Multi-Core Partial-Order Reduction for LTL Model Checking

Alfons Laarman
alfons@laarman.com

joint work with **Anton Wijs** (Eindhoven University of Technology)

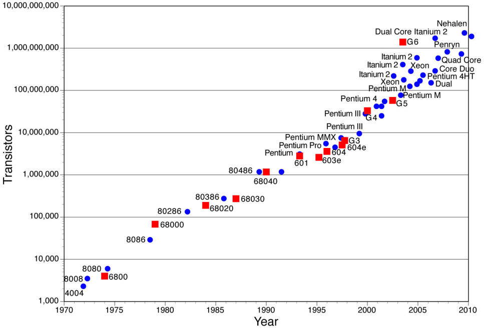
Formal Methods in Systems Engineering
Vienna University of Technology

May 5, 2015

Goals

Combine:

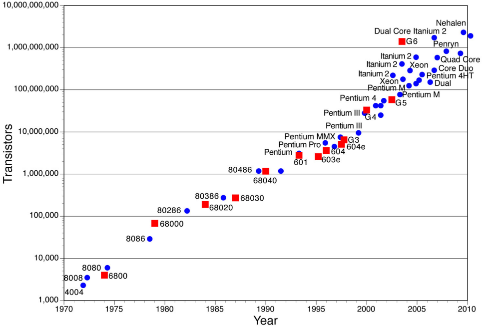
- Parallel model checking (exponential gains)
- Partial-Order Reduction (POR) (exponential gains)



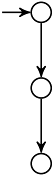
Goals

Combine:

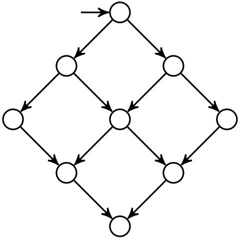
- Parallel model checking (exponential gains)
- Partial-Order Reduction (POR) (exponential gains)



P_i



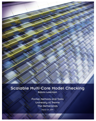
$P_1 \| P_2$



Scalable Multi-Core Model Checking

Research questions

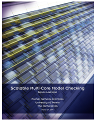
- Can model checking scale on modern multi-cores?
- Retain compatibility with different optimizations?



Scalable Multi-Core Model Checking

Research questions

- Can model checking scale on modern multi-cores?
- Retain compatibility with different optimizations?

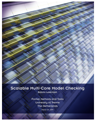


1. On-the-fly
2. Partial-order reduction
3. State compression
4. OR Symbolic with BDDs
   [van Dijk, L, van de Pol, 2013]

Scalable Multi-Core Model Checking

Research questions

- Can model checking scale on modern multi-cores?
- Retain compatibility with different optimizations?



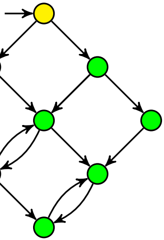
1. On-the-fly
2. Partial-order reduction
3. State compression
4. OR Symbolic with BDDs [van Dijk, L, van de Pol, 2013]

Formalism	Property	Explicit state	+ Compression	+ On-the-fly	+ POR	Symbolic
Plain	Reachability	✓	✓	✓	✓	✓
	Liveness	✓	✓	✓	?	✓
Timed	Reachability	✓	✓	✓	✓	✓
	Liveness	✓	✓	✓	?	✓

1. Shared hash table approach (as opposed to distributed algorithms)
2. Lockless data structures
3. Parallel algorithms (Multi-Core Nested-DFS)

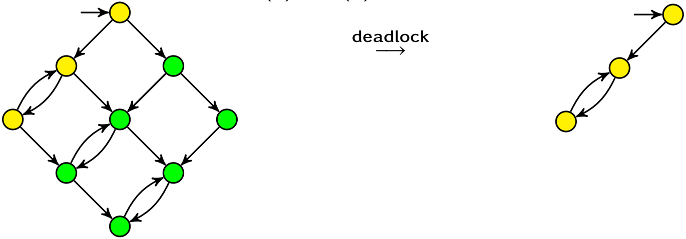
Partial-Order Reduction for LTL

State-space graph: $\mathcal{G} = (\mathcal{S}, T, s_0, AP)$
On-the-fly exploration: $en : \mathcal{S} \rightarrow \mathcal{S}$



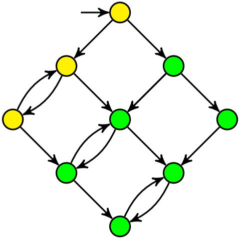
Partial-Order Reduction for LTL

State-space graph: $\mathcal{G} = (\mathcal{S}, T, s_0, AP)$
On-the-fly exploration: $en : \mathcal{S} \to \mathcal{S}$
Reduce successor function: $por(s) \subseteq en(s)$.

