Combining Partial Order Reduction with Bounded Model Checking

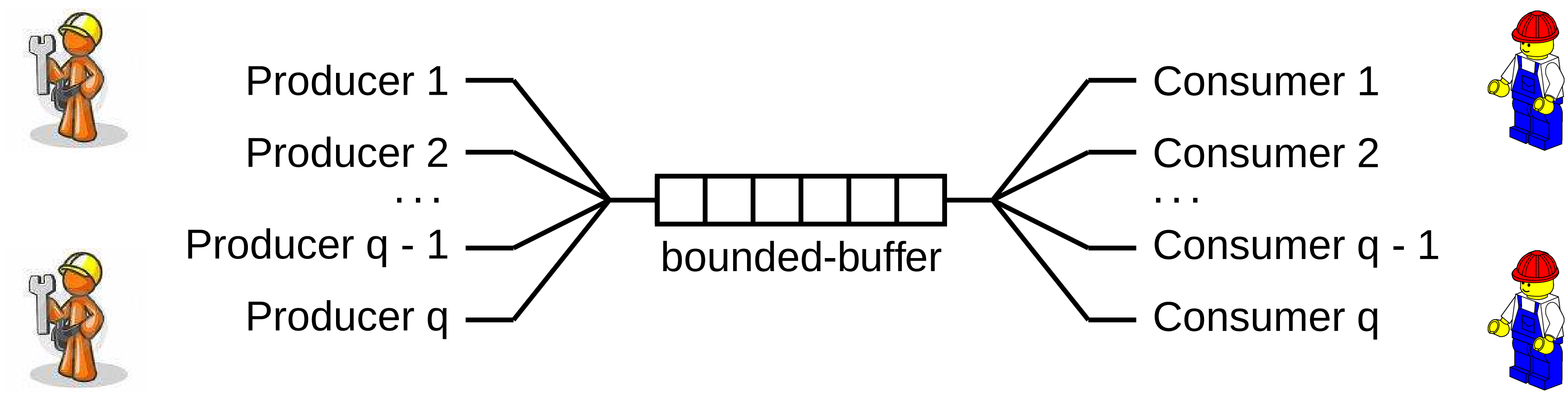
CPA 2009

José Vander Meulen and Charles Pecheur

UC Louvain

A Concurrent System

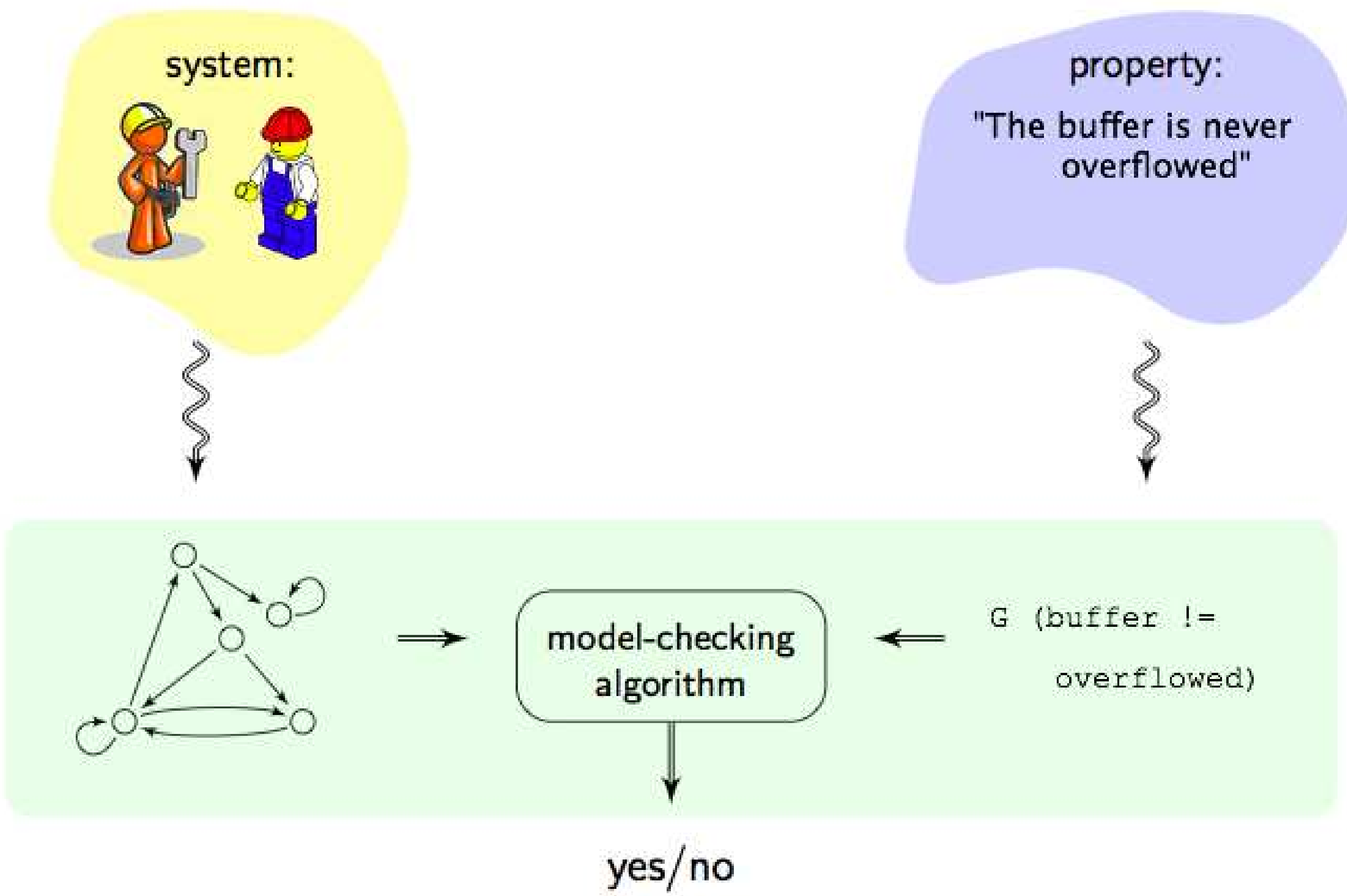
- Set of asynchronous and interacting processes



- Can we verify this system with Symbolic Model Checking?
- Up to what q?

Model Checking

- **Exhaustive** exploration of the state space of a system



