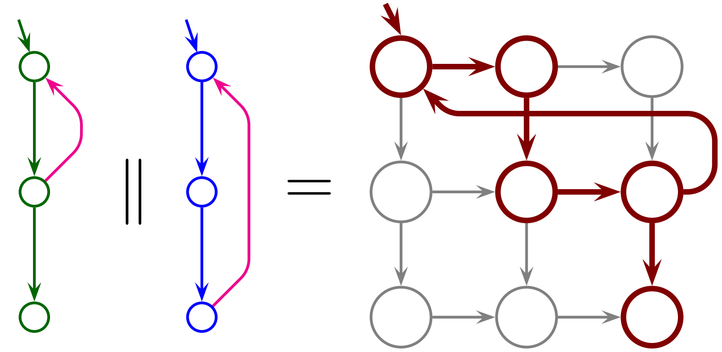
More Stubborn Set Methods for Process Algebras

Congratulations to Bill!

Antti Valmari

Tampere University of Technology
Department of Mathematics

1 Introduction



Stubborn set methods construct reduced state spaces preserving certain properties

Abbreviated history

- basic stubborn sets, algorithm 1988
  - deadlocks
  - $\exists$ inf. execution
- safety properties 1989
- linear temporal logic 1990
- nondeterministic actions 1992

$\Rightarrow$ Stubborn Set Methods for Process Algebras 1996
  - stable failures
  - CSP failures divergences
  - traces
  - chaos-free failures diverg.s
  - branching bisimilarity