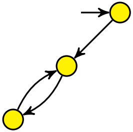


Partial-Order Reduction for LTL

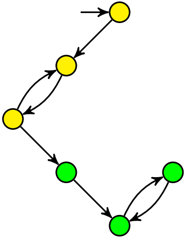
State-space graph: $\mathcal{G} = (\mathcal{S}, T, s_0, AP)$
On-the-fly exploration: $en : \mathcal{S} \to \mathcal{S}$
Reduce successor function: $por(s) \subseteq en(s)$.



deadlock $\longrightarrow$

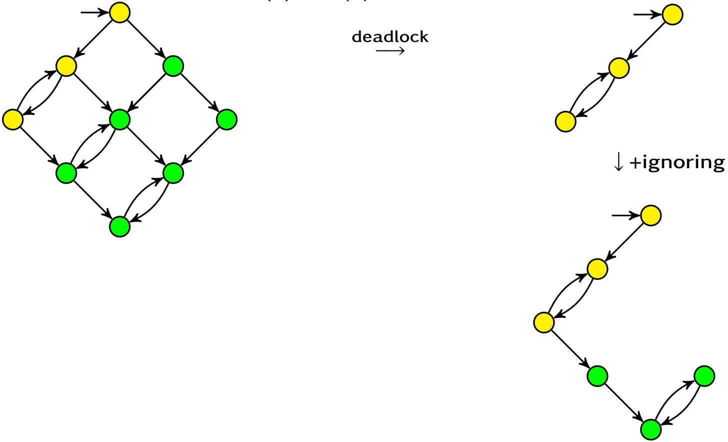


$\downarrow$ +ignoring



Partial-Order Reduction for LTL

State-space graph: $\mathcal{G} = (\mathcal{S}, T, s_0, AP)$
On-the-fly exploration: $en : \mathcal{S} \to \mathcal{S}$
Reduce successor function: $por(s) \subseteq en(s)$.



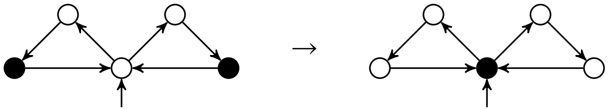
Smaller reduced set $por()$ leads to smaller state space.

DFS Stack Proviso

```
procedure DFS(s)
    for all s′ in por(s) do
        if s′ is not on stack and not visited then
            DFS(s′)
    if successor of s is on the stack then
        explore s fully ( por(s) := en(s) )
    mark s visited
```

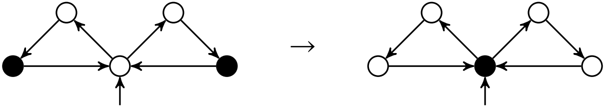
DFS Stack Proviso

```
procedure DFS(s)
    for all s′ in por(s) do
        if s′ is not on stack and not visited then
            DFS(s′)
    if successor of s is on the stack then
        explore s fully ( por(s) := en(s) )
    mark s visited
```



DFS Stack Proviso

procedure DFS(s)
    for all s′ **in** por(s) **do**
        if s′ is not on stack and not visited **then**
            DFS(s′)
    if successor of s is on the stack **then**
        explore s fully (por(s) := en(s))
    mark s visited

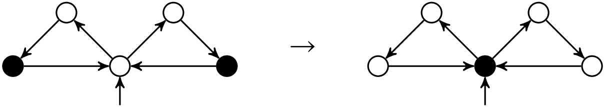


Why not anything else?

- (Minimal) feedback vertex set (FVS) → NP-complete
- Stack proviso is the best we can do on-the-fly and in linear time

DFS Stack Proviso

```
procedure DFS(s)
  for all s′ in por(s) do
    if s′ is not on stack and not visited then
      DFS(s′)
  if successor of s is on the stack then
    explore s fully ( por(s) := en(s) )
  mark s visited
```



Why not anything else?

