

Symmetry Reduction for temporal model checking

- Symmetry in distributed systems
- Quotient graphs
- Model checking properties
- Conclusion Discussion

Symmetry in distributed systems

General context to apply symmetry

A (labeled) Kripke Structure M :
and a set of atomic propositions AP

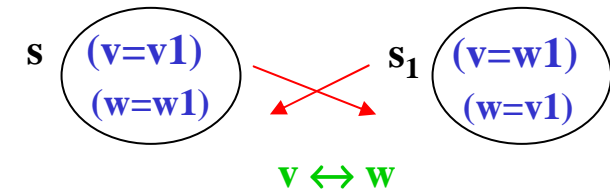
- M
- S set of states
 - R (\rightarrow) transition from a state to another
 - L labels of states – a label is a subset of AP
 - T labels of transition
 - S_0 initial states

$$AP = \{ (v=value) : v \in \text{Variables and } value \in \text{Domain}(v) \}$$

symmetry techniques focuses
on the atomic propositions labeling the Kripke structure

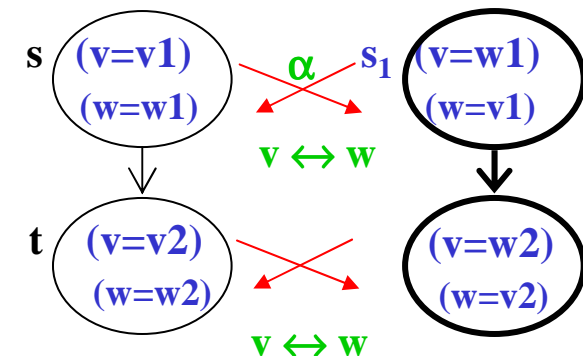
What is a behavioral symmetry ? (automorphism)

- A **symmetry** : a permutation on atomic properties
(interchange of AP values)
- **State symmetries** are permutations within the labels of states.



- **An automorphism** (behavioral symmetry)
= a state symmetries that preserve the transition relation.

$$\forall s, t \in S, \forall \alpha \in \text{Aut}(M), \\ s \rightarrow t \in R \Rightarrow \alpha(s) \rightarrow \alpha(t) \in R$$



[Aut(M) set of automorphisms of a kripke structure M]

Symmetrical system

Symmetrical system :

a system within which a given set of automorphisms applies everywhere

Example : 2-process mutual exclusion

3 variables :

- a global token : tok
- 2 local states variables
 - N -> NEUTRAL,
 - T -> TRY,
 - C -> CRITICAL ACCESS

☞ Condition for Process P_i to reach local state C :

- Token = i
- Local state (P_i) = T_i

☞ Token delivery rule :

- initial state : undeterministically
- after an access,
 - either undeterministically or
 - to the other process whenever it requests the access.

Example of symmetries on a system

*Kripke Structure for the
2-process mutual exclusion*

STATE

[localState(1), localState(2),
tok]

local state values :

N -> NEUTRAL,

T -> TRY,

C -> CRITICAL ACCESS

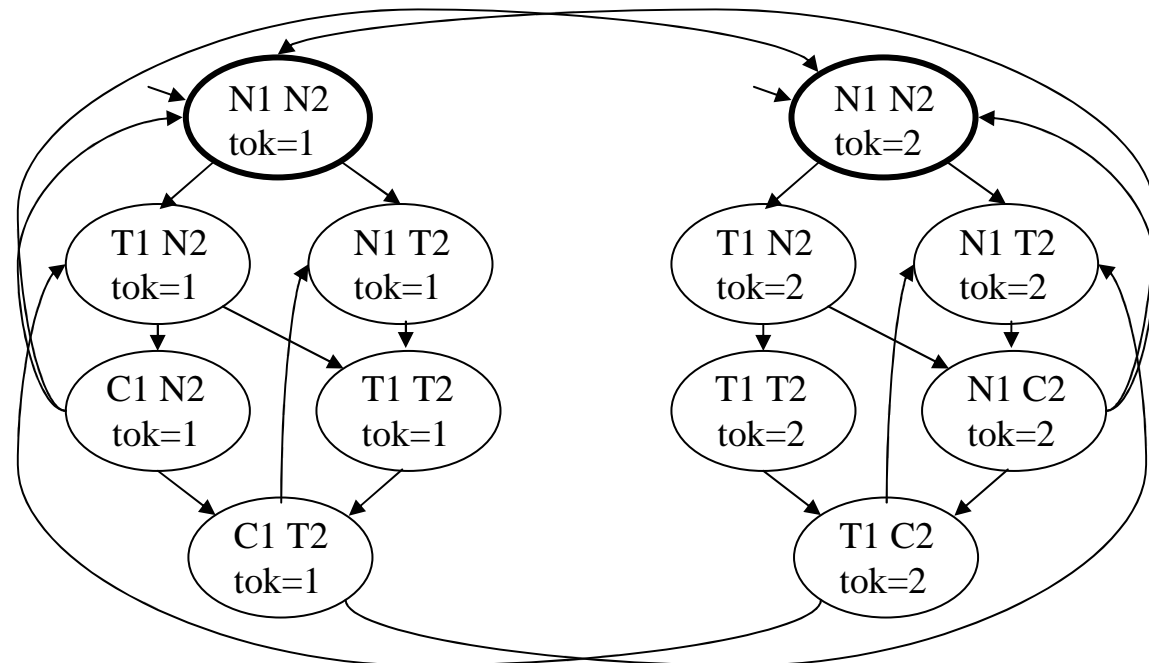
tok values :