Symbolic Model Checking

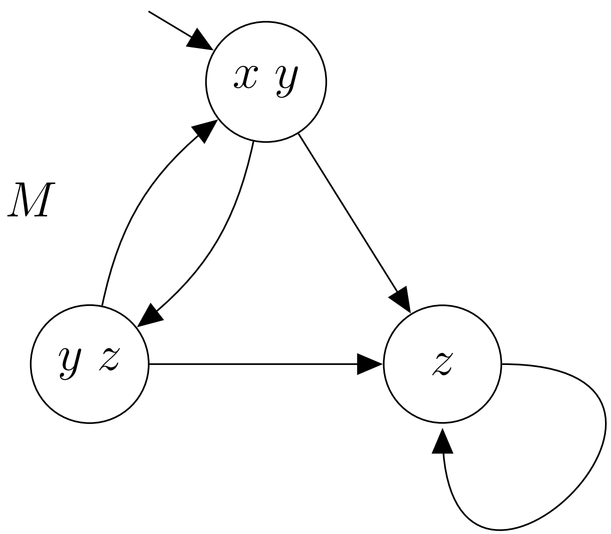
- Principle:
  - Compute **sets of states** (BDDs), or
  - Resolve a **SAT** problem (BMC)
- Brilliant results in the hardware domain [Biere $^+$ 03, Mc Millan 93]
- Conventional wisdom: Symbolic Model Checking methods are not well suited for asynchronous systems.
- How can we use symbolic Model Checking with asynchronous system?