- (Minimal) feedback vertex set (FVS) → NP-complete
- Stack proviso is the best we can do on-the-fly and in linear time

DFS is P-complete ⇒ inherently sequential (assuming P ≠ NC)

Related Work (Parallel LTL + POR)

Algorithm/Proviso	Reduction	Scalability
NDFS/Stack	++	
TwoPhase [Gopalakrishnan et al.]	+-	??
Topological sort [Barnat et al.]	+-	+
Sticky transitions [Peled et al]	-	+

Related Work (Parallel LTL + POR)

Algorithm/Proviso	Reduction	Scalability
NDFS/Stack	++	
TwoPhase [Gopalakrishnan et al.]	+-	??
Topological sort [Barnat et al.]	+-	+
Sticky transitions [Peled et al]	-	+
MC-NDFS/n/a	n/a	++

Challenge: do as good as DFS stack proviso in the parallel setting

Nested Depth-First Search for LTL

[Courcoubetis'93]

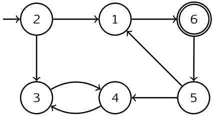
Büchi graph: $\mathcal{G} = (\mathcal{S}, \mathcal{F}, T, s_0, AP)$
On-the-fly exploration: $en : \mathcal{S} \rightarrow \mathcal{S}$
[Vardi et al, 1996]

Nested Depth-First Search for LTL

[Courcoubetis'93]

Büchi graph: $\mathcal{G} = (\mathcal{S}, \mathcal{F}, T, s_0, AP)$
On-the-fly exploration: $en : \mathcal{S} \rightarrow \mathcal{S}$
[Vardi et al, 1996]

Accepting cycle detection in Büchi automaton ($6 \in \mathcal{F}$):

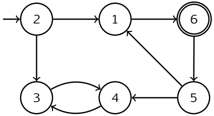


Nested Depth-First Search for LTL

[COURCOUBETIS'93]

Büchi graph: $\mathcal{G} = (\mathcal{S}, \mathcal{F}, T, s_0, AP)$
On-the-fly exploration: $en : \mathcal{S} \rightarrow \mathcal{S}$
[Vardi et al, 1996]

Accepting cycle detection in Büchi automaton ($6 \in \mathcal{F}$):



accepting-cycles($\mathcal{G}$) $\subseteq$ cycles($\mathcal{G}$)

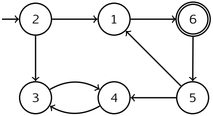
Nested Depth-First Search for LTL

[Courcoubetis'93]

procedure DFSblue(s)
  s.cyan := true
  for all s′ **in** en(s) **do**
    if ¬s′.blue∧¬s′.cyan **then**
      DFSblue(s′)
  if $s \in \mathcal{F}$ **then**
    DFSred(s)
  s.blue := true
  s.cyan := false
procedure DFSred(s)
  s.red := true
  for all s′∈en(s) **do**
    if s′.cyan **then** ExitCycle
    if ¬s′.red **then** DFSred(s′)

Büchi graph: $\mathcal{G} = (\mathcal{S}, \mathcal{F}, T, s_0, AP)$
On-the-fly exploration: $en : \mathcal{S} \rightarrow \mathcal{S}$
[Vardi et al, 1996]

Accepting cycle detection in Büchi automaton ($6 \in \mathcal{F}$):



accepting-cycles($\mathcal{G}$) ⊆ cycles($\mathcal{G}$)

Nested DFS (NDFS)

- Linear time

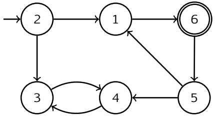
Nested Depth-First Search for LTL

[Courcoubetis'93]

procedure DFSblue(s)
  s.cyan := true
  for all s′ **in** en(s) **do**
    if ¬s′.blue∧¬s′.cyan **then**
      DFSblue(s′)
  if $s \in \mathcal{F}$ **then**
    DFSred(s)
  s.blue := true
  s.cyan := false
procedure DFSred(s)
  s.red := true
  for all s′∈en(s) **do**
    if s′.cyan **then** ExitCycle
    if ¬s′.red **then** DFSred(s′)