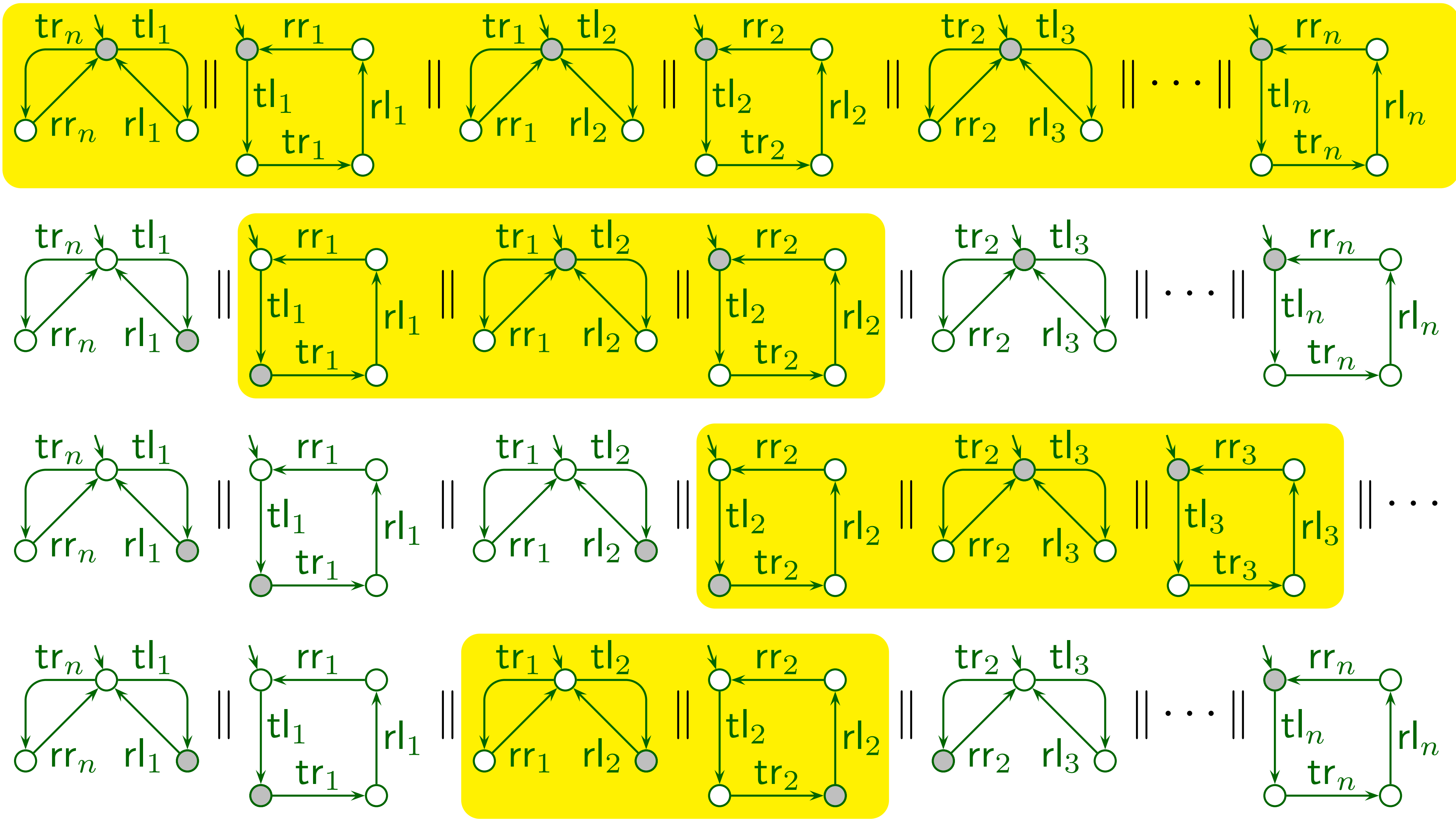
- persistent sets 1990, ample sets 1993

This talk: new results 2015 – 2017

- fair testing equivalence, tree failures $\Rightarrow$ ordinary failures
- an improvement to the terminal strong component condition for traces and fair t.
  - visibility-driven stubborn sets
- automata-theoretic visibility
- how to avoid the costly cycle / terminal sc conditions in many cases
- “remembering” divergencs instead of re-constructing them in later states

2 Intuition of Stubborn Sets

n dining philosophers



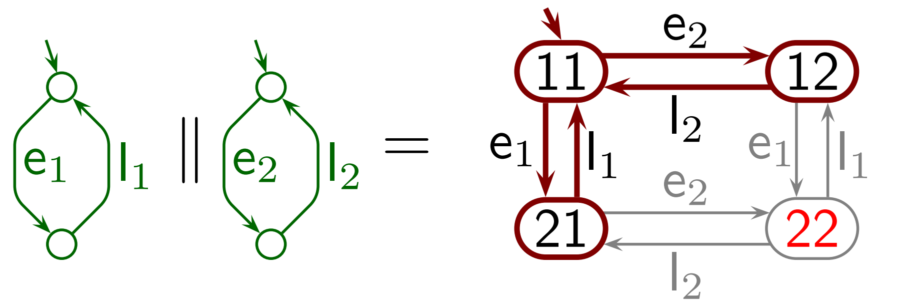
$3^n - 1 \rightsquigarrow 3n^2 - 3n + 2$ states $\rightsquigarrow 3n - 1$ with also symmetries

3 Why More Than One Method?

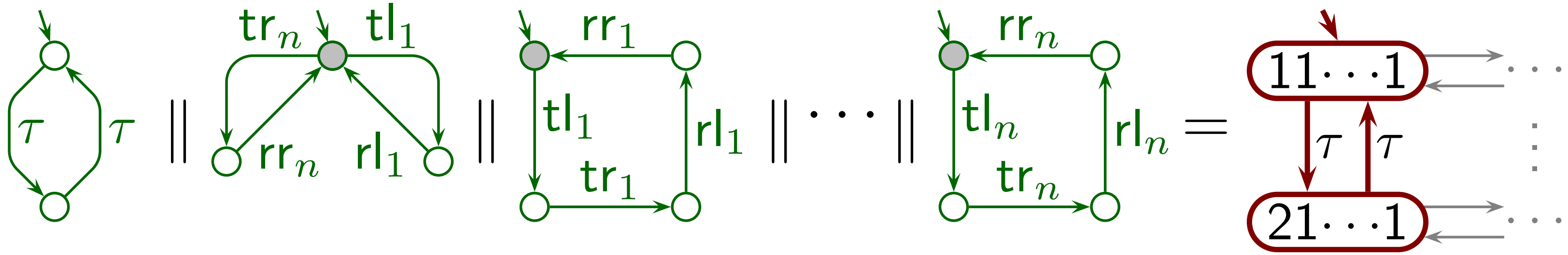
The basic method preserves all deadlocks and yes/no to “are there infinite executions?”

Not more because

1. the ordering of concurrent events may be important to the property
   - let e_i and l_i denote entering and leaving a critical section
   - this problem is rather mild
   - one of the new results addresses this directly



2. there is the ignoring problem



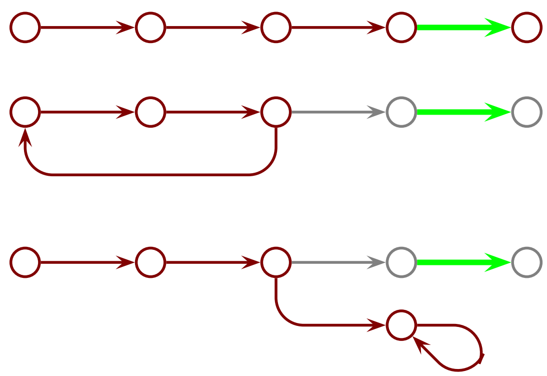
   - the basic method may terminate after investigating just the τ-cycle
   - this is a tough and versatile one!
   - two of the new results address this directly

