)} \cdots \approx N)$
Compresses states of lenght K to almost 2!

Recursive indexing

[HOLZMANN 97][BLOM ET AL. 08]



- ✓ Combinatorial $\implies$ balanced tree $(N + 2\sqrt{N} + 4\sqrt[4]{(N)} \cdots \approx N)$
  Compresses states of lenght K to almost 2!
- X Hard to parallelize (flatliners)

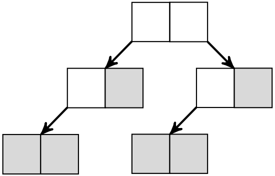
Parallel Tree Compression

Laarman, van de Pol, Weber [spin11]

Solution

- Temporary binary tree structure on stack

$\langle 3, 5, 5, 4, 1, 3 \rangle$

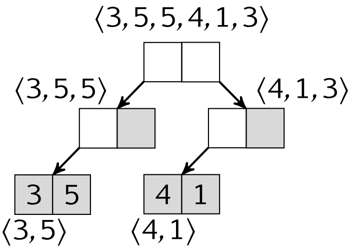


Parallel Tree Compression

Laarman, van de Pol, Weber [spin11]

Solution

- Temporary binary tree structure on stack

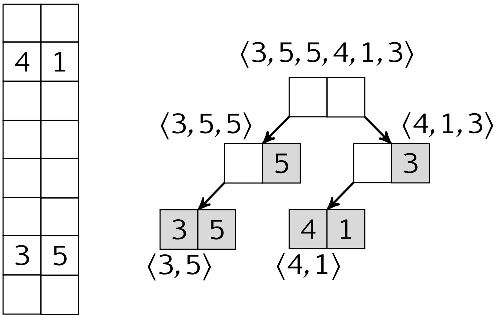


Parallel Tree Compression

LAARMAN, VAN DE POL, WEBER [SPIN11]

Solution

- Temporary binary tree structure on stack
- Reuse lockless hash table (merge tables)

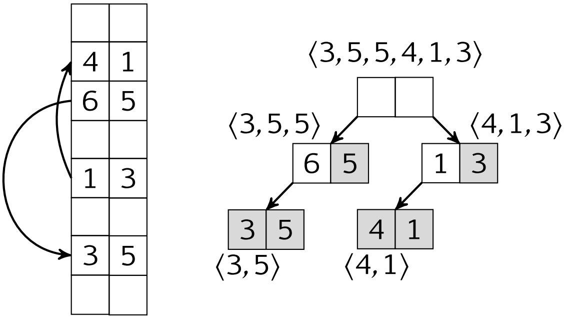


Parallel Tree Compression

LAARMAN, VAN DE POL, WEBER [SPIN11]

Solution

- Temporary binary tree structure on stack
- Reuse lockless hash table (merge tables)

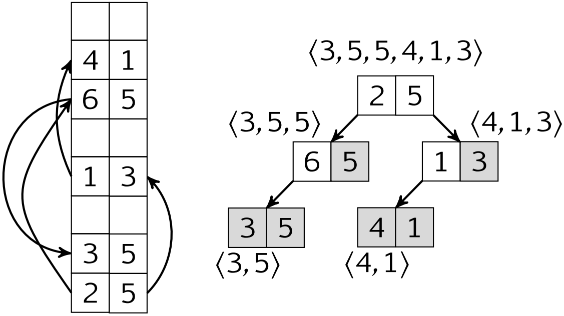


Parallel Tree Compression

Laarman, van de Pol, Weber [spin11]

Solution

- Temporary binary tree structure on stack
- Reuse lockless hash table (merge tables)

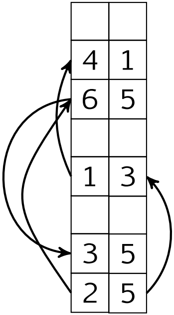


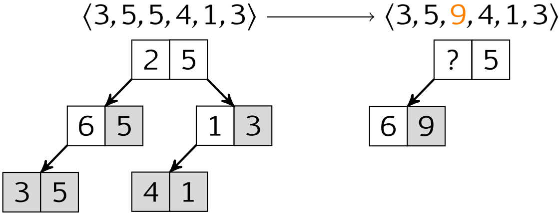
Parallel Tree Compression

Laarman, van de Pol, Weber [spin11]

Solution

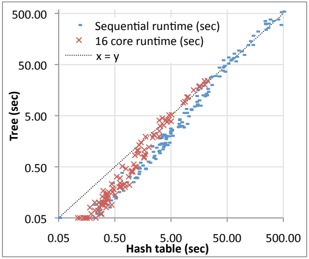
- Temporary binary tree structure on stack
- Reuse lockless hash table (merge tables)
- Incremental updates: $(K-1) \rightarrow \log_2(K-1)$ lookups





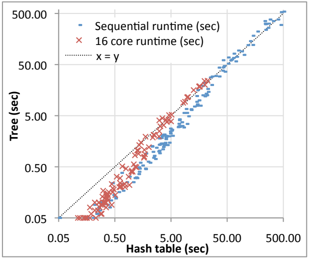
Experiments from 2011 [BEEM database]

