Scalable Multi-core Model Checking: Technology & Applications of Brute Force part II: Liveness & Timed Systems

Jaco van de Pol
30, 31 October 2014

VTSA 2014, Luxembourg
Multi-core LTL model checking

1. Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Nested Depth First Search
   - Parallel Nested Depth First Search

2. Interim Evaluation: Exhaustive Brute Force

3. Timed Automata: subsumption of symbolic states
   - Timed Büchi automata and subsumption
   - Multi-core Implementation of Reachability
   - LTL model checking with subsumption
Recall LTL

LTL formulae are built using temporal operators

\(\phi\) and \(\psi\) are formulae, interpreted over infinite paths

- \(\mathbf{X}\phi\): \(\phi\) holds in the next state in this path
- \(\mathbf{F}\phi\): \(\phi\) holds somewhere in this path
- \(\mathbf{G}\phi\): \(\phi\) holds everywhere on this path
- \(\mathbf{U}\phi\psi\): \(\psi\) holds somewhere on this path, and \(\phi\) holds in all preceding states
- \(\mathbf{R}\phi\psi\): \(\psi\) holds as long as \(\phi\) did not hold before

\[\phi ::= p \mid \neg \phi \mid \phi \land \phi \mid \phi \lor \phi \mid \mathbf{X}\phi \mid \mathbf{F}\phi \mid \mathbf{G}\phi \mid \phi \mathbf{U}\phi \mid \phi \mathbf{R}\phi\]

Sufficient basis for LTL:

\[\phi ::= p \mid \neg \phi \mid \phi \land \phi \mid \mathbf{X}\phi \mid \phi \mathbf{U}\phi\]
Kripke Structures and Paths

Kripke Structures .......................... (just labeled graphs)

A Kripke structure is a tuple $M = (S, S_0, R, AP, L)$, where

- $S$ is a set of states
- $S_0 \subseteq S$ is set of initial states
- $R \subseteq S \times S$ is a (total) transition relation on $S$
- $AP$ is a set of atomic proposition labels
- $L : S \rightarrow \mathcal{P}(AP)$ assigns to each state a set of labels

Infinite Paths ................. (just sequences of connected states)

- A path $\pi$ in $M$ is an infinite sequence $(s_0, s_1, s_2, \ldots)$ through the Kripke structure $M$, so $\forall i. s_i R s_{i+1}$
- Notation: $\pi \in path(s)$ if $\pi$ starts with $s$ (i.e.: $s_0 = s$)
- Notation: $\pi^i$ is the suffix from $i$, i.e.: $(s_i, s_{i+1}, \ldots)$
Formal CTL* semantics: $M, s_0 \models \phi$

**Semantics of Path Formulas (given path $\pi$)**

- $\pi \models \phi \iff \pi(0) \models \phi$ if $\phi$ is a state formula
- $\pi \models X \phi \iff \pi^1 \models \phi$
- $\pi \models F \phi \iff$ for some $i \geq 0$, $\pi^i \models \phi$
- $\pi \models G \phi \iff$ for all $i \geq 0$, $\pi^i \models \phi$
- $\pi \models \phi U \psi \iff \exists i \geq 0. \pi^i \models \psi \land \forall j < i. \pi^j \models \phi$
- $\pi \models \phi R \psi \iff \forall j \geq 0. ((\forall i < j. \pi^i \not\models \phi) \Rightarrow \pi^j \models \psi)$

**Some examples of LTL properties**

- Every request will be acknowledged: $G (req \implies req U ack)$
- $GF p$: $p$ happens infinitely often
- $FG p$: $p$ is nearly always true
- Note duality: $\neg GF p \iff FG \neg p$
Basic Automata Theoretic Approach

Automata Theoretic Approach

- Kripke Structure $M$ (system); LTL formula $\phi$ (requirement)
- Construct an automaton $A$ that recognizes violations of $\phi$.
- In other words: $A$ accepts a word $\pi \iff \pi \models \neg \phi$
- $M \models \phi$ iff $L(M) \subseteq L(\phi)$ iff $M \times A$ accepts $\emptyset$
- Problem: How to deal with infinite words?

