

Partial-Order Reduction for General State Exploring Algorithms

Dragan Boanački

Eindhoven University of Technology

Stefan Leue

University of Konstanz

Alberto Lluch Lafuente

Empoli

Main Messages

- " **Partial order reduction for General State Exploring Algorithms**
Covers **Depth First Search, Breadth First Search** and different **Directed Search Heuristics**
Crucial novelty: **new cycle proviso**
Implementation in **HFS Spin**, an extension of **Spin** for **directed model checking**, with encouraging results

Model Checking

System

S

satisfies

p

property

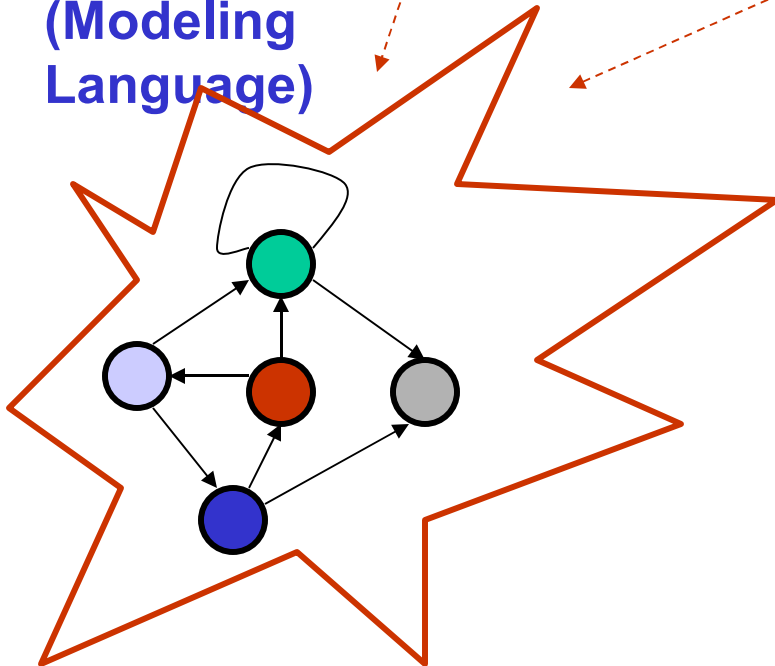
Model

M

Formal property

(Modeling
Language)

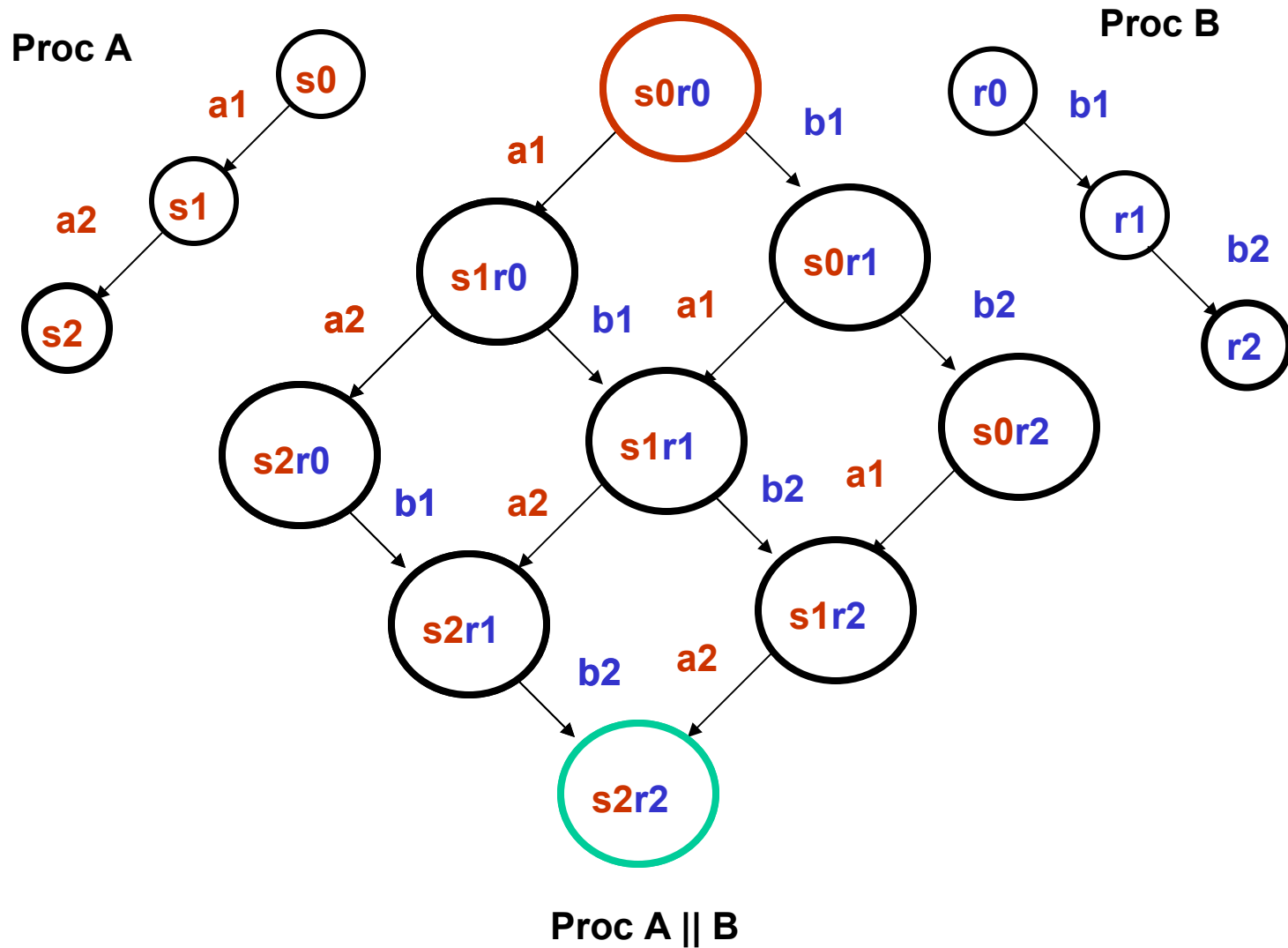
(Temporal
Logic)