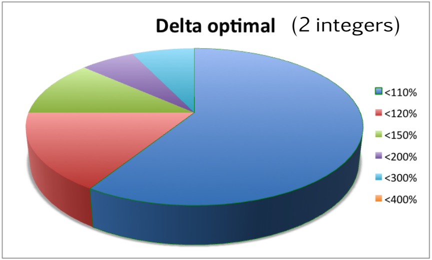
Laarman, van de Pol, Weber [spin11]



Experiments from 2011 [BEEM database]

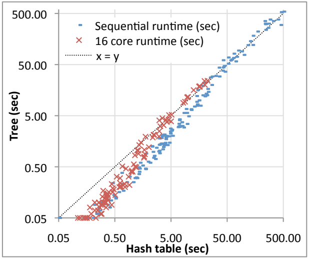
Laarman, van de Pol, Weber [spin11]

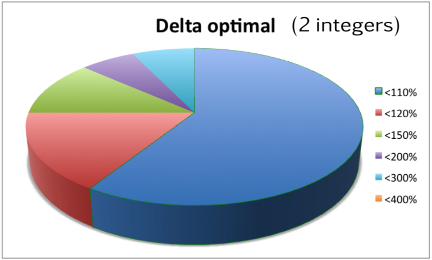




Experiments from 2011 [BEEM database]

Laarman, van de Pol, Weber [spin11]





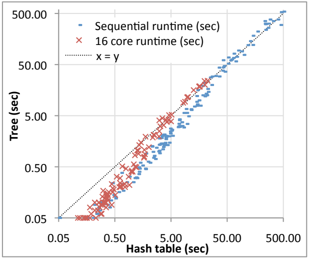
$\langle 3,5,5,4,1,3\rangle \longrightarrow \langle 3,5,9,4,1,3\rangle \longrightarrow \langle 3,5,9,3,2,3\rangle \longrightarrow$

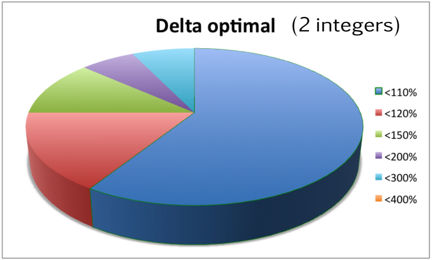
Information theoretical *lower bound*?

- View states as stream of variables: $\langle v_1^1, \dots v_K^1\rangle, \langle v_1^2, \dots v_K^2\rangle, \dots$ with $|v_j^i| = 2^{32}$

Experiments from 2011 [BEEM database]

LAARMAN, VAN DE POL, WEBER [SPIN11]





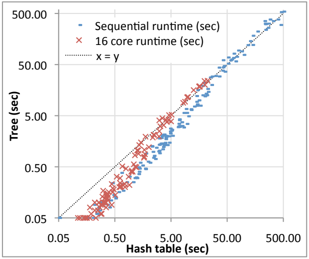
$\langle 3,5,5,4,1,3\rangle \xrightarrow{\frac{K-1}{K}} \langle 3,5,9,4,1,3\rangle \xrightarrow{\frac{1}{K}} \langle 3,5,9,3,2,3\rangle \longrightarrow$

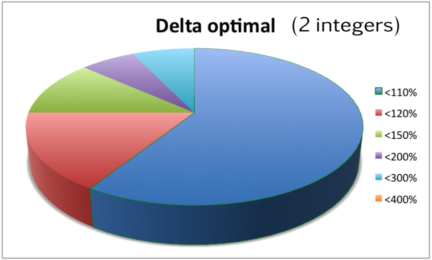
Information theoretical *lower bound*?

- View states as stream of variables: $\langle v_1^1, \dots v_K^1\rangle, \langle v_1^2, \dots v_K^2\rangle, \dots$ with $|v_j^i| = 2^{32}$
- $p(v_j^i = v_j^{i-1}) = \frac{K-1}{K}$ and $p(v_j^i \neq v_j^{i-1}) = \frac{1}{K}$ (*under-estimation*)

Experiments from 2011 [BEEM database]

LAARMAN, VAN DE POL, WEBER [SPIN11]





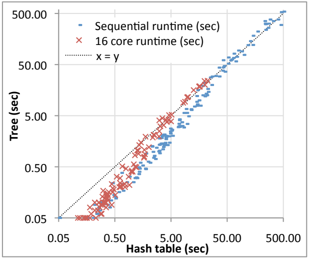
$\langle 3,5,5,4,1,3\rangle \xrightarrow{\frac{K-1}{K}} \langle 3,5,9,4,1,3\rangle \xrightarrow{\frac{1}{K}} \langle 3,5,9,3,2,3\rangle \longrightarrow$

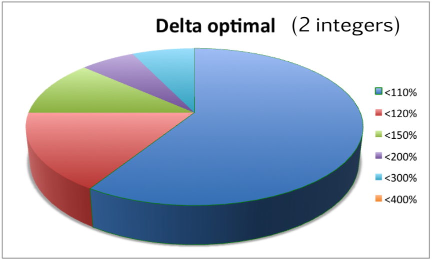
Information theoretical *lower bound*?

