Stubborn Sets with Frozen Actions

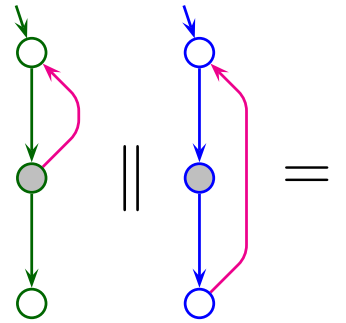
Antti Valmari

Tampere University of Technology
Mathematics

1 Aps Set Methods

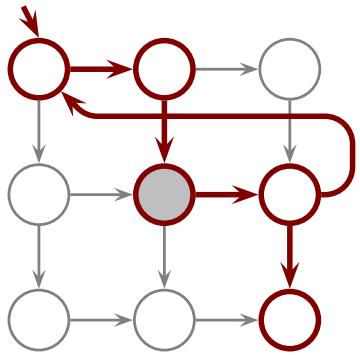
Aps set methods construct reduced state spaces by only firing a subset of enabled actions in each found state

- ample sets, persistent sets, stubborn sets
- widely but misleadingly called *partial order reduction*



Methods exist for various classes of properties

- deadlocks
- traces $=$ stuttering-insensitive safety
  - also preserves *fair testing* $\Rightarrow$ *may-progress*
- CSP failures–divergences semantics
- stuttering-insensitive linear temporal logic
- ...
- *the more properties are preserved, the less reduction is obtained*



Aps sets must satisfy certain abstract conditions

- more details on slide 4
- good algorithms for constructing such sets are known: $\rightsquigarrow_s$

The ignoring problem

2 The Famous Cycle Condition for Liveness

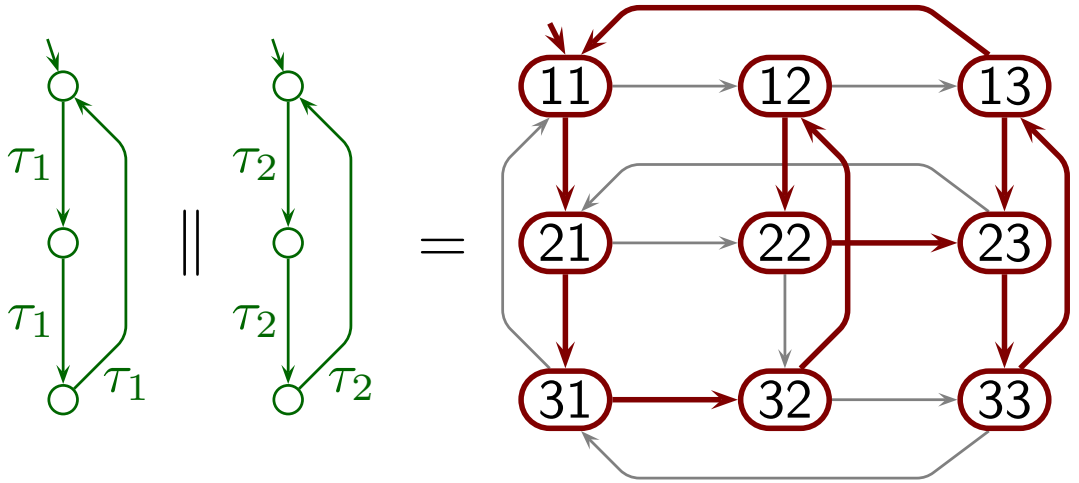
C3 Every r-cycle must contain a state s such that ample$(s) =$ en(s)

Implementation of C3 [Clarke & al. 1999]

- construct r-states and r-transitions in depth-first order
- if $a \in$ ample(s), $s \xrightarrow{a} s'$, and s' is in depth-first stack, choose ample$(s) =$ en(s)

A discouraging example

- try components from left to right
- sticking to a component helps a bit
  - [1999] does not tell to do so
  - fails badly with 3-dimensional case



This issue has received too little attention!

- observed in [Evangelista & Pajault 2010] (using another example)
- nobody knows how serious it really is
- we are not told how to deal with it

3 The Contribution of This Study

Two recent observations:

When preserving safety properties, solving the ignoring problem is seldom needed.

When, with safety properties, solving the ignoring problem is needed, earlier solutions tend to perform badly.

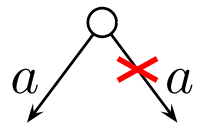
New contribution

We present a *freezing technique* that solves the above-mentioned performance problem when actions are deterministic.

