UNIVERSITY OF TWENTE.
formal methods & tools.

Improved Multi-core Nested Depth-First Search

Alfons Laarman

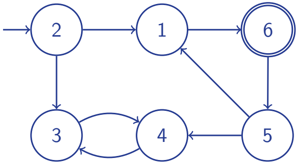
Joint with Jaco van de Pol (UTwente)
Sami Evangelista and Laure Pettruci (Paris 13 University)

Oct 5, 2012

ATVA'12, Thiruvananthapuram, India

LTL Model Checking

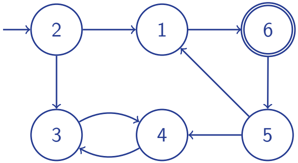
- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton [Vardi/Wolper 86]
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth-First Search (NDFS)



LTL Model Checking is Finding Accepting Cycles

LTL Model Checking

- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton [Vardi/Wolper 86]
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth-First Search (NDFS)



This talk

- Traditional solution: NDFS
- Multi-core NDFS flavors
- CNDFS and a comparison

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

Nested DFS

- Blue search
  - Visits all reachable states
  - Starts Red search on accepting states (seed) in post order

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

Nested DFS

- Blue search
  - Visits all reachable states
  - Starts Red search on accepting states (seed) in post order
- Red Search
  - Finds cycle (cyan)
  - Visits states at most once