- View states as stream of variables: $\langle v_1^1, \dots v_K^1\rangle, \langle v_1^2, \dots v_K^2\rangle, \dots$ with $|v_j^i| = 2^{32}$
- $p(v_j^i = v_j^{i-1}) = \frac{K-1}{K}$ and $p(v_j^i \neq v_j^{i-1}) = \frac{1}{K}$ (*under-estimation*)
- Entropy per state: $K \times H(s_j^i) \approx \log_2(2^{32}) + \log_2(K)$ bits $\approx 1 + \epsilon$ integer

Experiments from 2011 [BEEM database]

LAARMAN, VAN DE POL, WEBER [SPIN11]





$\frac{K-1}{K}$ $\frac{1}{K}$

$\langle 3,5,5,4,1,3\rangle \longrightarrow \langle 3,5,9,4,1,3\rangle \longrightarrow \langle 3,5,9,3,2,3\rangle \longrightarrow$

Information theoretical *lower bound*?

- View states as stream of variables: $\langle v_1^1,\dots v_K^1\rangle,\langle v_1^2,\dots v_K^2\rangle,\dots$ with $|v_j^i| = 2^{32}$
- $p(v_j^i = v_j^{i-1}) = \frac{K-1}{K}$ and $p(v_j^i \neq v_j^{i-1}) = \frac{1}{K}$ (*under-estimation*)
- Entropy per state: $K \times H(s_j^i) \approx \log_2(2^{32}) + \log_2(K)$ bits $\approx 1+\epsilon$ integer
- Halve the root table with *Cleary compact hash table* [MEMICS11]

Reachability

- Scalability from merging tables & incremental updates
- Correctness proved by hand
  - The recursive tree function is an injection [SPIN11]

	Explicit state	+ Compression	+ POR	+ On-the-fly
Reachability	✓	✓	✓	✓

Reachability

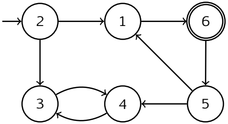
- Scalability from merging tables & incremental updates
- Correctness proved by hand
  - The recursive tree function is an injection [SPIN11]

	Explicit state	+ Compression	+ POR	+ On-the-fly
Reachability	✓	✓	✓	✓
LTL	?	?	?	?

Still only safety...

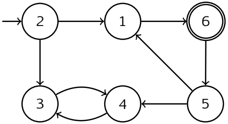
LTL

The ω-language of the Büchi automaton represents all counter examples [Vardi et Wolper 86]



LTL

The ω-language of the Büchi automaton represents all counter examples [Vardi et Wolper 86]



"It is as yet an open problem how a liveness verification algorithm could be generalized to the use of more than two processing cores while retaining a low search complexity."

[Holzmann '07]

"One of the most important open problems of parallel LTL model checking is to design an on-the-fly scalable parallel algorithm with linear time complexity."

[Brim, Barnat et Ročkai '11]

Nested Depth-First Search for LTL

[Courcoubetis'93]

```
procedure DFSblue(s)
    s.cyan := true
    for all s′ in post(s) do
        if ¬t.blue∧¬t.cyan then
            DFSblue(s′)
    if accepting(s) then
        DFSred(s)
    s.blue := true
    s.cyan := false
procedure DFSred(s)
    s.red := true
    for all s′ ∈ post(s) do
        if t.cyan then ExitCycle
        if ¬t.red then DFSred(s′)
```

Nested DFS (NDFS)

- Linear time

Nested Depth-First Search for LTL

[Courcoubetis'93]

```
procedure DFSblue(s)
    s.cyan := true
    for all s′ in post(s) do
        if ¬t.blue∧¬t.cyan then
            DFSblue(s′)
    if accepting(s) then
        DFSred(s)
    s.blue := true
    s.cyan := false
procedure DFSred(s)
    s.red := true
    for all s′ ∈ post(s) do
        if t.cyan then ExitCycle
        if ¬t.red then DFSred(s′)
```

Nested DFS (NDFS)

- Linear time
- DFS itself is likely not parallelizable
  - DFS order is a P-complete problem
  - We assume: $P \neq NC$

Multi-core Nested Depth-First Search (Principle)

[ATVA11], [PDMC11], [ATVA12]

code for worker p:

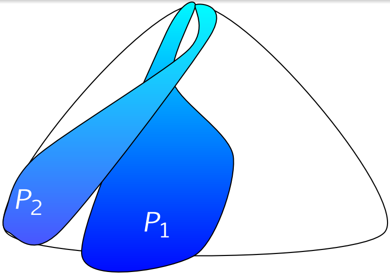
```
procedure DFSblue(s,p)
    s.cyan[p] := true
    for all s′ in shuffle(post(s)) do
        if ¬s′.blue∧¬t.cyan[p] then
            DFSblue(s′,p)
    if accepting(s) then
        DFSred(s,p)
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s,p)
    s.red[p] := true
    for all s′∈post(s) do
        if t.cyan[p] then ExitCycle
        if ¬t.red[p] then DFSred(s′,p)
```