4 Tree Failures and Fair Testing Equivalence

Fair Testing Equivalence [Vogler 1992, Rensink & Vogler 2007]

- is the weakest congruence that preserves ordinary (i.e., not necessarily stable) failures
- is the weakest congruence that preserves **AG EF** a
  - a canonical example of a branching-time property
- the only congruences below it are the traces, the alphabet, and the dullest
- facilitates checking a useful notion of progress without making fairness assumptions

About its definition



- tree failure: like ordinary, but a set of strings is refused
- tree failure equivalence compares them and the alphabets
- fair testing equivalence allows one system to “enter” the refusal set of the other system and refuse the suffixes

[Valmari & Vogler 2016]: the traditional method that preserves the traces also preserves fair testing

- a useful notion of progress for no additional cost
- surprising, because branching-time

[2017]: A small variant preserves also tree failures

5 System Model

$$(\bar{L}_1 \,||\, \cdots \,||\, \bar{L}_N) \;\setminus\; (H \cup \{\tau_1, \ldots, \tau_N\})$$

where

- $\bar{L}_i$ are τ-renamed labelled transitions systems $(S_i, \bar{\Sigma}_i, \bar{\Delta}_i, \hat{s}_i)$
  - τ_i is replaced for τ

$\Rightarrow$ each action belongs to a unique set of components

- Vis $=$ $(\Sigma_1 \cup \cdots \cup \Sigma_N) \setminus H$
- Inv $=$ $H \cup \{\tau_1, \ldots, \tau_N\}$
- Acts $=$ Vis $\cup$ Inv

Additional notation

- $s \,{-}a_1 \cdots a_n{\rightarrow}\; s'$ path from s to s' with also invisible actions listed
- $s \,{=}a_1 \cdots a_n{\Rightarrow}\; s'$ path from s to s' with invisible actions not listed
- $s \,{-}a_1 \cdots a_n{\rightarrow}$ there is s' such that $s \,{-}a_1 \cdots a_n{\rightarrow}\, s'$
- $s \,{-}\tau^\omega{\rightarrow}$ s diverges
- en(s) $=$ $\{a \mid s \,{-}a{\rightarrow}$ in full system$\}$ $=$ enabled actions
- en$_i(s)$ $=$ $\{a \mid s \,{-}a{\rightarrow}$ in $\bar{L}_i\}$

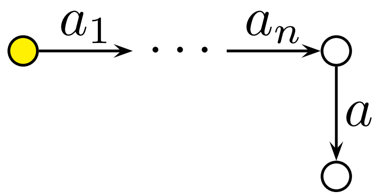
6 Definition of Stubborn Sets

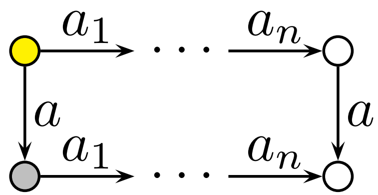
Reduced LTS: r-states, r-transitions, r-paths, ...

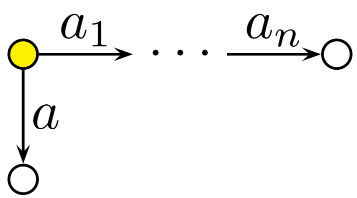
- put $\hat{s}$ in S_r
- for each $s \in S_\mathsf{r}$, construct $\mathcal{A}(s) \subseteq$ Acts
  - stubborn set
  - for each a and s', if $a \in \mathcal{A}(s)$ and $s \,{-a\!\!\rightarrow}\, s'$, put s' in S_r and (s, a, s') in $\bar{\Delta}_\mathsf{r}$

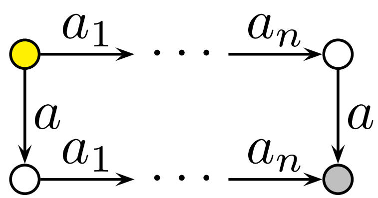
Depending on the method, $\mathcal{A}(s)$ must satisfy **D1**, (variant of) **D2**, and other conditions