1, 2

Note :

- 2 possible initial states

(w.r.t. the two tok's value)

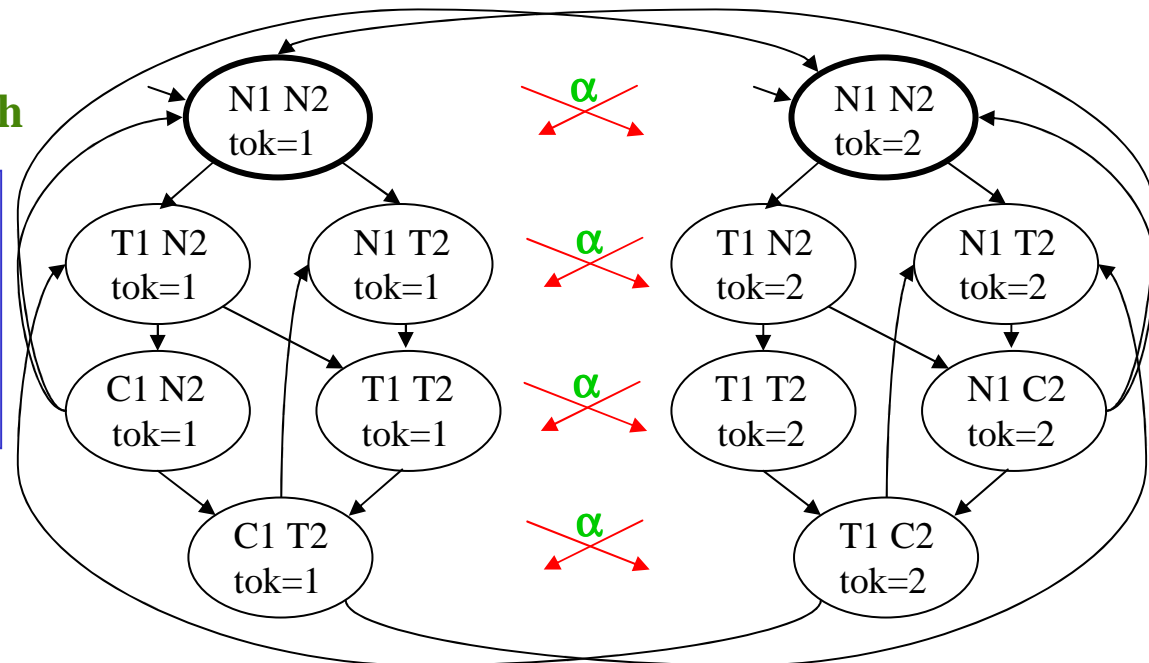


Example of symmetries on a system

Consider the symmetry
 $\alpha = (\text{LocalState}(1) \leftrightarrow \text{LocalState}(2)$
 and $\text{tok}=1 \leftrightarrow \text{tok}=2)$
 Apply it on all the reachable graph

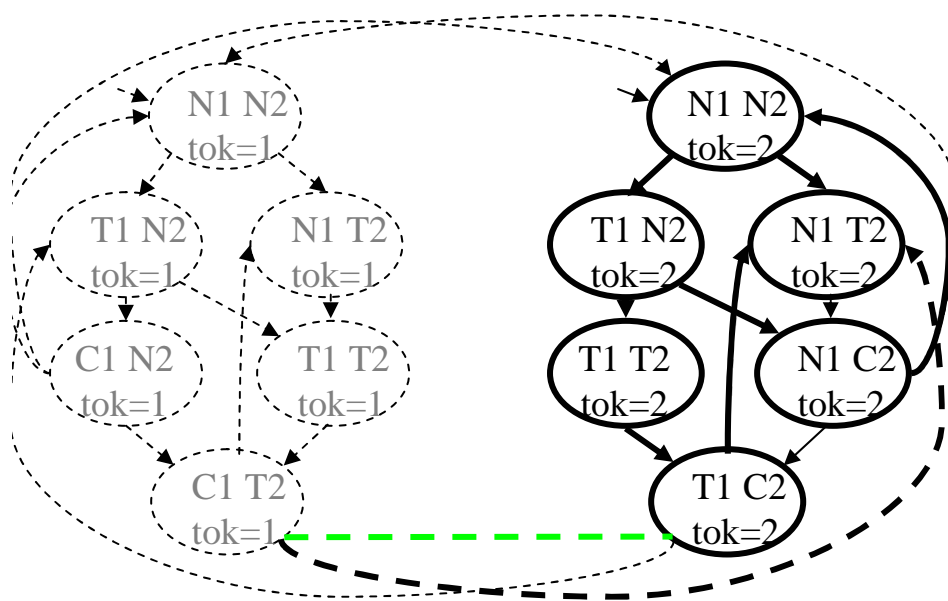
α is an automorphism and
 and
 the structure is left unchanged
 under the action of α .