Büchi automata for accepting infinite words

- Just like an normal automaton (NFA), with accepting states
- Accept words that hit an accepting state infinitely often
Examples of Büchi automata

almost always: $F G p$

```
\begin{tikzpicture}
\node[draw] (p) {$p$};
\node[draw] (neg_p) [right of=p] {$\neg p$};
\node[draw] (p_neg_p) [below of=neg_p] {$p$};
\node[draw] (neg_neg_p) [below of=p_neg_p] {$\neg q, \neg p$};
\node[draw] (q_neg_p) [right of=neg_neg_p] {$q, \neg p$};
\node[draw] (p_q_neg_p) [right of=q_neg_p] {$p$};
\node[draw] (neg_p_p) [right of=neg_p] {$p$};

\draw[->] (p) to (neg_p);
\draw[->] (neg_p) to (p_neg_p);
\draw[->] (p_neg_p) to (neg_neg_p);
\draw[->] (neg_neg_p) to (q_neg_p);
\draw[->] (q_neg_p) to (p_q_neg_p);
\draw[->] (p_q_neg_p) to (neg_p_p);
\end{tikzpicture}
```

infinitely often: $G F p$

```
\begin{tikzpicture}
\node[draw] (neg_p) {$\neg p$};
\node[draw] (p) [right of=neg_p] {$p$};
\node[draw] (neg_p_p) [right of=neg_p] {$p$};

\draw[->] (neg_p) to (p);
\draw[->] (neg_p_p) to (neg_p);
\end{tikzpicture}
```

infinitely often with guarantee: $G(q U p)$

```
\begin{tikzpicture}
\node[draw] (q_neg_p) {$q, \neg p$};
\node[draw] (p) [right of=q_neg_p] {$p$};
\node[draw] (neg_q_neg_p) [below of=q_neg_p] {$\neg q, \neg p$};
\node[draw] (neg_q_neg_p_q_neg_p) [below of=neg_q_neg_p] {$\neg q, \neg p$};

\draw[->] (q_neg_p) to (p);
\draw[->] (p) to (neg_q_neg_p);
\draw[->] (neg_q_neg_p) to (neg_q_neg_p_q_neg_p);
\draw[->] (neg_q_neg_p_q_neg_p) to (q_neg_p);
\end{tikzpicture}
```
Model Checking by 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
  - $S \subseteq \mathcal{P}$ iff $S \cap \overline{\mathcal{P}} = \emptyset$ iff $S \times \neg \mathcal{P}$ has no accepting cycle
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth First Search (NDFS)

Properties of NDFS

- NDFS runs in linear time
- Inherently depends on post-order
- Post-order is P-complete [Reif’85]
- Not parallelizable (unless P=NC)
Recall: Nested Depth First Search

[CVWY’92] [Holzmann’92]

- **Blue search**: explore graph in DFS order
  - states on the blue search stack are cyan
  - on backtracking from an accepting state:
- **Red search**: find an accepting cycle
  - exit as soon as the cyan stack is reached
- Linear time, depends on post-order

**Blue search**

1: `procedure dfsBlue(s)`
2: add $s$ to Cyan
3: for all successors $t$ of $s$ do
4: if $t \notin \text{Blue} \cup \text{Cyan}$ then
5: $\text{dfsBlue}(t)$
6: if $s$ is accepting then
7: $\text{dfsRed}(s)$
8: move $s$ from Cyan to Blue

**Red search**

1: `procedure dfsRed(s)`
2: add $s$ to Red
3: for all successors $t$ of $s$ do
4: if $t \in \text{Cyan}$ then
5: Exit: cycle detected
6: if $t \notin \text{Red}$ then
7: $\text{dfsRed}(t)$
Table of Contents

1 Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Nested Depth First Search
   - Parallel Nested Depth First Search

2 Interim Evaluation: Exhaustive Brute Force

3 Timed Automata: subsumption of symbolic states
   - Timed Büchi automata and subsumption
   - Multi-core Implementation of Reachability
   - LTL model checking with subsumption
Simple idea: Swarmed Nested Depth First Search

Laarman, Langerak, van de Pol, Wijs [ATVA’11]

