A *Complete* Method for Symmetry Reduction in Safety Verification

Duc-Hiep Chu and Joxan Jaffar
National University of Singapore

Motivation

- Settings:
  - Concurrent program is defined parametrically
  - The number of processes is known (n)
- The interleaving space contains many symmetric subtrees
  - A subtree might have up to n! symmetric images
- Consequence: If symmetry reduction is properly exploited, the benefit is HUGE

Background

- Given an n-process system, let
  - I = [1 … n] denote its process indices
  - π denote some *permutation* on I

    A permutation π *acts* on object (formula) F by simultaneously replacing each occurrence of index i by π(i)

    E.g. Let n = 2, π = {1 -> 2, 2 -> 1}.

    $\pi(id_1 < 3 \wedge id_2 > 4 \wedge x = 10) = (id_2 < 3 \wedge id_1 > 4 \wedge x = 10)$
  - π^{-1} denote the inverse of π

Traditional Symmetry Reduction

Strong Symmetry

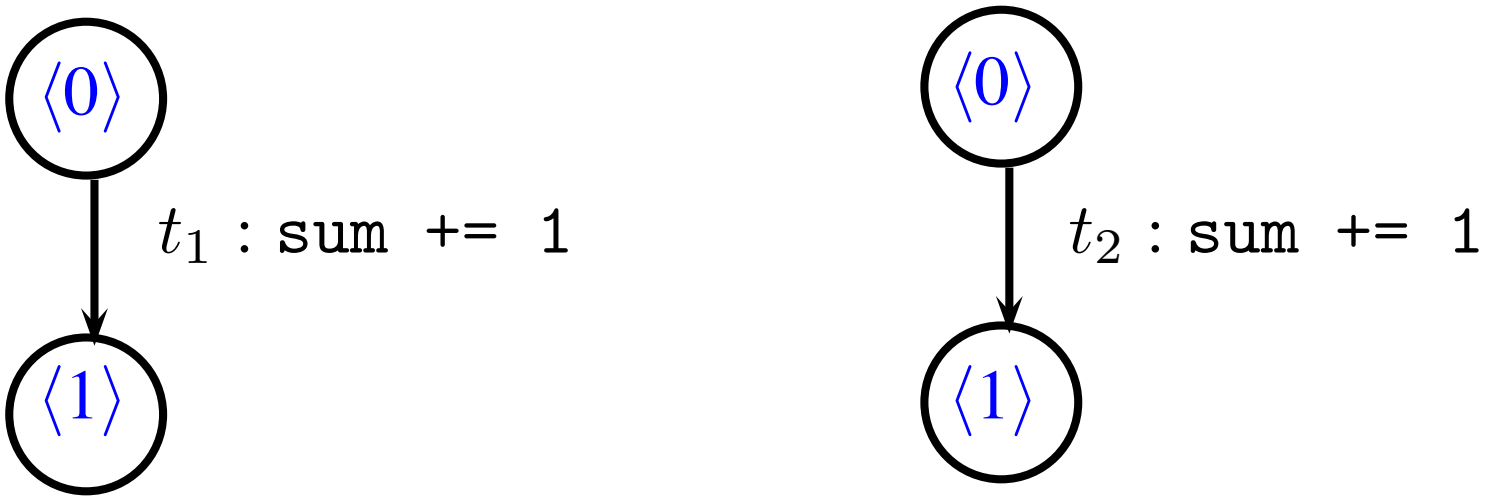
Def: Given safety condition ψ such that π(ψ) is equivalent to ψ, state s is strongly π-similar to s' if :

1. π(s) = s'
2. for each transition t, s -- t --> d, we have
   s' -- π(t) --> d', d is strongly π-similar to d'
3. for each transition t', s' -- t' --> d', we have
   s -- π^{-1}(t') --> d, d is strongly π-similar to d'