According to the
 mutual exclusion property,
 the left and right part
 are equivalent.



idée : take benefit from automorphisms
 to compact the reachability representation
 (provided the

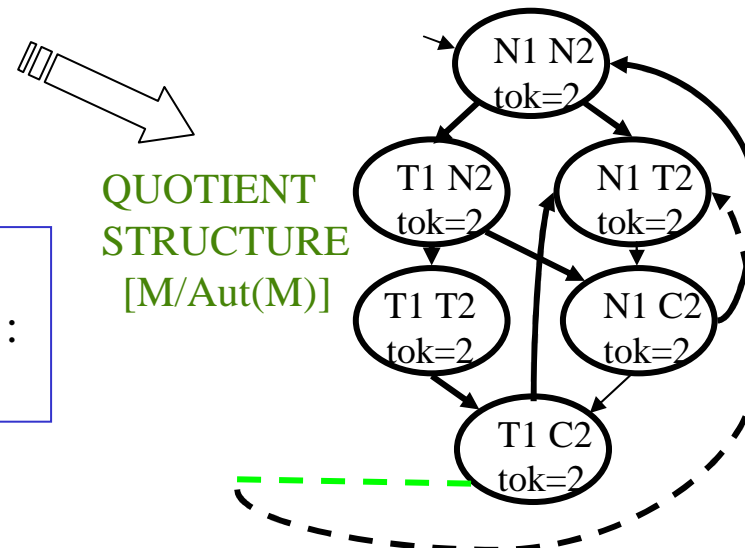
Bases of symmetry reduction techniques



Any state can be retrieved by applying symmetries on an EC representative :
 $\alpha(T1, N2, tok=2) = (N1, T2, tok=1) \dots$

According to a group of automorphisms $Aut(M)$

- Define the equivalent classes (EC) of states [An EC represents a set of states]
- Use representatives of classes to build a reduced structure.
- Define an arc (ECA) between EC1 and EC2 whether a state in EC1 reaches a state in EC2 [An ECA represents a set of arcs]



Quotient graph representation

Quotient Graph definition

Let M a kripke structure and
 $G = \text{Aut}(M)$ a subgroup of automorphisms over the states of M

Quotient Kripke structure M_G of M :

- S_G the states are the set of representatives $\{\text{rep}([s]_G) : s \in G\}$
- $R_G (\rightarrow)$ the transitions between representatives $\{(\text{rep}([s]_G), \text{rep}([t]_G)) : s \rightarrow t\}$
- $L_G(\text{rep}([s]_G))$ the labels of representatives
- S_G^0 the subset of S_G concerning the initial states

What is the reduction factor : $\text{SIZE}(S_G) = \frac{\text{SIZE}(S)}{\text{SIZE}(G)}$ (The size of every equivalence class $[s]_G$ is bounded by the size of G)

Efficiency depends on the system !

- ☺ Theoretically exponential
 (For n components in a highly symmetrical systems, $\text{SIZE}(G) = n!$ permutations)
- ☹ Could be no reduction
 (for an asymmetric system, G is reduced to the identity permutation)

3 components
 \Rightarrow
 $3! = 6$
 symmetrical
 states

Interest of group of automorphisms for the representation of states

- A group G acting on any set S defines a **partition** of the elements of S into equivalence classes.
- whatever the size of an equivalence class, it can be represented by using a single or multiple **representatives**.
- It is possible to define a **canonization function** to ensure the unicity of the selected representatives to build the quotient graph (based on a lexicographic ordering on variables)

An algorithm to build a quotient kripke structure

```

reached := {rep([s]) : s ∈ S0}
unexplored := {rep([s]) : s ∈ S0}
WHILE (unexplored ≠ ∅) DO
  Rep = unexplored.pop
  FORALL (successor states q of Rep) DO
    IF (rep([q]) ∉ reached)
      append rep([q]) to reached
      append rep([q]) to unexplored
    END (IF)
  END (FOR)
END (WHILE)
return reached

```

Assuming that $\text{Aut}(M)$ is identified, the quotient graph building mimics the reachability graph algorithm.

It is then possible to explore the states even though the state space is intractably large

To be solved :

- identifying **Aut(M)**
- solving the **orbit problem**
 - test whether $\text{rep}([q])$ is already in reached*
 - or whether a state r is already visited s.t. $\text{rep}([q]) = \text{rep}([r])$*
- checking **Temporal logic properties**

Identifying the automorphisms of M

- **Solution 1** : (deeply) **From a kripke structure** (testing any possible symmetry)
Costly : hard problem with no polynomial time algorithm

- **Solution 2** : **From the specification of the system,**

Some data types as a documentation that certain symmetries hold.

- define (symmetrical) data types (e.g. **scalarsets** of Murϕ)

from which the actions of the associated functions preserve symmetries.

From a state $s1$ having a scalarset variable, and a function F with param.

if $s1 \xrightarrow{F(\text{param})} s2$

then $\forall \alpha : \alpha(s1) \xrightarrow{F(\alpha(\text{param}))} \alpha(s2)$

- Define automorphisms from permutations over the values of the scalarset variables

- **Note** : Basically, high level Petri Nets can also define symmetrical data types close from scalarsets (see **Well-formed Nets**).

What is a scalarset in Murø program

A **scalar set is an integer subrange 1..i** with restricted operations :

- A term of a scalarset type **must be a variable reference**
(a scalarset cannot appear as an operand to + or any operator)
e.g. each process can be represented by a scalarset variable
- An **array can be indexed by a scalarset variable**,
e.g to store information of different processes.
- **comparison using "="** with a term of the exactly the same type
- Similar for **assignment :=**
- If a scalar set is used as an index of a **FOR statement**,
the result of the execution must be **independent** of the order of iteration

Verifying properties on quotient graphs

Verifying deadlock on a quotient graph

An example with a wrong behaviour :

- a set of ($n=2$) processes
- a shared (distributed) critical section

Local states :

{N,T,W}

Shared state :

{C}

For Processes 1 and 2 :