We show that in the traditional process-algebraic setting with stubborn sets, actions can be considered deterministic for this purpose.

Deterministic actions

- for every s, s_1, and s_2, if $s \xrightarrow{a} s_1$ and $s \xrightarrow{a} s_2$, then $s_1 = s_2$

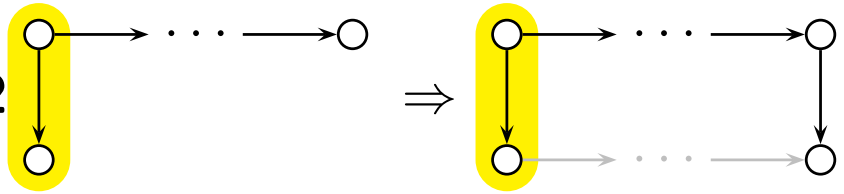


4 Stubborn Set Conditions for This Study

D1

- the first action from the stubborn set "moves to the front"

D2



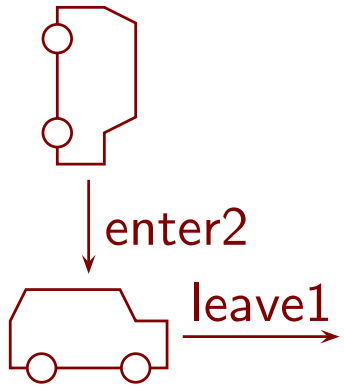
- outside actions cannot disable an enabled action in the stubborn set

V if stubb(s) contains an enabled visible action, then it contains all visible actions

- preserves the order of occurrences of visible actions, like leave1 and enter2 a crossing

D0V if there are enabled actions, then stubb(s) contains either an enabled action or all visible actions

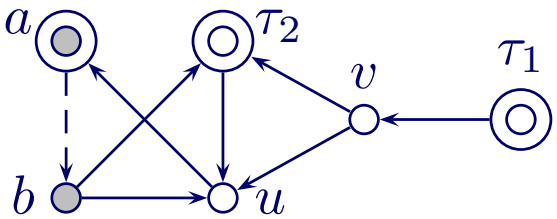
- the either-part "keeps the analysis going"
- the or-part implements the idea that if no visible action can be enabled in the future, then the future need not be investigated



S is a complicated condition that solves the ignoring problem but is hard to implement

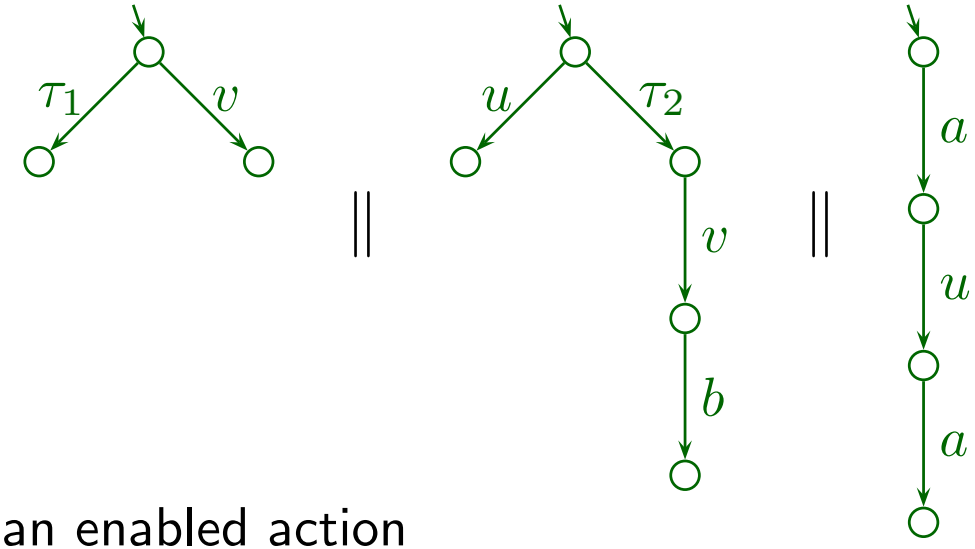
- its implementation is based on terminal strong components of the reduced state space

5 Construction of Stubborn Sets



$\rightsquigarrow_s \subseteq \text{Acts} \times \text{Acts}$

- if $a \notin \text{en}(s)$, choose i such that $\bar{L}_i$ disables a, and make $a \rightsquigarrow_s b$ for every $b \in \text{en}_i(s)$
- if $a \in \text{en}(s)$, then, for every i such that $a \in \bar{\Sigma}_i$ and for every $b \in \text{en}_i(s)$, make $a \rightsquigarrow_s b$
- it does not matter whether $a \rightsquigarrow_s a$



