UNIVERSITY OF TWENTE.

Formal Methods & Tools.

Model checking LTL for Timed Automata

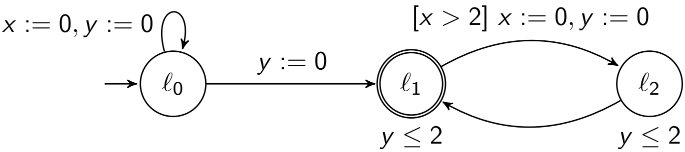
– Multi-core Nested DFS with Subsumption –

Alfons Laarman, Mads Olesen,
Andreas Dalsgaard, Kim Larsen,
Jaco van de Pol

CAV'13, St. Petersburg

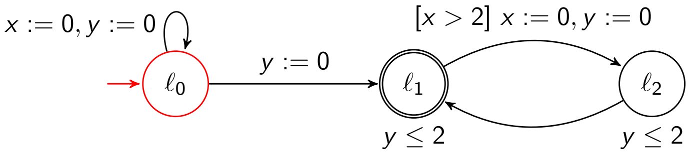
Kim celebrating the CAV 2013 Award

Timed Büchi Automata



Ingredients

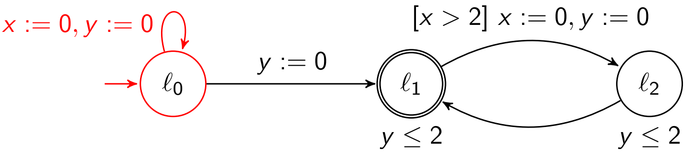
- locations (ℓ_0, ℓ_1, ℓ_2), can be initial, accepting or neither
- transitions, governed by real-valued clocks (x, y)
- timed runs should respect clock guards, resets, invariants



Ingredients

- locations (ℓ_0, ℓ_1, ℓ_2), can be initial, accepting or neither
- transitions, governed by real-valued clocks (x, y)
- timed runs should respect clock guards, resets, invariants

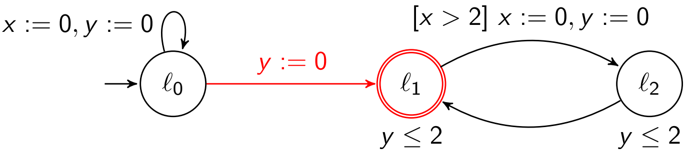
$$\ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$



Ingredients

- locations (ℓ_0, ℓ_1, ℓ_2), can be initial, accepting or neither
- transitions, governed by real-valued clocks (x, y)
- timed runs should respect clock guards, resets, invariants

$$\ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{2.7} \ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$



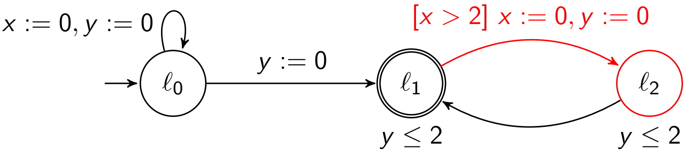
Ingredients

- locations (ℓ_0, ℓ_1, ℓ_2), can be initial, accepting or neither
- transitions, governed by real-valued clocks (x, y)
- timed runs should respect clock guards, resets, invariants

$$\ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{2.7} \ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{1.8} \ell_1, \begin{pmatrix} 1.8 \\ 0 \end{pmatrix}$$

Timed Büchi Automata

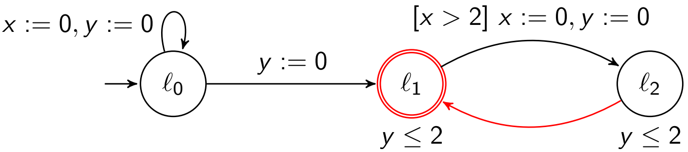
[Alur,Dill'94]



Ingredients

- locations (ℓ_0, ℓ_1, ℓ_2), can be initial, accepting or neither
- transitions, governed by real-valued clocks (x, y)
- timed runs should respect clock guards, resets, invariants

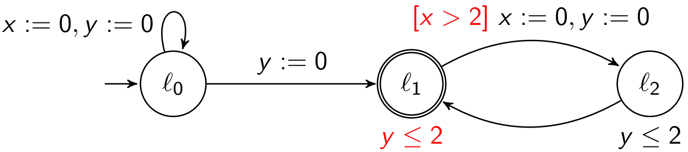
$$\ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{2.7} \ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{1.8} \ell_1, \begin{pmatrix} 1.8 \\ 0 \end{pmatrix} \xrightarrow{0.5} \ell_2, \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$