Multi-core Swarmed NDFS

- \( W \) workers perform independent random NDFS
  - Visited states are stored in a shared hashtable
  - All workers use their own set of colors (2W bits per state)
  - Speeds up bug hunting only

Blue search

1: **procedure** \( \text{dfsBlue}(s, i) \)
2: \( \text{add } s \text{ to } \text{Cyan}[i] \)
3: **for all** successors \( t \) of \( s \) **do**
4: \( \text{if } t \notin \text{Blue}[i] \cup \text{Cyan}[i] \text{ then} \)
5: \( \text{dfsBlue}(t, i) \)
6: \( \text{if } s \text{ is accepting then} \)
7: \( \text{dfsRed}(s, i) \)
8: move \( s \) from \( \text{Cyan}[i] \) to \( \text{Blue}[i] \)

Red search

1: **procedure** \( \text{dfsRed}(s, i) \)
2: \( \text{add } s \text{ to } \text{Red}[i] \)
3: **for all** successors \( t \) of \( s \) **do**
4: \( \text{if } t \in \text{Cyan}[i] \text{ then} \)
5: **Exit:** cycle detected
6: \( \text{if } t \notin \text{Red}[i] \text{ then} \)
7: \( \text{dfsRed}(t, i) \)
Multi-core Nested Depth First Search
Laarman, van de Pol,...[ATVA’11][PDMC’11]; Evangelista,L,vdP [ATVA’12]

Multi-core NDFS (several variations)

- Collaboration between NDFS workers
  - Share red and/or blue globally
  - Workers backtrack on parts finished by others
  - Correctness: Complicated to restore post-order
  - Performance: Reasonable scalability

---

Blue search

1: procedure dfsBlue(s, i)
2: add s to Cyan[i]
3: for all successors t of s do
4: if t $\not\in$ Blue $\cup$ Cyan[i] then
5: dfsBlue(t, i)
6: if s is accepting then
7: dfsRed(s, i)
8: move s from Cyan[i] to Blue

Red search

1: procedure dfsRed(s, i)
2: add s to Red
3: for all successors t of s do
4: if t $\in$ Cyan[i] then
5: Exit: cycle detected
6: if t $\not\in$ Red then
7: dfsRed(t, i)
Swarmed NDFS versus Parallel NDFS
Experiments from [ATVA’11] on BEEM benchmarks on 16 cores

Conclusions

- Swarmed NDFS speeds up bug hunting
- Parallel NDFS also speeds up verification
# Table of Contents

1. Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Nested Depth First Search
   - Parallel Nested Depth First Search

2. Interim Evaluation: Exhaustive Brute Force

3. Timed Automata: subsumption of symbolic states
   - Timed Büchi automata and subsumption
   - Multi-core Implementation of Reachability
   - LTL model checking with subsumption
**Nested Depth First Search**

**procedure** DFSblue(s)

\[
\text{s.blue := true}
\]

\[
\text{for all } t \in \text{post}(s) \text{ do}
\]

\[
\text{if } \neg t.\text{blue then DFSblue}(t)
\]

\[
\text{if } s \in \text{Accepting then}
\]

\[
\text{seed := s}
\]

\[
\text{DFSred(s)}
\]

**procedure** DFSred(s)

\[
\text{s.red := true}
\]

\[
\text{for all } t \in \text{post}(s) \text{ do}
\]

\[
\text{if } t = \text{seed then ExitCycle}
\]

\[
\text{if } \neg t.\text{red then DFSred}(t)
\]

---

**Nested DFS**

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

- **Red Search**
  - Finds cycle through seed
  - Visits states at most once

- **Linear time, on-the-fly**

- **Blue is inherently depth-first**
Swarmed Multi-core Nested Depth First Search

code for worker $i$

**procedure** DFSblue($s, i$)

$s$.blue[$i$] := true

for all $t \in$ post($s$) do

if $\neg t$.blue[$i$] then DFSblue($t, i$)

if $s \in$ Accepting then

seed[$i$] := $s$

DFSred($s, i$)

**procedure** DFSred($s, i$)

$s$.red[$i$] := true

for all $t \in$ post($s$) do

if $t = $ seed[$i$] then ExitCycle

if $\neg t$.red[$i$] then DFSred($t, i$)

**Multi-core Swarmed NDFS**

- $N$ workers perform parallel search **independently** [G. Holzmann etal.]