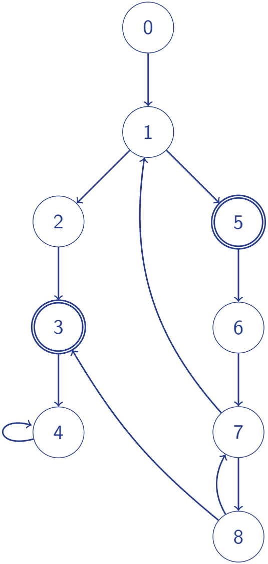
```
procedure DFSblue(s)
    s.cyan := true
    for all t ∈ post(s) do
        if ¬ t.blue ∧¬ t.cyan then
            DFSblue(t)
    if s ∈ Accepting then
        DFSred(s)
    s.blue := true
    s.cyan := false
procedure DFSred(s)
    s.red := true
    for all t ∈ post(s) do
        if t.cyan then ExitCycle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

Nested DFS

- Blue search
  - Visits all reachable states
  - Starts Red search on accepting states (seed) in post order
- Red Search
  - Finds cycle (cyan)
  - Visits states at most once
- Linear time, on-the-fly
- Blue is inherently depth-first

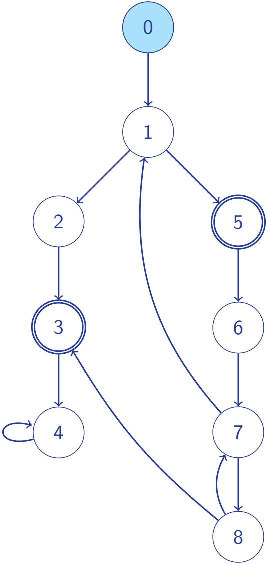
NDFS example

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



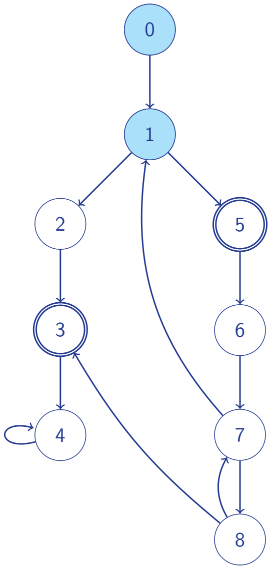
NDFS example

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



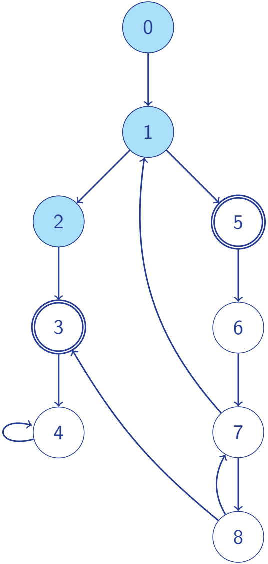
NDFS example

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



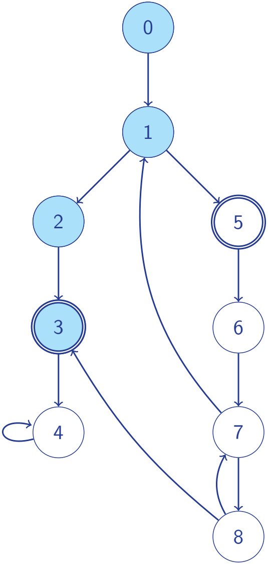
NDFS example

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



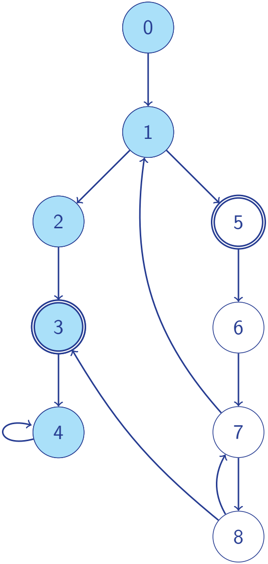
NDFS example

procedure DFSblue(s)
    s.cyan := true
    for all $t \in post(s)$ **do**
        if $\neg t.blue \wedge \neg t.cyan$ **then**
            DFSblue(t)
    if $s \in Accepting$ **then**
        DFSred(s)
    s.blue := true
    s.cyan := false
procedure DFSred(s)
    s.red := true
    for all $t \in post(s)$ **do**
        if t.cyan **then** ExitCycle
        if $\neg t.red$ **then** DFSred(t)



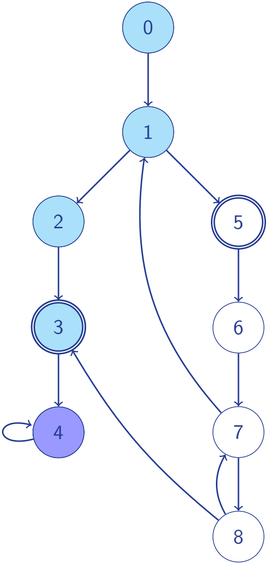
NDFS example

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



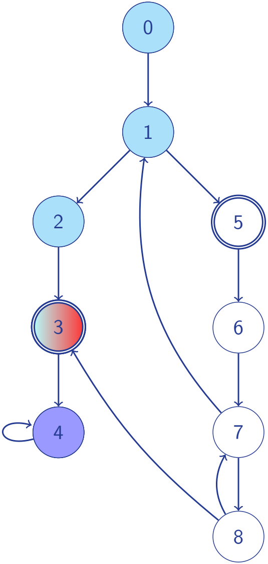
NDFS example

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



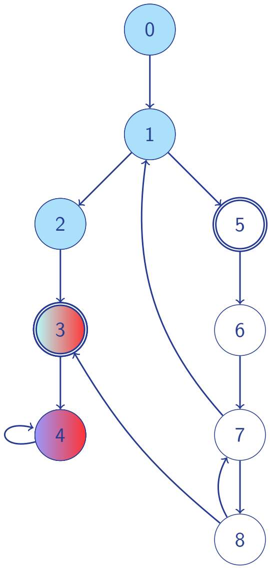
NDFS example

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



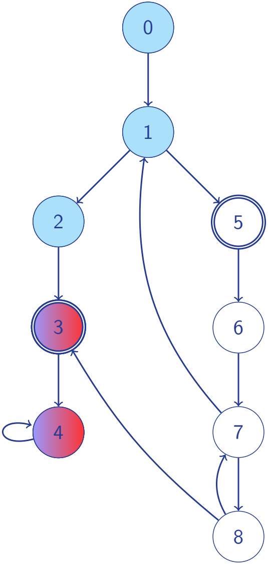
NDFS example

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



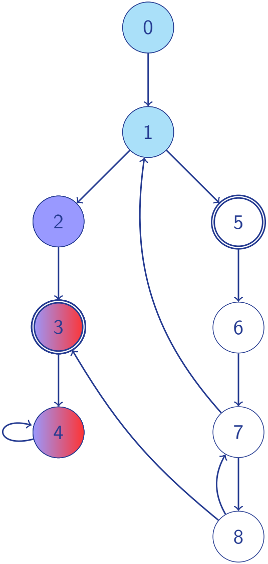
NDFS example

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



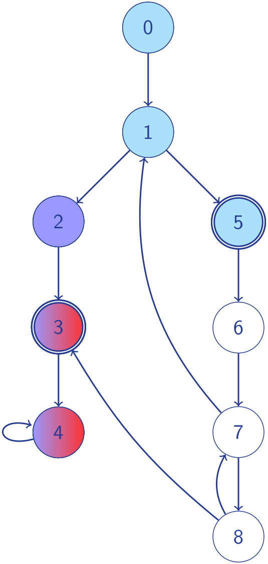
NDFS example

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



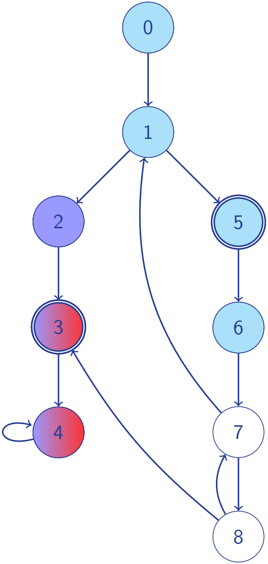
NDFS example

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



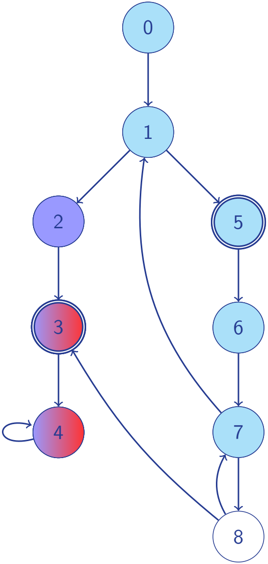
NDFS example

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



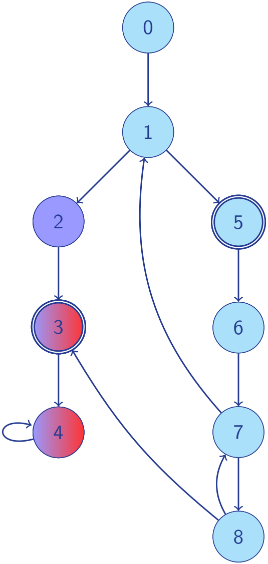
NDFS example

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



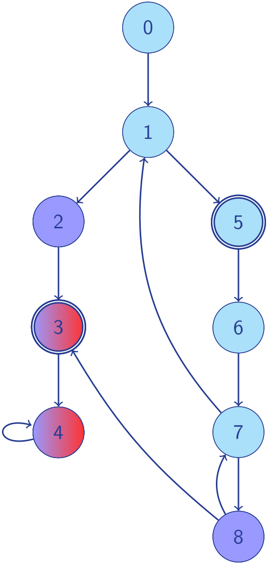
NDFS example

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