Multi-core Nested Depth-First Search (Principle)

[ATVA11], [PDMC11], [ATVA12]

code for worker p:

procedure DFSblue(s,p)
    s.cyan[p] := true
    for all s′ **in** shuffle(post(s)) **do**
        if ¬s′.blue∧¬t.cyan[p] **then**
            DFSblue(s′,p)
    if accepting(s) **then**
        DFSred(s,p)
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s,p)
    s.red[p] := true
    for all s′∈post(s) **do**
        if t.cyan[p] **then** ExitCycle
        if ¬t.red[p] **then** DFSred(s′,p)

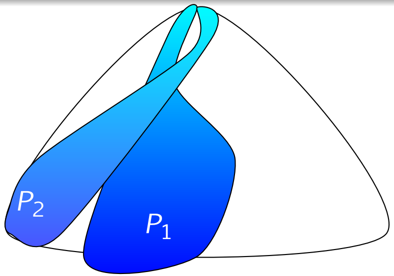


Multi-core Nested Depth-First Search (Principle)

[ATVA11], [PDMC11], [ATVA12]

code for worker p:

```
procedure DFSblue(s,p)
    s.cyan[p] := true
    for all s′ in shuffle(post(s)) do
        if ¬s′.blue∧¬t.cyan[p] then
            DFSblue(s′,p)
    if accepting(s) then
        DFSred(s,p)
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s,p)
    s.red[p] := true
    for all s′∈post(s) do
        if t.cyan[p] then ExitCycle
        if ¬t.red[p] then DFSred(s′,p)
```



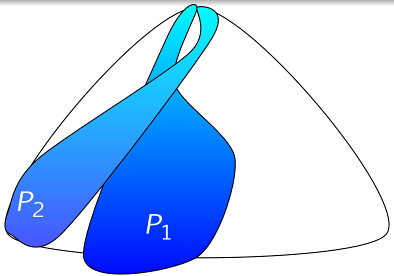
- In reality more synchronization!
- LAARMAN, WIJS ET AL. [ATVA11]
  LAARMAN ET VAN DE POL [PDMC11]
  EVANGELISTA, LAARMAN ET AL. [ATVA12]

Multi-core Nested Depth-First Search (Principle)

[ATVA11], [PDMC11], [ATVA12]

code for worker p:

procedure DFSblue(s,p)
    s.cyan[p] := true
    for all s′ **in** shuffle(post(s)) **do**
        if ¬s′.blue∧¬t.cyan[p] **then**
            DFSblue(s′,p)
    if accepting(s) **then**
        DFSred(s,p)
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s,p)
    s.red[p] := true
    for all s′∈post(s) **do**
        if t.cyan[p] **then** ExitCycle
        if ¬t.red[p] **then** DFSred(s′,p)



- In reality more synchronization!
- LAARMAN, WIJS ET AL. [ATVA11]
  LAARMAN ET VAN DE POL [PDMC11]
  EVANGELISTA, LAARMAN ET AL. [ATVA12]
- **Lemma 4**: Blue states have blue or cyan successors: $Blue \subseteq \bigcup_p \Box(Blue \cup Cyan_p)$.

LTL and Partial-Order Reduction

- Scalability due to hash/tree table (linear-time)
- Correctness proved by hand [ATVA11][PDMC11][ATVA12]

LTL and Partial-Order Reduction

- Scalability due to hash/tree table (linear-time)
- Correctness proved by hand [ATVA11][PDMC11][ATVA12]

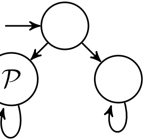
For partial-order reduction, we need to solve ignoring

LTL and Partial-Order Reduction

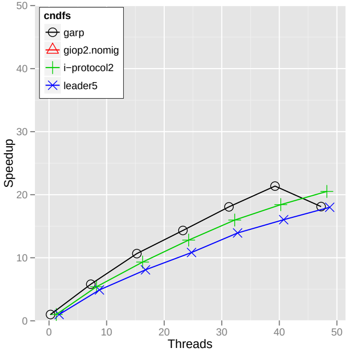
- Scalability due to hash/tree table (linear-time)
- Correctness proved by hand [ATVA11][PDMC11][ATVA12]

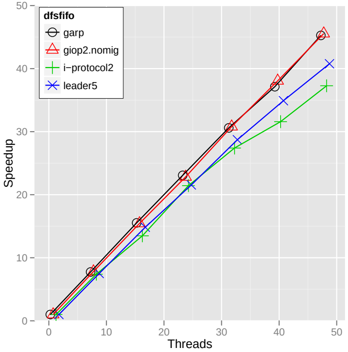
For partial-order reduction, we need to solve ignoring

- For livelocks ($\supset$ LTL), any unfair cycle is a counter example!
- Parallel DFS_{FIFO} Laarman et Faragó [NFM13]