Büchi graph: $\mathcal{G} = (\mathcal{S}, \mathcal{F}, T, s_0, AP)$
On-the-fly exploration: $en : \mathcal{S} \to \mathcal{S}$
[Vardi et al, 1996]

Accepting cycle detection in Büchi automaton ($6 \in \mathcal{F}$):



accepting-cycles($\mathcal{G}$) ⊆ cycles($\mathcal{G}$)

Nested DFS (NDFS)

- Linear time
- DFS itself is likely not parallelizable
  - DFS order is P-complete

Swarm Nested Depth-First Search

[Holzmann, 2010]

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

Multi-core Nested Depth-First Search

[ATVA11], [PDMC11], [ATVA12]

```
code for worker p:
procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ in shuffle(en(s)) do
        if ¬s′.blue∧¬t.cyan[p] then
            DFSblue(s′, p)
    if s ∈ F then
        R := ∅
        DFSred(s, p)

        mark all in R red
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s, p)
    R := R ∪ {s}
    for all s′ ∈ shuffle(en(s)) do
        if s′.cyan[p] then ExitCycle
        if s′ ∉ R ∧ ¬s.red then DFSred(s′, p)
```

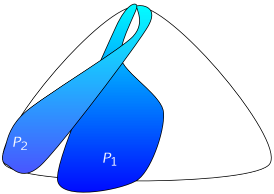
Multi-core Nested Depth-First Search

[ATVA11], [PDMC11], [ATVA12]

code for worker p:

```
procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ in shuffle(en(s)) do
        if ¬s′.blue∧¬t.cyan[p] then
            DFSblue(s′, p)
    if s ∈ F then
        R := ∅
        DFSred(s, p)

        mark all in R red
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s, p)
    R := R ∪ {s}
    for all s′ ∈ shuffle(en(s)) do
        if s′.cyan[p] then ExitCycle
        if s′ ∉ R ∧ ¬s.red then DFSred(s′, p)
```

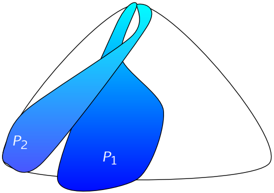


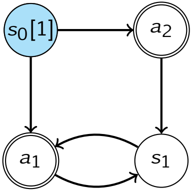
Multi-core Nested Depth-First Search

[ATVA11], [PDMC11], [ATVA12]

```
code for worker p:
procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ in shuffle(en(s)) do
        if ¬s′.blue∧¬t.cyan[p] then
            DFSblue(s′, p)
    if s ∈ F then
        R := ∅
        DFSred(s, p)

        mark all in R red
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s, p)
    R := R ∪ {s}
    for all s′ ∈ shuffle(en(s)) do
        if s′.cyan[p] then ExitCycle
        if s′ ∉ R ∧ ¬s.red then DFSred(s′, p)
```





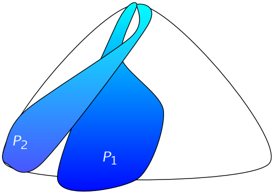
Multi-core Nested Depth-First Search

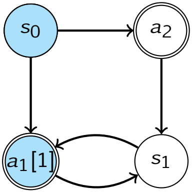
[ATVA11], [PDMC11], [ATVA12]

code for worker p:

procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ **in** shuffle(en(s)) **do**
        if ¬s′.blue∧¬t.cyan[p] **then**
            DFSblue(s′, p)
    if $s \in \mathcal{F}$ **then**
        $R := \emptyset$
        DFSred(s, p)

        mark all in R red
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s, p)
    $R := R \cup \{s\}$
    for all s′ ∈ shuffle(en(s)) **do**
        if s′.cyan[p] **then** ExitCycle
        if $s' \notin R \wedge \neg s.red$ **then** DFSred(s′, p)