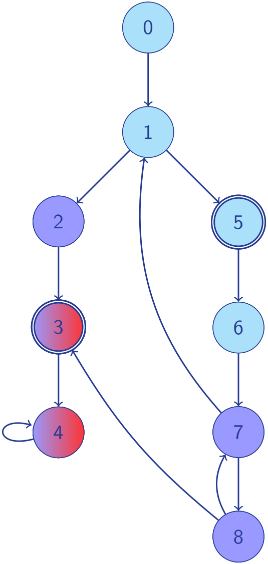
NDFS example

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



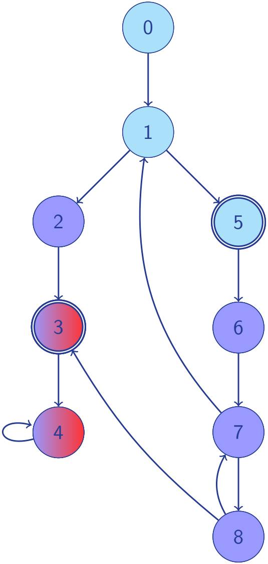
NDFS example

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



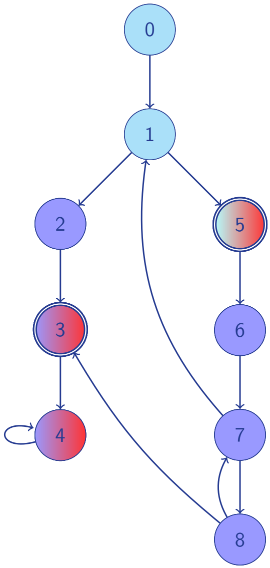
NDFS example

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



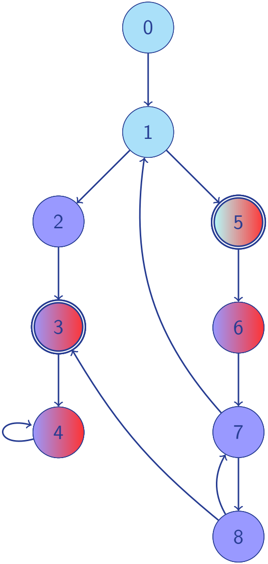
NDFS example

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



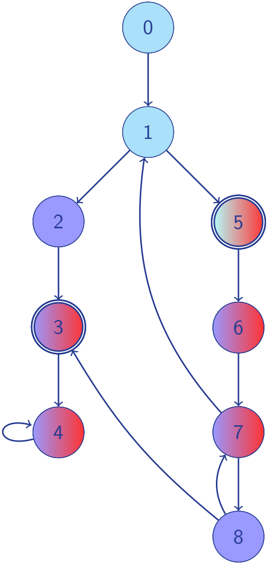
NDFS example

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



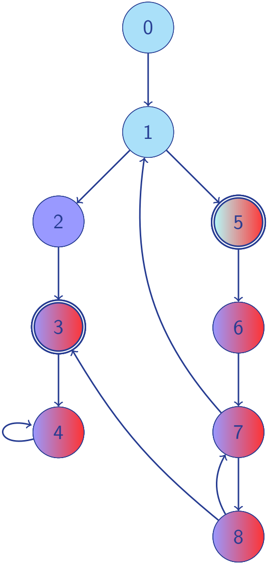
NDFS example

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



NDFS example

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



NDFS example

NDFS properties

- Time complexity: $|Blue| + |Red|$ (linear)
- Memory usage: 2 bits per state [Schwoon et al.]
- DFS-order $\implies$ Hard to parallelize

NDFS example

NDFS properties

- Time complexity: $|Blue| + |Red|$ (linear)
- Memory usage: 2 bits per state [Schwoon et al.]
- DFS-order $\implies$ Hard to parallelize
- Completeness: DFS-order $\implies$ one accepting state in each red search (safe to reuse the red color)

Swarmed and Multi-core Nested Depth-First Search

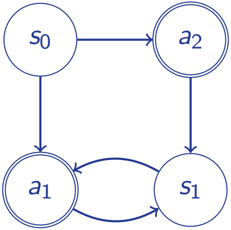
code for worker i:

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

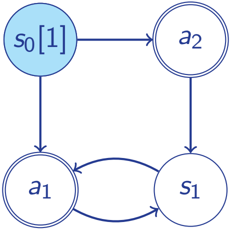
Multi-core NDFS

- Independent (swarmed) $\Longrightarrow$ no speedup for full verification
- States in shared hash table [FMCAD10]
- Share red $\Longrightarrow$ good speedup iff $|Red| = |Blue|$
- Share blue $\Longrightarrow$ lose post-order

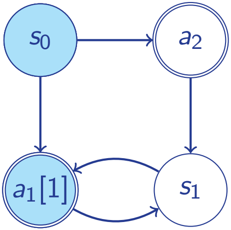
Sharing Blue Counter Example



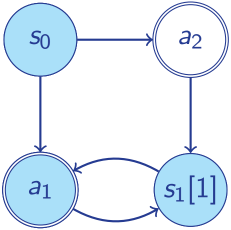
Sharing Blue Counter Example



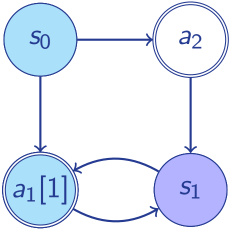
Sharing Blue Counter Example



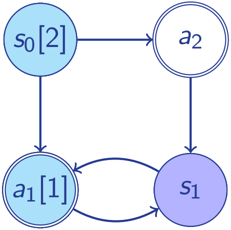
Sharing Blue Counter Example



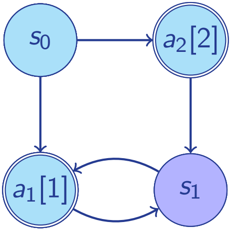
Sharing Blue Counter Example



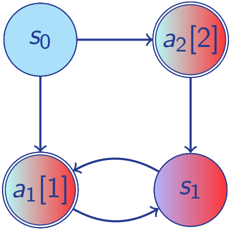
Sharing Blue Counter Example



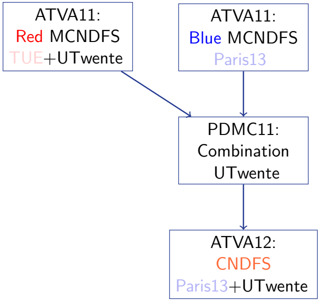
Sharing Blue Counter Example

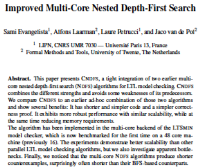


Sharing Blue Counter Example



A History of Multi-core NDFSs





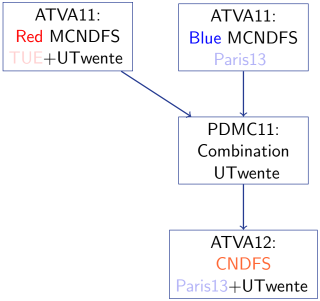
Improved Multi-Core Nested Depth-First Search

Sami Evangelista[1], Alfons Laarman[2], Laure Petrucci[1], and Jaco van de Pol[2]

[1] LIPN, CNRS UMR 7030 — Université Paris 13, France
[2] Formal Methods and Tools, University of Twente, The Netherlands

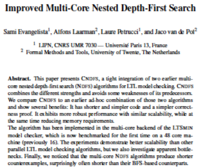