N_1, N_2 : neutral

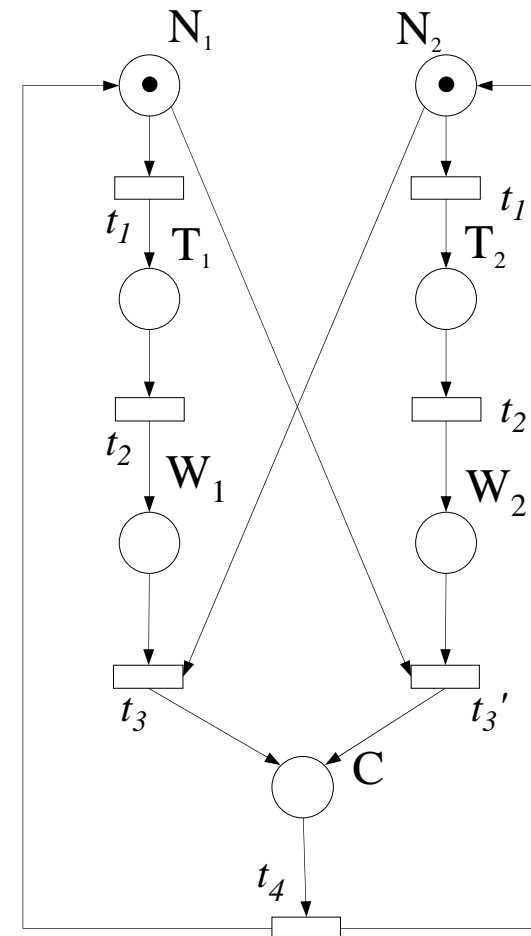
T_1, T_2 : try

W_1, W_2 : wait

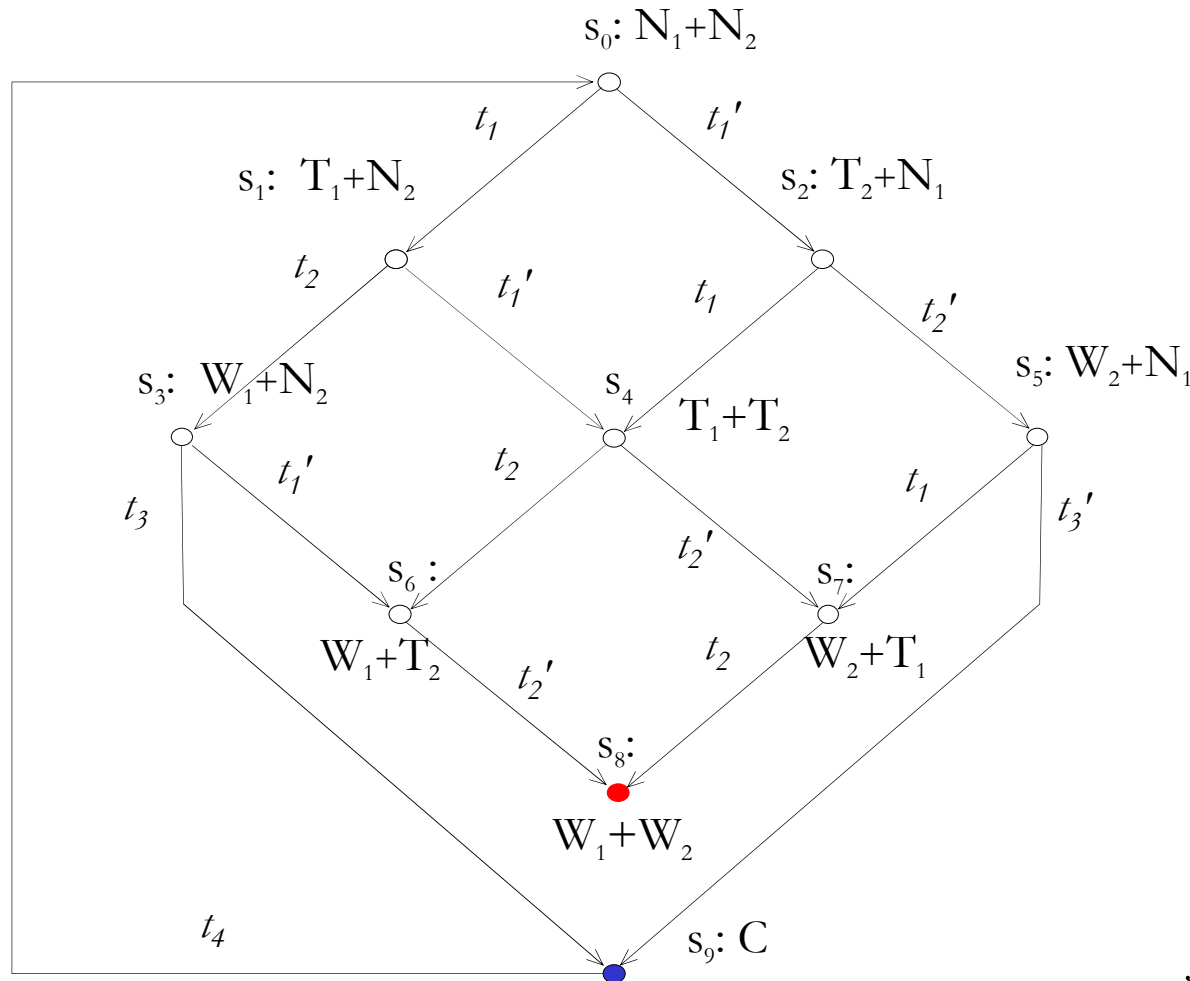
C : critical section