D0 if there are enabled actions, $\mathcal{A}(s)$ must contain at least one

D1 $a \in \mathcal{A}(\circ)$, $a_i \notin \mathcal{A}(\circ)$ and 

$\Rightarrow$ there is $\circ$ such that 

D2 $a \in \mathcal{A}(\circ)$, $a_i \notin \mathcal{A}(\circ)$ and 

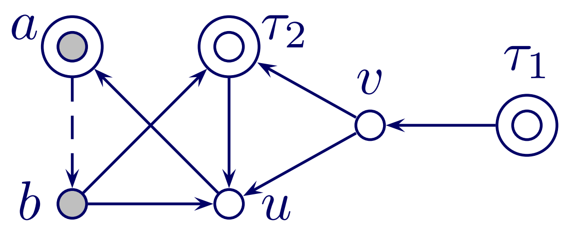
$\Rightarrow$ there is $\circ$ such that 

V if $\mathcal{A}(s)$ contains an enabled visible action, then Vis $\subseteq \mathcal{A}(s)$

I if there are enabled invisible actions, $\mathcal{A}(s)$ must contain at least one

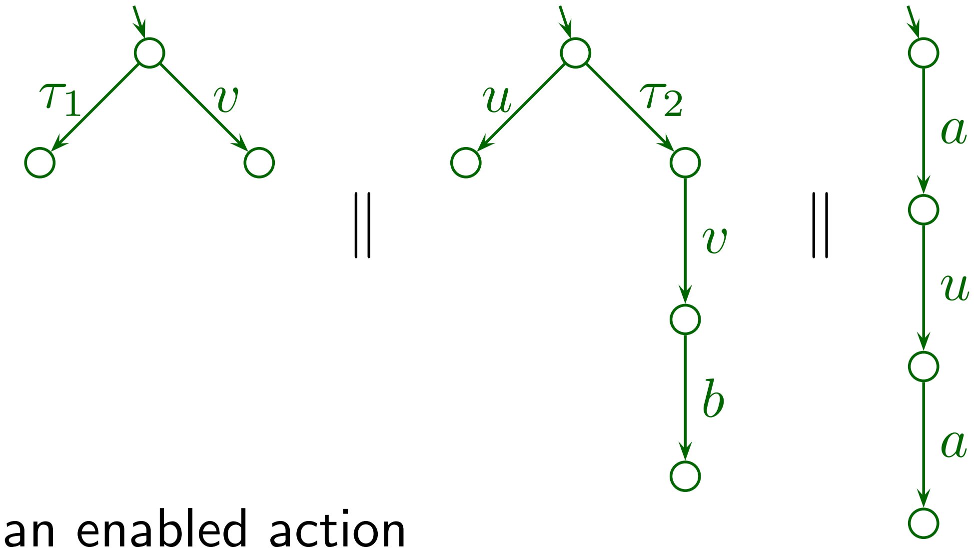
some condition for preventing ignoring: **Sen**, **SV**, **L**, **C3**

7 Construction of Stubborn Sets



"$\leadsto_s$" $\subseteq$ Acts $\times$ Acts

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



clsr(s, a) $=$ the closure of a w.r.t. "$\leadsto_s$"

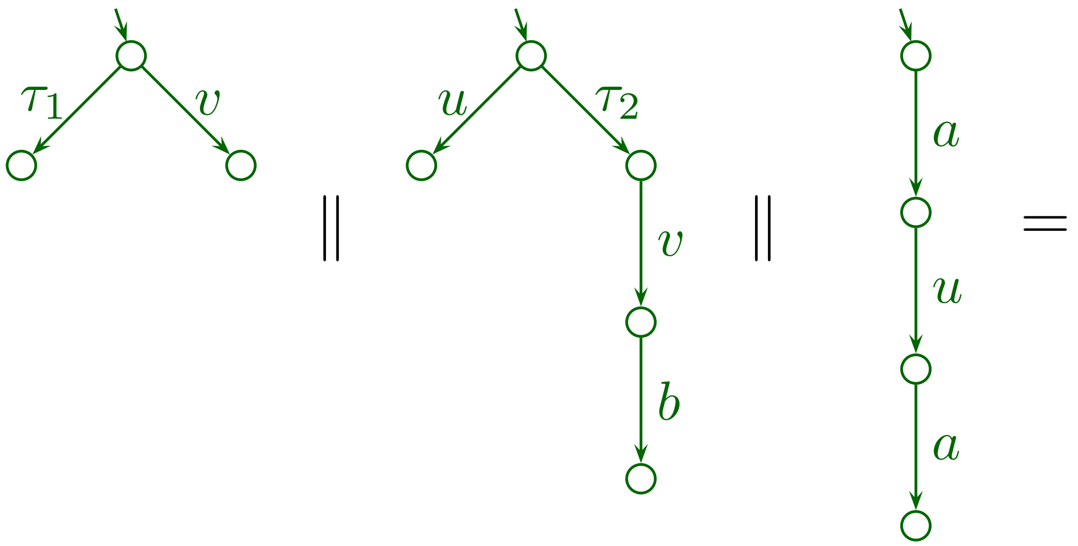
- satisfies **D1** and **D2**
- satisfies also **V** if $a \leadsto_s b$ is added
  for every $a \in$ Vis $\cap$ en(s) and $b \in$ Vis