Abstract. This paper presents CNDFS, a tight integration of two earlier multi-core nested depth-first search (NDFS) algorithms for LTL model checking. CNDFS combines the different strengths and avoids some weaknesses of its predecessors. We compare CNDFS to an earlier ad-hoc combination of those two algorithms and show several benefits: It has shorter and simpler code and a simpler correctness proof. It exhibits more robust performance with similar scalability, while at the same time reducing memory requirements.
The algorithm has been implemented in the multi-core backend of the LTSMIN model checker, which is now benchmarked for the first time on a 48 core machine (previously 16). The experiments demonstrate better scalability than other parallel LTL model checking algorithms, but we also investigate apparent bottlenecks. Finally, we noticed that the multi-core NDFS algorithms produce shorter counterexamples, surprisingly often shorter than their BFS-based counterparts.

A History of Multi-core NDFSs



Blue MCNDFS

- o Share Blue + sequential repair procedure
- ± Often good speedups sometimes speeddowns
- ± Memory usage: $4 \cdot P + 3$ bits per state



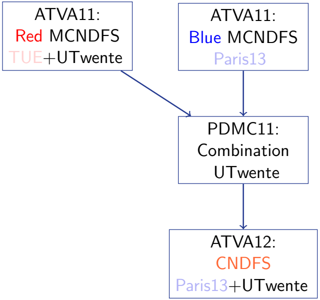
Improved Multi-Core Nested Depth-First Search

Sami Evangelista[1], Alfons Laarman[2], Laure Petrucci[1], and Jaco van de Pol[2]

[1] LIPN, CNRS UMR 7030 — Université Paris 13, France
[2] Formal Methods and Tools, University of Twente, The Netherlands

Abstract. This paper presents CNDFS, a tight integration of two earlier multi-core nested depth-first search (NDFS) algorithms for LTL model checking. CNDFS combines the different strengths and avoids some weaknesses of its predecessors. We compare CNDFS to an earlier ad-hoc combination of those two algorithms and show several benefits: It has shorter and simpler code and a simpler correctness proof. It exhibits more robust performance with similar scalability, while at the same time reducing memory requirements.
The algorithm has been implemented in the multi-core backend of the LTSMIN model checker, which is now benchmarked for the first time on a 48 core machine (previously 16). The experiments demonstrate better scalability than other parallel LTL model checking algorithms, but we also investigate apparent bottlenecks. Finally, we noticed that the multi-core NDFS algorithms produce shorter counterexamples, surprisingly often shorter than their BFS-based counterparts.

A History of Multi-core NDFSs

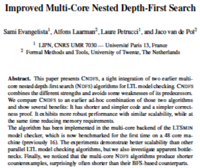


Blue MCNDFS

- o Share Blue + sequential repair procedure
- ± Often good speedups sometimes speeddowns
- ± Memory usage: $4 \cdot P + 3$ bits per state

Red MCNDFS

- o Share Red + little synchronization
- ± Ideal speedups but in rare cases ($|Red| = |Blue|$)