$\text{clsr}(s, a)$ = the closure of a w.r.t. "$\rightsquigarrow_s$"

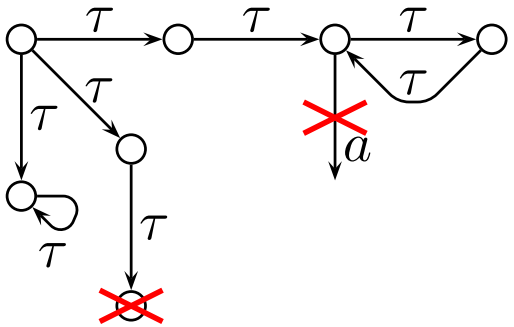
- satisfies **D1** and **D2**
- satisfies also **V** if $a \rightsquigarrow_s b$ is added for every $a \in \text{Vis} \cap \text{en}(s)$ and $b \in \text{Vis}$

$\text{gsc}(s, a, \ldots)$ = "good strong component"

- finds a $\subseteq$-minimal closed set that contains an enabled action or replies that such a set does not exist
- additional parameters tune it for future needs (may be called more than once in the same state)
- $O(|\rightsquigarrow|)$

6 With Safety, Solving the Ignoring is Seldom Needed

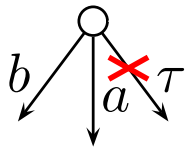
A state is only-diverging iff no deadlocks and no occurrences of visible actions can be reached from it



Theorem If a trace is lost using **D1** to **D0V**, then its prefix leads to a state that is only-diverging in the reduced state space.

⇒ ignoring a trace leaves an easily detectable symptom in the reduced state space

- however, the symptom does not necessarily imply that a trace was ignored



A state is stable iff it has no τ-transitions

Theorem If a trace leads to a stable state, then the trace is not lost using **D1** to **D0V**.

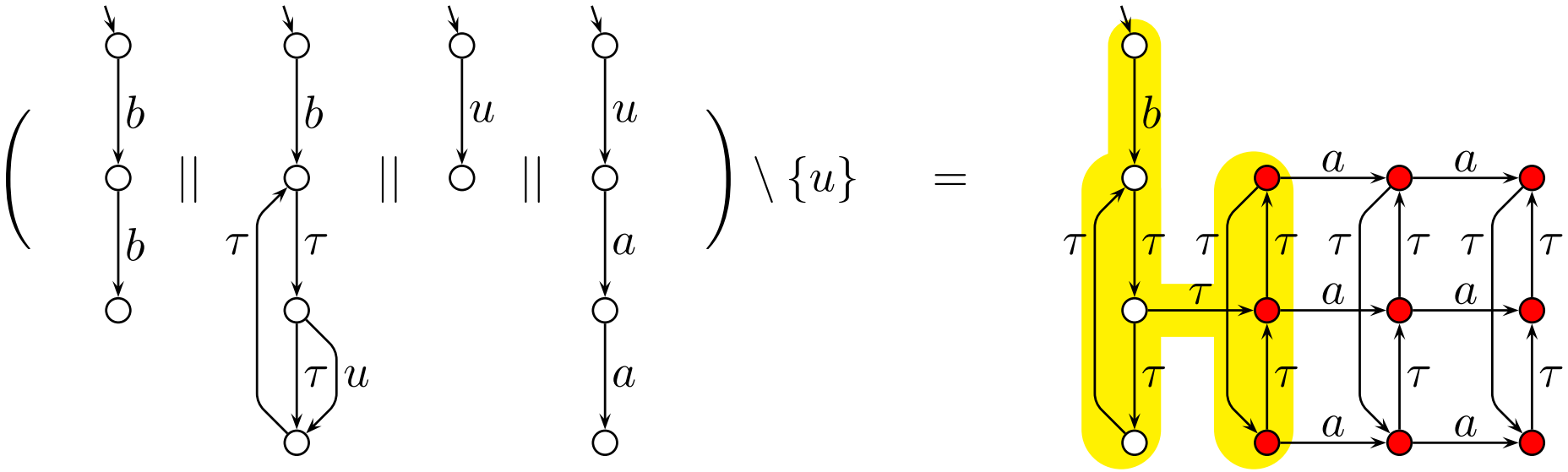
All prefixes of preserved traces are trivially preserved ⇒