State space explosion

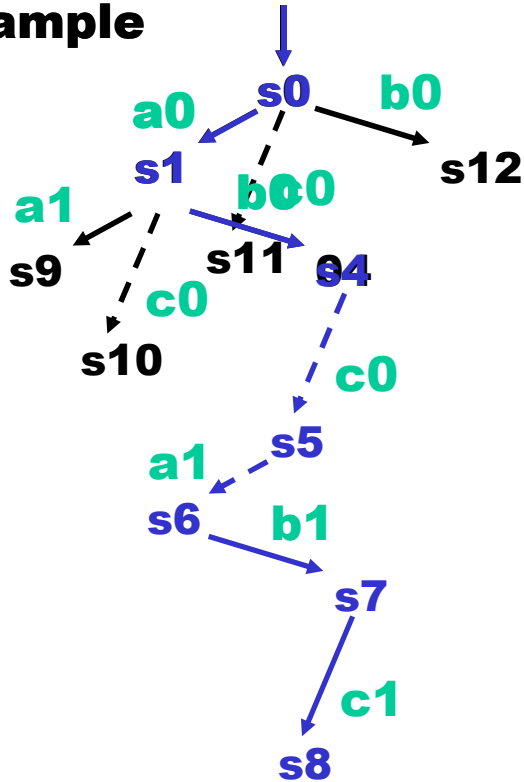
- reduction techniques needed

Partial Order Reduction



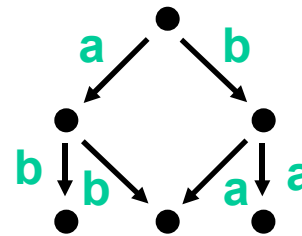
Partial Order Reduction

reduction function r assigns to a state the set of actions being considered
example (reduced action set)



partial-order reduction
exploits independence of actions
of concurrent processes

independence



restrictions

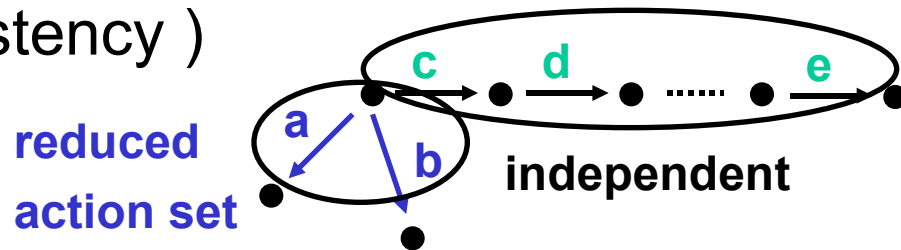
needed to guarantee that reduction preserves
properties of interest

Partial-Order Reduction

restrictions on reduced action set

C0: empty iff no actions enabled

C1 (persistency)

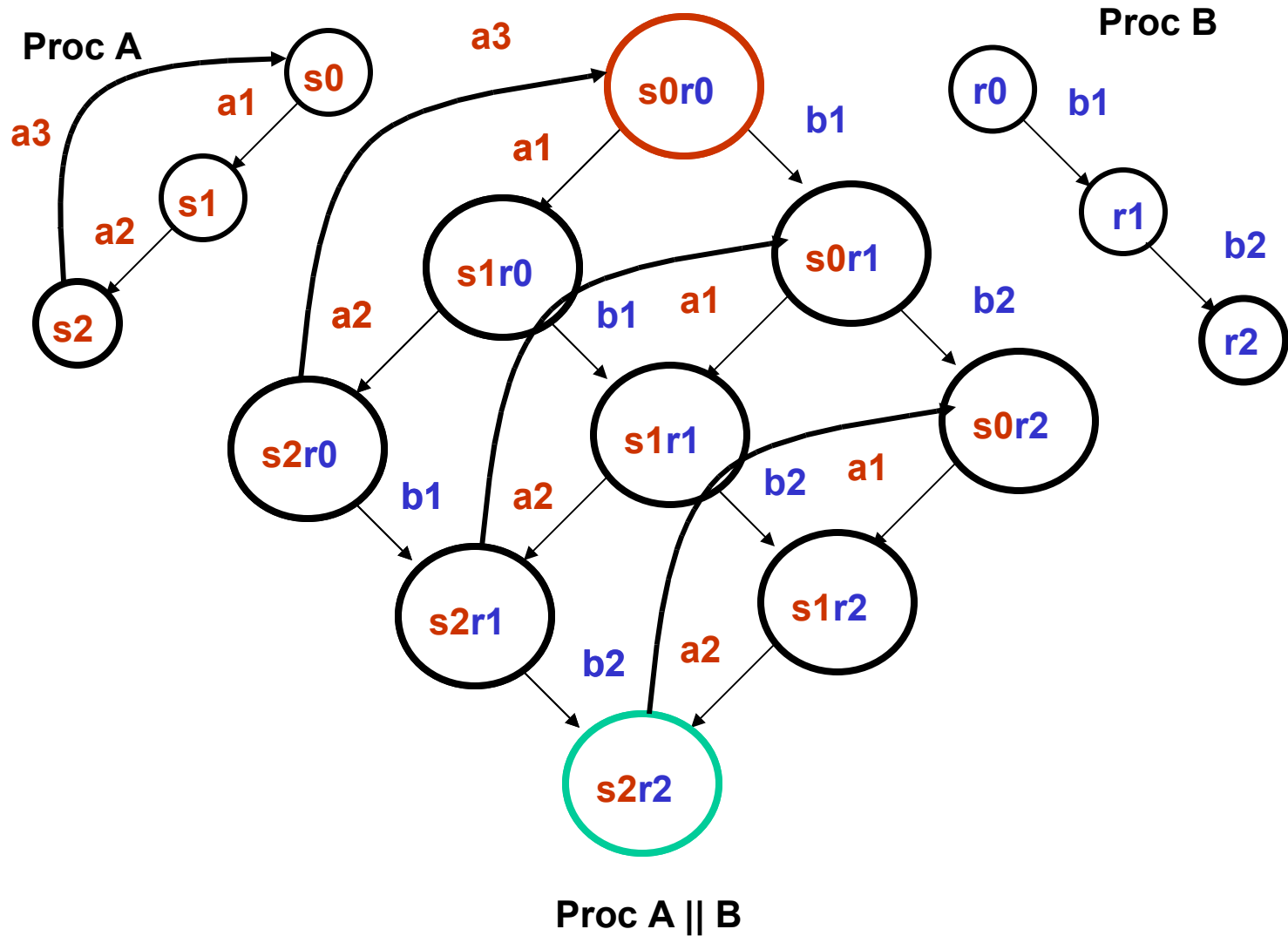


☐ **deadlock preservation**

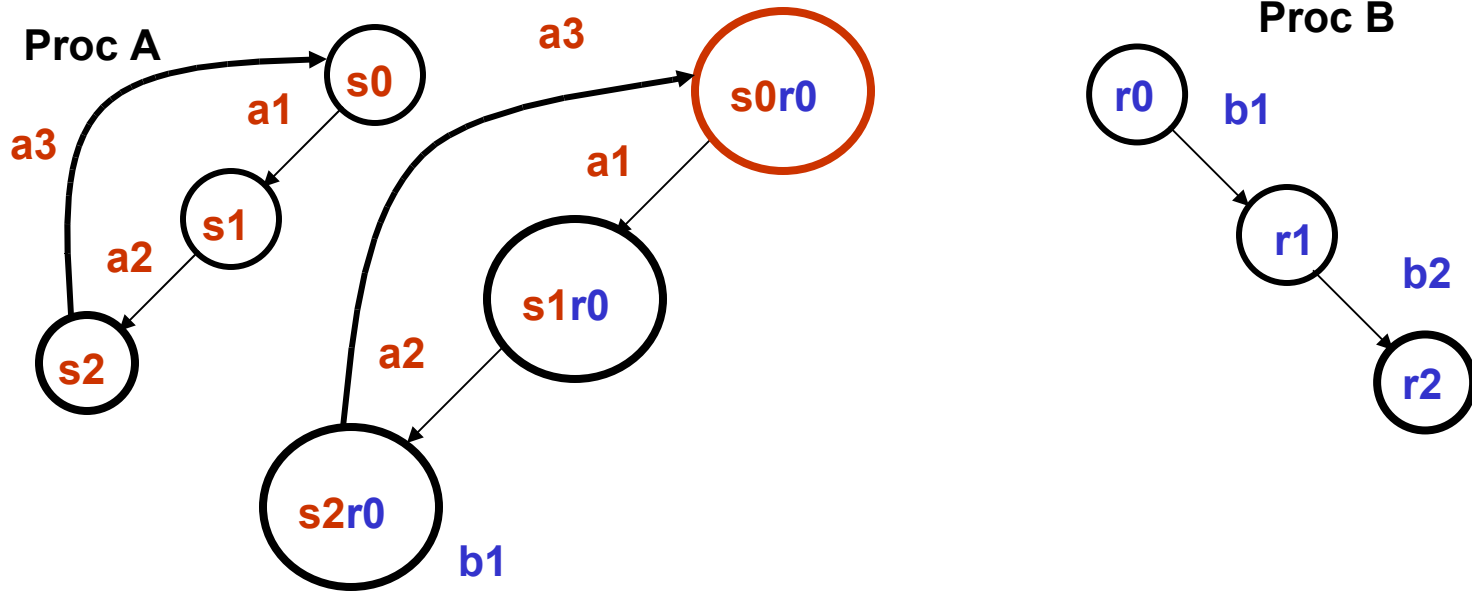
?