- **Condition for Process i to reach the shared state C :**

- Local state (i) = W
- Local state (j : $j \neq i$) = N



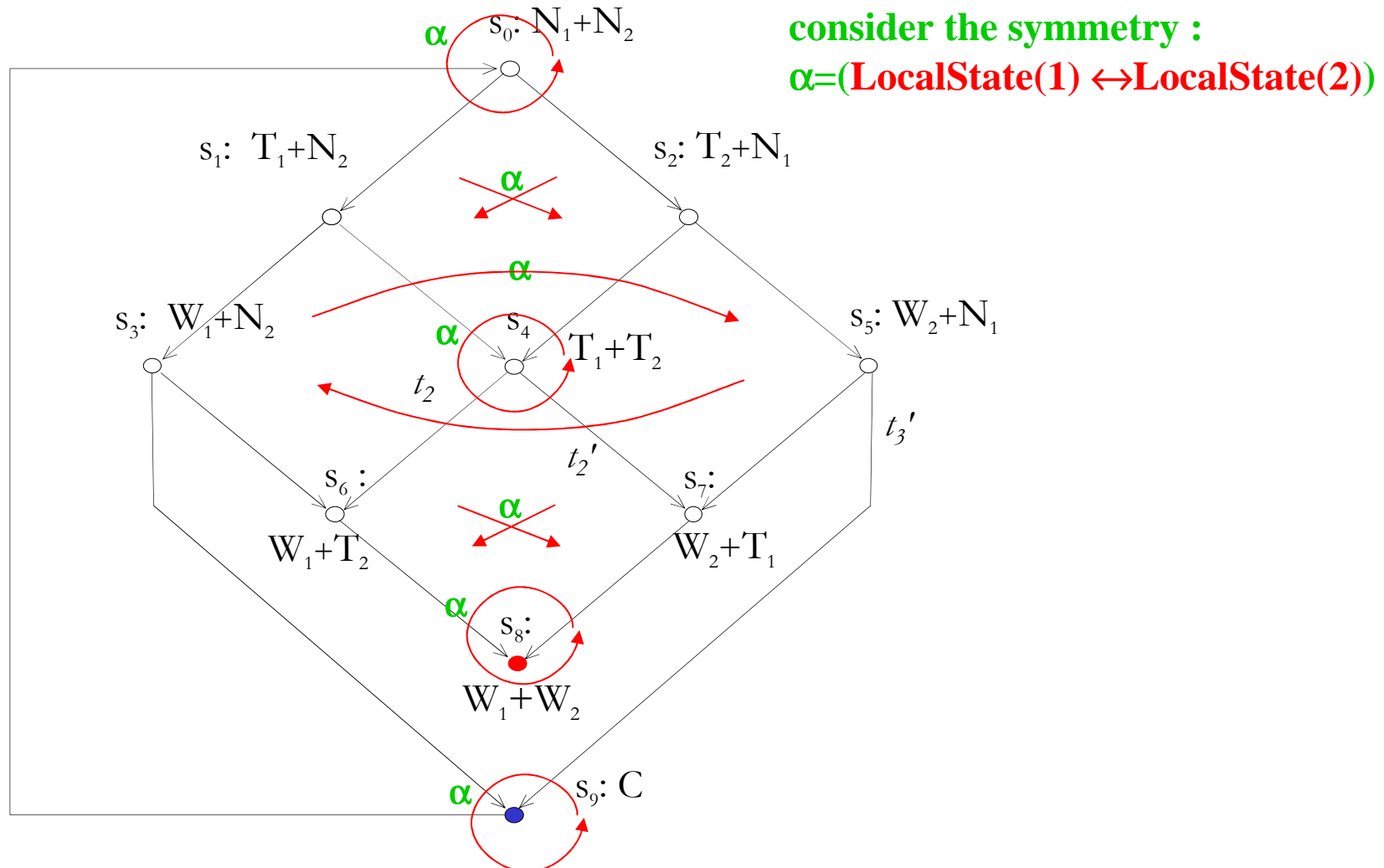
Kripke structure of the example with fault