- if a system always has the *possibility* of eventually **AG EF**
  - yielding output with no invisible alternative activity, or
  - stopping to wait for new input or for good,

  then all traces are preserved
- if not, we either see it from the reduced state space, or no traces are lost

⇒ **S** is not needed with most practical systems

7 A Problematic Example



Initially only b is enabled, then only τ is enabled, and then there are two possibilities

- $\tau\tau$ takes back to an earlier state
- u takes to a state, where aa is concurrent with the $\tau\tau\tau$-cycle
  - u becomes τ by "$\setminus\{u\}$"

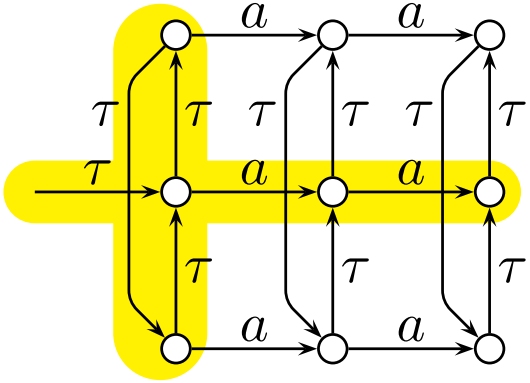
In each red state s, stubborn set construction goes

- $a \rightsquigarrow_s b$ by **V**
- $b \rightsquigarrow_s \tau$ (and $b \rightsquigarrow_s u$) by **D1**

$\Rightarrow$ the useless $\tau\tau\tau$-cycle is investigated

S forces to investigate a, but then no reduction is obtained

8 Frozen Actions – Idea



To avoid the performance problem on the previous slide, we freeze all actions in all stubborn sets of all useless cycles

- frozen actions are treated as if they did not exist
  - not fired
  - not taken into account in stubborn set construction
- when a new state is found, it inherits the frozen set of the previous state

Computation of stubb(s)

- in the first time, stubb(s) is computed as usual
- when it is the time to backtrack from s, the algorithm tests whether
  - s is a root of a terminal strong component of the reduced state space, and
  - that component does not contain an occurrence of a visible action
- if that holds, instead of backtracking, new actions are put to stubb(s)
  - also the expanded set must satisfy **D0V**, **D1**, **D2**, and **V** (excluding the frozen)
  - only actions that are relevant for enabling visible actions are considered: $\rightsquigarrow_s$, ...
  - if an enabled action can be added, do so, otherwise backtrack
- this implements **S**

9 Frozen Actions – Algorithm

DFS(s, old_frozen)

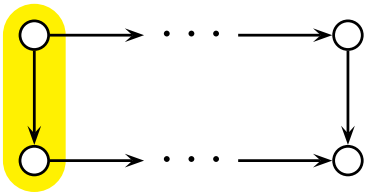
```
 1   S_r := S_r ∪ {s}
 2   new_frozen := old_frozen
 3   done := false
 4   while ¬ done do
 5       more_stubborn := compute or expand stubb(s, new_frozen)
 6       for a ∈ more_stubborn ∩ en(s) do
 7           for s' such that s −a→ s' do
 8               Δ_r := Δ_r ∪ {(s, a, s')}
 9               if s' ∉ S_r then DFS(s', new_frozen)
10       if   more_stubborn ∩ en(s) = ∅
11           ∨  s is not a root of a terminal strong component of (S_r, Δ_r, ŝ)
12           ∨  ∃s' ∈ R_r(s) : stubb(s') ∩ V ∩ en(s') ≠ ∅
13       then done := true
14       else new_frozen := new_frozen ∪ stubb(R_r(s))
```

Depth-first search -based reduced state space construction with additions

$\mathcal{R}_r(s)$ = the states that are r-reachable from s

stubb(X) = $\bigcup_{x \in X}$ stubb(x)

10 Frozen Actions – Correctness Proof

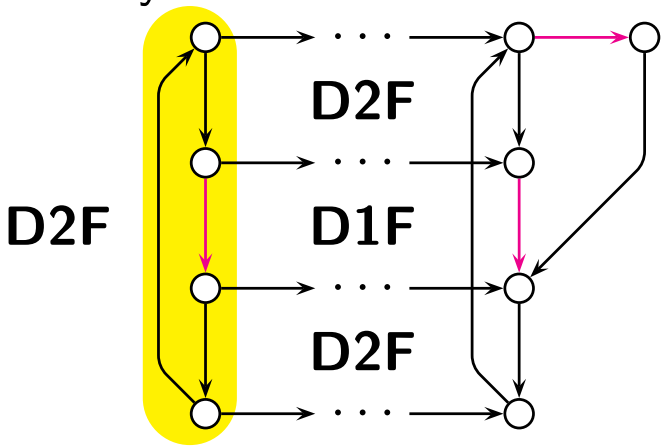


D1 and **D2** are replaced by **D1F** and **D2F** with neither the horizontal nor the vertical actions frozen