Ingredients

- locations (ℓ_0, ℓ_1, ℓ_2), can be initial, accepting or neither
- transitions, governed by real-valued clocks (x, y)
- timed runs should respect clock guards, resets, invariants

$$\ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{2.7} \ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{1.8} \ell_1, \begin{pmatrix} 1.8 \\ 0 \end{pmatrix} \xrightarrow{0.5} \ell_2, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{2.0} \ell_1, \begin{pmatrix} 2.0 \\ 2.0 \end{pmatrix} \not\rightarrow$$

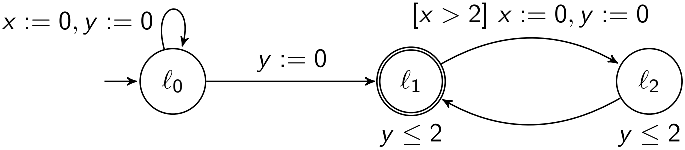


Ingredients

- locations (ℓ_0, ℓ_1, ℓ_2), can be initial, accepting or neither
- transitions, governed by real-valued clocks (x, y)
- timed runs should respect clock guards, resets, invariants

$$\ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{2.7} \ell_0, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{1.8} \ell_1, \begin{pmatrix} 1.8 \\ 0 \end{pmatrix} \xrightarrow{0.5} \ell_2, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{2.0} \ell_1, \begin{pmatrix} 2.0 \\ 2.0 \end{pmatrix} \not\rightarrow$$

Timed Büchi Automata [Alur,Dill'94]



Ingredients

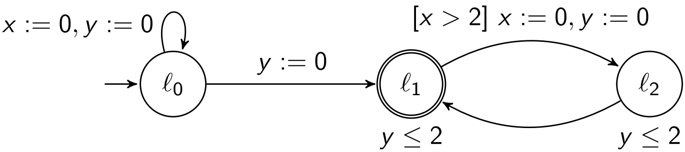
- locations (ℓ_0, ℓ_1, ℓ_2), can be initial, accepting or neither
- transitions, governed by real-valued clocks (x, y)
- timed runs should respect clock guards, resets, invariants

$$\ell_0, \begin{pmatrix}0\\0\end{pmatrix} \xrightarrow{2.7} \ell_0, \begin{pmatrix}0\\0\end{pmatrix} \xrightarrow{1.8} \ell_1, \begin{pmatrix}1.8\\0\end{pmatrix} \xrightarrow{0.5} \ell_2, \begin{pmatrix}0\\0\end{pmatrix} \xrightarrow{2.0} \ell_1, \begin{pmatrix}2.0\\2.0\end{pmatrix} \not\to$$

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

Does a (non-zeno) timed run exist that visits an accepting state infinitely often?

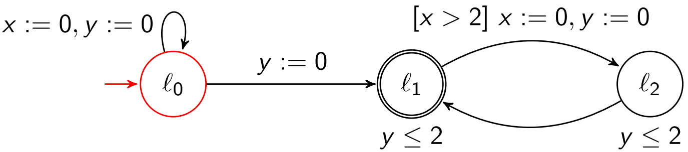
Finite representation: zone abstraction, extrapolation



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

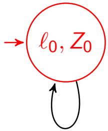
- A zone is a set of constraints
- Finite abstractions: k-extrapolation, LU-abstraction (taking into account Lower/Upperbounds in the TBA)

Finite representation: zone abstraction, extrapolation



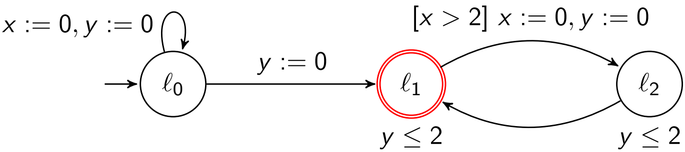
Finite representation by zones (DBM) [Dill'89] [Daws,Tripakis'98]

- A zone is a set of constraints
- Finite abstractions: k-extrapolation, LU-abstraction (taking into account Lower/Upperbounds in the TBA)



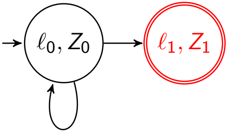
$$Z_0 \quad := \quad y = x$$

Finite representation: zone abstraction, extrapolation



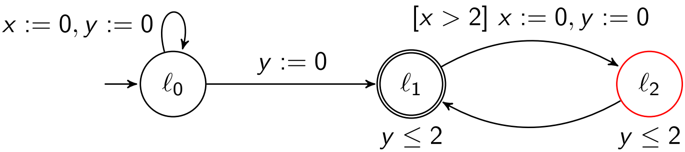
Finite representation by zones (DBM) [Dill'89] [Daws, Tripakis'98]

- A zone is a set of constraints
- Finite abstractions: k-extrapolation, LU-abstraction
  (taking into account Lower/Upperbounds in the TBA)