☐ **assertion (local-property) preservation**

Action Ignoring Partial Order Reduction



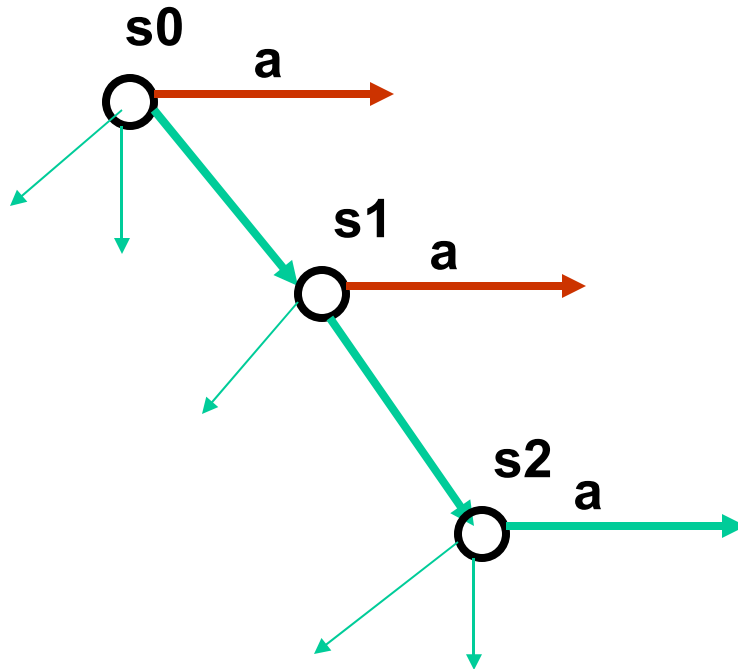
Action Ignoring Partial Order Reduction



Reduced Proc A || B

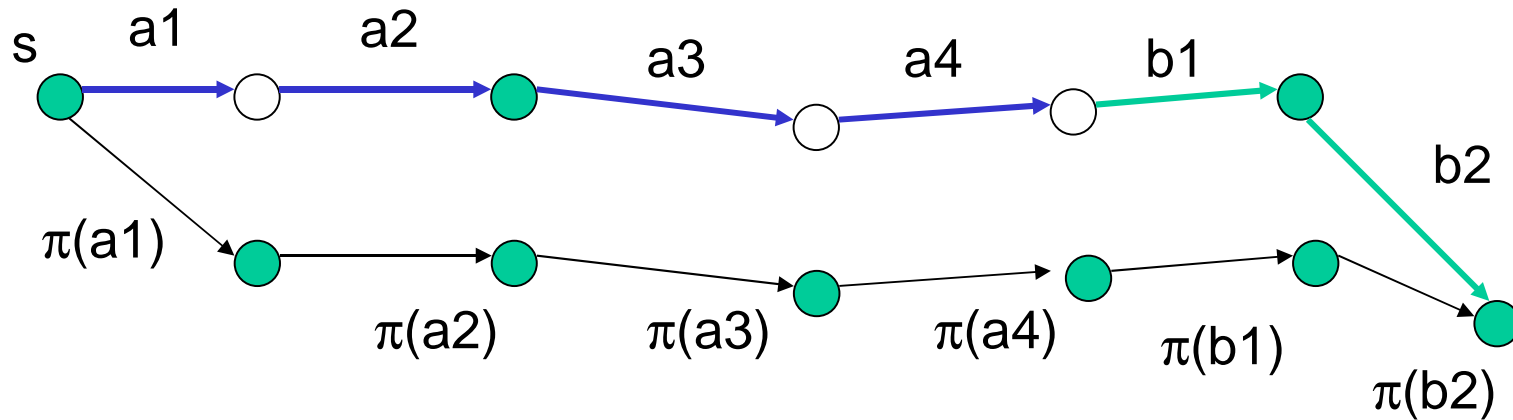
Action Ignoring Solution [Valmari 89]

Any action which is temporarily ignored in a given state s must be **eventually executed** in some state reachable from s

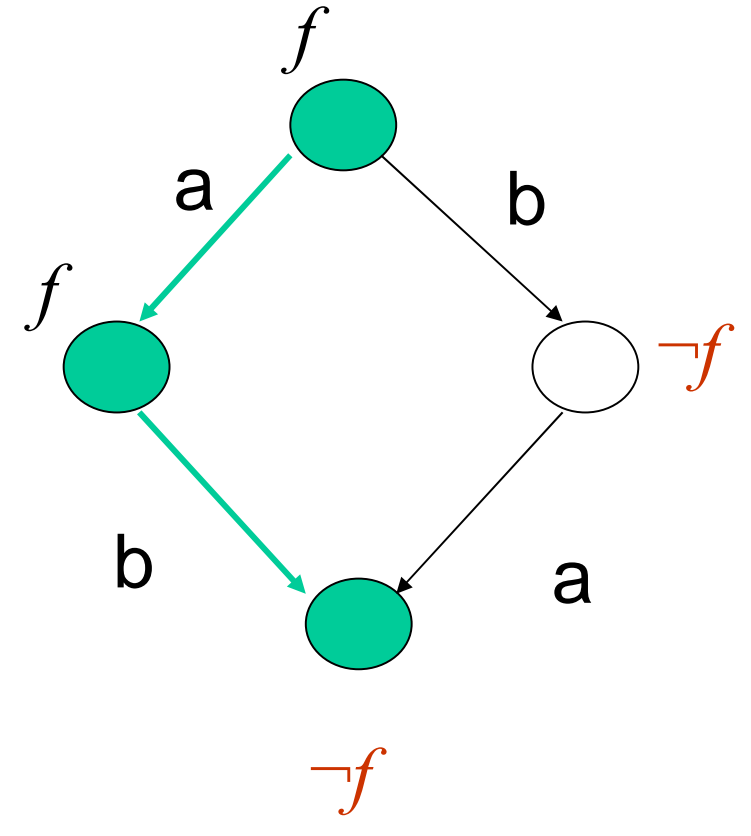
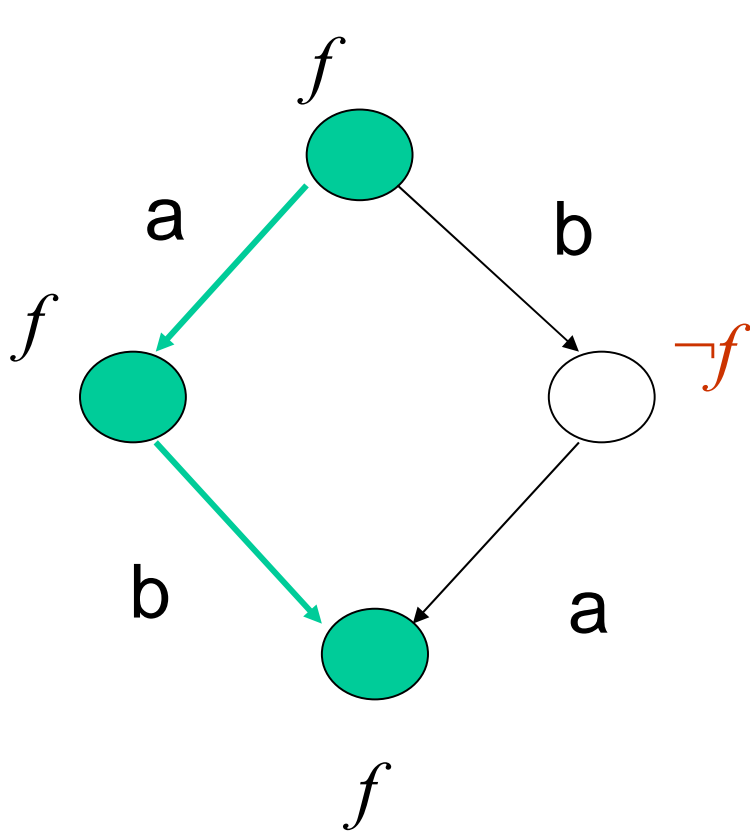