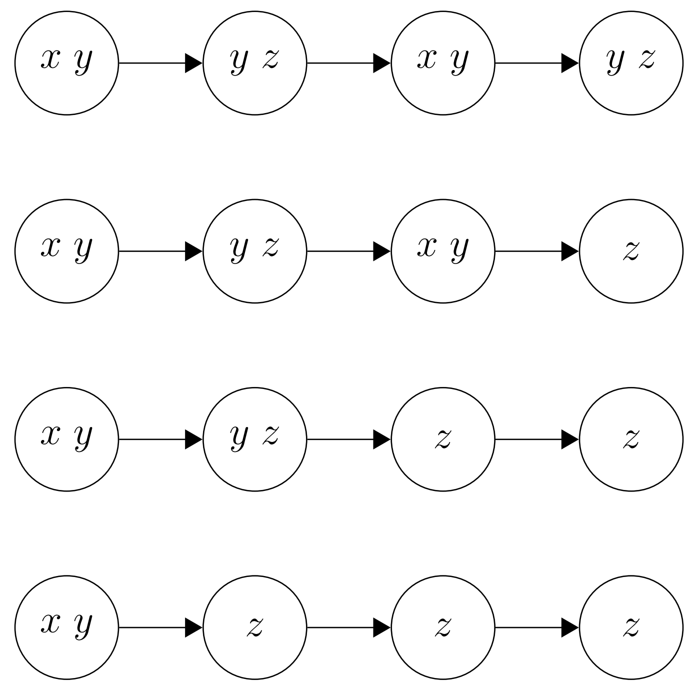
Outline

- Background
  - Bounded Model Checking
  - Partial Order Reduction
- Combining Partial Order Reduction with Bounded Model Checking
- Experimental results
- Conclusion
- Perspectives

Bounded Model Checking [Biere + 99]

- Search for a counterexample in executions whose length $= k$
  - e.g. paths of length 3





Bounded Model Checking [Biere + 99]

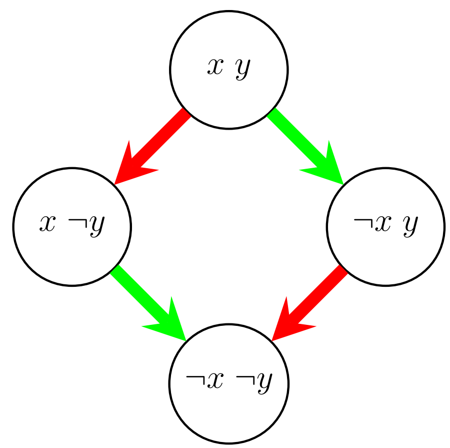
- Reduce model checking problem to a SAT problem
- Unfold the transition relation k times to obtain a boolean formula $[\![M]\!]_k$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \wedge \cdots \wedge T(\vec{x}_{k-1}, \vec{x}_k)$$

- Translate the negation of a LTL property f to a Boolean formula $[\![\neg f]\!]_k$
- If $[\![M]\!]_k \wedge [\![\neg f]\!]_k$ is satisfiable, an error is found

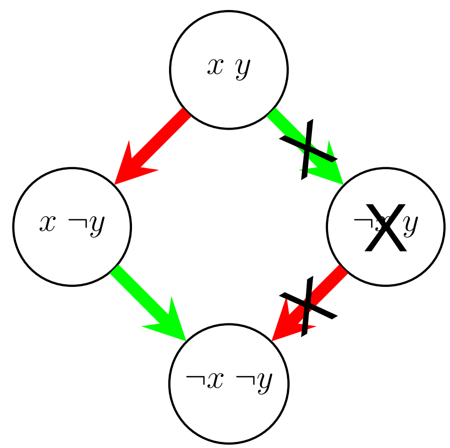
Partial Order Reduction

- **Partial order reduction** methods are best suited for asynchronous systems
  - Can we use these methods with BMC and LTL?
- **Verification** = only check some interleavings of a transition system
- Based on **independence** between transitions and **invisibility** of a transition