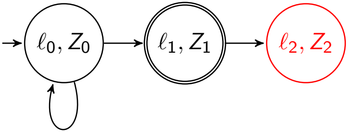
$$
\begin{aligned}
Z_0 \quad &:= \quad y = x \\
Z_1 \quad &:= \quad y \leq x \wedge y \leq 2
\end{aligned}
$$

Finite representation: zone abstraction, extrapolation



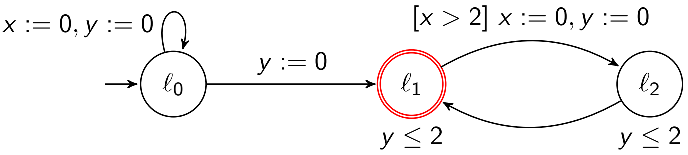
Finite representation by zones (DBM) [Dill'89] [Daws,Tripakis'98]

- A zone is a set of constraints
- Finite abstractions: k-extrapolation, LU-abstraction
  (taking into account Lower/Upperbounds in the TBA)



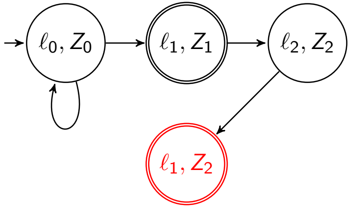
$$
\begin{aligned}
Z_0 &:= y = x \\
Z_1 &:= y \leq x \wedge y \leq 2 \\
Z_2 &:= y = x \wedge y \leq 2
\end{aligned}
$$

Finite representation: zone abstraction, extrapolation



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

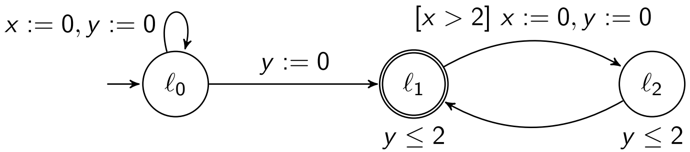
- A zone is a set of constraints
- Finite abstractions: k-extrapolation, LU-abstraction
  (taking into account Lower/Upperbounds in the TBA)



$$
\begin{aligned}
Z_0 &:= y = x \\
Z_1 &:= y \leq x \wedge y \leq 2 \\
Z_2 &:= y = x \wedge y \leq 2
\end{aligned}
$$

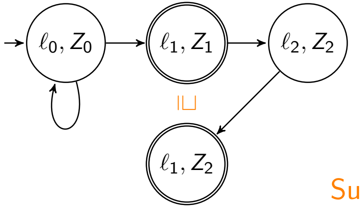
No accepting run!

Finite representation: zone abstraction, extrapolation



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

- A zone is a set of constraints
- Finite abstractions: *k*-extrapolation, LU-abstraction
  (taking into account Lower/Upperbounds in the TBA)



$$
\begin{aligned}
Z_0 &:= y = x \\
Z_1 &:= y \leq x \wedge y \leq 2 \\
Z_2 &:= y = x \wedge y \leq 2
\end{aligned}
$$

Subsumption:

$Z_2 \subseteq Z_1$, so $(\ell_1, Z_2) \sqsubseteq (\ell_1, Z_1)$

No accepting run!

Subsumption, or inclusion abstraction

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

Known results [Behrmann et al'04] [Tripakis'09] [Li'09]

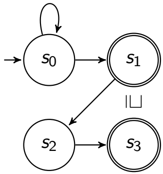
- k-extrapolation and LU-abstraction preserves reachability of locations
- k-extrapolation and LU-abstraction also preserve Büchi emptiness
- subsumption preserves reachability of locations as well

Subsumption, or inclusion abstraction

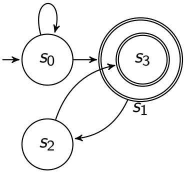
Why explore a state again, if it is subsumed by a previous state?

Known results [Behrmann et al'04] [Tripakis'09] [Li'09]

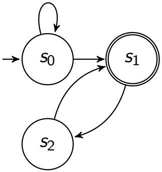
- k-extrapolation and LU-abstraction preserves reachability of locations
- k-extrapolation and LU-abstraction also preserve Büchi emptiness
- subsumption preserves reachability of locations as well



Zone abstraction



$s_3 \sqsubseteq s_1$



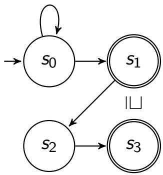
subsumption

Subsumption, or inclusion abstraction

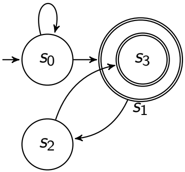
Why explore a state again, if it is subsumed by a previous state?