- **Multi-core:** store visited states in a shared hash table [FMCAD 2010, SPIN 2011]

- Scales well in the presence of accepting cycles (bugs)

- Otherwise, all workers traverse the whole graph
Approaches to Parallel LTL Model Checking

**Speedup of Swarmed NDFS**
(1 versus 16 cores)

![Graph showing speedup of Swarmed NDFS](image)

**Alternatives**

- **Swarm verification with NDFS**
  - Effective, only for bug finding

- **Dual-core NDFS [Holzmann]**
  - Red search on 2nd CPU
  - Speedup of at most factor 2

- **Red Search as parallel reachability**
  - Speedup still $\leq 2$: $|G| + |G|/N$

- **Can one do better?**
  - Post-order is P-Complete, so
  - DFS not efficiently parallelizable

- **Breadth-first based:**
  - OWCTY, MAP [Brno]
  - Not linear ($|G| \cdot h$), not on-the-fly
New NDFS with Cyan and Pink [à la Schwoon/Esparza]

s.bc: white → cyan → blue
s.rc: white → pink → red

procedure DFSblue(s)
    s.bc := cyan
    for all t ∈ post(s) do
        if t.bc=white then DFSblue(t)
        if s ∈ Acc then DFSred(s)
    s.bc := blue

procedure DFSred(s)
    s.rc := pink
    for all t ∈ post(s) do
        if t.bc=cyan then ExitCycle
        if t.rc=white then DFSred(t)
    s.rc := red
What goes wrong if the DFS order is violated?

What if:

- Red search starts from 1, no Cyan state is encountered
- On the backtrack, the states are colored red
- A new red search starts from 2, but terminates immediately

No accepting cycle is detected!
Parallel NDFS: share the red color (first try)

\[ s.\text{color}[i] : \text{white} \rightarrow \text{cyan} \rightarrow \text{blue} \]
\[ s.\text{pink}[i], s.\text{red} : \text{Boolean} \]

**procedure** DFSblue(s,i)
- \[ s.\text{color}[i] := \text{cyan} \]
- **for all** \( t \in \text{post}(s) \) **do**
  - **if** \( t.\text{color}[i] = \text{white} \) and \( \neg t.\text{red} \) **then** DFSblue(t,i)
- **if** \( s \in \text{Acc} \) **then** DFSred(s,i)
- \[ s.\text{color}[i] := \text{blue} \]

**procedure** DFSred(s,i)
- \[ s.\text{pink}[i] := \text{true} \]
- **for all** \( t \in \text{post}(s) \) **do**
  - **if** \( t.\text{color}[i] = \text{cyan} \) **then** ExitCycle
  - **if** \( \neg t.\text{pink}[i] \) and \( \neg t.\text{red} \) **then** DFSred(t,i)
- \[ s.\text{red} := \text{true} \] (unfortunately incorrect)
Example: what is the meaning of red? (2 workers)

All accepting cycles contain red:

Accepting states on cycles get red:

No problem: path pink \rightarrow cyan
Synchronisation is necessary: third worker strikes!

Workers 1, 2 proceed as before

Worker 3 starts Red search in 1, 0
No cycle will be detected!
Parallel NDFS: share the red color (correct version)

procedure DFSblue(s,i)
    s.color[i] := cyan
    for all t ∈ post(s) do
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
    if s ∈ Acc then DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.pink[i] := true
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if ¬t.pink[i] and ¬t.red then DFSred(t,i)
    pink[i] := false
    if s ∈ Acc then await ∀j : ¬s.pink[j]
    s.red := true