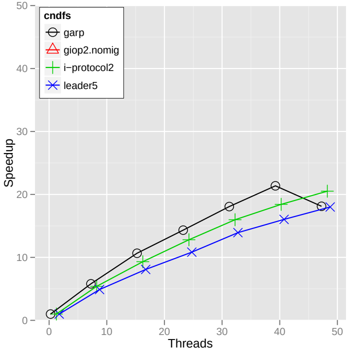


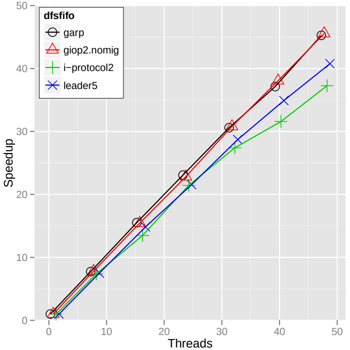
Experiments: LTL with Partial-Order Reduction





Experiments: LTL with Partial-Order Reduction

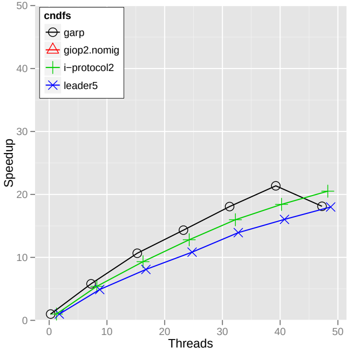


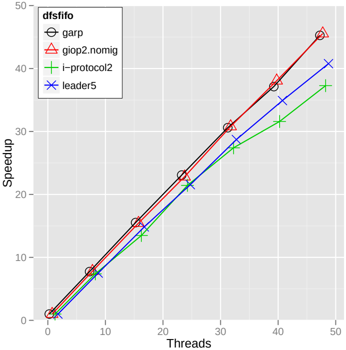


Partial-order reductions:

	LTSMIN DFS$_{FIFO}$	SPIN NDFS
leader	0.49%	1.15%
garp	2.18%	12.73%
giop	1.86%	2.42%
i-prot	31.83%	41.37%

Experiments: LTL with Partial-Order Reduction





Partial-order reductions:

	LTSMIN DFS$_{FIFO}$	SPIN NDFS
leader	0.49%	1.15%
garp	2.18%	12.73%
giop	1.86%	2.42%
i-prot	31.83%	41.37%

Max. model size explored in 30 min.

cores	LTSMIN DFS$_{FIFO}$	DIVINE OWCTY
1	12	9
48	15	11

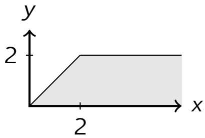
DFS$_{FIFO}$ VS OWCTY + POR [BRIM ET AL '10]

Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly
Plain	Reachability	✓	✓	✓	✓
	LTL	✓	✓	X	✓
	..... Livelocks	✓	✓	✓	✓

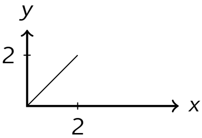
Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly
Plain	Reachability	✓	✓	✓	✓
	LTL	✓	✓	X	✓
	. . . . . Livelocks	✓	✓	✓	✓
Timed	Reachability	?	?	?	?
	LTL	?	?	?	?

Timed Automata

States are semi-symbolic: $s = \langle d, \sigma \rangle$ (finite continuous-time abstraction)



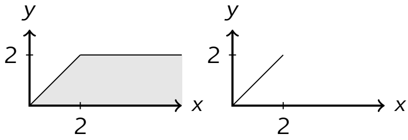
$Z_1 :=$
$y - x \leq 0 \wedge y \leq 2$



$Z_2 := Z_3 :=$
$y - x = 0 \wedge y \leq 2$

Timed Automata

States are semi-symbolic: $s = \langle d, \sigma \rangle$ (finite continuous-time abstraction)



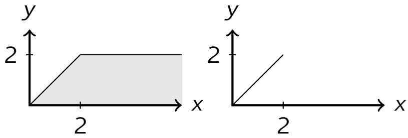
$Z_1 :=$ $y - x \leq 0 \wedge y \leq 2$

$Z_2 := Z_3 :=$ $y - x = 0 \wedge y \leq 2$

This introduces a new subsumption relation: $s \sqsubseteq s'$, iff $d = d' \wedge \sigma \sqsubseteq \sigma'$

Timed Automata

States are semi-symbolic: $s = \langle d, \sigma \rangle$ (finite continuous-time abstraction)



$Z_1 :=$ $y - x \leq 0 \wedge y \leq 2$

$Z_2 := Z_3 :=$ $y - x = 0 \wedge y \leq 2$

This introduces a new subsumption relation: $s \sqsubseteq s'$, iff $d = d' \wedge \sigma \sqsubseteq \sigma'$

Subsumption is a simulation relation which allows another, dynamic abstraction

Timed Automata

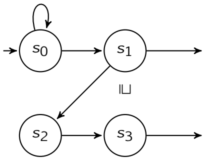
Dalsgaard, Laarman, Olesen, Larsen, van de Pol [formats12]

✓ For reachability, we implemented a lockless multi-map [format12]

Timed Automata

Dalsgaard, Laarman, Olesen, Larsen, van de Pol [FORMATS12]

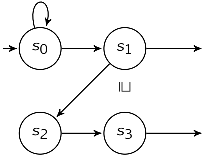
✓ For reachability, we implemented a lockless multi-map [FORMAT12]