Known results [Behrmann et al'04] [Tripakis'09] [Li'09]

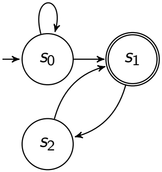
- k-extrapolation and LU-abstraction preserves reachability of locations
- k-extrapolation and LU-abstraction also preserve Büchi emptiness
- subsumption preserves reachability of locations as well



Zone abstraction



$s_3 \sqsubseteq s_1$



subsumption

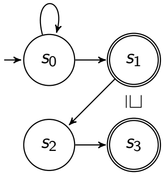
Open problem posed in [Tripakis'09]

Is emptiness of Timed Büchi Automata preserved by subsumption?

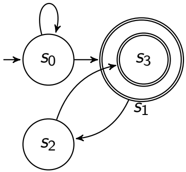
Why explore a state again, if it is subsumed by a previous state?

Known results [Behrmann et al'04] [Tripakis'09] [Li'09]

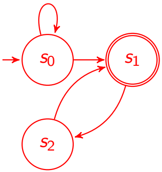
- k-extrapolation and LU-abstraction preserves reachability of locations
- k-extrapolation and LU-abstraction also preserve Büchi emptiness
- subsumption preserves reachability of locations as well



Zone abstraction



$s_3 \sqsubseteq s_1$



subsumption

Open problem posed in [Tripakis'09]

Is emptiness of Timed Büchi Automata preserved by subsumption? NO

Analysis of accepting cycles/spirals with subsumption

$\sqsubseteq$ is a simulation relation:

$$\begin{array}{ccc} s' & & \\ \sqcup\!\mid & & \\ s & \rightarrow & t \end{array}$$

Analysis of accepting cycles/spirals with subsumption

$\sqsubseteq$ is a simulation relation:

$$\begin{array}{ccc} s' & \rightarrow & t' \\ \sqcup\!\!| & & \sqcup\!\!| \\ s & \rightarrow & t \end{array}$$

$\sqsubseteq$ is a finite abstraction

Analysis of accepting cycles/spirals with subsumption

$\sqsubseteq$ is a simulation relation:

$$\begin{array}{ccc} s' & \rightarrow & t' \\ \sqcup\!\mid & & \sqcup\!\mid \\ s & \rightarrow & t \end{array}$$

$\sqsubseteq$ is a finite abstraction

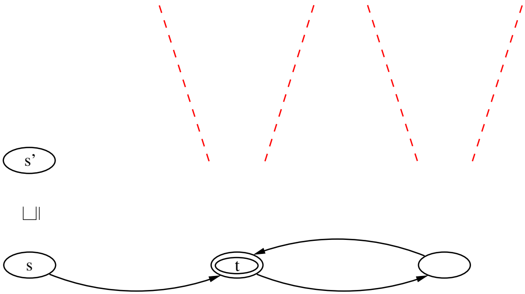
Lemma: If s has an accepting cycle then any $s' \sqsupseteq s$ has it as well
Lemma: If t' has an accepting spiral then t' has an accepting cycle

Analysis of accepting cycles/spirals with subsumption

$\sqsubseteq$ is a simulation relation:

$$\begin{tikzcd} s' \arrow[r] & t' \\ s \arrow[r] \arrow[u, phantom, sloped, "\sqsubseteq"] & t \arrow[u, phantom, sloped, "\sqsubseteq"] \end{tikzcd}$$

$\sqsubseteq$ is a finite abstraction



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

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

Preservation of accepting cycles — Proof Sketch

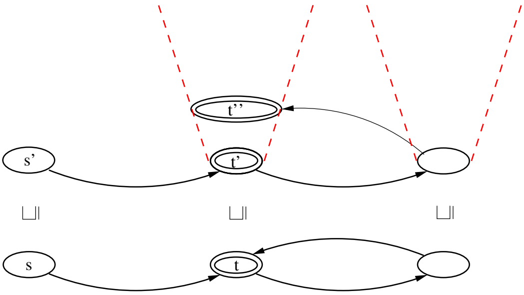
$$\begin{tikzcd} s' & & \\ s \arrow[u, phantom, sloped, "\sqsubseteq"] \arrow[r, "*"] & t \arrow[r, "+"] & t \end{tikzcd}$$

Analysis of accepting cycles/spirals with subsumption

$\sqsubseteq$ is a simulation relation:

$$\begin{array}{ccc} s' & \rightarrow & t' \\ \rotatebox[origin=c]{90}{$\sqsubseteq$} & & \rotatebox[origin=c]{90}{$\sqsubseteq$} \\ s & \rightarrow & t \end{array}$$

$\sqsubseteq$ is a finite abstraction



Lemma: If s has an accepting cycle then any $s' \sqsupseteq s$ has it as well
Lemma: If t' has an accepting spiral then t' has an accepting cycle