Traditional Symmetry Reduction

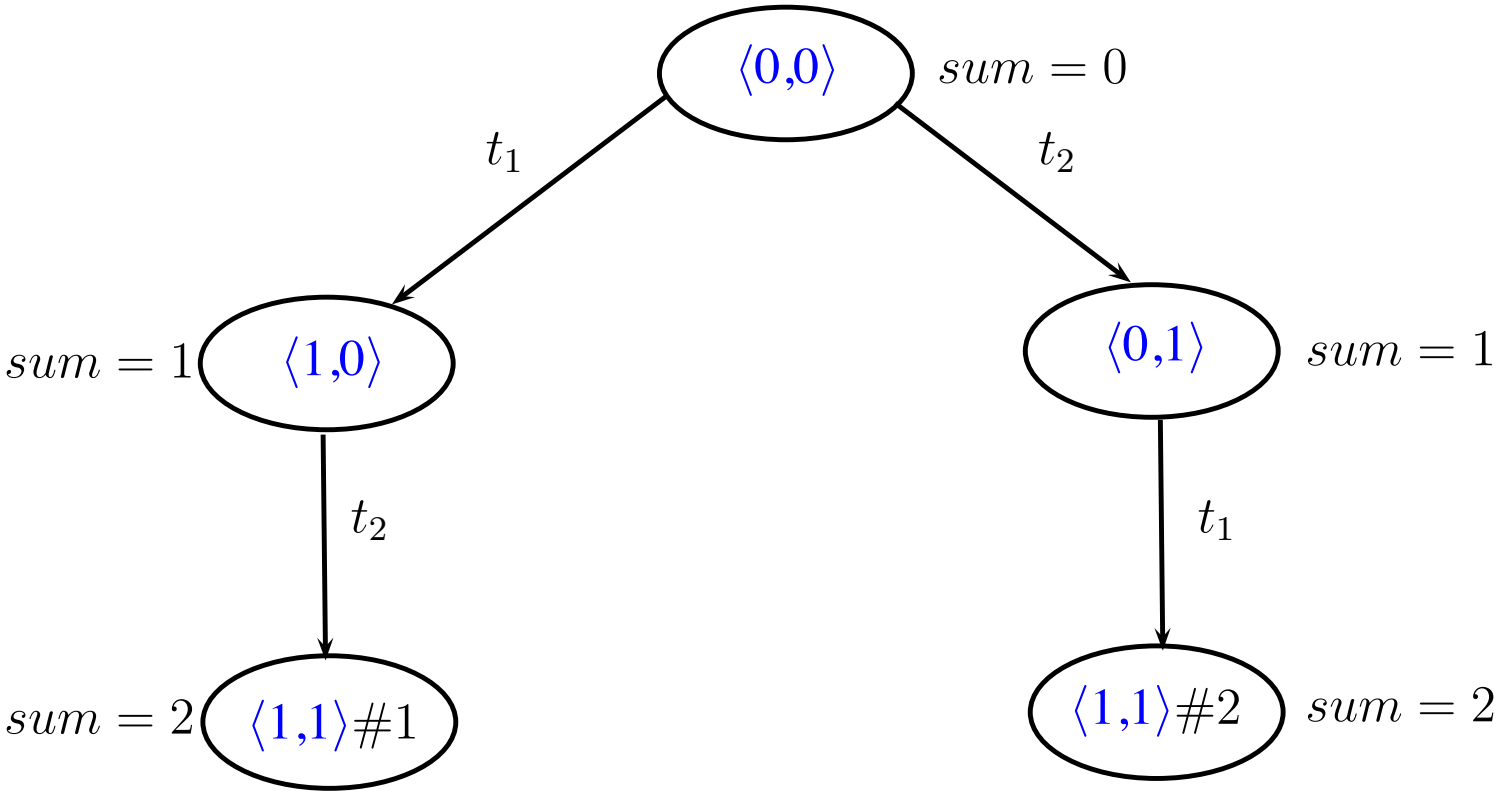
- Detecting 2. and 3. is hard
- Rely on all processes being *identical*

Example (Increment)