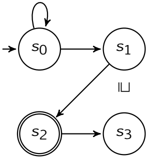
Timed Automata

Dalsgaard, Laarman, Olesen, Larsen, van de Pol [FORMATS12]

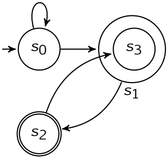
✓ For reachability, we implemented a lockless multi-map [FORMAT12]



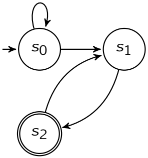
X Subsumption does not preserve Büchi emptiness! [Tripakis'09]



Timed abstraction



$s_3 \sqsubseteq s_1$



subsumption

Analysis of accepting cycles/spirals with subsumption

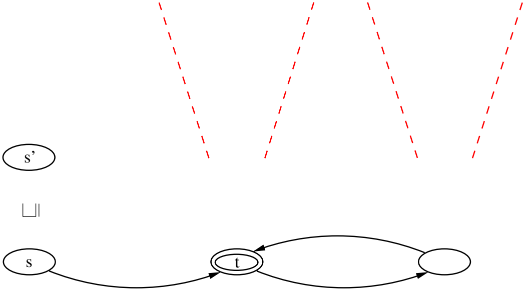
LAARMAN, OLESEN, DALSGAARD, LARSEN, VAN DE POL [CAV13]

Lemma: If s has an accepting cycle then any $s' \sqsupseteq s$ has it as well

Analysis of accepting cycles/spirals with subsumption

LAARMAN, OLESEN, DALSGAARD, LARSEN, VAN DE POL [CAV13]

Lemma: If s has an accepting cycle then any $s' \sqsupseteq s$ has it as well



Preservation of accepting cycles — Proof Sketch

s'

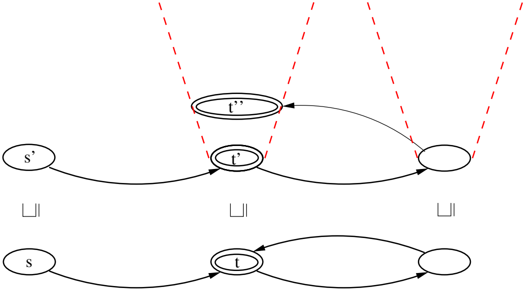
$\sqsupseteq$

$s \quad \rightarrow^* \quad t \quad \rightarrow^+ \quad t$

Analysis of accepting cycles/spirals with subsumption

Laarman, Olesen, Dalsgaard, Larsen, van de Pol [cav13]

Lemma: If s has an accepting cycle then any $s' \sqsupseteq s$ has it as well



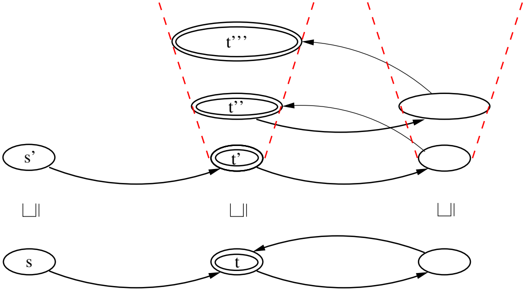
Preservation of accepting cycles — Proof Sketch

$$
\begin{array}{ccccc}
s' & \rightarrow^* & t' & \rightarrow^+ & t'' \\
\sqsupseteq & & \sqsupseteq & & \sqsupseteq \\
s & \rightarrow^* & t & \rightarrow^+ & t
\end{array}
$$

Analysis of accepting cycles/spirals with subsumption

LAARMAN, OLESEN, DALSGAARD, LARSEN, VAN DE POL [CAV13]

Lemma: If s has an accepting cycle then any $s' \sqsupseteq s$ has it as well



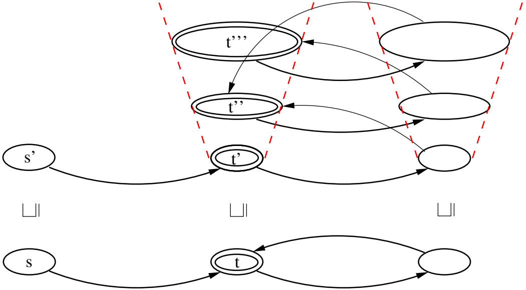
Preservation of accepting cycles — Proof Sketch

s'	$\rightarrow^*$	t'	$\rightarrow^+$	t''	$\rightarrow^+$	$\cdots\cdots$	$\rightarrow^+$	t'''
$\sqsupseteq$		$\sqsupseteq$		$\sqsupseteq$				$\sqsupseteq$
s	$\rightarrow^*$	t	$\rightarrow^+$	t	$\rightarrow^+$	$\cdots\cdots$	$\rightarrow^+$	t

Analysis of accepting cycles/spirals with subsumption

LAARMAN, OLESEN, DALSGAARD, LARSEN, VAN DE POL [CAV13]