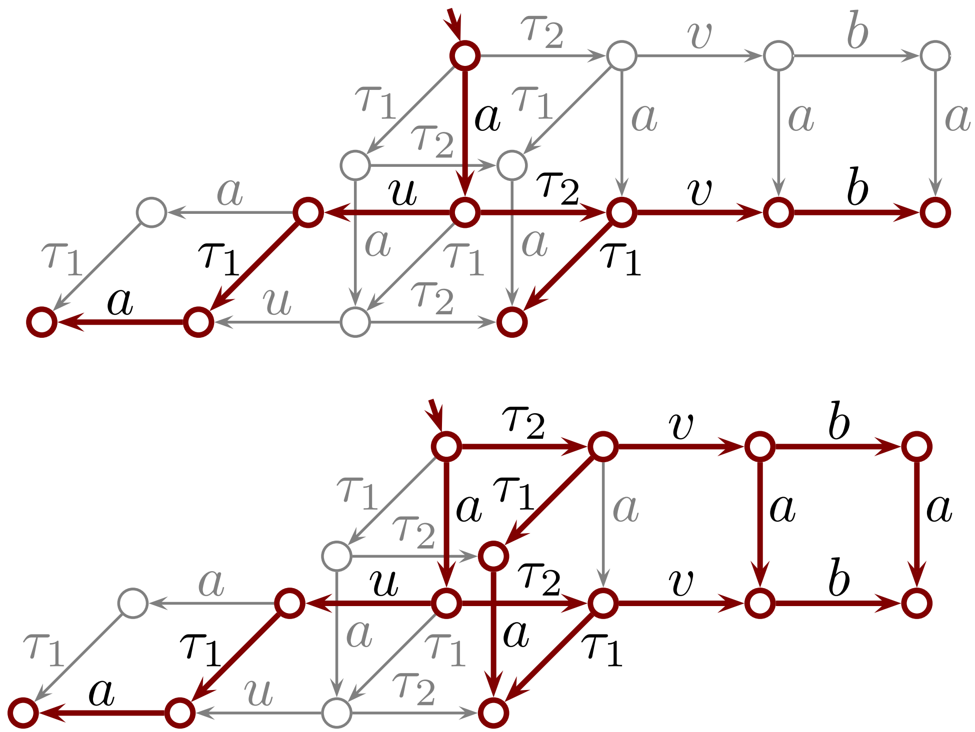
gsc$(s, a, \ldots)$ $=$ "good strong component"

- 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)
- $O(|\leadsto|)$

8 Why Disabled Actions in Stubborn Sets?

Only **D0**, **D1**, and **D2** are obeyed, starting points from left to right





Also **V** is obeyed, where $V = \{a, b\}$

- although initially $a \in V$ is taken, τ_1 is not

$\Rightarrow$ **V** is better than **C2** in ample set theory
  - if ample(s) contains an enabled visible action, then ample(s) = en(s)

$\Rightarrow$ Allowing disabled actions in stubborn sets facilitates formulation of better conditions