[ATVA 2011]
Optimization 1: Early detection and $2N+1+\log(N)$ bits

\begin{verbatim}
procedure DFSblue(s,i)
s.color[i] := cyan
for all t ∈ post(s) do
    if t.color[i]=cyan and s or t ∈ Acc then ExitCycle
    if t.color[i]=white and ¬t.red then DFSblue(t,i)
if s ∈ Acc then s.count++; DFSred(s,i)
s.color[i] := blue

procedure DFSred(s,i)
s.color[i] := pink
for all t ∈ post(s) do
    if t.color[i]=cyan then ExitCycle
    if t.color[i]≠pink and ¬t.red then DFSred(t,i)
if s ∈ Acc then s.count--; await s.count=0
s.red := true
\end{verbatim}
Optimization 2: Sprinkle red paint

procedure DFSblue(s,i)
    s.color[i] := cyan
    all_successors_red := true
    for all t \in post(s) do
        if t.color[i]=cyan and s or t \in Acc then ExitCycle
        if t.color[i]=white and \neg t.red then DFSblue(t,i)
        if \neg t.red then all_successors_red := false
    if all_successors_red then s.red := true
    else if s \in Acc then s.count++; DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.color[i] := pink
    for all t \in post(s) do
        if t.color[i]=cyan then ExitCycle
        if t.color[i]\neq pink and \neg t.red then DFSred(t,i)
        if s \in Acc then s.count--; await s.count=0
        s.red := true
Multi-core LTL model checking

Interim Evaluation: Exhaustive Brute Force

Timed Automata

Swarmed NDFS versus Parallel NDFS

Swarmed NDFS (1 versus 16-core)

Parallel NDFS (1 versus 16-core)

UNIVERSITY OF TWENTE.

Multi-core Model Checking

30, 31 October 2014  26 / 45
OWCTY and Swarmed NDFS versus Parallel NDFS

Swarmed versus Parallel NDFS (both 16 cores)

OWCTY versus Parallel NDFS (both 16 cores)
Experiments extended to 48 cores

From [PDMC’12]. See fmt.cs.utwente.nl/tools/ltsmin/performance/

Promela: Bakery protocol

Promela: Elevator controller
Table of Contents

1 Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Nested Depth First Search
   - Parallel Nested Depth First Search

2 Interim Evaluation: Exhaustive Brute Force

3 Timed Automata: subsumption of symbolic states
   - Timed Büchi automata and subsumption
   - Multi-core Implementation of Reachability
   - LTL model checking with subsumption
## Interim Evaluation: what did we learn?

### Reachability: Implementation matters, keep it simple
- Leave workers alone when possible; load balancing
- Rely on randomness to avoid “duplicate work”
- Careful design of concurrent data structures

### LTL model checking
- Previous parallel algorithms (OWCTY) used BFS: $O(N^2)$
- Now: linear, speedups ... $P = NC$, or what did we do?
  - $W \to \infty$ versus $W = 48$
  - Worst case $O(N \cdot W)$, no speedup

### Remaining theoretical questions
- Average (randomized) runtime/scalability analysis
- Why doesn’t this work for Strongly Connected Components?
### Practical Evaluation: Solved multi-core model checking?

#### Multi-core MC is compatible
- On-the-fly
- Partial-order reduction
- State compression
- Symbolic model checking

#### Quite general
- Arbitrary state/edge labels
- mCRL2, Promela, DVE, GSPN,
- LLVM, C, xUML, POOSL, ??
- Domain Specific Languages?

#### Remaining Questions
- Even better speedup – especially for symbolic model checking
- Quite restricted to explicit state model checking
- Infinite state systems? data, recursion, time, BDDs, ...
Table of Contents

1 Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Nested Depth First Search
   - Parallel Nested Depth First Search

2 Interim Evaluation: Exhaustive Brute Force

3 Timed Automata: subsumption of symbolic states
   - Timed Büchi automata and subsumption
   - Multi-core Implementation of Reachability
   - LTL model checking with subsumption
Model checking LTL for Timed Automata

Laarman, Olesen, Dalsgaard, Kim Larsen, vdPol [FORMATS’12] [CAV’13]

Handling Timed Automata

- Work with timed zones (DBM) for Timed Büchi Automata
- Checking LTL properties for Uppaal timed automata
  - Use subsumption to prune Nested DFS where possible
  - Multi-core NDFS algorithm for Timed Büchi Automata

Tool support

- Open source through OPAAL and LTSMIN
  - opaal-modelchecker.com/
  - fmt.cs.utwente.nl/tools/ltsmin/
# Table of Contents

1. **Multi-core LTL model checking**
   - Büchi automata for LTL model checking
   - Nested Depth First Search
   - Parallel Nested Depth First Search