- the proof that **D1** to **S** preserve all traces goes through with them

Invariant property: if $s \,{-}\sigma{\rightarrow}$ in the full state space and σ_s is obtained from σ by removing all frozen elements, then $s \,{-}\sigma_s{\rightarrow}$ in the full state space

- the proof is by induction along the path via which s was first found
- in the initial state nothing is frozen, so the claim holds trivially
- assume that s was first found via $z \,{-}a{\rightarrow}\, s$
  $\Rightarrow$ a was not frozen in z
- if $s \,{-}\sigma{\rightarrow}$, then $z \,{-}a\sigma{\rightarrow}$
- by the induction assumption $z \,{-}a\sigma_z{\rightarrow}$
- if s has additional frozen actions, they were in the stubborn sets of a strong component with root z
  $\Rightarrow$ they can be removed by traversing in the component, by **D1F** and **D2F**



$\Rightarrow z \,{-}a\sigma_s{\rightarrow}$

- because actions are deterministic, $z \,{-}a{\rightarrow}\, s \,{-}\sigma_s{\rightarrow}$

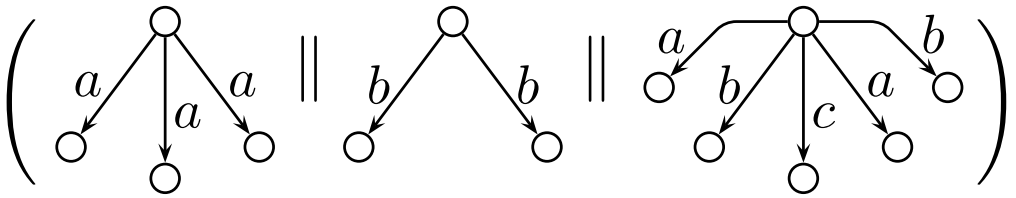
11 Restoring Nondeterministic Actions

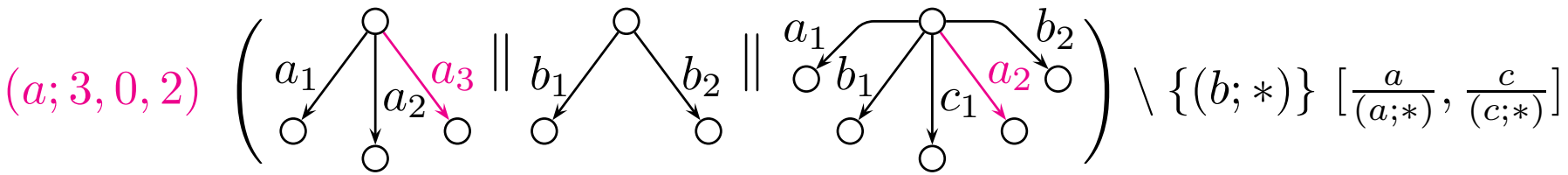
Stubborn sets for process algebras deal with systems of the form

$$(L_1 \,||\, \cdots \,||\, L_N) \setminus H$$

- the L_i are LTSs, $||$ is parallel composition, H is a set, and $\setminus$ denotes hiding

Recording the cause of an action makes it deterministic





The added information does not affect the construction of stubborn sets

$\Rightarrow$ The freezing algorithm applies to parallel compositions of nondeterministic LTSs

12 Conclusion

Good old conditions for safety and liveness are worse than we have thought for 25 years

- this applies to aps sets / partial order reduction in general, not just stubborn sets

With safety

- most of the time the condition is not needed in the end!
- the remaining cases are exceptionally nasty
- we solved them in the present study
- our solution is less elegant than we hoped
  - nondeterminism was dealt with by trickery

  $\Rightarrow$ smaller application scope, fortunately wide enough
  - it would be better to capture the relevant feature similarly to how **D1F**, etc., do
- implementation and experiments ...

With liveness, the corresponding work has not yet been done

- fortunately, the safety method applies to fair testing $\Rightarrow$ many liveness properties

Thank you for attention! Questions?