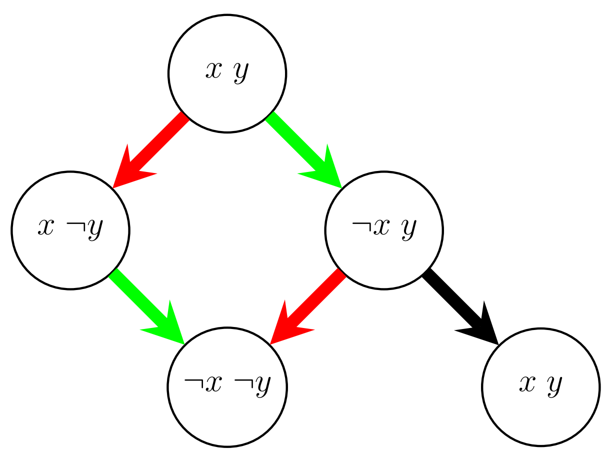
Partial Order Reduction

- **Partial order reduction** methods are best suited for asynchronous systems
  - Can we use these methods with BMC and LTL?
- **Verification** = only check some interleavings of a transition system
- Based on **independence** between transitions and **invisibility** of a transition



Partial Order Reduction

- **Algorithm**: modified depth-first search (DFS)
  - At each step s, a subset of the successors is selected: $ample(s)$
  - $ample(s)$ has to respect a set of conditions
- **c1**: Along every path in the full state graph that starts at s: a transition that is dependent on a transition in $ample(s)$ cannot be executed without a transition in $ample(s)$ occurring first.

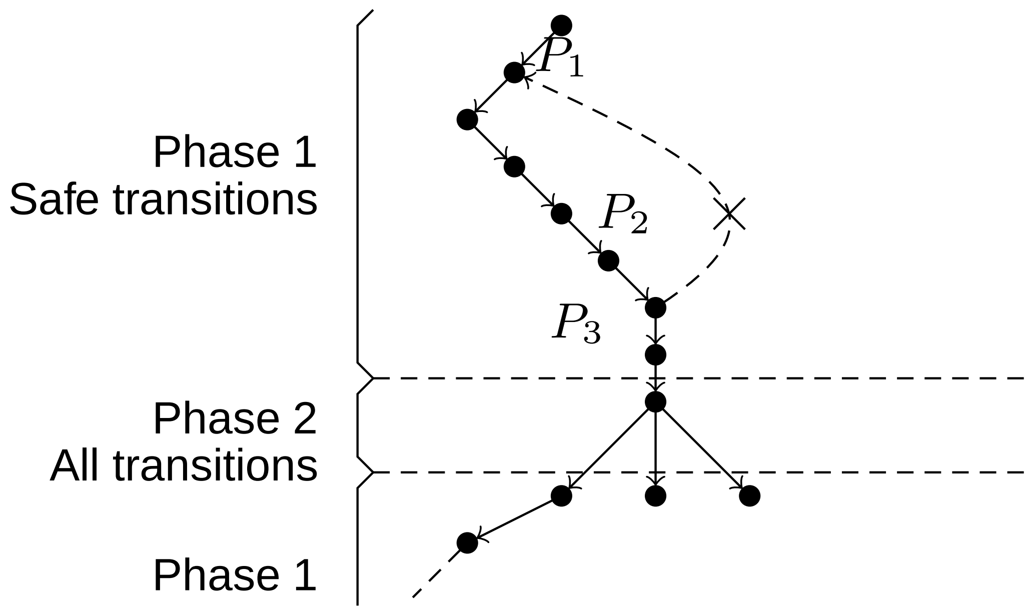


Partial Order Reduction

- **c2** at least one state s per cycle is fully expanded
- **c3** If $ample(s) \neq enable(s)$, all transitions in $ample(s)$ are invisible.
- **c4** if $ample(s) \neq enable(s)$, then $ample(s)$ is a singleton
  - **C1 – C3** preserve deadlocks, LTL_X properties
  - **C1 – C4** preserve CTL_X properties