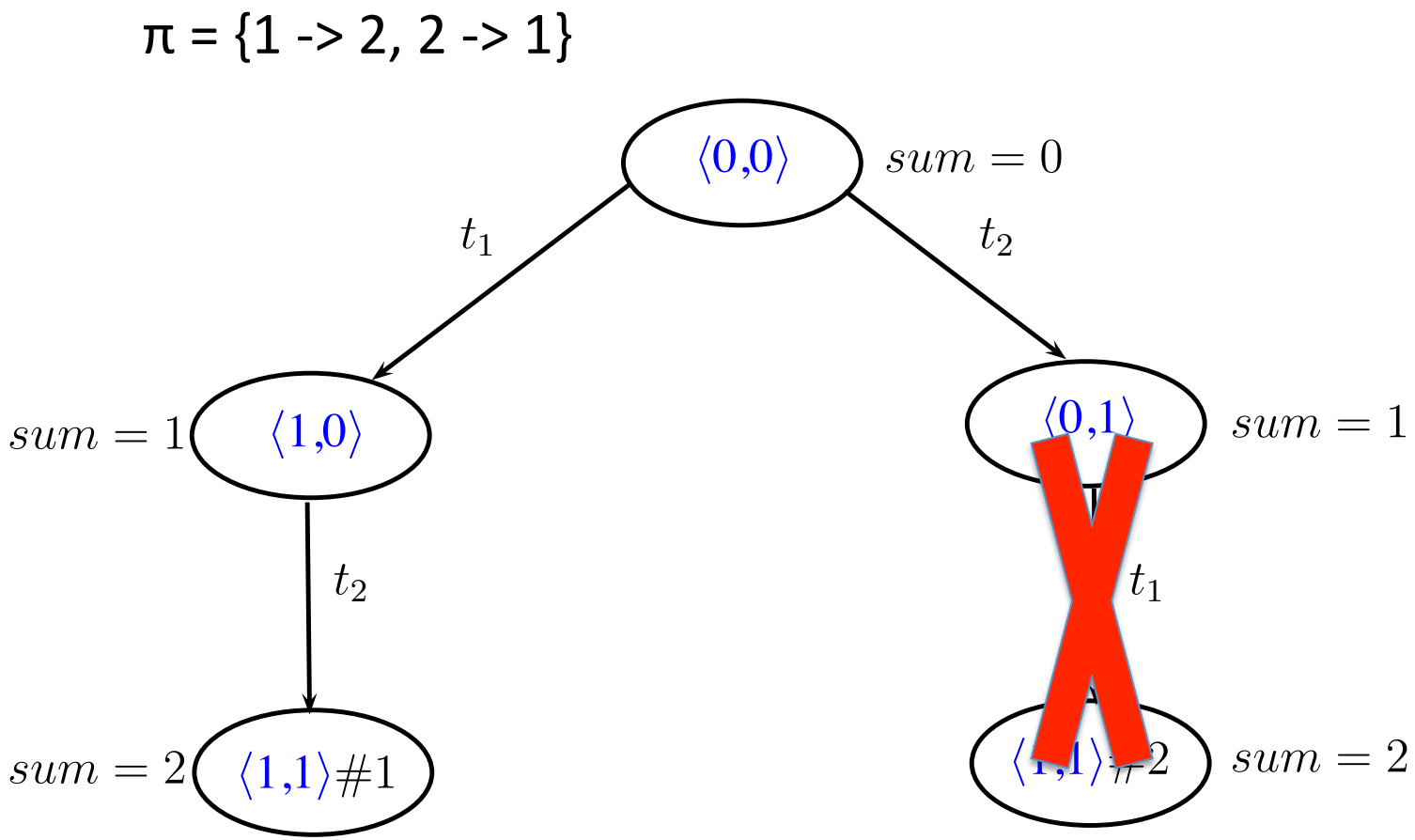


Example (Increment)

π = {1 -> 2, 2 -> 1}



Example (Increment)



We don’t always have identical processes

```
process(id) {
   if (id == master_id) {
      /* code for master process */
   else {
      /* code for slave processes */
   }
}
```

- It is an unreasonable assumption
  - Excludes many systems

Related Work

- Traditional symmetry reduction methods exploit *perfect* symmetry, relying on the fact that all component processes are identical
- [Emerson, 99] considered *near* and *rough* symmetry, which later generalized to *virtual symmetry* [Emerson, 00]. No implementation provided
- [Sistla, 04] and [Wahl, 07] are closest to us, in allowing behaviors of processes to range from totally identical to arbitrarily divergent
- All of them attempt to capture *strong* symmetry

Our Symmetry Reduction

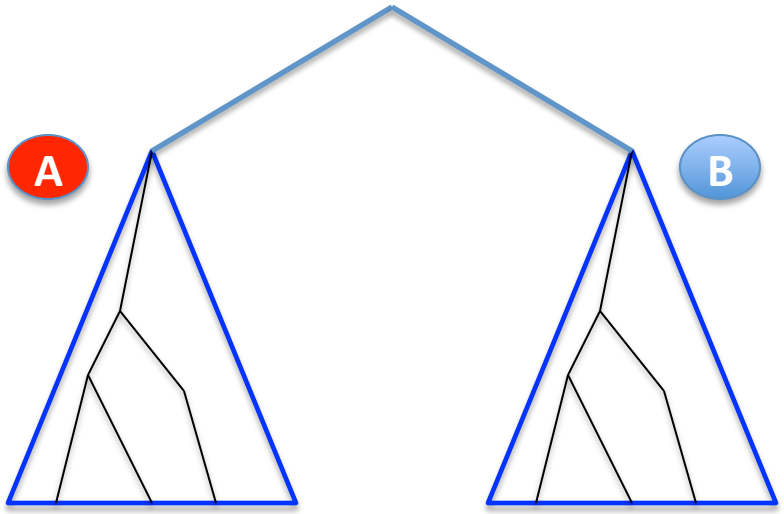
Weak Symmetry (property driven)

Def: Given safety condition ψ such that $\pi(\psi)$ is equivalent to ψ, state s is weakly π-similar to s' if :