- and better algorithms: $\leadsto_s$, gsc

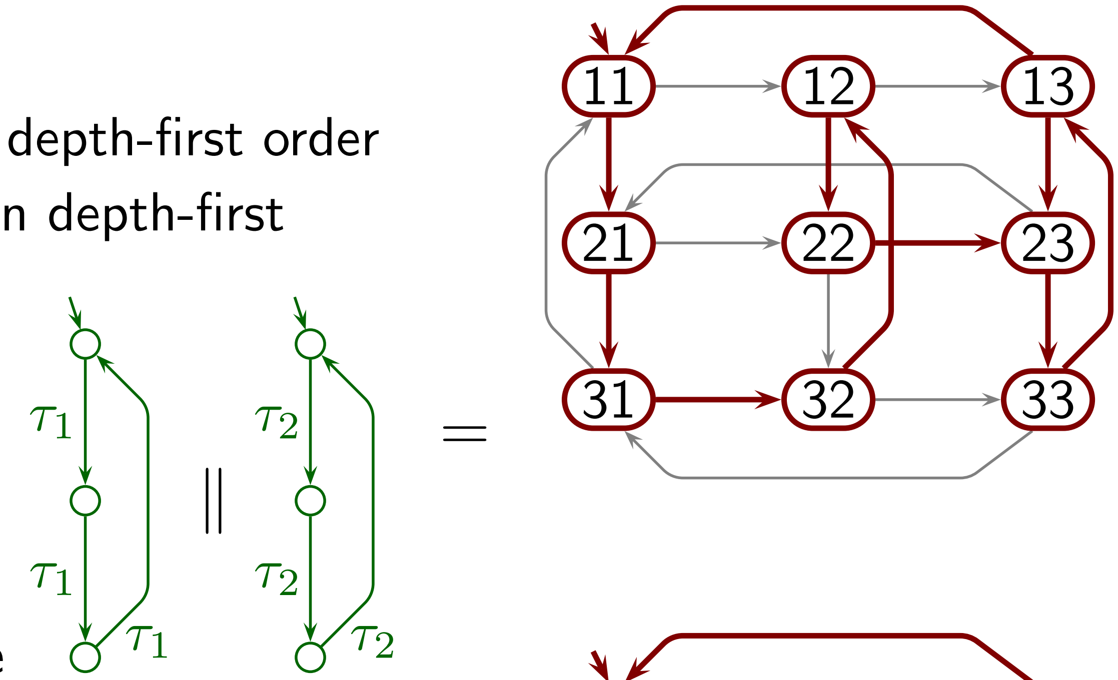
9 The Famous Cycle Condition for Liveness

L For every $a \in \mathsf{Vis}$, every r-cycle must contain a state s such that $a \in \mathcal{A}(s)$

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

Implementation of C3 [Clarke & al. 1999]

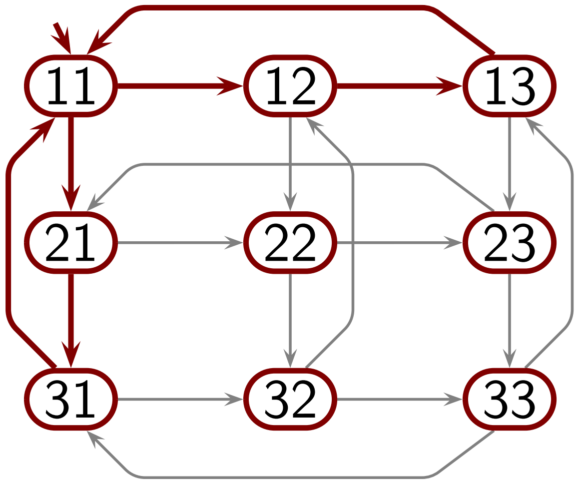
- construct r-states and r-transitions in depth-first order
- if $a \in \mathsf{ample}(s)$, $s -a\rightarrow s'$, and s' is in depth-first stack, choose $\mathsf{ample}(s) = \mathsf{en}(s)$



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

We can expand s' instead (Theorem: DFS), seems better

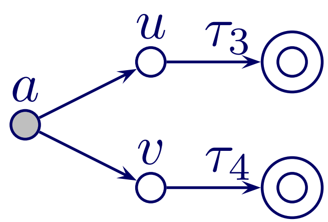


This issue has received too little attention!

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

10 Terminal Strong Component Conditions for Safety

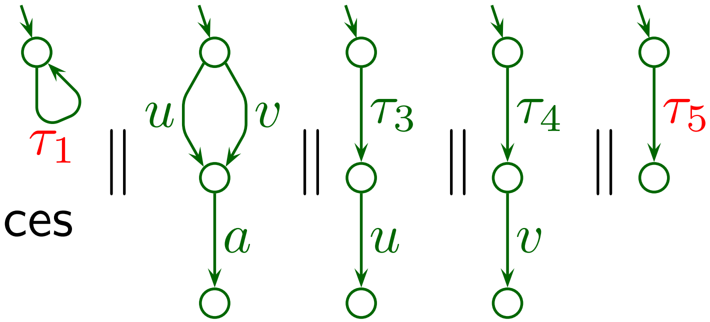
Sen For each r-terminal strong component C and each $a \in \mathsf{en}(r)$ where r is the root of C, there is $s \in C$ such that $a \in \mathcal{A}(s)$



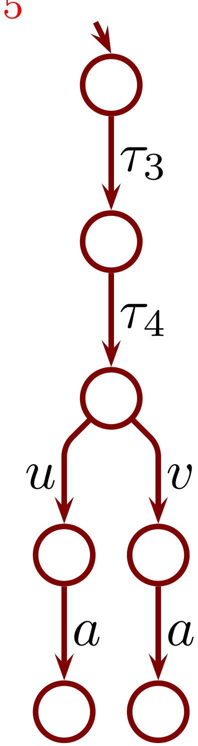
SV For each r-terminal strong component C and each $a \in \mathsf{Vis}$, there is $s \in C$ such that $a \in \mathcal{A}(s)$