Main Theorem

Each execution sequence σ from the original state space S that begins in a state s of the reduced state space S_r has a representative execution sequence σ in the reduced state space which contains a permutation of σ .



Local Properties /Assertions



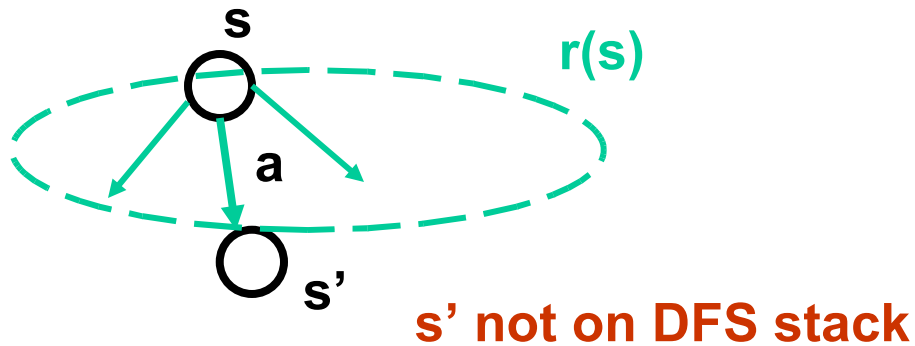
Efficient Provisos

Because of efficiency reasons we need **a locally checkable** version of the condition that prevents action ignoring

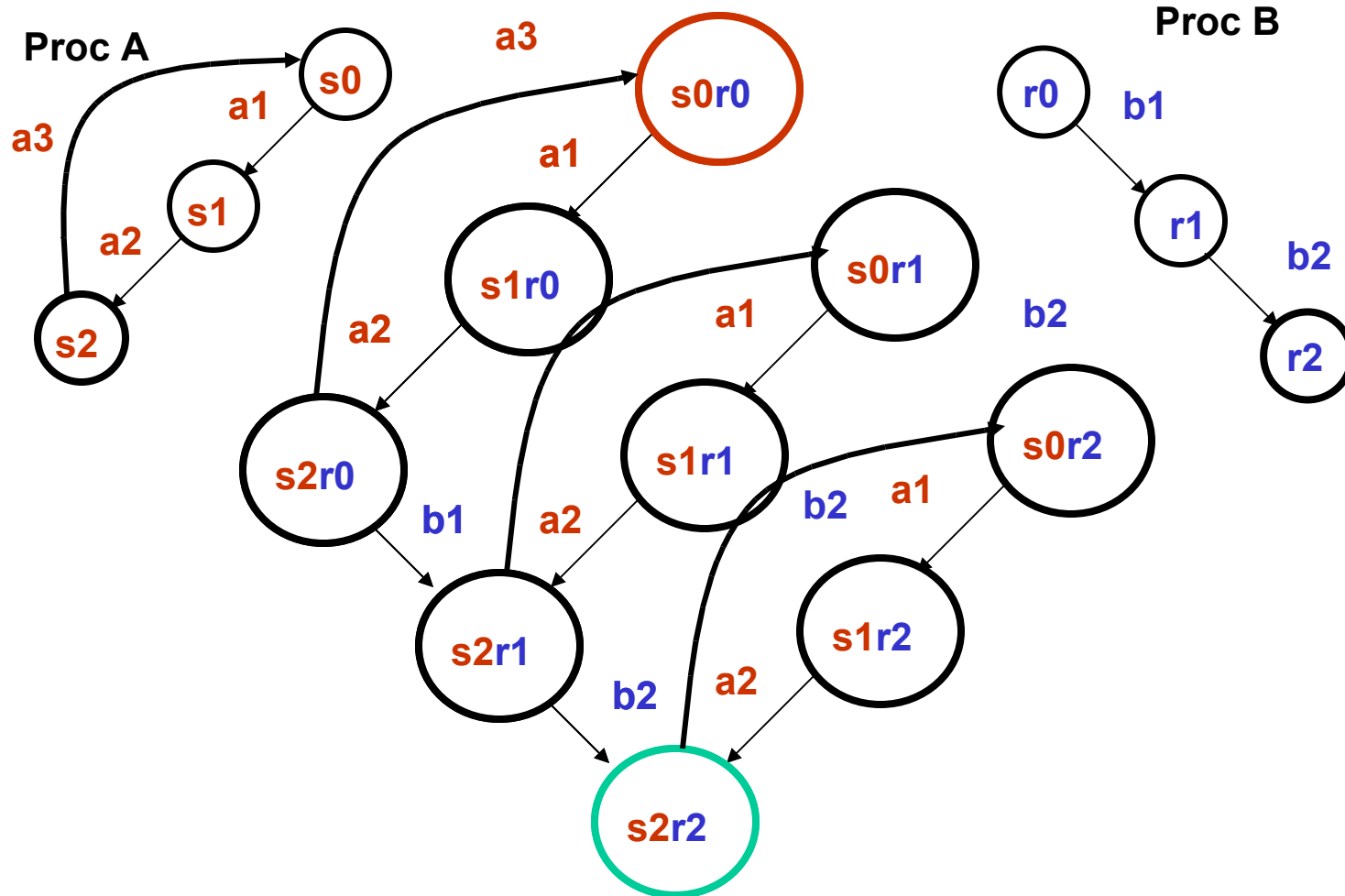
Stack Proviso

C2s: [Godefroid/Wolper 91, Godefroid 96,
Holzmann/Godefroid/Pirottin 92]

For any s in S_r , there **exists at least one action a** in $r(s)$ and state s' such that **$s \xrightarrow{a} s'$** and **s' is not on the DFS stack**, i.e., s' is not in $stack(s)$. Otherwise, $r(s) = enabled_T(s)$.