1. π(program point of s) = program point of s'
2. s models ψ iff s' models $\pi(\psi)$
3. for each transition t, s -- t --> d, we have
   s' -- π(t) --> d', d is weakly π-similar to d'
3. for each transition t', s' -- t' --> d', we have
   s -- π^{-1}(t') --> d, d is weakly π-similar to d'

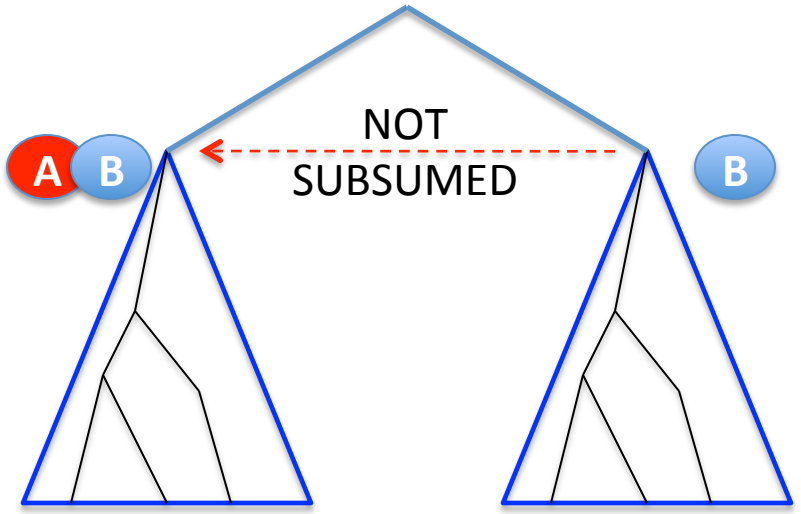
State Interpolation

A and B are sibling sub-trees (same program point, different context)



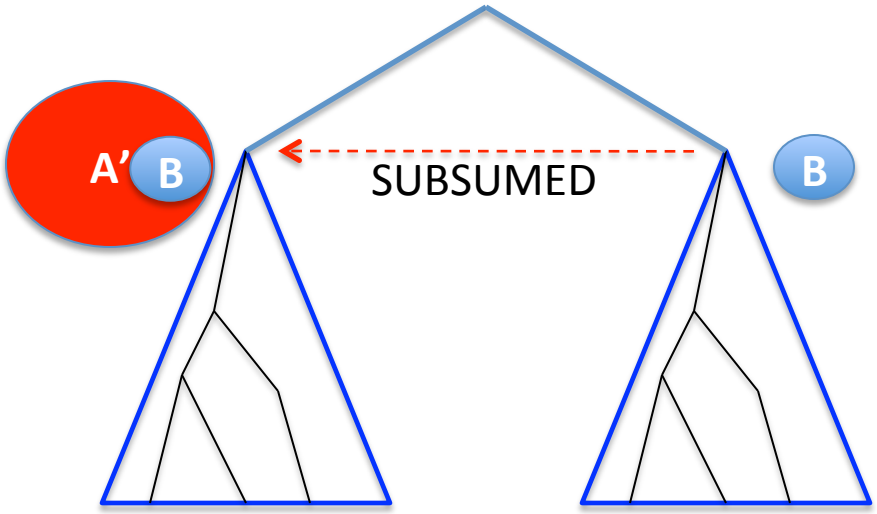
State Interpolation

A and B are sibling sub-trees (same program point, different context)



State Interpolation

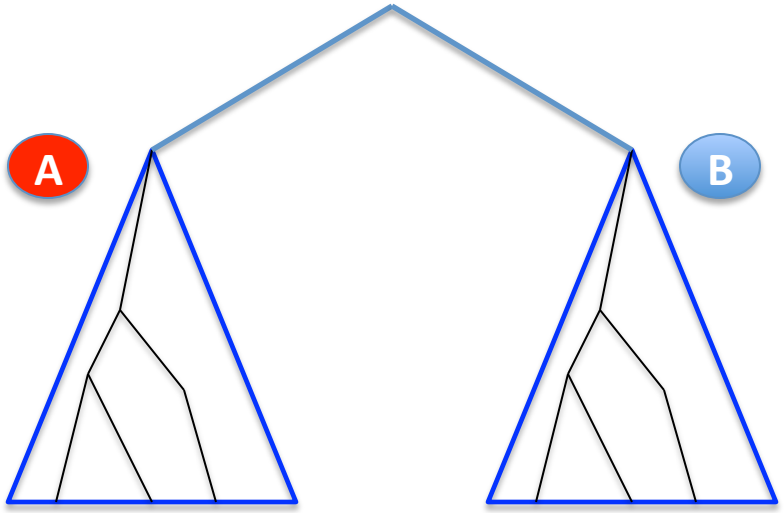
A and B are sibling sub-trees (same program point, different context)