2. **Interim Evaluation: Exhaustive Brute Force**

3. **Timed Automata: subsumption of symbolic states**
   - Timed Büchi automata and subsumption
   - Multi-core Implementation of Reachability
   - LTL model checking with subsumption
Timed Büchi Automata

Ingredients

- locations \((\ell_0, \ell_1, \ell_2)\), can be initial or accepting
- transitions, governed by real-valued clocks \((x, y)\)
- timed runs should respect clock guards, resets, invariants

\[
\begin{align*}
\ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} &\xrightarrow{2.7} \ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} &\xrightarrow{1.8} \ell_1, \begin{pmatrix} 1.8 \\ 0 \end{pmatrix} &\xrightarrow{0.5} \ell_2, \begin{pmatrix} 0 \\ 0 \end{pmatrix} &\xrightarrow{2.0} \ell_1, \begin{pmatrix} 2.0 \\ 2.0 \end{pmatrix} \\
\ell_1 &\xrightarrow{y \leq 2} \ell_1 &\xrightarrow{y \leq 2} \ell_2
\end{align*}
\]

Question: is the Büchi language empty? . . . . . . no counterexample

Does a (non-zeno) timed run exist that visits an accepting state infinitely often?
Finite representation: zone abstraction, extrapolation

\[ x := 0, y := 0 \]

\[ \ell_0 \]

\[ y := 0 \]

\[ \ell_1 \]

\[ [x > 2] \]

\[ x := 0, y := 0 \]

\[ \ell_2 \]

\[ y \leq 2 \]

\[ y \leq 2 \]

Finite representation by zones (DBM) [Dill’89] [Daws, Tripakis’98]

- A zone is a set of constraints
- finite by taking into account the lower/upperbounds

\[ Z_0 := y = x \]

\[ Z_1 := y \leq x \land y \leq 2 \]

\[ Z_2 := y = x \land y \leq 2 \]

Subsumption:

\[ Z_2 \subseteq Z_1, \text{ so } (\ell_1, Z_2) \sqsubseteq (\ell_1, Z_1) \]
Subsumption, or inclusion abstraction

Why explore a state again, if it is subsumed by a previous state?

Zone abstraction

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \]

\[ s_0 \rightarrow s_3 \rightarrow s_1 \]

\[ s_0 \rightarrow s_1 \]

Subsumption

\[ s_3 \sqsubseteq s_1 \]

Known results

- finite zone abstraction preserves reachability of locations
- finite zone abstraction also preserve Büchi emptiness
- subsumption preserves reachability of locations as well

Open problem

Is emptiness of Timed Büchi Automata preserved by subsumption? NO
Table of Contents

1. Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Nested Depth First Search
   - Parallel Nested Depth First Search

2. Interim Evaluation: Exhaustive Brute Force

3. Timed Automata: subsumption of symbolic states
   - Timed Büchi automata and subsumption
   - Multi-core Implementation of Reachability
   - LTL model checking with subsumption
Extension to Multi-core Reachability . . . [FORMATS’12]

- Timed zones captured in Difference Bound Matrices (DBM)
- For LTSmin, extend discrete state vector \( s \) with a pointer to a DBM \( (s, \sigma) \)
- Extend the PINS API with a function \( \text{Covers}(\sigma, \tau) \)
- Hash based on discrete parts, keep list of maximal zones
- Can be generalized to other symbolic domains (lattice model checking)
Table of Contents

1 Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Nested Depth First Search
   - Parallel Nested Depth First Search

2 Interim Evaluation: Exhaustive Brute Force

3 Timed Automata: subsumption of symbolic states
   - Timed Büchi automata and subsumption
   - Multi-core Implementation of Reachability
   - LTL model checking with subsumption
Analysis of accepting spirals with subsumption

\[ \sqsubseteq \text{ is a simulation relation:} \]

\[
\begin{align*}
s' & \rightarrow t' \\
\sqsubseteq & \sqsubseteq \\
s & \rightarrow t
\end{align*}
\]

\[ \sqsubseteq \text{ is a finite abstraction} \]

Lemma: If \( s \) has an accepting cycle then any \( s' \sqsubseteq s \) has it as well.