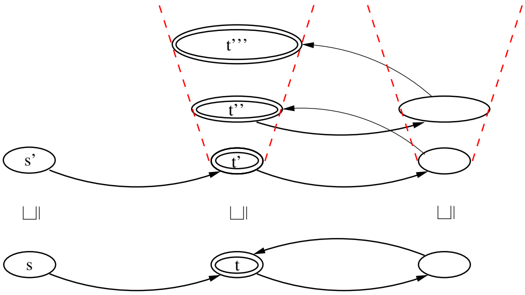
Preservation of accepting cycles — Proof Sketch

$$\begin{array}{ccccc} s' & \rightarrow^* & t' & \rightarrow^+ & t'' \\ \rotatebox[origin=c]{90}{$\sqsubseteq$} & & \rotatebox[origin=c]{90}{$\sqsubseteq$} & & \rotatebox[origin=c]{90}{$\sqsubseteq$} \\ s & \rightarrow^* & t & \rightarrow^+ & t \end{array}$$

Analysis of accepting cycles/spirals with subsumption

$\sqsubseteq$ is a simulation relation:

$$\begin{array}{ccc} s' & \rightarrow & t' \\ \sqsupseteq & & \sqsupseteq \\ s & \rightarrow & t \end{array}$$



$\sqsubseteq$ is a finite abstraction

Lemma: If s has an accepting cycle then any $s' \sqsupseteq s$ has it as well
Lemma: If t' has an accepting spiral then t' has an accepting cycle

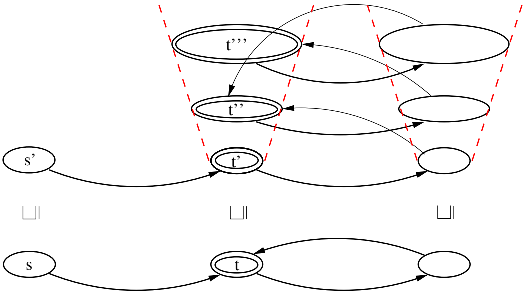
Preservation of accepting cycles — Proof Sketch

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

Analysis of accepting cycles/spirals with subsumption

$\sqsubseteq$ is a simulation relation:

$$\begin{array}{ccc} s' & \rightarrow & t' \\ \rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} \\ s & \rightarrow & t \end{array}$$



$\sqsubseteq$ is a finite abstraction

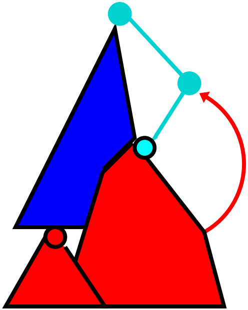
Lemma: If s has an accepting cycle then any $s' \sqsupseteq s$ has it as well

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

Preservation of accepting cycles — Proof Sketch

$$\begin{array}{ccccccccccc} s' & \rightarrow^* & t' & \rightarrow^+ & t'' & \rightarrow^+ & \cdots x \cdots & \rightarrow^+ & t''' & \rightarrow^+ & x \\ \rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} & & & & \rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} \\ s & \rightarrow^* & t & \rightarrow^+ & t & \rightarrow^+ & \cdots\cdots & \rightarrow^+ & t & \rightarrow^+ & t \end{array}$$

- Blue search: explore graph in DFS order
  - states on the blue search stack are cyan

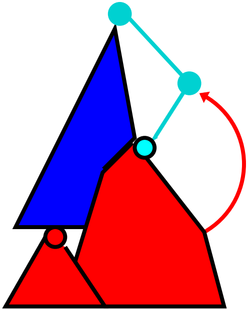


Blue search

1: **procedure** *dfsBlue*(s)
2: add s to *Cyan*

8: move s from *Cyan* to *Blue*

- Blue search: explore graph in DFS order
  - states on the blue search stack are cyan

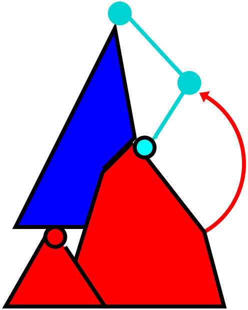


Blue search

procedure *dfsBlue*(s)
    add s to *Cyan*
    for all successors t of s **do**
        if $t \notin Blue \cup Cyan$ **then**
            dfsBlue(t)
    move s from *Cyan* to *Blue*

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:

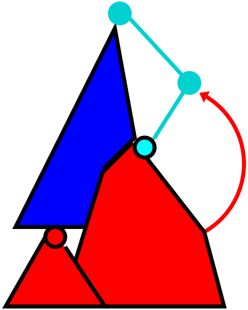


Blue search

procedure *dfsBlue*(s)
  add s to *Cyan*
  for all successors t of s **do**
    if $t \notin \mathit{Blue} \cup \mathit{Cyan}$ **then**
      dfsBlue(t)
  if s is accepting **then**
    dfsRed(s)
  move s from *Cyan* to *Blue*