Lemma: If s has an accepting cycle then any $s' \sqsupseteq s$ has it as well



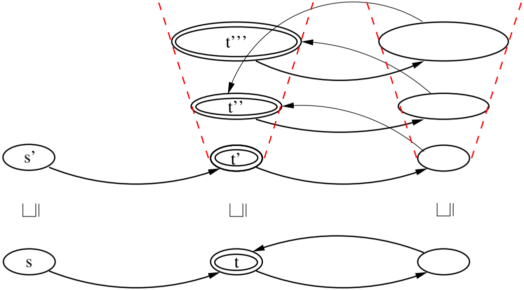
Preservation of accepting cycles — Proof Sketch

s'	$\rightarrow^*$	t'	$\rightarrow^+$	t''	$\rightarrow^+$	$\cdots x \cdots$	$\rightarrow^+$	t'''	$\rightarrow^+$	x
$\sqcup\!\!\mid$		$\sqcup\!\!\mid$		$\sqcup\!\!\mid$				$\sqcup\!\!\mid$		$\sqcup\!\!\mid$
s	$\rightarrow^*$	t	$\rightarrow^+$	t	$\rightarrow^+$	$\cdots\cdots$	$\rightarrow^+$	t	$\rightarrow^+$	t

Analysis of accepting cycles/spirals with subsumption

LAARMAN, OLESEN, DALSGAARD, LARSEN, VAN DE POL [CAV13]

Lemma: If s has an accepting cycle then any $s' \sqsupseteq s$ has it as well



Preservation of accepting cycles — Proof Sketch

s'	$\rightarrow^*$	t'	$\rightarrow^+$	t''	$\rightarrow^+$	$\cdots x \cdots$	$\rightarrow^+$	t'''	$\rightarrow^+$	x
$\sqsupseteq$		$\sqsupseteq$		$\sqsupseteq$				$\sqsupseteq$		$\sqsupseteq$
s	$\rightarrow^*$	t	$\rightarrow^+$	t	$\rightarrow^+$	$\cdots\cdots$	$\rightarrow^+$	t	$\rightarrow^+$	t

Lemma: If t' has an accepting spiral then t' has an accepting cycle

Results with Parallel Timed Reachabilty / LTL

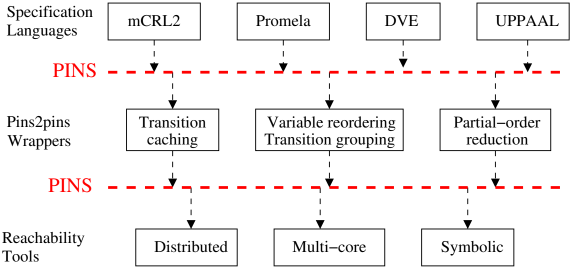
Laarman, Olesen, Dalsgaard, Larsen, van de Pol [cav13][formats2012]

- Add full LTL to timed automata
- Runtimes 60x faster than UPPAAL on 48 cores
- Up to 70x reductions due to subsumption
- Tree compression for large discrete states

LTSmin

LTSMIN BLOM, VAN DE POL, WEBER [CAV09]

http://fmt.cs.utwente.nl/tools/ltsmin/ (open source)



Other work

- Guard-based POR . . . . . PATER, LAARMAN, VAN DE POL [SPIN13]
- PROMELA formalism . . . . VAN DER BERG ET LAARMAN [PDMC12]
- LTSMIN tool . . . . LAARMAN, WEBER, VAN DE POL [NFM11]

Contributions

Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly	publications
Plain	Reachability	✓	✓	✓	✓	[FMCAD10][SPIN11][MEMICS11]
	LTL	✓	✓	1/2	✓	[ATVA11][PDMC11][ATVA12]
	.... Livelocks	✓	✓	✓	✓	[SPIN13][NFM13]
Timed	Reachability	✓	✓	–	✓	[FORMATS12]
	LTL	✓	✓	–	✓	[CAV13]

Contributions

Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly	Symbolic	publications
Plain	Reachability	✓	✓	✓	✓	✓	[FMCAD10][SPIN11][MEMICS11]
	LTL	✓	✓	1/2	✓	?	[ATVA11][PDMC11][ATVA12]
	.... Livelocks	✓	✓	✓	✓	?	[SPIN13][NFM13]
Timed	Reachability	✓	✓	–	✓	?	[FORMATS12]
	LTL	✓	✓	–	✓	?	[CAV13]

Other work

- Multi-core BDDs .... van Dijk, Laarman, van de Pol [PDMC12]

Contributions

Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly	Symbolic	publications
Plain	Reachability	✓	✓	✓	✓	✓	[FMCAD10][SPIN11][MEMICS11]
	LTL	✓	✓	1/2	✓	?	[ATVA11][PDMC11][ATVA12]
	.... Livelocks	✓	✓	✓	✓	?	[SPIN13][NFM13]
Timed	Reachability	✓	✓	–	✓	?	[FORMATS12]
	LTL	✓	✓	–	✓	?	[CAV13]

Other work

- Multi-core BDDs .... van Dijk, Laarman, van de Pol [PDMC12]
- One-Way-Catch-Them Young (LTL) .... [Barnat,Brim,Ročkai'01]
- Topological sort proviso (POR) .... [Barnat,Brim,Ročkai'10]
- CTL .... [Saad et al'12]

Future work

Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly
Plain	Reachability	✓	✓	✓	✓
	LTL	✓	✓	1/2	✓
Timed	Reachability	✓	✓	–	✓
	LTL	✓	✓	–	✓

Future work

Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly
Plain	Reachability	✓	✓	✓	✓
	LTL	✓	✓	1/2	✓
	CTL	?	?	?	?
Timed	Reachability	✓	✓	–	✓
	LTL	✓	✓	–	✓
	CTL	?	?	?	?