Generalize A (to A') while preserving safety

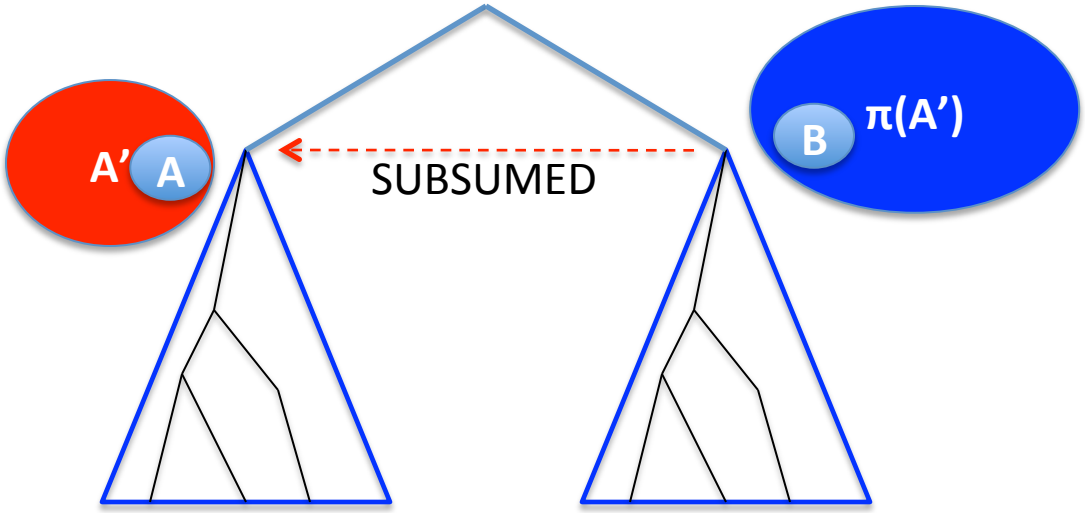
Pruning with Weak Symmetry

A (program point p_A) and B (program point p_B) are siblings and $\boldsymbol{\pi}(p_A) = p_B$ i.e. symmetric program points



Pruning with Weak Symmetry

A (program point p_A) and B (program point p_B) are siblings and $\boldsymbol{\pi}(p_A) = p_B$ i.e. symmetric program points



Generalize A (to A') while preserving safety
Apply π to A'

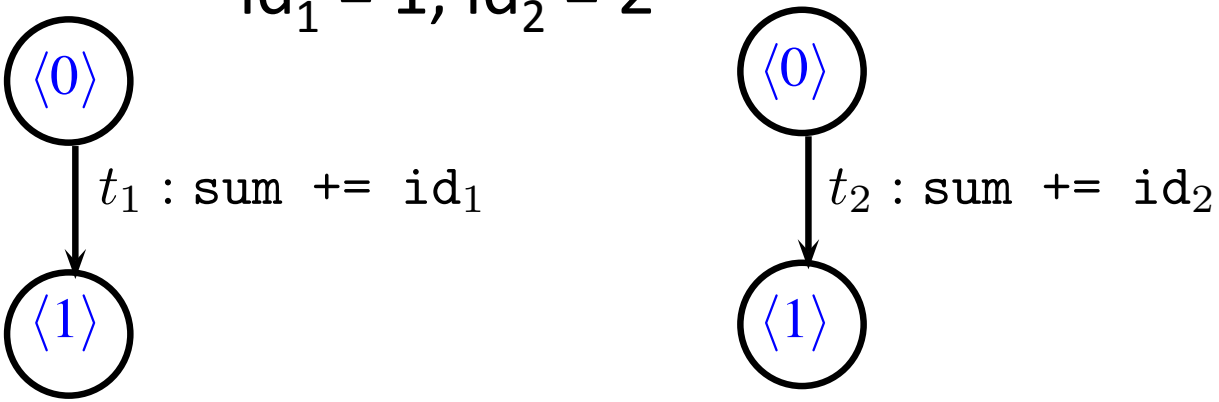
Our Language

- Allow the use of variable id
  - id is initialized to a unique value in each process
  - for simplicity, id ranges from 1 ... n
  - value of id can not be changed
- The behaviors of processes can range from totally identical to arbitrarily divergent