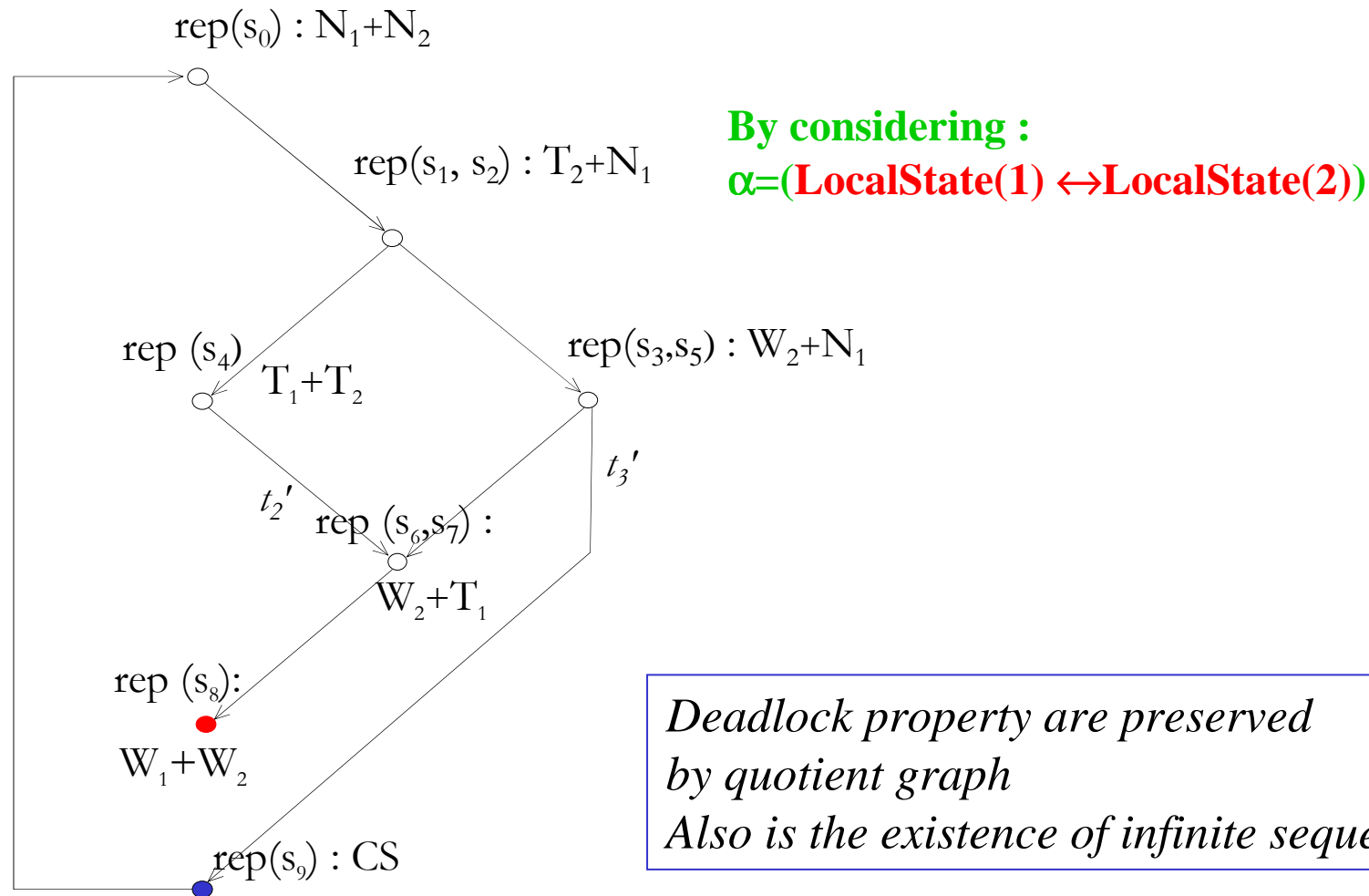
A deadlock optionally occurs whenever both processes are Waiting.

Diamond property : The state explosion could make difficult to find it.

Kripke structure of the example with fault



Quotient structure of the example with fault



Verifying CTL* symmetric property on a quotient graph

Symmetric CTL* properties are checked directly

In a symmetric CTL* formula,
every maximal propositional subformula is invariant
w.r.t. the action of G.

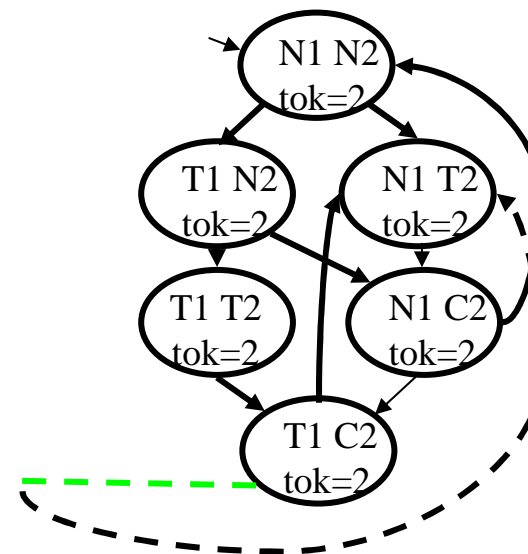
For a system M composed of 2 Processes

☺ $\Phi_1 = \mathbf{AG} (\neg (C1 \wedge C2))$ is symmetrical

☹ but $\Phi_2 = \mathbf{AG} (\neg (C1))$ is not
(permutations $1 \leftrightarrow 2$ yields : $\mathbf{AG} (\neg (C2))$)

Φ_1 is true on both M and $M/Aut(M)$

Φ_2 is true on $M/Aut(M)$ but not on M



Extension for symmetric formulas

Indexation or quantification can be used within propositional formulas

- ☺ The language of atomic propositions can be extended (indexed or quantified) according to the variables and the values to be permuted

$AG (\neg (\wedge_{i=1..2} Ci))$ (\wedge_i means for all processes, \vee_i for some)