Future work

Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly	Symbolic	Distributed
Plain	Reachability	✓	✓	✓	✓	✓	?
	LTL	✓	✓	1/2	✓	?	?
	CTL	?	?	?	?	?	?
Timed	Reachability	✓	✓	–	✓	?	?
	LTL	✓	✓	–	✓	?	?
	CTL	?	?	?	?	?	?

Future work

Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly	Symbolic	Distributed
Plain	Reachability	✓	✓	✓	✓	✓	?
	LTL	✓	✓	1/2	✓	?	?
	CTL	?	?	?	?	?	?
Timed	Reachability	✓	✓	–	✓	?	?
	LTL	✓	✓	–	✓	?	?
	CTL	?	?	?	?	?	?
Stoch.	Reachability	?	?	?	?	?	?
	LTL	?	?	?	?	?	?
	CTL	?	?	?	?	?	?

Future work

Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly	Symbolic	Distributed
Plain	Reachability	✓	✓	✓	✓	✓	?
	LTL	✓	✓	✓	✓	?	?
Stoch.	Reachability	?	?	?	?	?	?
	LTL	?	?	?	?	?	?
	CTL	?	?	?	?	?	?

Other questions

- Can our parallel DFS-based solutions be generalized?
  - (Bottom-)SCC detection
  - Emptiness of {Tree, Rabin, Streett} automata, etc.
  - What search-order property is preserved?

Multi-core Nested Depth-First Search (Principle)

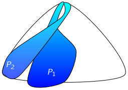
[ATVA11][PDMC11][ATVA12]

code for worker p:

procedure DFSblue(s,p)
 s.cyan[p] := true
 for all s′ **in** shuffle(post(s)) **do**
  if ¬s′.blue∧¬t.cyan[p] **then**
   DFSblue(s′,p)
 if accepting(s) **then**
  DFSred(s,p)
 s.blue := true
 s.cyan[p] := false
procedure DFSred(s,p)
 s.red[p] := true
 for all s′∈post(s) **do**
  if t.cyan[p] **then** ExitCycle
  if ¬t.red[p] **then** DFSred(s′,p)

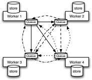
Multi-core NDFS

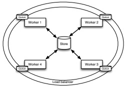
- States in shared tree/table
- Independent forward search
- Share blue color
- repair DFS order (not shown)



Static partitioning or shared hash table

[FMCAD10]





Static partitioning

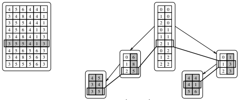
X On-the-fly (BFS)
± Scalability (communication on queues)

Shared hash table

✓ On-the-fly: (pseudo) DFS & BFS
? Scalability

Recursive indexing

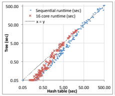
[HOLZMANN 97][BLOM ET AL. 08]

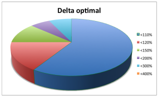


✓ Combinatorial $\Longrightarrow$ balanced tree ($N + 2\sqrt{N} + 4\sqrt[4]{(N)} \cdots \approx N$)
Compresses states of lenght K to almost 2!
X Hard to parallelize (flatliners)

Experiments from 2011 [BEEM database]

[SPIN11]





$\langle 3,5,5,4,1,3\rangle \longrightarrow \langle 3,5,9,4,1,3\rangle \xrightarrow{\frac{K-1}{K}} \langle 3,5,9,3,2,3\rangle \longrightarrow$

Information theoretical optimum?

- View states as K-periodic stream of 2^{32}-valued variables
- Information entropy per state: $\log_2(2^{32}) + \log_2(K)$ bits $\approx 1 + \epsilon$ integer
- Halve root table with *compact hash table* [MEMICS11]

Contributions

Formalism	Property	Explicit state	+ Compression	+ POR	+ On-the-fly	Symbolic	publications
Plain	Reachability	✓	✓	1/2	✓	✓	[FMCAD10][SPIN11][MEMICS11]
	LTL	✓	✓	1/2	✓	?	[ATVA11][PDMC11][ATVA12]
	.... Livelocks	✓	✓	✓	✓	?	[SPIN13][NFM13]
Timed	Reachability	✓	✓	–	✓	?	[FORMATS12]
	LTL	✓	✓	–	✓	?	[CAV13]

Related work

- Multi-core BDDs .... [PDMC13(1)]
- One-Way-Catch-Them Young (LTL) .... [BARNAT,BRIM,ROČKAI'01]
- Topological sort proviso (POR) .... [BARNAT,BRIM,ROČKAI'10]
- CTL .... [SAAD ET AL 12]