Partial Order Reduction with Stack Proviso

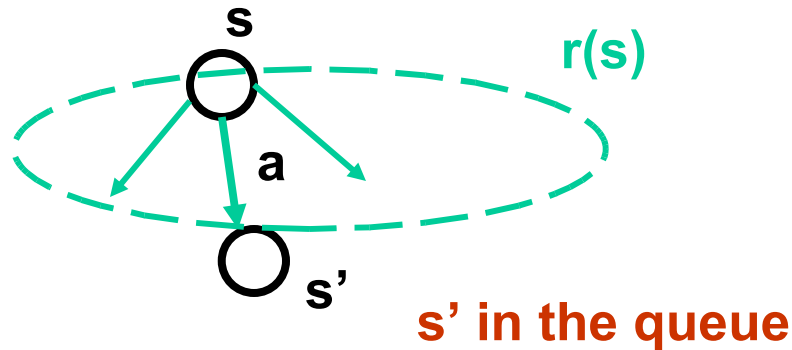


Stack Proviso Reduced Proc A || B

Queue Proviso

C2q: [Boanački/Holzmann 05,
Alur/Brayton/Henzinger/Qadeer/Rajamani 97]

For any s in S_r there **exists at least one action a** in $r(s)$ and state s' such that **$s \xrightarrow{a} s'$** and **s' is in the BFS queue**, i.e., s is in **queue** (s). Otherwise, $r(s) = \text{enabled}_T(s)$.



General State Exploring/Expanding Algorithm

procedure *GSEA*(*s*)

Closed = \emptyset ; *Open* = {*s*}

while not *Open.empty()* **do**

u \leftarrow *Open.extract()*; *Closed.insert(u)*;

If *goal(u)* **then return solution**;

For each *a* in *enabled*~~*T*~~(*u*) **do**

□ *v* \leftarrow $\tau(u, a)$; *process(v)*;

If *reopenOK(v)* **then** *Closed.delete(v)*;

If *v* **not in** *Closed* **and** *v* **not in** *Open* **then** *Open.Insert(v)*;



r(u)

A* Search Algorithm

procedure $A^*(s)$

$Closed = \emptyset; Open = \{s\}; s.g \leftarrow 0; s.f \leftarrow h(s); Open.insert(s);$

while not $Open.empty()$ **do**

$u \leftarrow Open.extractmin(); Closed.insert(u);$

if $goal(u)$ **then return solution;**

for each a **in** ~~$enabled_T(u)$~~ **do**

$v \leftarrow t(u,a); v.g \leftarrow u.g + cost(a); f' \leftarrow v.g + h(v);$

if v **in** $Open$ **then**

if $(f' < v.f)$ **then** $v.f \leftarrow f';$

else if v **in** $Closed$ **then**

if $(f' < v.f)$ **then** $v.f \leftarrow f'; Closed.delete(v); Open.insert(v);$

else $v.f \leftarrow f'; Open.insert(v);$

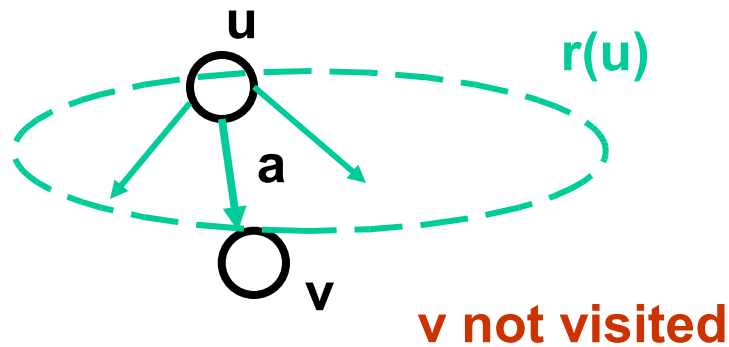


$r(u)$

Visited Proviso

C2v: [Edelkamp/Leue/Lluch-Lafuente 02]

For any u in S_r there **exists at least one action a** in $r(u)$ and state v such that $u-a \rightarrow v$ and v has **not been visited** by the search, i.e., v is not in $Closed \cup Open$. Otherwise, $r(u) = enabled_T(u)$.

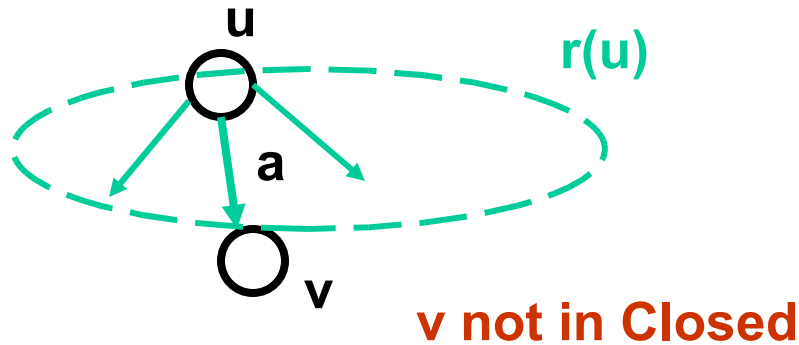


Closed Proviso

C2c:

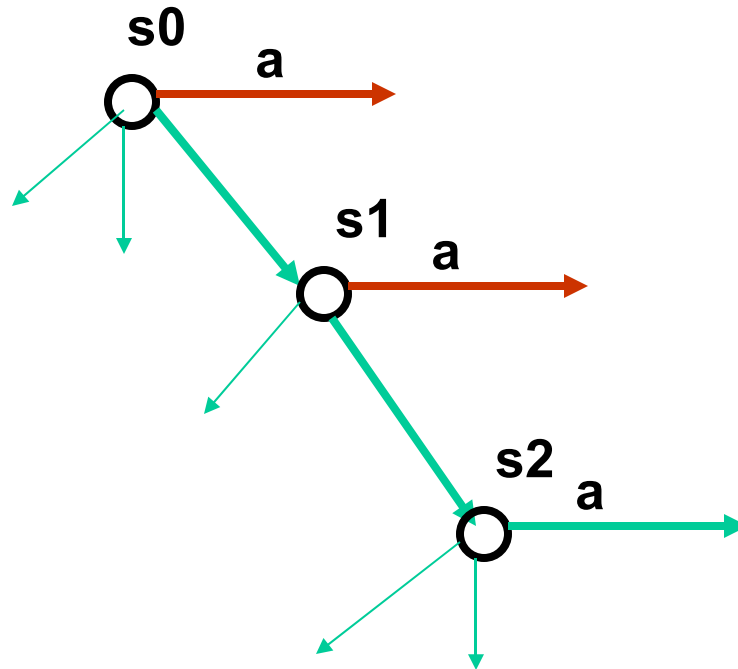
For any u in S_r there **exists at least one action** a in $r(u)$ and state v in S_r such that $u-a \rightarrow v$ and v is not in **Closed**

Otherwise, $r(s) = \text{enabled_}T(s)$.