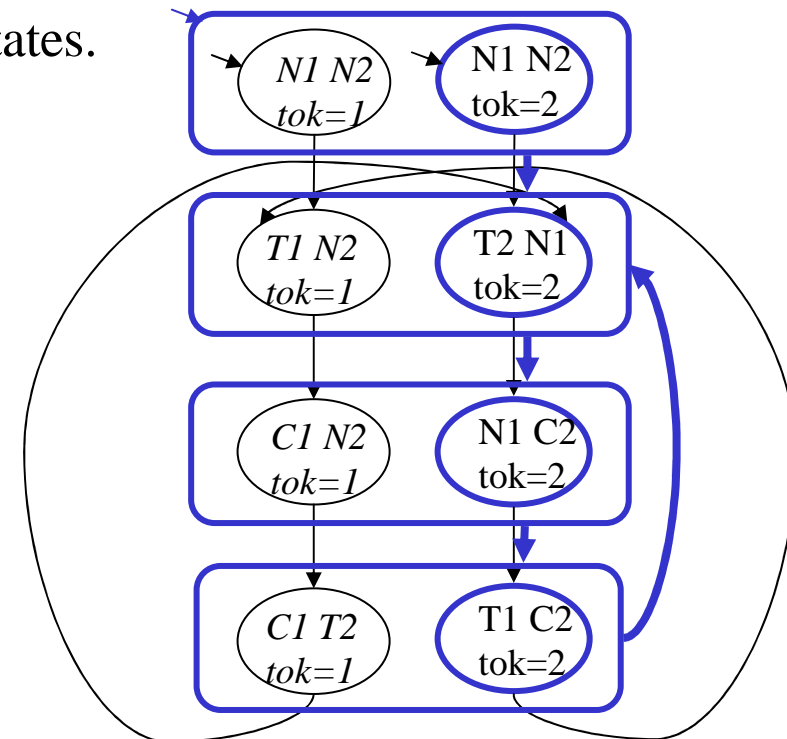
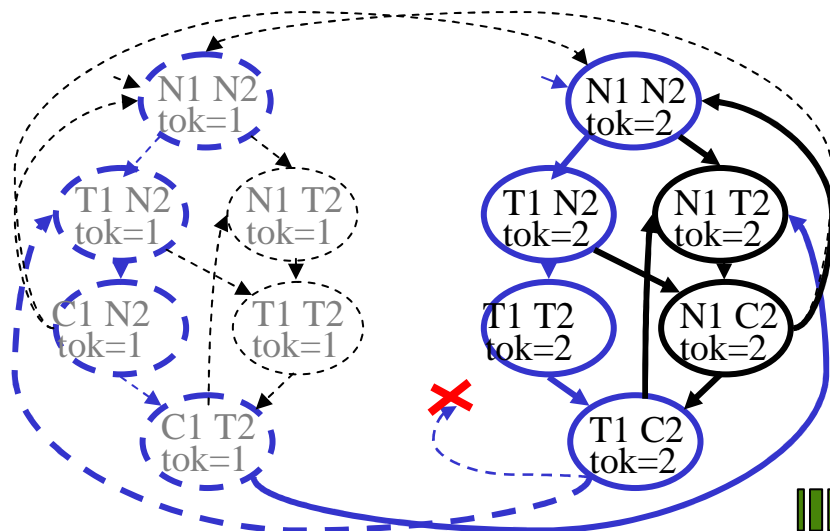
$AG ((\vee_{i=1..2} Ci) \Rightarrow AF (\wedge_{i=1..2} Ni))$

- ☹ **Be careful**, indexation does not concern the temporal operators

$\wedge_{i=1..2} AG ((Ti) \Rightarrow AF (Ni))$ cannot be checked directly on the quotient graph

What is the problem for unrestricted temporal properties

The quotient graph does not preserve real arcs
but existence of arcs between classes of states.



Superposition of
(some) arcs of M
and arcs in the quotient graph



Verifying unrestricted LTL properties

General idea :

Do not apply the automorphisms

which does not preserve the property to be checked

Basic techniques :

- Partition the atomic propositions that are not used identically
- Let automorphisms be applied **but** on each part separately.

The static method

☺ **From the analysis of the property :**

- Non symmetric atomic propositional formula leads to partition the atomic properties
- Allow the symmetries which act on each part separately

Example for a system of 3 processes

$\Phi = [\text{AG}(\text{C1}) \Rightarrow \text{AF}(\text{N1 and N2 and N3})]$

- Allow permutations between values of localState(2) et localState(3)
- Avoid permutations between values of localState(1) and the others



- **Partition AP** in $\text{AP}_1 \cup \text{AP}_2 \cup \dots$ with

$\text{AP}_1 = \{\text{C1}, \text{N1}, \text{T1}\}$

$\text{AP}_2 = \{\text{N2}, \text{N3}, \text{C2}, \text{C3}, \text{T2}, \text{T3}\}$

- Build $\text{Aut}(\text{M})$ from all the permutations

$1 \leftrightarrow 1,$

$2 \leftrightarrow 3, 3 \leftrightarrow 2$

- Check $\Phi' = [\text{AG}(\text{C1}) \Rightarrow \text{AF}(\text{N1 and } \bigwedge_{i=2..3} \text{Ni})]$ **directly on $\text{M}/\text{Aut}(\text{M})$**

☹ **Main problem :** $\text{Aut}(\text{M})$ could rapidly be trivial (reduced to identity)

$\Phi = [\text{AG}(\text{C1}) \Rightarrow \text{AF}(\text{N1 and N2})]$

The dynamic method

The partitioning is realized at any stage of an exploration of a quotient structure :

e.g. during the computation of successors or predecessors

According to the verification of a property.

- ☺ the set of symmetries can be adapted according to each state
- ☺ taking asymmetry between processes becomes possible.

Principal difficulty : grouping operation (inverse of partitioning)

to allow new permutations whenever they become possible.