Two-phase algorithm [Nalumasu + 97]

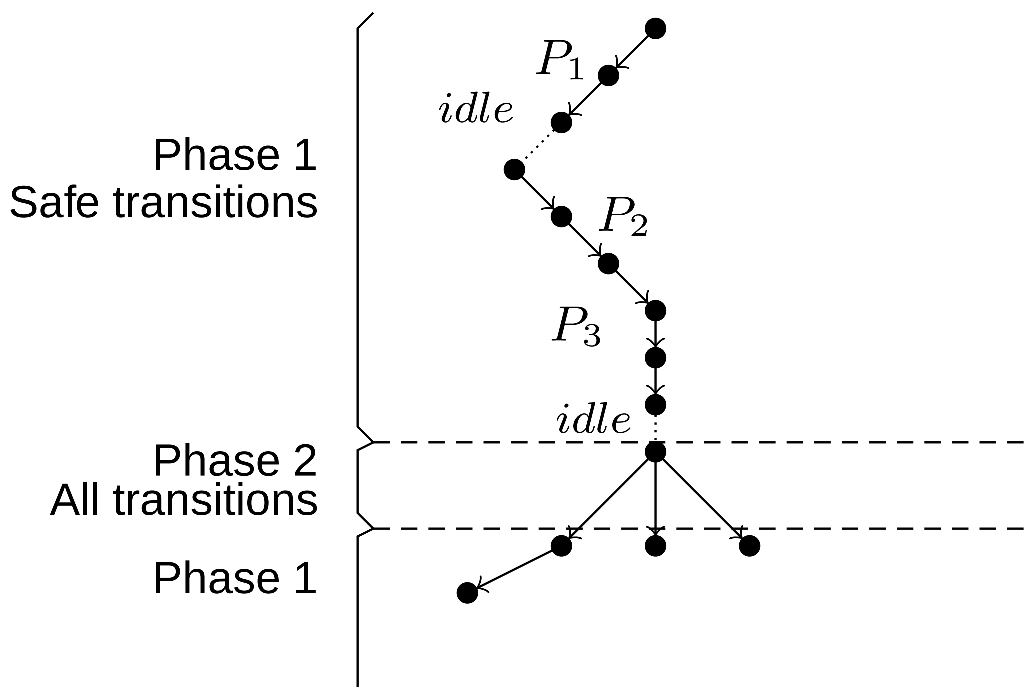
- A modified DFS: performs alternatively 2 phases
  - Phase-1: explore for each process as many safe transitions (**C1, C4**) as possible
  - Phase-2: fully expand the current state