- ± Memory usage: $2 \cdot P + \log(P) + 1$ bits per state



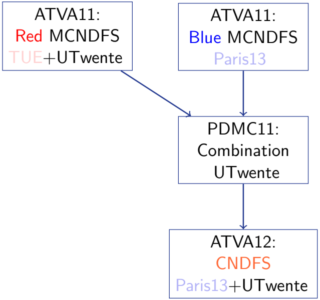
Improved Multi-Core Nested Depth-First Search

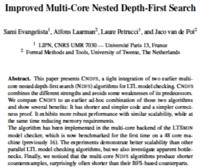
Sami Evangelista[1], Alfons Laarman[2], Laure Petrucci[1], and Jaco van de Pol[2]

[1] LIPN, CNRS UMR 7030 — Université Paris 13, France
[2] Formal Methods and Tools, University of Twente, The Netherlands

Abstract. This paper presents CNDFS, a tight integration of two earlier multi-core nested depth-first search (NDFS) algorithms for LTL model checking. CNDFS combines the different strengths and avoids some weaknesses of its predecessors. We compare CNDFS to an earlier ad-hoc combination of those two algorithms and show several benefits: It has shorter and simpler code and a simpler correctness proof. It exhibits more robust performance with similar scalability, while at the same time reducing memory requirements.
The algorithm has been implemented in the multi-core backend of the LTSMIN model checker, which is now benchmarked for the first time on a 48 core machine (previously 16). The experiments demonstrate better scalability than other parallel LTL model checking algorithms, but we also investigate apparent bottlenecks. Finally, we noticed that the multi-core NDFS algorithms produce shorter counterexamples, surprisingly often shorter than their BFS-based counterparts.

A History of Multi-core NDFSs





Improved Multi-Core Nested Depth-First Search

Sami Evangelista[1], Alfons Laarman[2], Laure Petrucci[1], and Jaco van de Pol[2]

[1] LIPN, CNRS UMR 7030 — Université Paris 13, France
[2] Formal Methods and Tools, University of Twente, The Netherlands

Abstract. This paper presents CNDFS, a tight integration of two earlier multi-core nested depth-first search (NDFS) algorithms for LTL model checking. CNDFS combines the different strengths and avoids some weaknesses of its predecessors. We compare CNDFS to an earlier ad-hoc combination of those two algorithms and show several benefits: It has shorter and simpler code and a simpler correctness proof. It exhibits more robust performance with similar scalability, while at the same time reducing memory requirements.
The algorithm has been implemented in the multi-core backend of the LTSMIN model checker, which is now benchmarked for the first time on a 48 core machine (previously 16). The experiments demonstrate better scalability than other parallel LTL model checking algorithms, but we also investigate apparent bottlenecks. Finally, we noticed that the multi-core NDFS algorithms produce shorter counterexamples, surprisingly often shorter than their BFS-based counterparts.

Blue MCNDFS