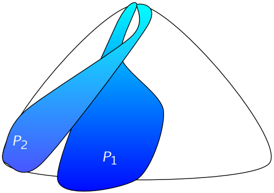


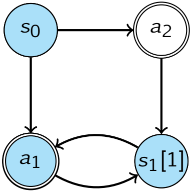
Multi-core Nested Depth-First Search

[ATVA11], [PDMC11], [ATVA12]

```
code for worker p:
procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ in shuffle(en(s)) do
        if ¬s′.blue∧¬t.cyan[p] then
            DFSblue(s′, p)
    if s ∈ ℱ then
        R := ∅
        DFSred(s, p)

        mark all in R red
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s, p)
    R := R ∪ {s}
    for all s′ ∈ shuffle(en(s)) do
        if s′.cyan[p] then ExitCycle
        if s′ ∉ R ∧ ¬s.red then DFSred(s′, p)
```



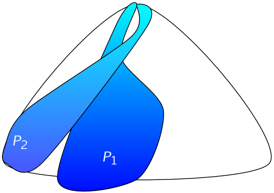


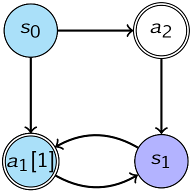
Multi-core Nested Depth-First Search

[ATVA11], [PDMC11], [ATVA12]

code for worker p:
procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ **in** shuffle(en(s)) **do**
        if ¬s′.blue∧¬t.cyan[p] **then**
            DFSblue(s′, p)
    if $s \in \mathcal{F}$ **then**
        $R := \emptyset$
        DFSred(s, p)

        mark all in R red
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s, p)
    $R := R \cup \{s\}$
    for all s′ ∈ shuffle(en(s)) **do**
        if s′.cyan[p] **then** ExitCycle
        if $s' \notin R \wedge \neg s.\mathit{red}$ **then** DFSred(s′, p)



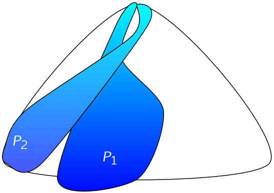


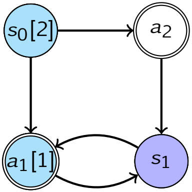
Multi-core Nested Depth-First Search

[ATVA11], [PDMC11], [ATVA12]

code for worker p:
procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ **in** shuffle(en(s)) **do**
        if ¬s′.blue∧¬t.cyan[p] **then**
            DFSblue(s′, p)
    if $s \in \mathcal{F}$ **then**
        $R := \emptyset$
        DFSred(s, p)

        mark all in R red
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s, p)
    $R := R \cup \{s\}$
    for all s′ ∈ shuffle(en(s)) **do**
        if s′.cyan[p] **then** ExitCycle
        if $s' \notin R \wedge \neg s.red$ **then** DFSred(s′, p)



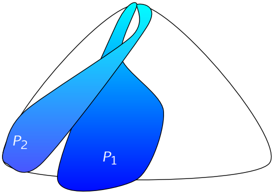


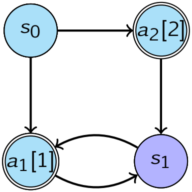
Multi-core Nested Depth-First Search

[ATVA11], [PDMC11], [ATVA12]

```
code for worker p:
procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ in shuffle(en(s)) do
        if ¬s′.blue∧¬t.cyan[p] then
            DFSblue(s′, p)
    if s ∈ 𝓕 then
        R := ∅
        DFSred(s, p)

        mark all in R red
    s.blue := true
    s.cyan[p] := false
procedure DFSred(s, p)
    R := R ∪ {s}
    for all s′ ∈ shuffle(en(s)) do
        if s′.cyan[p] then ExitCycle
        if s′ ∉ R ∧ ¬s.red then DFSred(s′, p)
```



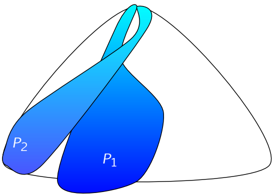


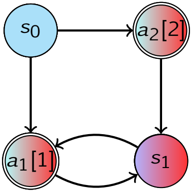
Multi-core Nested Depth-First Search

[ATVA11], [PDMC11], [ATVA12]

code for worker p:
procedure DFSblue(s, p)
  s.cyan[p] := true
  for all s′ **in** shuffle(en(s)) **do**
    if ¬s′.blue∧¬t.cyan[p] **then**
      DFSblue(s′, p)
  if $s \in \mathcal{F}$ **then**
    $R := \emptyset$
    DFSred(s, p)

    mark all in R red
  s.blue := true
  s.cyan[p] := false
procedure DFSred(s, p)
  $R := R \cup \{s\}$
  for all s′ ∈ shuffle(en(s)) **do**
    if s′.cyan[p] **then** ExitCycle
    if $s' \notin R \wedge \neg s.red$ **then** DFSred(s′, p)