- Two-phase algorithm can check CTL_X properties

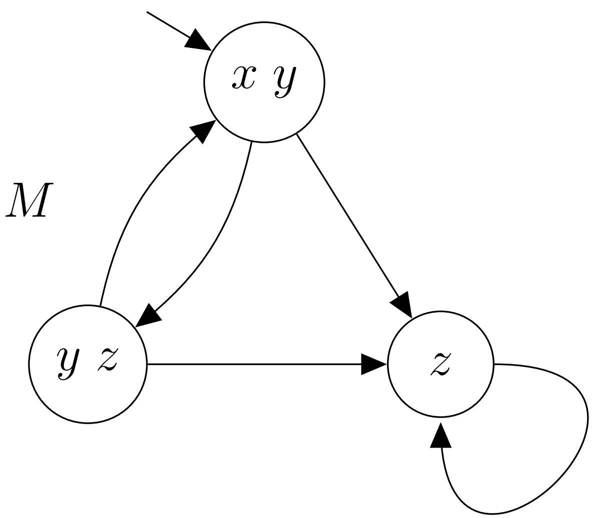
SBTP

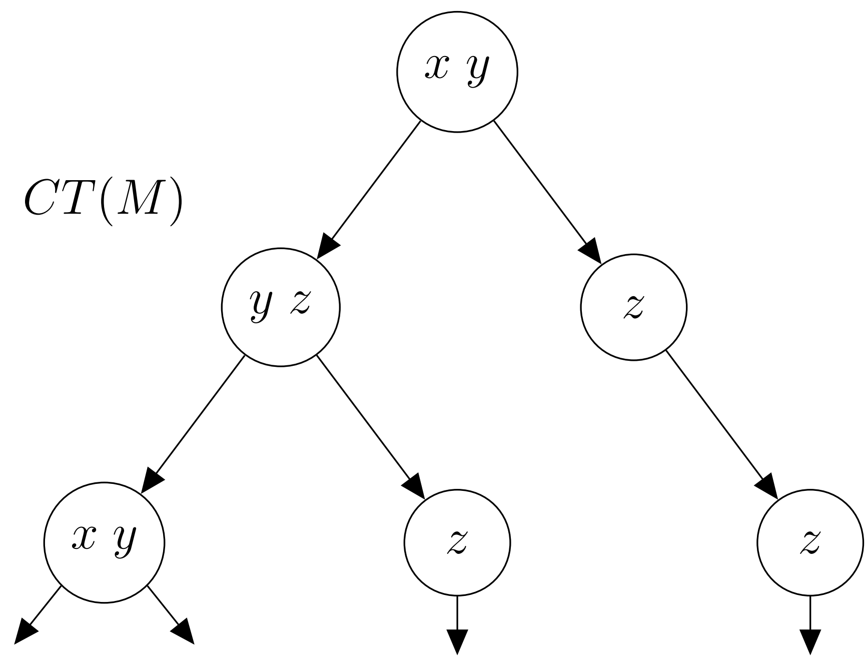
- Algorithm combining POR with BMC:
  - SBTP: Phase-1 performs a fixed number n of partial expansions for each process
  - A process might not be able to produce n safe transitions ($idle$ transitions)



SBTP

- From a transition system to a computation tree



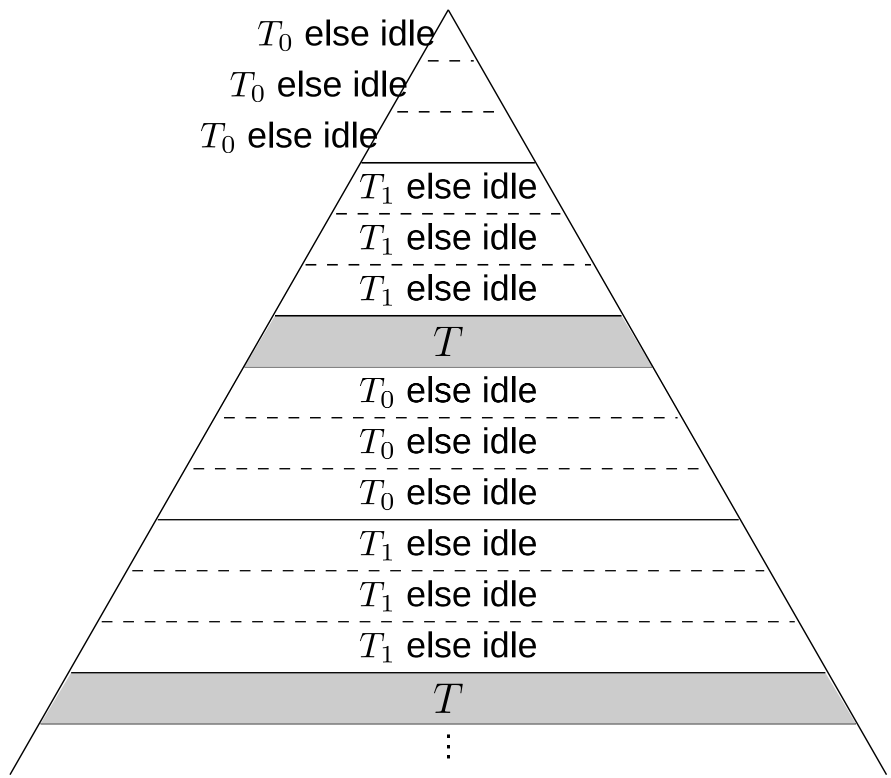


- M and $CT(M)$ are equivalent

SBTP

- A modified computation tree ($\approx CT(M)$)
- Given p processes, a fixed number n of partial expansions, construct a reduced computation tree.
  - e.g number of processes $p = 2$, and $n = 3$