- o Share Blue + sequential repair procedure
- ± Often good speedups sometimes speeddowns
- ± Memory usage: $4 \cdot P + 3$ bits per state

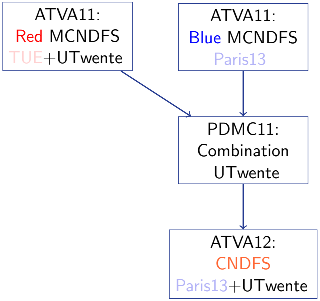
Red MCNDFS

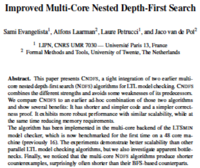
- o Share Red + little synchronization
- ± Ideal speedups but in rare cases ($|Red| = |Blue|$)
- ± Memory usage: $2 \cdot P + \log(P) + 1$ bits per state

Combination

- o Replace repair in Blue MCNDFS with Red MCNDFS
- \+ Good, but unstable, speedups
- – Combined memory usage

A History of Multi-core NDFSs





Improved Multi-Core Nested Depth-First Search

Sami Evangelista[1], Alfons Laarman[2], Laure Petrucci[1], and Jaco van de Pol[2]

[1] LIPN, CNRS UMR 7030 — Université Paris 13, France
[2] Formal Methods and Tools, University of Twente, The Netherlands

Abstract. This paper presents CNDFS, a tight integration of two earlier multi-core nested depth-first search (NDFS) algorithms for LTL model checking. CNDFS combines the different strengths and avoids some weaknesses of its predecessors. We compare CNDFS to an earlier ad-hoc combination of those two algorithms and show several benefits: It has shorter and simpler code and a simpler correctness proof. It exhibits more robust performance with similar scalability, while at the same time reducing memory requirements.
The algorithm has been implemented in the multi-core backend of the LTSMIN model checker, which is now benchmarked for the first time on a 48 core machine (previously 16). The experiments demonstrate better scalability than other parallel LTL model checking algorithms, but we also investigate apparent bottlenecks. Finally, we noticed that the multi-core NDFS algorithms produce shorter counterexamples, surprisingly often shorter than their BFS-based counterparts.

Blue MCNDFS

- o Share Blue + sequential repair procedure
- ± Often good speedups sometimes speeddowns
- ± Memory usage: $4 \cdot P + 3$ bits per state

Red MCNDFS

- o Share Red + little synchronization
- ± Ideal speedups but in rare cases ($|Red| = |Blue|$)
- ± Memory usage: $2 \cdot P + \log(P) + 1$ bits per state

Combination

- o Replace repair in Blue MCNDFS with Red MCNDFS
- + Good, but unstable, speedups
- − Combined memory usage

CNDFS

- o Avoid repair in Blue MCNDFS by late Red coloring
- + Good, stable speedups
- + Memory usage: $P + 2 + \epsilon$ bits per state

CNDFS

procedure DFSblue(s,i)
    s.cyan[i] := true
    for all t $\in$ post(s) **do**
        if $\neg$t.cyan[i] $\wedge\neg$t.blue **then** DFSblue(t,i)
    if s $\in$ Acc **then**
        $R[i] := \emptyset$
        DFSred(s,i)
        await $\forall r \in R[i]: r \in$Acc$\wedge r \neq s \implies$ r.red