- implemented efficiently using depth-first order and Tarjan's algorithm

Each suffers from a problem



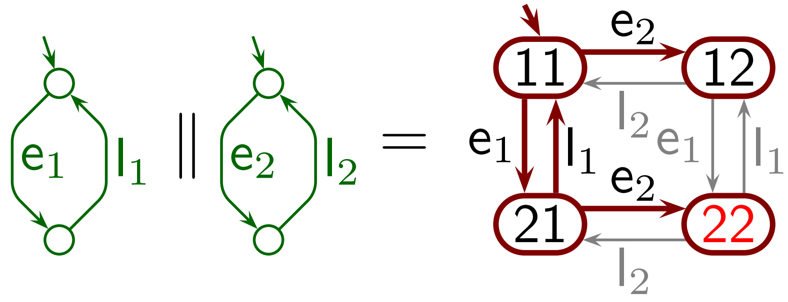
- **Sen** may force to fire irrelevant actions
  - e.g., when $\mathsf{clsr}(\mathsf{Vis}) = \emptyset$ when preserving traces
- **SV** may yield big stubborn sets



[Valmari & Hansen 2016]: The advantages of **Sen** and **SV** can be combined

- consider extending the root r of a terminal strong component
- compute an "enabling-closed" upper approximation $\mathsf{VIS}(r)$ of Vis
  - if $a \in \mathsf{VIS}(r) \setminus \mathsf{en}(r)$, choose i such that $\bar{L}_i$ disables a, and add every $b \in \mathsf{en}_i(r)$ to $\mathsf{VIS}(r)$
  - try $\mathsf{gsc}(r, a, \ldots)$ for each $a \in \mathsf{en}(r) \cap \mathsf{VIS}(r)$ until r is no longer a root or $\mathsf{VIS}(r)$ is exhausted

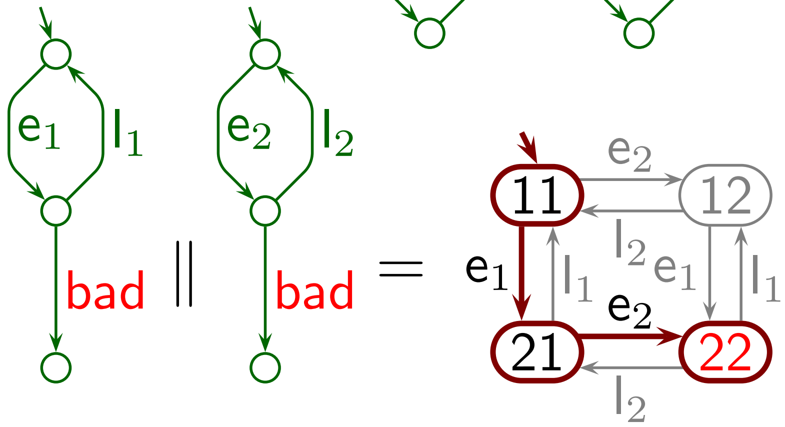
11 Automata-Theoretic Visibility



Assume that bad states are recognized immediately

With the trace-preserving method

- $\mathsf{Vis} = \{\mathsf{e}_1, \mathsf{e}_2, \mathsf{l}_1, \mathsf{l}_2\}$
- **V** forces $\mathsf{e}_1 \rightsquigarrow_{\hat{s}} \mathsf{e}_2 \rightsquigarrow_{\hat{s}} \mathsf{e}_1$



Using bad to drive analysis

- $\mathsf{e}_1 \not\rightsquigarrow_{\hat{s}} \mathsf{e}_2 \not\rightsquigarrow_{\hat{s}} \mathsf{e}_1$

$\Rightarrow$ smaller reduced LTS

$\Rightarrow$ Property- and state-dependent "incomplete" visibility suffices

Finite automata

- sufficiently keep track of the state, catch errors on-the-fly
- block irrelevant branches of the LTS
- problem: "if $a \in \mathsf{en}(s)$, ... for every $b \in \mathsf{en}_i(s)$, make $a \rightsquigarrow_s b$" acts like **V**

[Hansen & Valmari 2016]: the automaton need not cause $a \rightsquigarrow_s b$, if every error-detecting path starting with b in it remains error-detecting if a is added or moved to its front

- implemented by pre-processing the automaton

12 Insight on the Ignoring Problem

Infinite executions

- if traces are preserved and the reduced LTS is finite, then infinite traces are preserved
  - König's Lemma type of reasoning
- **I** (with **D0**, **D1**, **D2b**, and **V**) suffices for preserving minimal divergence traces