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



Blue search

```
procedure dfsBlue(s)
    add s to Cyan
    for all successors t of s do
        if t ∉ Blue ∪ Cyan then
            dfsBlue(t)
        if s is accepting then
            dfsRed(s)
    move s from Cyan to Blue
```

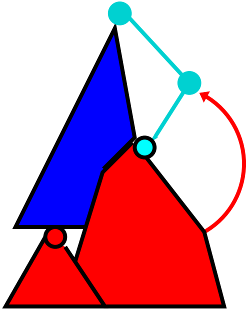
Red search

```
procedure dfsRed(s)
    add s to Red
    for all successors t of s do

            if t ∉ Red then
                dfsRed(t)
```

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



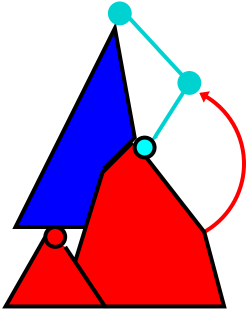
Blue search

procedure *dfsBlue*(s)
    add s to *Cyan*
    for all successors t of s **do**
        if $t \notin \mathit{Blue} \cup \mathit{Cyan}$ **then**
            dfsBlue(t)
        if s is accepting **then**
            dfsRed(s)
    move s from *Cyan* to *Blue*

Red search

procedure *dfsRed*(s)
    add s to *Red*
    for all successors t of s **do**
        if $t \in \mathit{Cyan}$ **then**
            Exit: cycle detected
        if $t \notin \mathit{Red}$ **then**
            dfsRed(t)

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

procedure *dfsBlue*(s)
    add s to *Cyan*
    for all successors t of s **do**
        if $t \notin \mathit{Blue} \cup \mathit{Cyan}$ **then**
            dfsBlue(t)
        if s is accepting **then**
            dfsRed(s)
    move s from *Cyan* to *Blue*

Red search

procedure *dfsRed*(s)
    add s to *Red*
    for all successors t of s **do**
        if $t \in \mathit{Cyan}$ **then**
            Exit: cycle detected
        if $t \notin \mathit{Red}$ **then**
            dfsRed(t)

Subsumption in Nested Depth First Search

Blue search find accepting states in post order

procedure *dfsBlue*(s)
  $Cyan := Cyan \cup \{s\}$
  for all successors t of s **do**
    if $t \notin Blue \cup Cyan$ **then**
      dfsBlue(t)
  if s is accepting **then**
    dfsRed(s)
  $Blue, Cyan := Blue \cup \{s\}, Cyan \setminus \{s\}$

Red search find cycles on accepting states

procedure *dfsRed*(s) Postcondition: no accepting spiral reachable
  $Red := Red \cup \{s\}$
  for all successors t of s **do**
    if $t \in Cyan$ **then**
      Exit: cycle detected
    if $t \notin Red$ **then**
      dfsRed(t)

Blue search — find accepting states in post order

procedure *dfsBlue*(s)
  $Cyan := Cyan \cup \{s\}$
  for all successors t of s **do**
    if $t \notin Blue \cup Cyan$ **then**
      dfsBlue(t)
  if s is accepting **then**
    dfsRed(s)
  $Blue, Cyan := Blue \cup \{s\}, Cyan \setminus \{s\}$

Red search — find cycles on accepting states

procedure *dfsRed*(s) Postcondition: no accepting spiral reachable
  $Red := Red \cup \{s\}$
  for all successors t of s **do**
    if $t \sqsupseteq Cyan$ **then** Accepting spiral found!
      Exit: cycle detected
    if $t \notin Red$ **then**
      dfsRed(t)

Subsumption in Nested Depth First Search

Blue search — find accepting states in post order

procedure *dfsBlue*(s)
  $Cyan := Cyan \cup \{s\}$
  for all successors t of s **do**
    if $t \notin Blue \cup Cyan$ **then**
      dfsBlue(t)
  if s is accepting **then**
    dfsRed(s)
  $Blue, Cyan := Blue \cup \{s\}, Cyan \setminus \{s\}$

Red search — find cycles on accepting states

procedure *dfsRed*(s) — Postcondition: no accepting spiral reachable
  $Red := Red \cup \{s\}$
  for all successors t of s **do**
    if $t \sqsupseteq Cyan$ **then** — Accepting spiral found!
      Exit: cycle detected
    if $t \not\sqsubseteq Red$ **then** — Spiral on t would give spiral from Red
      dfsRed(t)