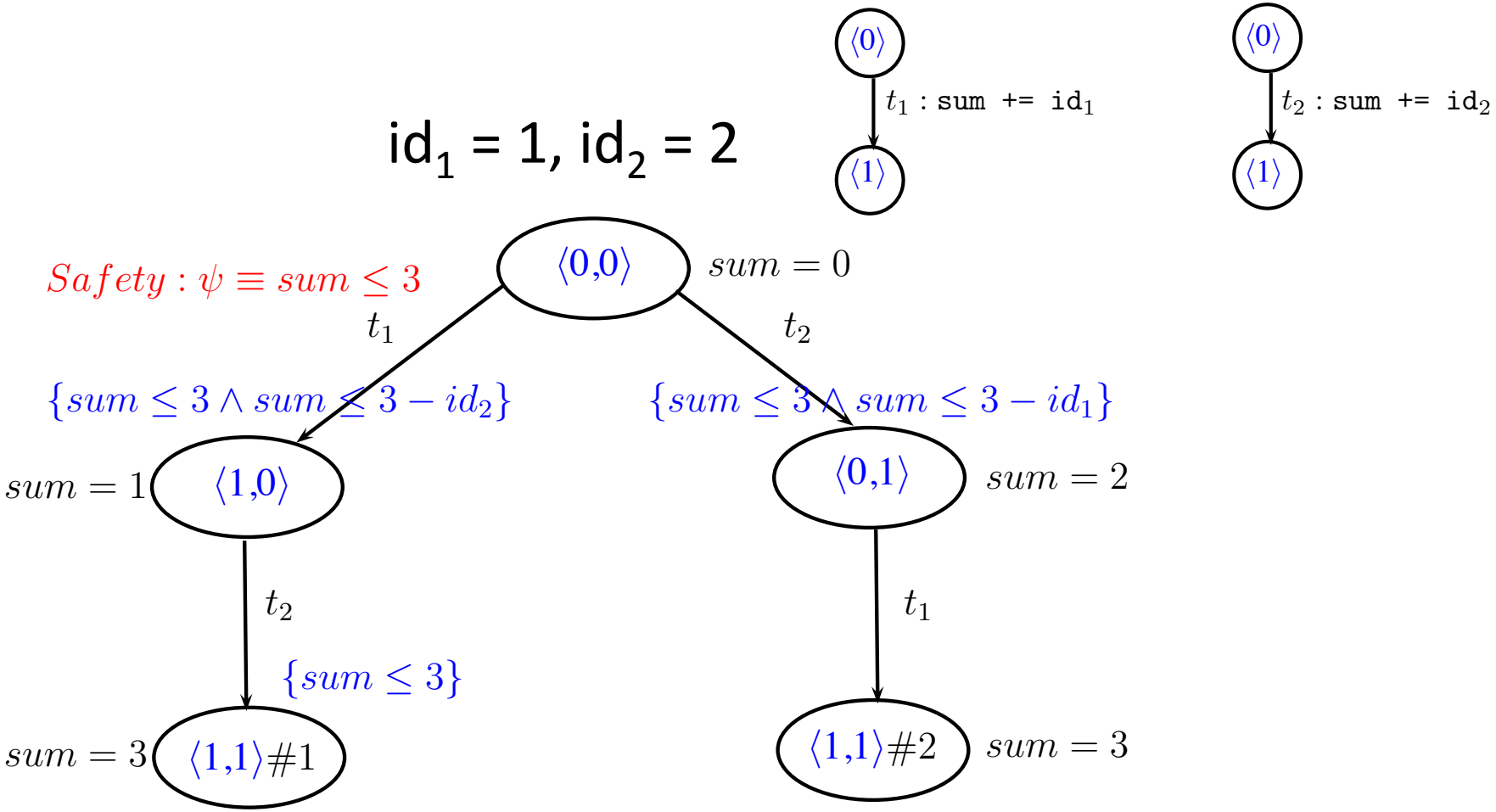
Example (Weak Symmetry)

```
sum = 0
process(id) {
   sum += id
}
```