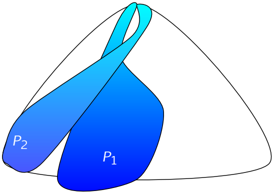


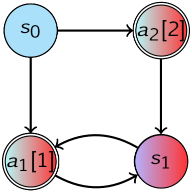


Multi-core Nested Depth-First Search

[ATVA11], [PDMC11], [ATVA12]

code for worker p:
procedure DFSblue(s, p)
  s.cyan[p] := true
  for all s′ **in** shuffle(en(s)) **do**
    if ¬s′.blue∧¬t.cyan[p] **then**
      DFSblue(s′, p)
  if $s \in \mathcal{F}$ **then**
    $R := \emptyset$
    DFSred(s, p)
    await all accepting in $R \setminus \{s\}$ **are red**
    mark all in R red
  s.blue := true
  s.cyan[p] := false
procedure DFSred(s, p)
  $R := R \cup \{s\}$
  for all s′ ∈ shuffle(en(s)) **do**
    if s′.cyan[p] **then** ExitCycle
    if $s' \notin R \wedge \neg s.red$ **then** DFSred(s′, p)





Multi-core Nested Depth-First Search

[ATVA11], [PDMC11], [ATVA12]

code for worker p:

procedure DFSblue(s, p)

  s.cyan[p] := true

  for all s′ **in** shuffle(en(s)) **do**

    if ¬ ...

  if $s \in$ ...

    R := ...

    DFSred(s, p)

    await all accepting in $R \setminus \{s\}$ **are red**

    mark all in R red

  s.blue := true

  s.cyan[p] := false

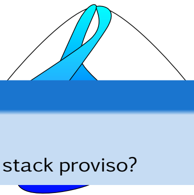
procedure DFSred(s, p)

  $R := R \cup \{s\}$

  for all s′ ∈ shuffle(en(s)) **do**

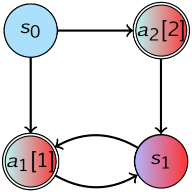
    if s′.cyan[p] **then** ExitCycle

    if $s' \notin R \wedge \neg s.red$ **then** DFSred(s′, p)



Conclusions

- MC-NDFS scales in practice and uses DFS
- Does it preserve enough order to implement stack proviso?



Stack Proviso in Parallel

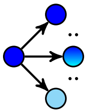
```
procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ in shuffle(por(s)) do
        if ¬s′.blue ∧ ¬t.cyan[p] then
            DFSblue(s′, p)
    if s ∈ 𝓕 then
        R := ∅
        DFSred(s, p)
        await all accepting in R \ {s} are red
        mark all in R red
    if ∃s′ ∈ por(s) : s′.cyan then
        explore s fully with DFSblue
    s.blue := true
    s.cyan[p] := false
```

Stack Proviso in Parallel

```
procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ in shuffle(por(s)) do
        if ¬s′.blue ∧ ¬t.cyan[p] then
            DFSblue(s′, p)
    if s ∈ F then
        R := ∅
        DFSred(s, p)
        await all accepting in R \ {s} are red
        mark all in R red
    if ∃s′ ∈ por(s) : s′.cyan then
        explore s fully with DFSblue
    s.blue := true
    s.cyan[p] := false
```

Soundness trivial
Completeness

$Blue \subseteq \Box(Blue \cup \bigcup_p Cyan_p)$



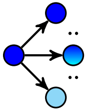
Stack Proviso in Parallel

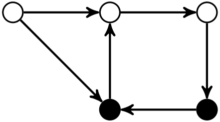
```
procedure DFSblue(s, p)
    s.cyan[p] := true
    for all s′ in shuffle(por(s)) do
        if ¬s′.blue ∧ ¬t.cyan[p] then
            DFSblue(s′, p)
    if s ∈ 𝓕 then
        R := ∅
        DFSred(s, p)
        await all accepting in R \ {s} are red
        mark all in R red
    if ∃s′ ∈ por(s) : s′.cyan then
        explore s fully with DFSblue
    s.blue := true
    s.cyan[p] := false
```