        forall $r \in R[i]$ **do** r.red := true
    s.blue := true
    s.cyan[i] := false
procedure DFSred(s,i)
    $R[i] := R[i] \cup \{s\}$
    for all t $\in$ post(s) **do**
        if t.cyan[i]=true **then** ExitCycle
        if t $\notin R[i]$ $\wedge$ $\neg$t.red **then** DFSred(t,i)

CNDFS

procedure DFSblue(s,i)
  s.cyan[i] := true
  for all t ∈ post(s) **do**
    if ¬t.cyan[i] ∧¬t.blue **then** DFSblue(t,i)   Loses post order!
  if s ∈ Acc **then**
    $R[i] := \emptyset$
    DFSred(s,i)
    await $\forall r \in R[i]: r \in$Acc$\wedge r \neq s \implies$ r.red
    forall $r \in R[i]$ **do** r.red := true
  s.blue := true
  s.cyan[i] := false
procedure DFSred(s,i)
  $R[i] := R[i] \cup \{s\}$
  for all t ∈ post(s) **do**
    if t.cyan[i]=true **then** ExitCycle
    if t $\notin R[i]$ ∧ ¬t.red **then** DFSred(t,i)

CNDFS

procedure DFSblue(s,i)
  s.cyan[i] := true
  for all t $\in$ post(s) **do**
    if $\neg$t.cyan[i] $\wedge\neg$t.blue **then** DFSblue(t,i) Loses post order!
  if s $\in$ Acc **then**
    $R[i] := \emptyset$
    DFSred(s,i)
    await $\forall r \in R[i]\colon r \in \text{Acc} \wedge r \neq s \implies$ r.red
    forall $r \in R[i]$ **do** r.red := true
  s.blue := true
  s.cyan[i] := false
procedure DFSred(s,i) Collects states in $R[i]$
  $R[i] := R[i] \cup \{s\}$
  for all t $\in$ post(s) **do**
    if t.cyan[i]=true **then** ExitCycle
    if t $\notin R[i] \wedge \neg$t.red **then** DFSred(t,i)

CNDFS

procedure DFSblue(s,i)
  s.cyan[i] := true
  for all t ∈ post(s) **do**
    if ¬t.cyan[i] ∧¬t.blue **then** DFSblue(t,i)   Loses post order!
  if s ∈ Acc **then**
    $R[i] := \emptyset$
    DFSred(s,i)
    await $\forall r \in R[i]$: $r \in$Acc$\wedge r \neq s \implies$ r.red   Waiting repairs order!
    forall $r \in R[i]$ **do** r.red := true
  s.blue := true
  s.cyan[i] := false
procedure DFSred(s,i)   Collects states in $R[i]$
  $R[i] := R[i] \cup \{s\}$
  for all t ∈ post(s) **do**
    if t.cyan[i]=true **then** ExitCycle
    if t $\notin R[i]$ ∧ ¬t.red **then** DFSred(t,i)

CNDFS Correctness Argument

Completeness & soundness:
$\exists \textbf{acc\_cycle}\colon s_0 \rightarrow^* \textbf{acc\_cycle} \Leftrightarrow \mathit{cndfs}(s_0, N) = \textbf{report}(\textbf{cycle})$

CNDFS Correctness Argument

Completeness & soundness:
$\exists \textbf{acc\_cycle} \colon s_0 \rightarrow^* \textbf{acc\_cycle} \Leftrightarrow \mathit{cndfs}(s_0, N) = \textbf{report}(\textbf{cycle})$

CNDFS Correctness Argument

Completeness & soundness:
$\exists \textbf{acc\_cycle}\colon s_0 \rightarrow^* \textbf{acc\_cycle} \Leftrightarrow \mathit{cndfs}(s_0, N) = \textbf{report}(\textbf{cycle})$

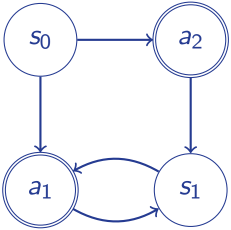