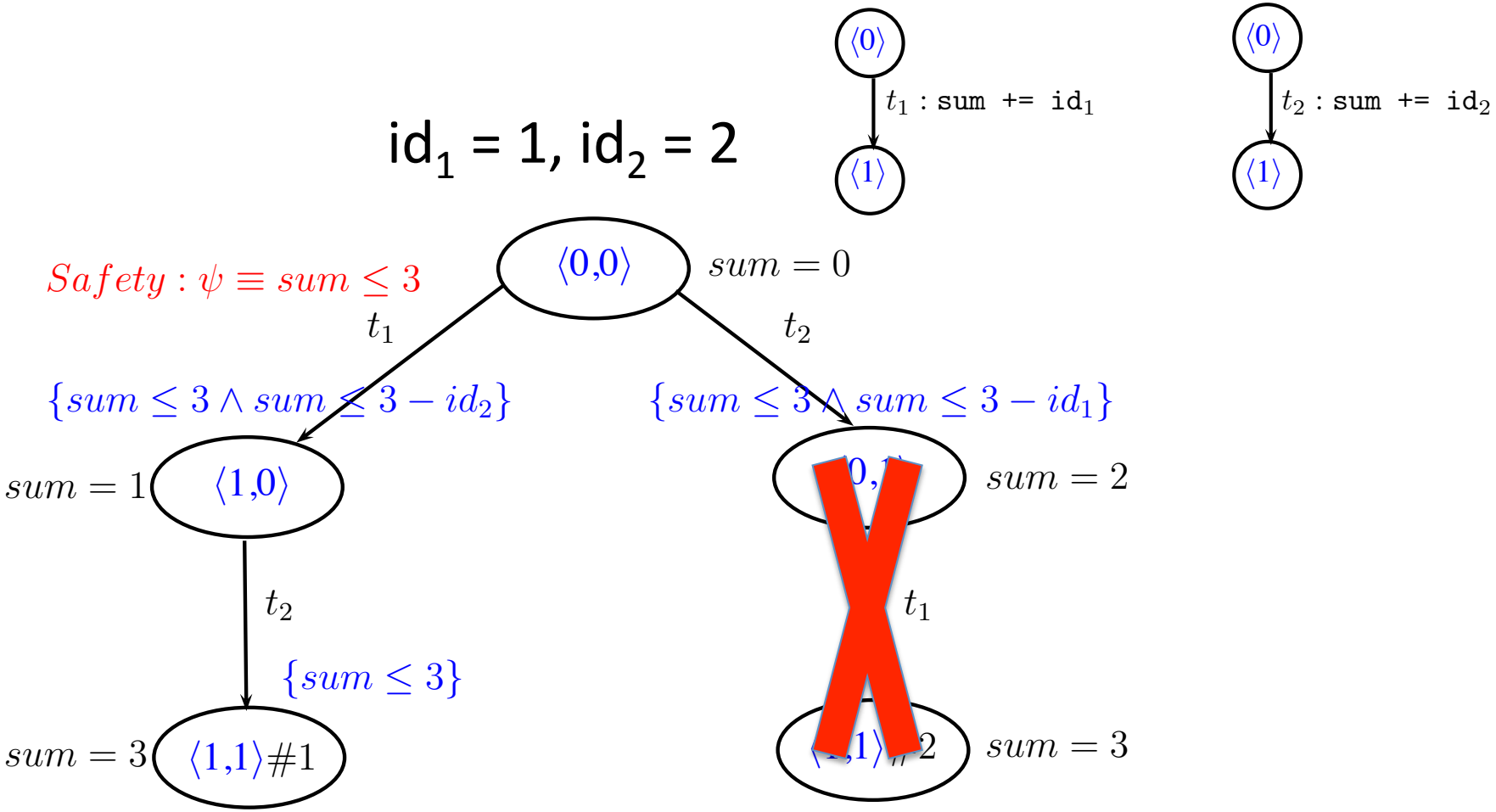
$id_1 = 1, id_2 = 2$



Example (Weak Symmetry)



Example (Weak Symmetry)



Example (Violation of Symmetry)