Intuition

Any action which is temporarily ignored in a given state **s** must be **eventually executed** in some state reachable from **s**



At least one transition leads to a state which will be processed **later**

When and Why it Works

- " Assumption: GSEA **terminates**
- " *Open* will **eventually** become **empty**
- " We **cannot postpone** the execution **forever** by choosing states in *Open* (actually: outside *Closed*)
- " Proof by **inductoin** on the **decreasing order** in which the states are **removed** from *Open*

General State Exploring/Expanding Algorithm

procedure *GSEA*(*s*)

Closed = \emptyset ; *Open* = {*s*}

while not *Open.empty()* **do**

u \leftarrow *Open.extract()*; *Closed.insert(u)*;

If *goal(u)* **then return solution**;

For each *a* **in** *enabled_T(u)* **do**

□ *v* \leftarrow $\tau(u, a)$; *process(v)*;

If *reopenOK(v)* **then** *Closed.delete(v)*;

If *v* **not in** *Closed* **and** *v* **not in** *Open* **then** *Open.Insert(v)*;

Comparison Stack, Queue and Closed Provisos

C2q:

For any s in S_r there exists at least one action a in $r(s)$ and state s such that $s \xrightarrow{a} s$ and s is in the **BFS queue**. Otherwise, $r(s) = \text{enabled}_T(s)$.

C2s:

For any s in S_r , there exists at least one action a in $r(s)$ and state s such that $s \xrightarrow{a} s$ and s is not on the **DFS stack**. Otherwise, $r(s) = \text{enabled}_T(s)$.

Partial-Order Reduction for Liveness (LTL)

restrictions on an ample set

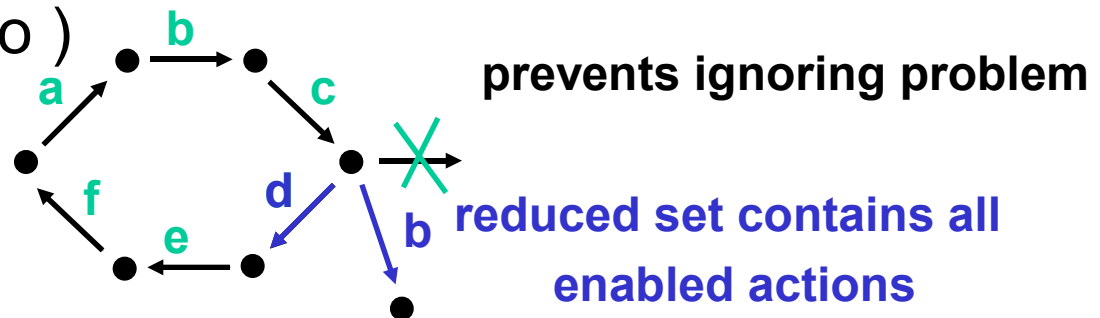
C0: empty iff no actions enabled

C1 (persistency)



□ **deadlock preservation**

C2 (cycle proviso)



C3 (invisibility)

- . all actions in the reduced set globally “invisible” or
- . reduced set contains all enabled actions

□ **next-time-free LTL preservation**

Liveness Cycle Proviso

For any cycle in the reduced state space in some state s_i of the cycle $r(s_i) = \text{enabled_}T(s_i)$

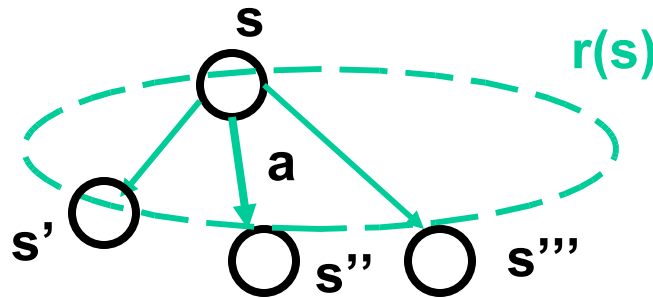
Liveness Closed Proviso

C2ql:

For any s in S_r for all actions a in $r(s)$ and states s such that

$s \xrightarrow{a} s'$ we require that s' *is not in Closed*

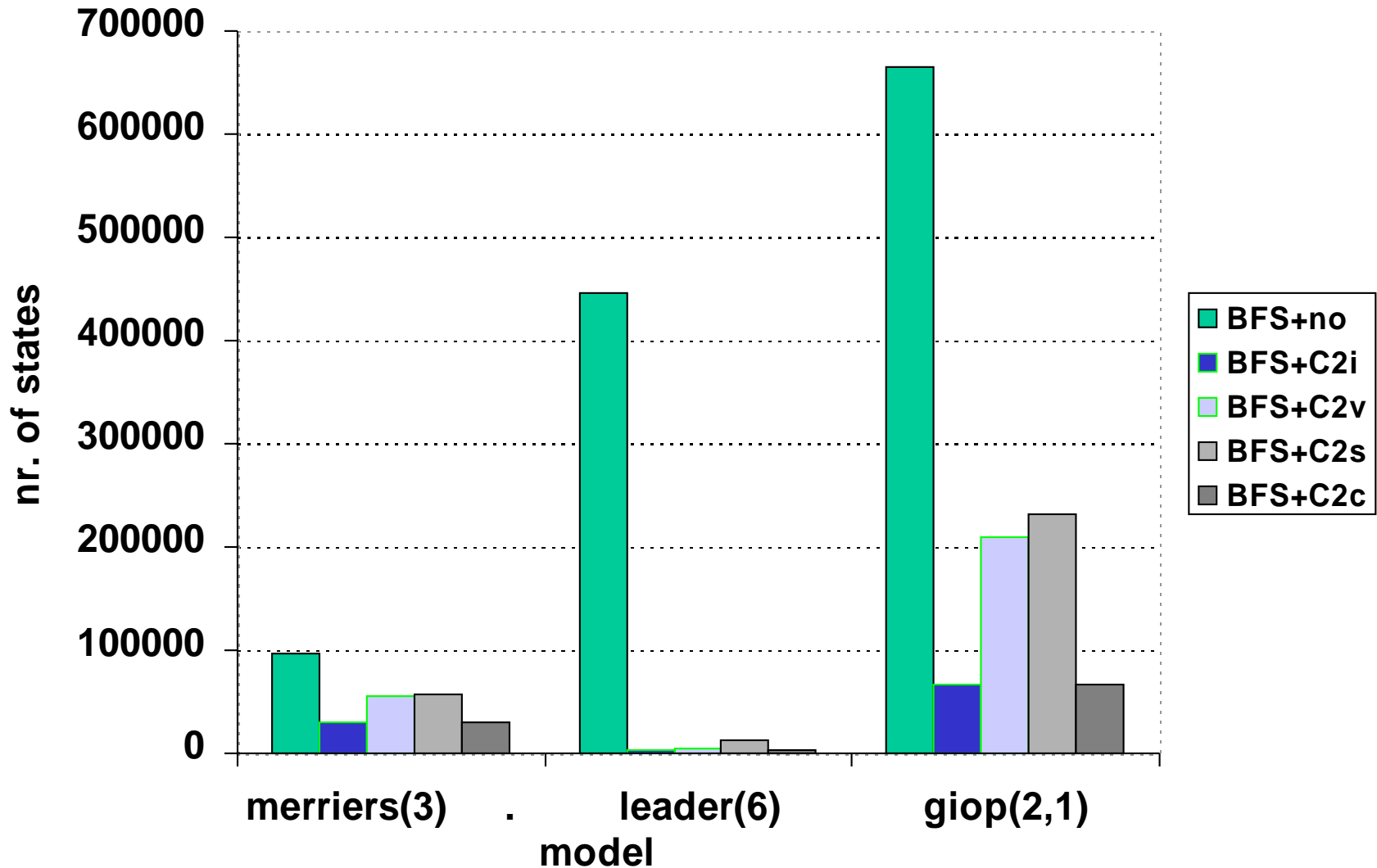
Otherwise, $r(s) = \text{enabled_T}(s)$.