CNDFS Correctness Argument

Completeness & soundness:

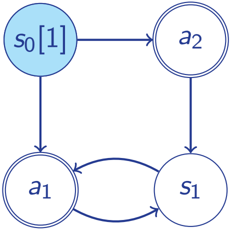
$\exists \textbf{acc\_cycle}\colon s_0 \rightarrow^* \textbf{acc\_cycle} \Leftrightarrow \mathit{cndfs}(s_0, N) = \textbf{report}(\textbf{cycle})$

Termination?

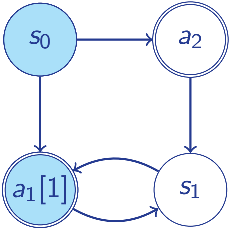
CNDFS Example



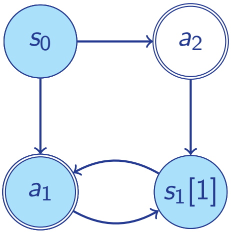
CNDFS Example



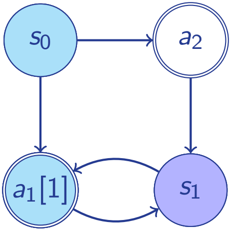
CNDFS Example



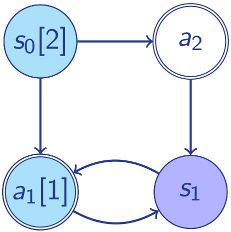
CNDFS Example



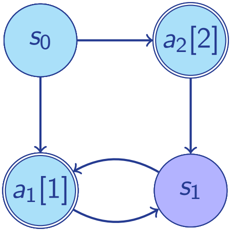
CNDFS Example



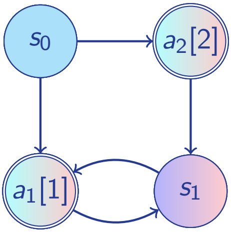
CNDFS Example



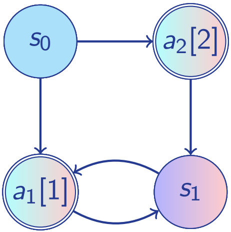
CNDFS Example



CNDFS Example



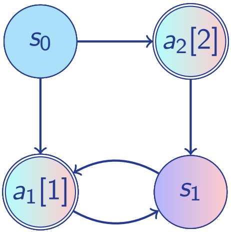
CNDFS Example



$a_1 \in R[2]$

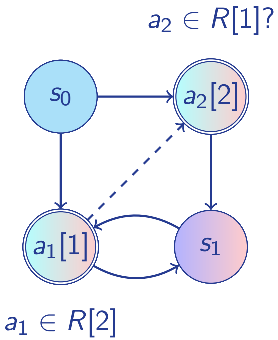
CNDFS Example

$a_2 \in R[1]?$



$a_1 \in R[2]$

CNDFS Example

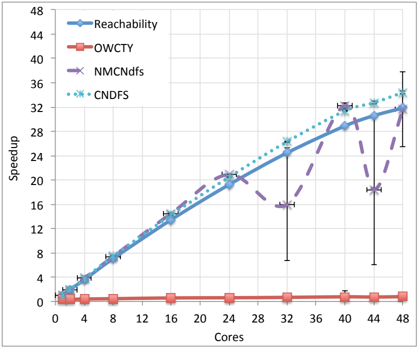


Experiments

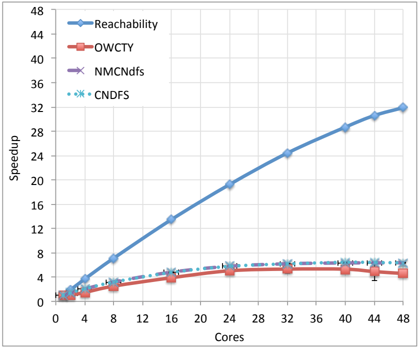
Experimental setup

- 48-core machine (NUMA)
- Compare: CNDFS and NMCNDFS in LTSmin, and OWCTY in DiVinE [Barnat et al.]
- 400+ models from the BEEM database
- Full verification and counter examples

Experiments: CNDFS vs OWCTY vs Reachability



leader_filters.6.prop2



lup.4.prop2

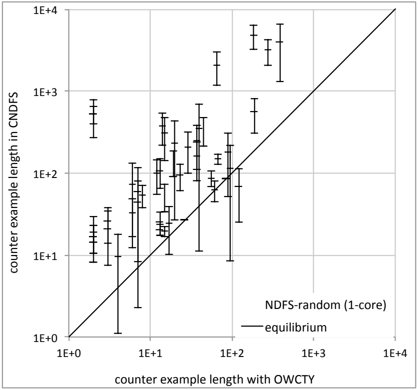
Full verification!

Experiments: CNDFS on-the-fly

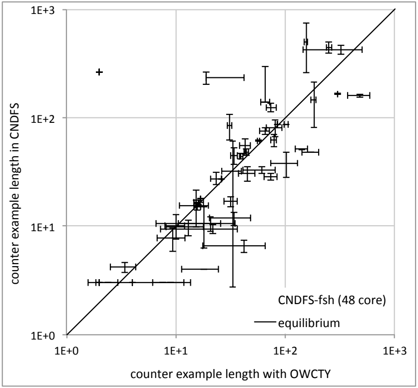
		NDFS	CNDFS		OWCTY	
		1 core	48 core		1 core	48 core
	model	Rand.	Rand.	Fsh.	Static	Rand.
Runtimes (sec)	anderson.8.prop3	36.4	4.1	0.2	2858.8	1433.2
	bakery.7.prop3	3.2	0.3	0.2	2.2	5.2
	bakery.8.prop4	15.7	0.6	0.3	73.4	14.3
	elevator2.3.prop3	8.4	1.4	0.2	432.3	192.5
	extinction.4.prop2	2.2	0.1	0.1	1.8	1.7
	peterson.6.prop4	29.1	0.9	0.5	668.4	705.7
	szymanski.5.prop4	1.7	1.3	0.2	2.1	376.4
Speedups	anderson.8.prop3		8.8	175.0		2.0
	bakery.7.prop3		10.9	21.2		0.4
	bakery.8.prop4		26.2	48.9		5.1
	elevator2.3.prop3		5.9	52.1		2.2
	extinction.4.prop2		18.5	28.8		1.0
	peterson.6.prop4		33.0	62.4		0.9
	szymanski.5.prop4		1.3	10.9		0.0

Super-linear speedups for bug hunting!

Experiments: Counterexample Length



Sequential



With 48x parallelism

Conclusions

- We have improved/investigated multi-core NDFS

Conclusions

- We have improved/investigated multi-core NDFS

Future work

- Define class of scaling inputs
- Can we do better? Guaranteed speedup?

Conclusion

Conclusions

- We have improved/investigated multi-core NDFS

Future work

- Define class of scaling inputs
- Can we do better? Guaranteed speedup?

Availability

- The implementation is available (open source) at:

  http://fmt.cs.utwente.nl/tools/ltsmin/

- See also: CAV'10, FMCAD'10, ATVA'11, NFM'11, SPIN'11, PDMC'11, PDMC'12