Soundness trivial

Completeness

$Blue \subseteq \Box(Blue \cup \bigcup_p Cyan_p)$



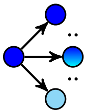


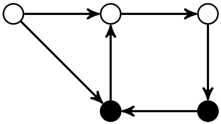
Stack Proviso in Parallel

procedure DFSblue(s, p)
  s.cyan[p] := true
  for all s′ **in** shuffle(por(s)) **do**
    if ¬s′.blue∧¬t.cyan[p] **then**
      DFSblue(s′, p)
  if $s \in \mathcal{F}$ **then**
    $R := \emptyset$
    DFSred(s, p)
    await all accepting in $R \setminus \{s\}$ **are red**
    mark all in R red
  if ∃s′ ∈por(s): s′.cyan **then**
    explore s fully with DFSblue
  s.blue := true
  s.cyan[p] := false
procedure DFSred(s, p)
  $R := R \cup \{s\}$
  for all s′∈shuffle(por(s)) **do**
    if s′.cyan[p] **then** ExitCycle
    if $s' \notin R \wedge \neg s.red$ **then**DFSred(s′, p)
  if successor of s is on DFSred_p **stack then**
    explore s fully with DFSred

Soundness trivial
Completeness

$Blue \subseteq \Box(Blue \cup \bigcup_p Cyan_p)$





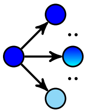
Re-visiting problem
[Holzmann et al., 1996 –
On nested depth-first search]

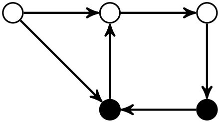
Stack Proviso in Parallel

procedure DFSblue(s, p)
  s.cyan[p] := true
  for all s′ **in** shuffle(por(s)) **do**
    if ¬s′.blue∧¬t.cyan[p] **then**
      DFSblue(s′, p)
  if $s \in \mathcal{F}$ **then**
    $R := \emptyset$
    DFSred(s, p)
    await all accepting in $R \setminus \{s\}$ **are red**
    mark all in R red
  if ∃s′ ∈por(s): s′.cyan **then**
    explore s fully with DFSblue
  s.blue := true
  s.cyan[p] := false
procedure DFSred(s, p)
  $R := R \cup \{s\}$
  for all s′∈shuffle(por(s)) **do**
    if s′.cyan[p] **then** ExitCycle
    if $s' \notin R \wedge \neg s.red$ **then**DFSred(s′, p)
  if successor of s is on DFSred$_p$ stack **then**
    explore s fully with DFSred

Soundness trivial
Completeness

$Blue \subseteq \Box(Blue \cup \bigcup_p Cyan_p)$



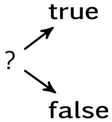


Re-visiting problem
[Holzmann et al., 1996 –
On nested depth-first search]

No termination!

The Parallel Cycle Proviso

Add a state proviso flag:

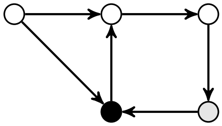



```
procedure dfsBlue(s, p)
    ...
    prov := successor of s is on the local DFSblue stack
    compare_and_swap (s.proviso, ?, prov)
    if s.proviso then
        explore s fully with dfsRed
    ...
procedure dfsRed(s, p)
    ...
    prov := successor of s is on the local DFSred stack
    compare_and_swap (s.proviso, ?, prov)
    if s.proviso then
        explore s fully with dfsRed
    ...
```

The Parallel Cycle Proviso

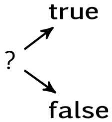
Add a state proviso flag: ? → **true** / **false**

Performance



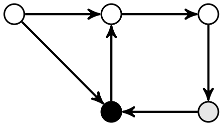
```
procedure dfsBlue(s, p)
    ...
    prov := successor of s is on the local DFSblue stack
    compare_and_swap (s.proviso, ?, prov)
    if s.proviso then
        explore s fully with dfsRed
    ...
procedure dfsRed(s, p)
    ...
    prov := successor of s is on the local DFSred stack
    compare_and_swap (s.proviso, ?, prov)
    if s.proviso then
        explore s fully with dfsRed
    ...
```

The Parallel Cycle Proviso

Add a state proviso flag: 

procedure dfsBlue(s, p)
...
prov := successor of s is on the local DFSblue stack
compare_and_swap (s.proviso, ?, prov)
if s.proviso **then**
explore s fully with dfsRed
...
procedure dfsRed(s, p)
...
prov := successor of s is on the local DFSred stack
compare_and_swap (s.proviso, ?, prov)
if s.proviso **then**
explore s fully with dfsRed
...