Lemma: If \( t' \) has an accepting spiral then \( t' \) has an accepting cycle.

Preservation of accepting cycles

<table>
<thead>
<tr>
<th>Preserves</th>
<th>Proof Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s' \rightarrow^* )</td>
<td>( t' \rightarrow^+ )</td>
</tr>
<tr>
<td>( t'' \rightarrow^+ )</td>
<td>( \ldots \rightarrow^* \times \rightarrow^+ )</td>
</tr>
<tr>
<td>( t'''' \rightarrow^+ )</td>
<td>( \times )</td>
</tr>
</tbody>
</table>

UNIVERSITY OF TWENTE.

Multi-core Model Checking

30, 31 October 2014
Subsumption in Nested Depth First Search

Blue search

1: \textbf{procedure} \textit{dfsBlue}(s)
2: \quad \textit{Cyan} := \textit{Cyan} \cup \{s\}
3: \quad \textbf{for all} successors $t$ of $s$ \textbf{do}
4: \quad \quad \textbf{if} $t \notin \text{Blue} \cup \text{Cyan} \land t \nsubseteq \text{Red}$ \textbf{then} \quad \text{Prune the blue search}
5: \quad \quad \textit{dfsBlue}(t)
6: \quad \textbf{if} $s$ is accepting \textbf{then}
7: \quad \quad \textit{dfsRed}(s)
8: \quad \textit{Blue, Cyan} := \textit{Blue} \cup \{s\}, \textit{Cyan} \setminus \{s\}

Red search

1: \textbf{procedure} \textit{dfsRed}(s) \quad \textbf{Postcondition: no accepting spiral reachable}
2: \quad \textit{Red} := \textit{Red} \cup \{s\}
3: \quad \textbf{for all} successors $t$ of $s$ \textbf{do}
4: \quad \quad \textbf{if} $t \in \text{Cyan} \land t \sqsupseteq \text{Cyan}$ \textbf{then} \quad \text{Accepting spiral found!}
5: \quad \quad \text{Exit: cycle detected}
6: \quad \textbf{if} $t \notin \text{Red} \land t \nsubseteq \text{Red}$ \textbf{then} \quad \text{Spiral on $t$ would give spiral from Red}
7: \quad \quad \textit{dfsRed}(t)
Subsumption on Blue is Unsound

Assume we would backtrack on $t$ as soon as $t \subseteq \text{Blue}$:

Accepting cycle $s_4 \rightarrow s_5$ not detected

- The blue search proceeds via $s_0, s_1, s_2$, then backtracks via $s_1$ to $s_3$
- Now since $s'_2 \subseteq \text{Blue}$, the blue search is pruned at $s_3$
- $s_3 \in \text{Acc}$, so a red search is started: $s_3, s'_2, s'_1, s_4, s_5$
- The only accepting cycle $s_4 \rightarrow s_5$ is erroneously made red
- Note: accepting states are not visited in post-order
Experiments: speedup up to 48 cores
Reachability: [FORMATS’12]. LTL model checking: [CAV’13]

BFS Reachability on Timed Automata

Checking LTL on Timed Automata

Experiments with **opaal** and **LTSmin** – open source
hours → minutes → seconds
Literature on LTSmin (liveness - LTL model checking)

LTL model checking

- Alfons Laarman, Rom Langerak, Jaco vd Pol, Michael Weber, A. Wijs, Multi-Core Nested Depth-First Search. .................. (ATVA 2011)
- Alfons Laarman, Jaco van de Pol, Variations on Multi-Core Nested Depth-First Search ........ (PDMC 2011)
- Sami Evangelista, Alfons Laarman, Laure Petrucci and Jaco van de Pol, Improved Multi-Core Nested Depth-First Search .......... (ATVA 2012)

Timed Automata

- A. Dalsgaard, A.W. Laarman, K.G. Larsen, M. Olesen, J. van de Pol, Multi-Core Reachability for Timed Automata ............ (FORMATS'12)
- Alfons Laarman, M. Olesen, A. Dalsgaard, K.G. Larsen, J. van de Pol, Multi-core emptiness checking of timed Büchi automata using inclusion abstraction ........................................... (CAV'13)