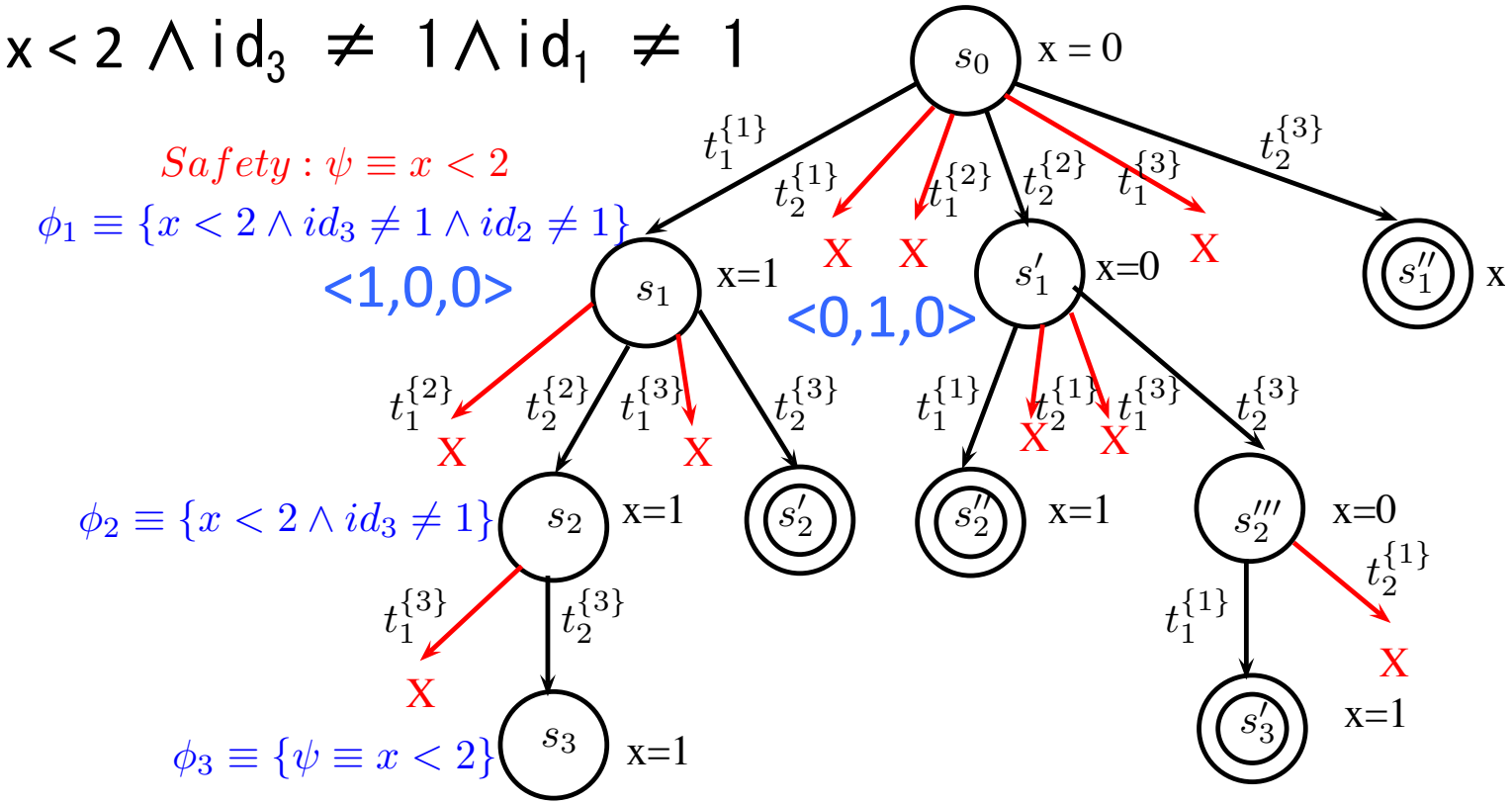
```
x = 0;
process(id) {
   if (id == 1) x++;
}
```

- Instantiated to n = 3 processes
- Safety: x < 2

Example (Violation of Symmetry)

$id_1 = 1, id_2 = 2, id_3 = 3$

$x < 2 \wedge id_3 \neq 1 \wedge id_1 \neq 1$



Note: $\phi_1' \equiv \{x < 1 \wedge id_1 = 1 \wedge id_3 \neq 1\}$

Completeness

- *Completeness* means that "given two states which are weakly symmetric, we *will not explore them both* in our search space"
- pre(t, φ) computes the precondition wrt. postcondition φ and transition t

Def: The precondition operator pre is said to be *monotonic* wrt. transition t if for all φ_1, φ_2: if φ_1 is weaker than φ_2 then pre(t, φ_1) is weaker than pre(t, φ_2)

Completeness

Theorem: Our symmetry reduction is *complete* wrt. weak symmetry if our precondition operator is monotonic wrt. every transition

Experiments
Sum-Of-Ids

	Complete Symmetry Reduction			SPIN (w/t Symmetry Reduction)		
#Processes	**Visited**	**Subsumed**	**T(s)**	**Visited**	**Subsumed**	**T(s)**
10	57	45	0.02	6146	4097	0.03
20	212	190	0.04	115334338	9437185	69.70
40	822	780	0.37	-	-	-
100	5052	4950	22.09	-	-	-

Experiments
Reader-Writer Protocol

		Complete Symmetry Reduction			Lazy Symmetry Reduction [Wahl, CAV07]	
#Readers	**#Writers**	**Visited**	**Subsumed**	**T(s)**	**Abstract States**	**T(s)**
2	1	35	20	0.01	9	0.01
4	2	226	175	0.19	41	0.10
6	3	779	658	9.93	79	67.80
8	4	1987	1750	3.23	165	81969.0
10	5	4231	3820	9.21	-	-

Experiments
Bakery Algorithm

	Complete Symmetry Reduction			State Interpolation		
# Processes	**Visited**	**Subsumed**	**T(s)**	**Visited**	**Subsumed**	**T(s)**
3	65	31	0.10	265	125	0.43
4	182	105	0.46	1925	1089	5.89
5	505	325	2.26	14236	9067	74.92
6	1423	983	11.10	-	-	-

Summary

- We weaken the notion of symmetry
  - Property-driven
- An interpolant for a subtree can be permuted to prune symmetric subtrees
- Our symmetry reduction algorithm is *complete* wrt. the notion of weak symmetry