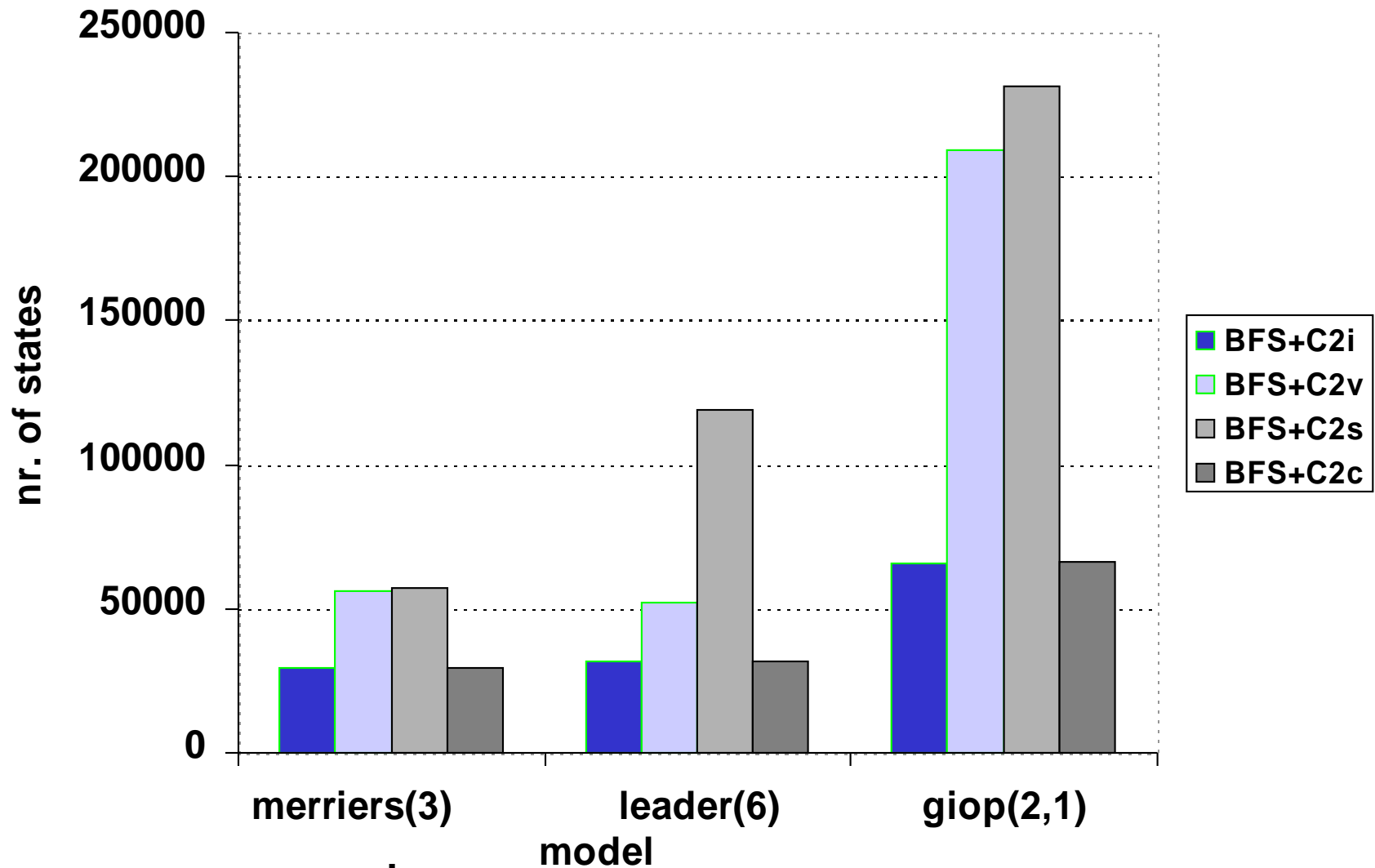
s' , s'' , s''' not in Closed

Again analogy between the DFS (stack) [Peled 94, Holzmann/Peled 94] and GSEA provisos

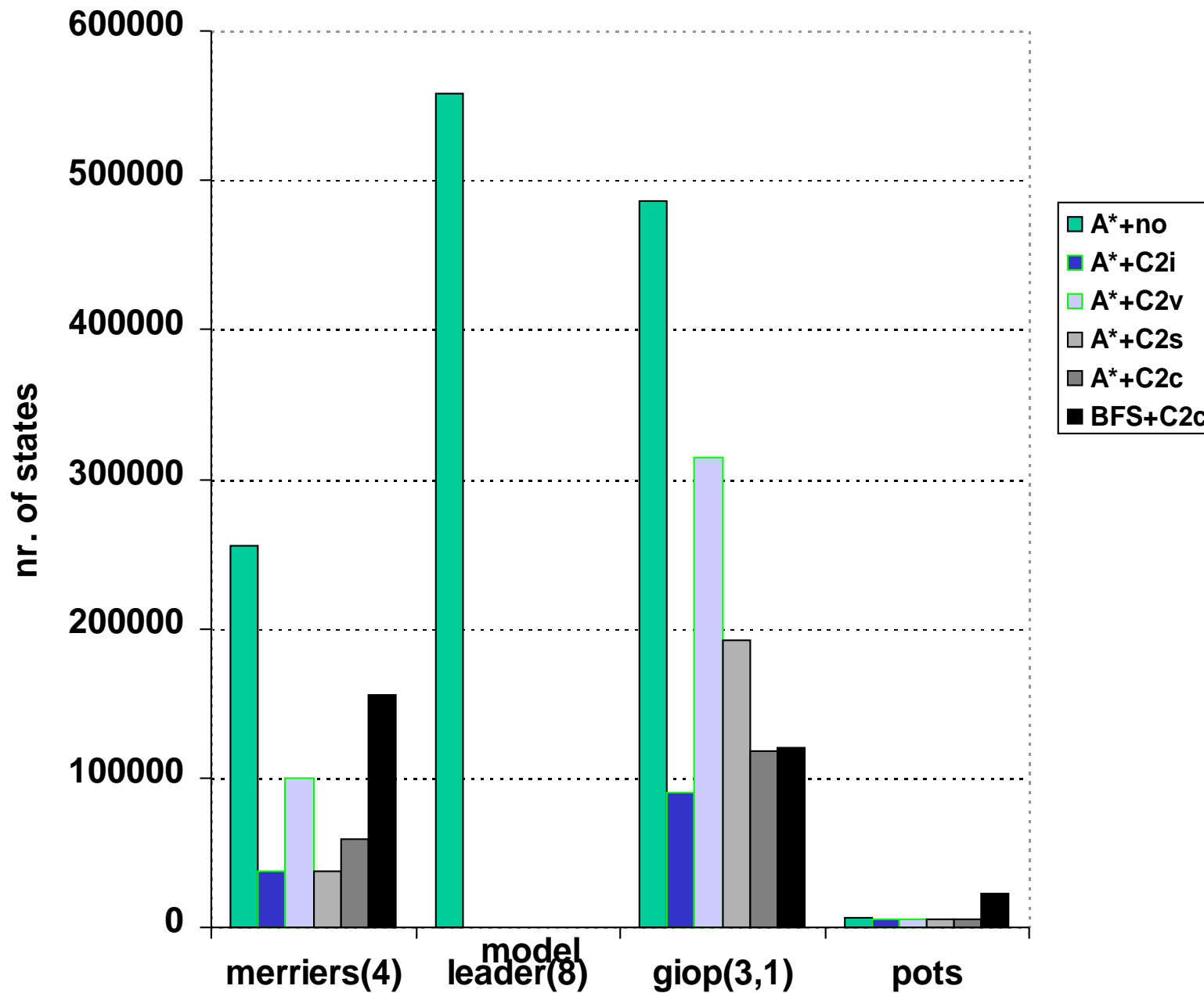
Experiments: BFS with Different Cycle Provisos