$SBTP(M, n)$



SBTP

- Given p processes, a fixed number n of partial expansions, and $k = m(p \times n + 1)$, apply m times the two phases to obtain $[\![M]\!]_{k,n}^{SBTP}$
  - e.g number of processes $p = 2$, and $n = 3$

$$\left[\bullet \xrightarrow{T_1^{idle}} \bullet \xrightarrow{T_1^{idle}} \bullet \xrightarrow{T_1^{idle}} \bullet \xrightarrow{T_2^{idle}} \bullet \xrightarrow{T_2^{idle}} \bullet \xrightarrow{T_2^{idle}} \bullet \xrightarrow{T} \bullet\right]^m$$

- Translate the negation of a LTL_X property f to a boolean formula $[\![\neg f]\!]_k$
- If $[\![M]\!]_{k,n}^{SBTP} \wedge [\![\neg f]\!]_k$ is satisfiable, an error is found

Justification

There exists k $\geq$ 0 such that $[\![M, \neg f]\!]_{k,n}^{SBTP}$ if and only if $M \not\models f$

Our method finds a true assignment satisfying $\neg f$

$$\Longleftrightarrow$$

Classical BMC on $SBTP(M, n)$ finds a true assignment satisfying $\neg f$

$$\Longleftrightarrow$$

$SBTP(M, n)$ does not satisfy f

$$\Longleftrightarrow$$

M does not satisfy f

Tool

- Implemented in Scala:
  - Smoothly integrates features of object-oriented and functional languages.
  - Fully interoperable with Java.
- SAT part uses the Yices SMT solver.
- Main Features:
  - Modelling language based on processes and synchronization by rendezvous
  - BMC of LTL properties
  - SBTP of LTL$_X$ properties

Case Study: Producer-Consumer

- A variant of the Producer-Consumer problem:
  - with q producers, q consumers, and $n = 8$
- P_2: in all cases the buffer will eventually contain more than one piece

		BMC property P_2		SBTP property P_2		
q	states	k	sec	k	cycles	sec
1	1,059	26	73	153	9	122
2	51,859	44	29,898	297	9	211
3	3,807,747	—	—	441	9	401
4	$\approx 10^8$	—	—	585	9	1,238
5	$\approx 10^{10}$	—	—	729	9	1,338
6	$\approx 10^{12}$	—	—	873	9	1,926
7	$\approx 10^{14}$	—	—	1,017	9	4,135

Case Study: Producer-Consumer

- Influence of the parameter n when the number of producers (resp. consumers) = 2

	property P_2			
n	k	# cycles	TIME (sec)	MEM (MB)
0	44	44	29,898	131
1	95	19	855	159
2	135	15	235	167
3	169	13	305	194
4	187	11	217	192
5	231	11	375	308
6	275	11	381	240
7	319	11	583	318
8	297	9	211	224
9	333	9	240	295

Conclusion

- Combining Partial Order Reduction with Bounded Model Checking
  - From 2 Producers/Consumers ($51,859$ states) to 7 Producers/Consumers ($\approx 10^{14}$ states)
  - How to choose the number n of partial expansions during Phase-1?
  - Need to apply SBTP to other case studies (more complex, more realistic)
- Appropriate algorithm to check asynchronous systems with symbolic model-checking

Perspectives

- Extend SBTP to handle models featuring variables on infinite domains (SMT solvers)
- Automatically determine the number n of partial expansions during Phase-1
- Consolidate our prototype:
  - Perform state-of-the-art BMC translations
  - Improve input language