Subsumption in Nested Depth First Search

Blue search — find accepting states in post order

procedure *dfsBlue*(s)
  $Cyan := Cyan \cup \{s\}$
  for all successors t of s **do**
    if $t \not\sqsubseteq Blue \cup Cyan$ **then** — This goes wrong, unfortunately!
      dfsBlue(t)
  if s is accepting **then**
    dfsRed(s)
  $Blue, Cyan := Blue \cup \{s\}, Cyan \setminus \{s\}$

Red search — find cycles on accepting states

procedure *dfsRed*(s) — Postcondition: no accepting spiral reachable
  $Red := Red \cup \{s\}$
  for all successors t of s **do**
    if $t \sqsupseteq Cyan$ **then** — Accepting spiral found!
      Exit: cycle detected
    if $t \not\sqsubseteq Red$ **then** — Spiral on t would give spiral from Red
      dfsRed(t)

Subsumption in Nested Depth First Search

Blue search — find accepting states in post order

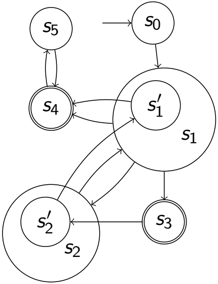
procedure $\mathit{dfsBlue}(s)$
  $\mathit{Cyan} := \mathit{Cyan} \cup \{s\}$
  for all successors t of s **do**
    if $t \notin \mathit{Blue} \cup \mathit{Cyan} \wedge t \not\sqsubseteq \mathit{Red}$ **then** — Prune the blue search
      $\mathit{dfsBlue}(t)$
  if s is accepting **then**
    $\mathit{dfsRed}(s)$
  $\mathit{Blue}, \mathit{Cyan} := \mathit{Blue} \cup \{s\}, \mathit{Cyan} \setminus \{s\}$

Red search — find cycles on accepting states

procedure $\mathit{dfsRed}(s)$ — Postcondition: no accepting spiral reachable
  $\mathit{Red} := \mathit{Red} \cup \{s\}$
  for all successors t of s **do**
    if $t \sqsupseteq \mathit{Cyan}$ **then** — Accepting spiral found!
      Exit: cycle detected
    if $t \not\sqsubseteq \mathit{Red}$ **then** — Spiral on t would give spiral from Red
      $\mathit{dfsRed}(t)$

Subsumption on Blue is Unsound

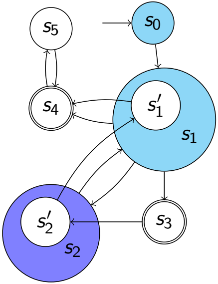
Assume we would backtrack on t as soon as $t \sqsubseteq Blue$:



Accepting cycle s_4–s_5 not detected

Subsumption on Blue is Unsound

Assume we would backtrack on t as soon as $t \sqsubseteq Blue$:

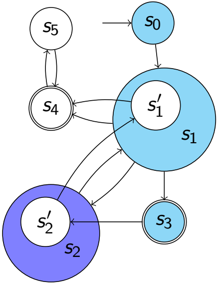


Accepting cycle s_4–s_5 not detected

- The blue search proceeds via $s_0, s_1, s_2,$

Subsumption on Blue is Unsound

Assume we would backtrack on t as soon as $t \sqsubseteq Blue$:

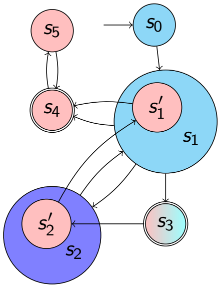


Accepting cycle s_4–s_5 not detected

- The blue search proceeds via s_0, s_1, s_2, then backtracks via s_1 to s_3
- Now since $s_2' \sqsubseteq Blue$, the blue search is pruned at s_3

Subsumption on Blue is Unsound

Assume we would backtrack on t as soon as $t \sqsubseteq Blue$:

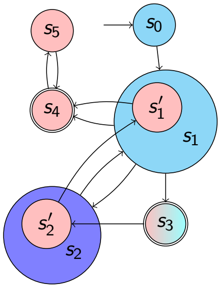


Accepting cycle s_4–s_5 not detected

- The blue search proceeds via s_0, s_1, s_2, then backtracks via s_1 to s_3
- Now since $s_2' \sqsubseteq Blue$, the blue search is pruned at s_3
- $s_3 \in Acc$, so a red search is started: s_3, s_2', s_1', s_4, s_5

Subsumption on Blue is Unsound

Assume we would backtrack on t as soon as $t \sqsubseteq Blue$:

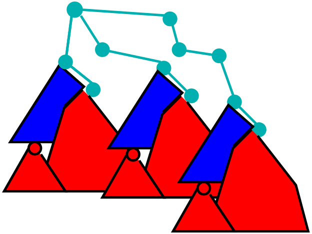


Accepting cycle s_4–s_5 not detected

- The blue search proceeds via s_0, s_1, s_2, then backtracks via s_1 to s_3
- Now since $s_2' \sqsubseteq Blue$, the blue search is pruned at s_3
- $s_3 \in Acc$, so a red search is started: s_3, s_2', s_1', s_4, s_5
- The only accepting cycle s_4–s_5 is erroneously made red
- Note: accepting states are not visited in post-order

Multi-core Nested DFS

[Laarman/Evangelista et al, ATVA'11,12]

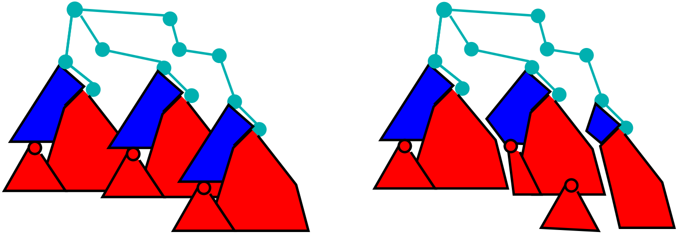


Parallel NDFS algorithm – shared hashtable

- Basic idea: *n* workers perform independent random NDF Search
  - Visited states are stored in a shared hashtable
  - All workers use their own separate set of colors
  - Speeds up bug hunting, what about full verification?
  - Better subsumption: visit larger states earlier due to BFS-effect

Multi-core Nested DFS

[Laarman/Evangelista et al, ATVA'11,12]

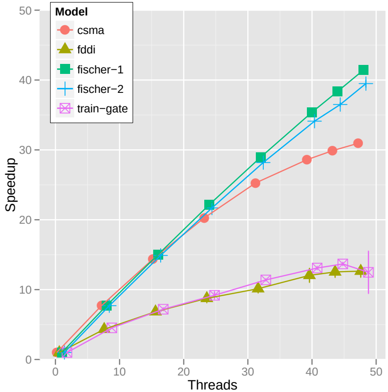


Parallel NDFS algorithm – shared hashtable

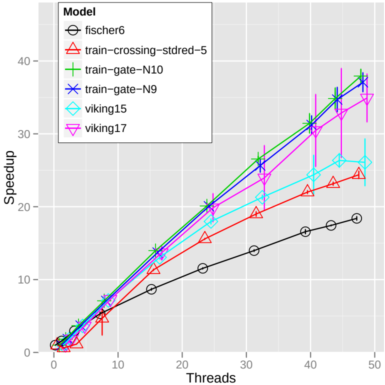
- Basic idea: *n* workers perform independent random NDF Search
  - Visited states are stored in a shared hashtable
  - All workers use their own separate set of colors
  - Speeds up bug hunting, what about full verification?
  - Better subsumption: visit larger states earlier due to BFS-effect
- Collaboration between NDFS workers
  - Share red and blue globally, workers keep their own cyan stack
  - Workers backtrack on parts finished by others
  - Complicated to restore post-order, reasonable scalability

Experiments: speedup up to 48 cores

Checking LTL on Timed Automata



BFS Reachability on Timed Automata



Experiments with OPAAL and LTSMIN – open source

hours ⟶ minutes ⟶ seconds

Multi-Core Reachability for Timed Automata, FORMATS'12

Conclusion

Contributions

- Subsumption in Timed Büchi Automata (open problem)
  - introduces spurious counter examples
  - preserves some structural properties

Conclusion

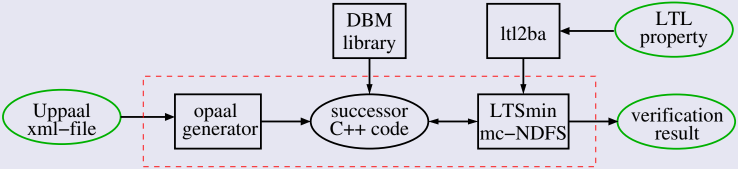
Contributions

- Subsumption in Timed Büchi Automata (open problem)
  - introduces spurious counter examples
  - preserves some structural properties
- Checking LTL properties for Uppaal timed automata
  - Use subsumption to prune Nested DFS where possible
  - Multi-core NDFS algorithm for Timed Büchi Automata

Conclusion

Contributions

- Subsumption in Timed Büchi Automata (open problem)
  - introduces spurious counter examples
  - preserves some structural properties
- Checking LTL properties for Uppaal timed automata
  - Use subsumption to prune Nested DFS where possible
  - Multi-core NDFS algorithm for Timed Büchi Automata



- Open source through OPAAL and LTSMIN
  - `opaal-modelchecker.com/`
  - `fmt.cs.utwente.nl/tools/ltsmin/`