Performance



Correctness
Backtracked states (blue and red):
$F = \{s \mid s.proviso \neq ?\}$
$V = \{s \mid s.proviso = \textbf{false}\}$
Lemma 6. Backtracked states have a proviso set: $(B \cup R) \subseteq F$.
Lemma 8. Successors of V states are backtracked: $V \subseteq \Box(B \cup R)$.

Conclusions

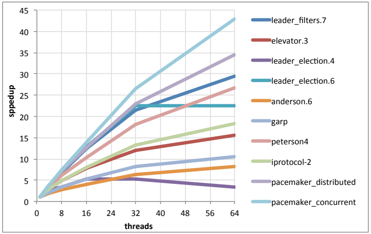
Parallel cycle proviso (% reduction)

Model	Threads 1	Threads 64
leader_filters.7	2.35	2.35
elevator.3	94.20	94.96
leader_election.4	3.02	3.02
leader_election.6	0.70	0.70
anderson.6	48.43	51.71
garp	18.69	20.79
peterson4	15.52	15.67
iprotocol-2	34.80	37.91
pacemaker_distributed	47.81	48.26
pacemaker_concurrent	45.90	46.00

Conclusions

Parallel cycle proviso (% reduction)

Model	Threads 1	Threads 64
leader_filters.7	2.35	2.35
elevator.3	94.20	94.96
leader_election.4	3.02	3.02
leader_election.6	0.70	0.70
anderson.6	48.43	51.71
garp	18.69	20.79
peterson4	15.52	15.67
iprotocol-2	34.80	37.91
pacemaker_distributed	47.81	48.26
pacemaker_concurrent	45.90	46.00

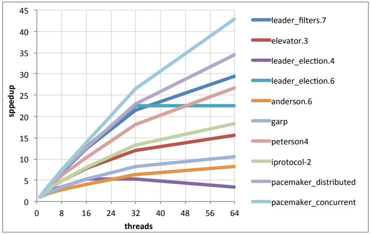


- DFS-proviso's reduction power is preserved
- Speedups maintained

Conclusions

Parallel cycle proviso (% reduction)

Model	Threads 1	Threads 64
leader_filters.7	2.35	2.35
elevator.3	94.20	94.96
leader_election.4	3.02	3.02
leader_election.6	0.70	0.70
anderson.6	48.43	51.71
garp	18.69	20.79
peterson4	15.52	15.67
iprotocol-2	34.80	37.91
pacemaker_distributed	47.81	48.26
pacemaker_concurrent	45.90	46.00

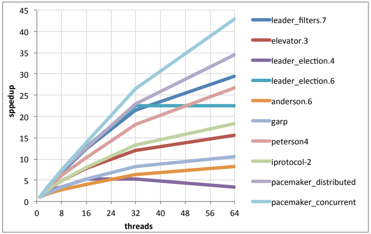


- DFS-proviso's reduction power is preserved
- Speedups maintained
- Demonstrates the strength of parallel DFS-based algortihms

Conclusions

Parallel cycle proviso (% reduction)

Model	Threads 1	Threads 64
leader_filters.7	2.35	2.35
elevator.3	94.20	94.96
leader_election.4	3.02	3.02
leader_election.6	0.70	0.70
anderson.6	48.43	51.71
garp	18.69	20.79
peterson4	15.52	15.67
iprotocol-2	34.80	37.91
pacemaker_distributed	47.81	48.26
pacemaker_concurrent	45.90	46.00



- DFS-proviso's reduction power is preserved
- Speedups maintained
- Demonstrates the strength of parallel DFS-based algortihms
- How much of the DFS order is preserved?
- On which type of graphs does CNDFS scale?