- implies the ability to define larger orbits

(thesis of Soheib Baarir – 2007)

(see GreatSPN model checker from WN)

Conclusion on Symmetry Reduction Techniques

- Symmetry are
 - mainly based on behaviour
 - some data symmetry are captured
 - (state) complexity worse case : $\text{SIZE}(M)$; best case $\text{SIZE}(M)/\text{SIZE}(G)_{\text{symm}}$
 - assymetry can be also handeld
- Symmetry model checker tools
 - GreatSPN+ with SPOT (explicit, from Well-formed PetriNets and LTL),
 - CPN-AMI symmetric tools with SPOT (symbolic, SDD, from WN+)
 - SMV (symbolic, from (symmetrical) scalarsets specification)
 - others : SMC (explicit, CTL), SymmSPIN (explicit), SYMM(symbolic, CTL)
- Other symmetry interest :
 - (stochastic) performance evaluation : GreatSPN+ from SWN formalisms

More conclusion on reduction techniques

- Common points between symmetry and observation
 - both reason on SETs of states
 - both apply to LTL model checking on the fly
 - Reduction techniques are heuristics (their efficiency depend on the system and property)
 - both have been implemented and intensively tested on use cases in Lip6 laboratory
- Specific to symmetry reduction
 - in practice, need a specification (to be handled efficiently and automatically)
 - well-adapted to distributed systems and algorithms : Ressource Management systems, Fault Tolerance Protocols, Cache Coherence Protocols,
- Specific to observation reduction
 - assumption : require few things to be observed at a time
 - LTL reduces to free next LTL
 - efficiency : related to what to be observed in some system
- Reduction techniques can be mixed, hoping even better efficiency
 - Symmetry (up to its representation), Observation, Abstraction : orthogonal
 - (real) time reduction techniques (class graph, zone graph) : orthogonal

Appendix

Basic group theory on permutations

Perm(AP)
Aut(M)

(G , ◦) is a group :

- ◦ : $G \times G \rightarrow G$ is associative
- neutral element : G contains a neutral element 1_G
- inverse : for each g , $\exists g^{-1} : g \circ g^{-1} = g^{-1} \circ g = 1_G$

Standard notations :

- composition : $(g1 \circ g2)(x) = g1(g2(x))$
- subgroup H $H \leq G$

Let

G be a non empty set

◦ : binary operation on G

Equiv
classes
over
the states

S be a finite set of states

and G be a group of permutations acting on S

- Orbit of an element s of S : $[s]_G$ the set $\{s' : \exists g \in G : g(s') = s\}$
- Equivalence : $s \sim_G s'$ s and s' are in the same orbit
- Representative : $\text{rep}([s]_G)$ representative of the orbit of s

A permutation is simply a bijection from S to S

The set of all permutations over S forms a group under the possible compositions

What is data types and functions in a Well-formed Petri Net

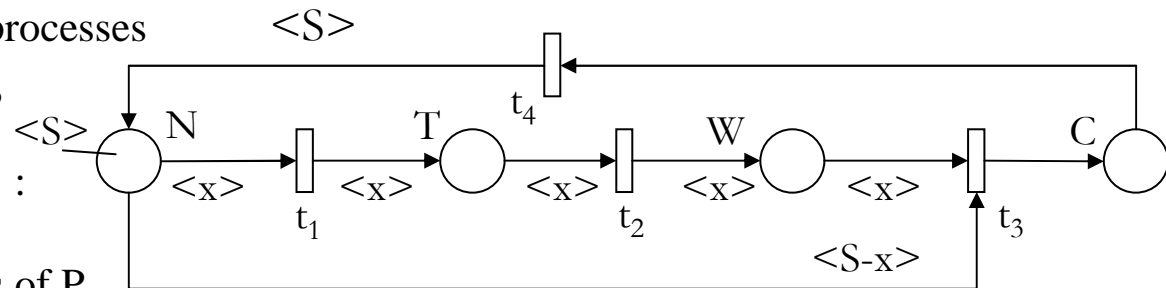
Analogy between program specifications and Well-formed Petri Net

- The variables the places
- The values of variables the colours in places
- The variable domains The different values that can be permuted
 - Partitions are used when it is required to avoid some permutations between colours (the allowed permutations preserves the colour partitions)
- The events the instances of the transitions of the model
- The function types The different colour functions used on the arcs of the SWN

Due to their colour functions and transitions inscriptions (guard,) a Well-formed Petri net can model either symmetrical system or and partially symmetrical one.

Example of Well-formed Petri Net

- $P = 1..n$ //A simple colour class for n processes
 - x : Parameter of fonction, with $D(x)=P$
 - Colour functions used (defined over P) :
 - $\langle x \rangle$ require a process of P
 - $\langle S \rangle$ require the colour of every process of P
 - $\langle S-x \rangle$ require all colours of processes by not the one represented by $\langle x \rangle$.
- Colour fonctions, then parameter instances are local to transition and arcs.



N1 : Neutral
 P2 : Try
 W3 : Wait
 C4 : critical section

- ☺ Note : More complex data are possible
- ☺ Note : priority and guard can be expressed attached to arc, transition or event, Then, the colour class can be partitioned to capture this "asymmetries" between processes.
 - statically or
 - dynamically.

This Well-formed Petri net features a mutual algorithm with deadlocks for any (finite) number of processes.

How to exploit BDD and symmetry

Assuming a representation of states using boolean variables.

- Any state is modeled as a vector in $[k]^n$

where k is the (maximum) size of the possible process local states.

- The action of a symmetry group on a boolean vector (x_1, x_2, \dots, x_m) correspond to the swapping of bit values.

For any symmetry α :

$$\alpha(x_1, x_2, \dots, x_m) = (x_{\alpha(1)}, x_{\alpha(2)}, \dots, x_{\alpha(m)})$$

- The orbit relation can be represented by a set of pair $\{(s,t) : t \in [s]\}$.
- The canonization function according to a symmetry group and a boolean vector x corresponds to find the lexicographic least element in the orbit of x (the constructive orbit problem)