$\Rightarrow$ the only problematic infinite executions are non-minimal divergence traces
  - nothing better is known than cycle conditions

- nothing is known for fairness assumptions

Finite executions

- **D0** or gsc(Vis) (with **D1**, **D2r**, and **V**) suffices for traces that lead to stable states
  - gsc($s, a, \ldots$) for each $a \in$ Vis until an enabled action is found or Vis is exhausted
- the only problematic case is permanently diverging states
  - s is permanently diverging iff every state reachable from s is diverging
  - that is, no stable state is reachable from s

$\Rightarrow$ without **S** and **L**,
  - either all traces (and fair testing and ...) are preserved,
  - or the method tells that the system may reach a permanently diverging state

$\Rightarrow$ if we want the system not to have such states, **S** is not needed

13 Avoiding Terminal Strong Component Conditions

So the following works for traces, fair testing equivalence, and so on

- construct a reduced LTS using gsc(Vis)
- if the result contains a terminal strong component that is not a deadlock and does not contain occurrences of visible actions, fix the system and try again

It seems that the actions in the component could be permanently frozen and gsc(Vis) executed again, but this is future research

For the time being: Stop It, and Be Stubborn! [Valmari 2015]

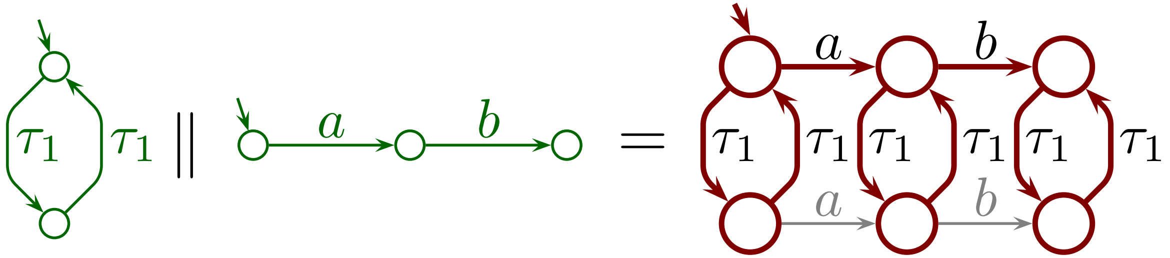
- add stop-transitions to terminal states to chosen states of chosen $\bar{L}_i$
  - they model clients deciding not to request for service
  - often this has to be done anyway, to not lose certain progress errors
- compute the reduced LTS
- if it contains a terminal sc that is not a deadlock, go back to first step
  - or of the above kind
- otherwise remove its stop-transitions

Deadlocks automatically guarantee **Sen** and **SV**

This trick works also for some liveness properties

14 Remembering Divergences

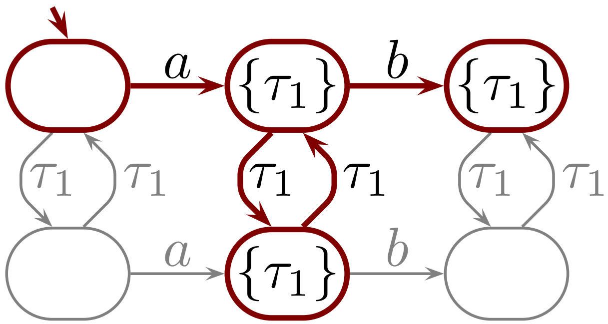
I may force to construct "the same" divergence many times



Solution (new):

- obey **D0**, **D1**, **D2**, **V**, and **L**
  - **D0** and **D2** may be weakened a bit
- store the union of $\mathcal{A}(s) \cap \mathsf{en}(s)$ of the states of the cycle to each state of the cycle
  - a state may have many div-sets
  - (or, to simplify implementation, at most one)
- when constructing a new r-state s' via $s -a\rightarrow s'$, copy from s the div-sets that lack a
- apply **I** on states where divergence matters that have no div-sets

By **D2**, each state with div-sets diverges



Example

- assume divergence matters always after a
- it is remembered after b

15 Discussion

The conditions in the present talk have the following roles:

- permutation correctness conditions: **D1**, **V**
- idling correctness conditions: variants of **D2**
- driving force: **D0**, the new **S**, **I**
- useful progress guarantee: variants of **S**, **L**

Most results of the talk are about avoiding progressing in a useless direction

- save from some state sub-explosions

The fair testing (and tree failures) result provides a useful fairness notion with surprisingly little additional cost

- the first realistic ample / persistent / stubborn set method for fairness
- surprising step towards branching time

Thank you for attention!
Questions?