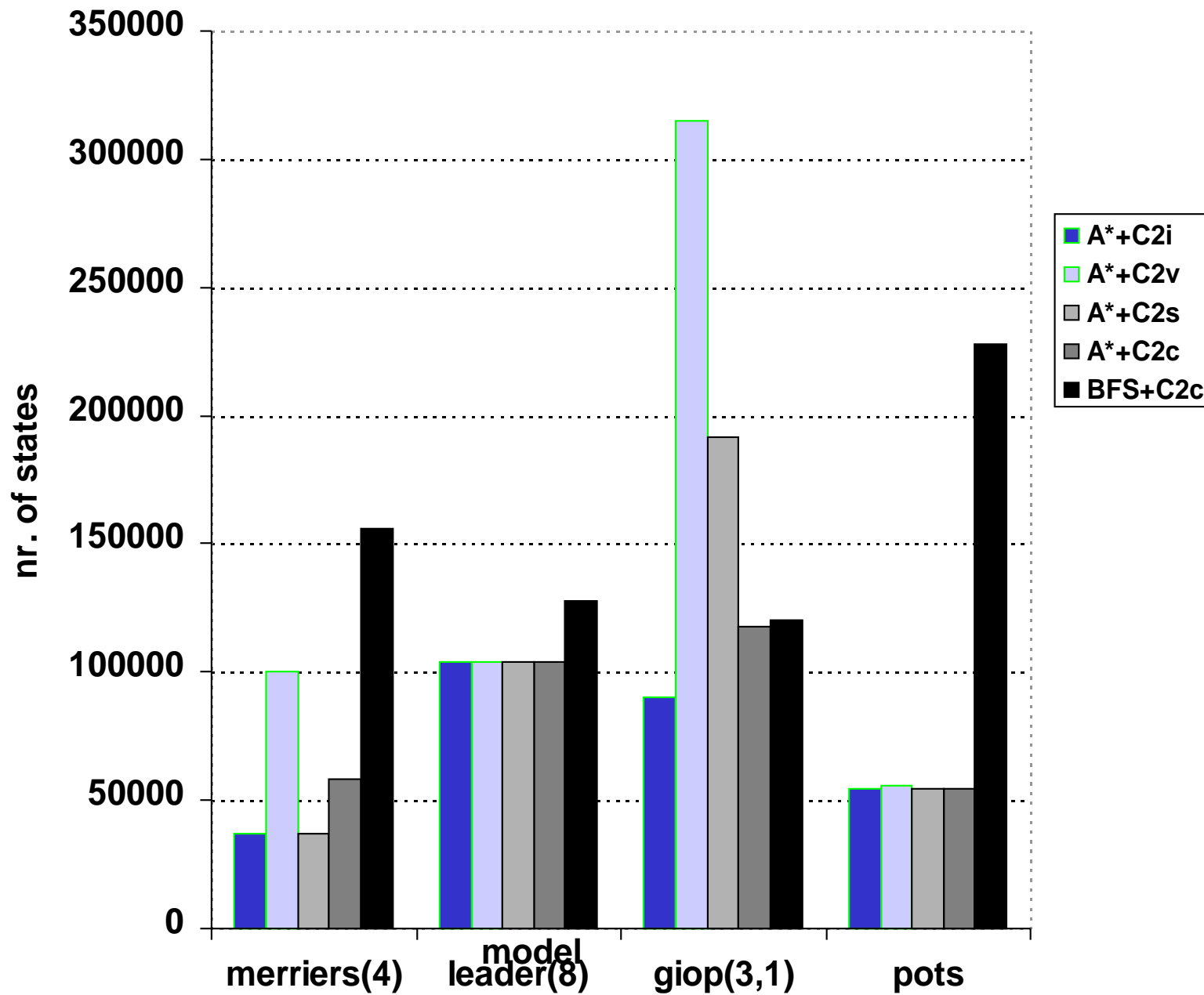
Experiments: BFS with Different Cycle Provisos



Experiments: A* with Different Provisos vs BFS



Experiments: A* with Different Provisos vs BFS



Related Work

- " Alur/Brayton/Henzinger/Qadeer/Rajamani 97
POR for Breadth First Search, symbolic model checking
- " Lluch-Lafuente/Edelkamp/Leue 02
visited proviso, directed model checking
- " Boanački/Holzmann 05
queue proviso, Breadth First Search
- " Kurshan/Levin/Minea/Peled/Yenigun 98
sticky transitions, syntactic level

- " Nalumasu/Gopalakrishnan 02
POR without proviso , DFS
- " Blom/van de Pol 02
Modification of Nalumasu/Gopalakrishnan
- " Palmer/Gopalakrishnan 02
Proviso for distributed model checking, BFS
- " Levin/Palmer/Qadeer/Rajamani 05
another POR without proviso, transactions, DFS

Related Work A Brief History of the Cycle Proviso

- " **Valmari 91, 98**
first introduced, safety, liveness, strongly connected components

- " **Godefroid/Wolper 91, Godefroid 96, Holzmann/Godefroid/Pirottin 92**
safety, stack

- " **Katz/Peled 92, Peled 94, Holzmann/Peled 94**
LTL, stack

- " **Willems/Wolper 96, Gerth/Kuiper/Peled/Penczek 96**
CTL*, stack

Conclusions

- " **Partial Order Reduction for General State Exploring Algorithms**
- " **Main novelty: a condition for avoiding action ignoring**
a generalization of the BFS proviso
- " **Version of the proviso for liveness properties**
- " **The improved version performs significantly better in the**
experiments for directed model checking (HFS-Spin)
- " **Algorithm independent of the implementation in HFS Spin**
- " **Future work:**
 - Improving the performance of the proviso by action**
reordering
 - Symbolic model checking, liveness properties**