Improving Parallel State Space Exploration Using Genetic Algorithms

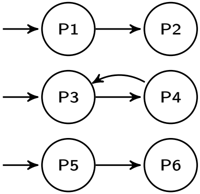
E. Renault

LRDE/EPITA

Tuesday, October 18th

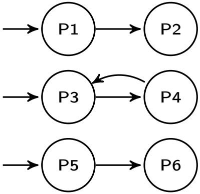
State Space & Property Checking

Let us consider a system:

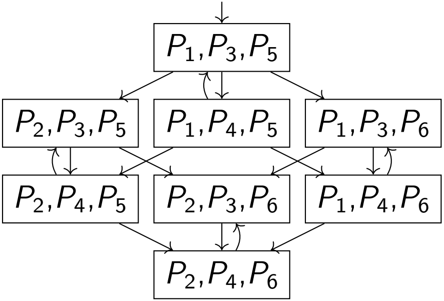


State Space & Property Checking

Let us consider a system:

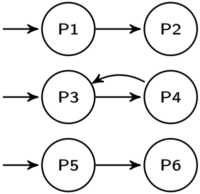


We obtain the following state space:

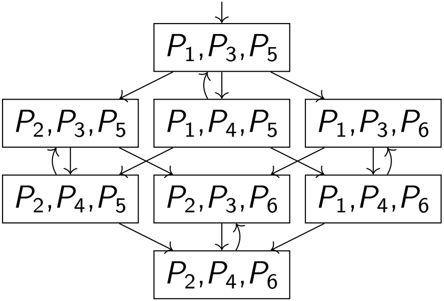


State Space & Property Checking

Let us consider a system:



We obtain the following state space:

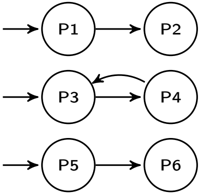


Property Checking

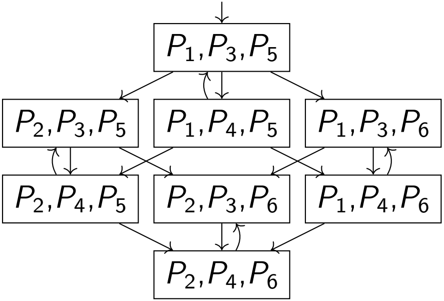
- **Safety**: involves only state space exploration

State Space & Property Checking

Let us consider a system:



We obtain the following state space:



Property Checking

- **Safety**: involves only state space exploration
- **Liveness**: involves the exploration of the synchronous product between the state-space and the (negated) property

On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.

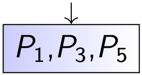
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



P_1,P_3,P_5

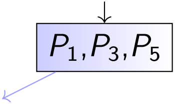
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



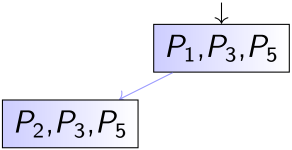
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



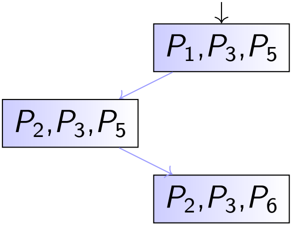
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



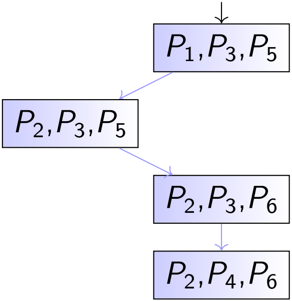
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



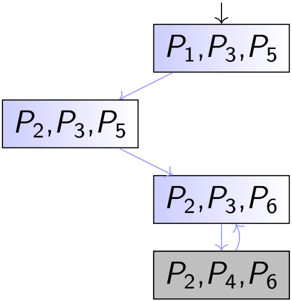
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



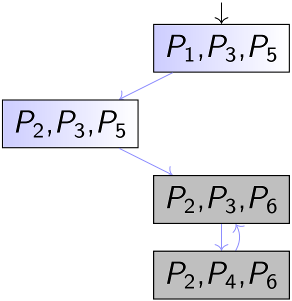
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



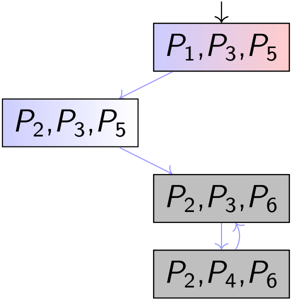
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



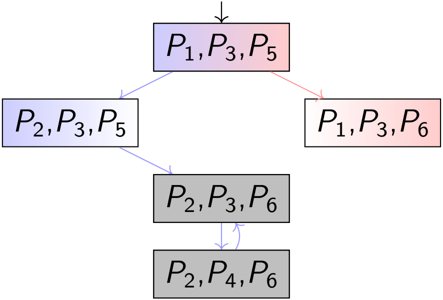
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



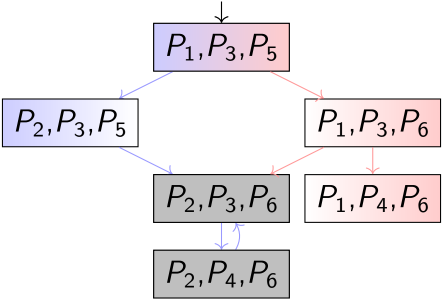
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



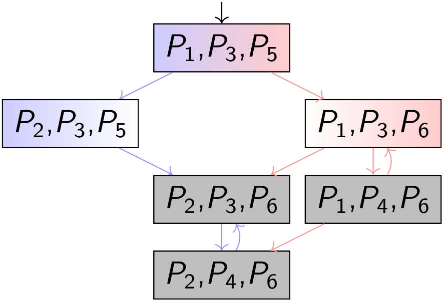
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



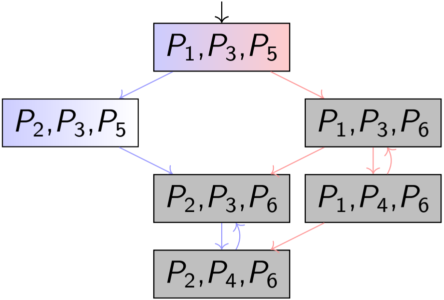
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



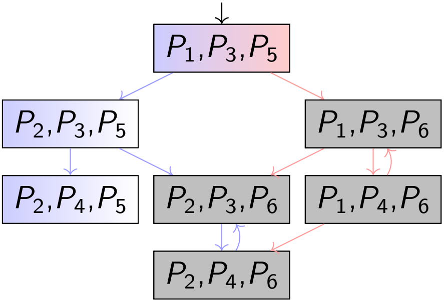
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



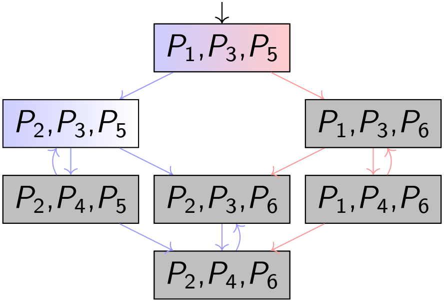
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



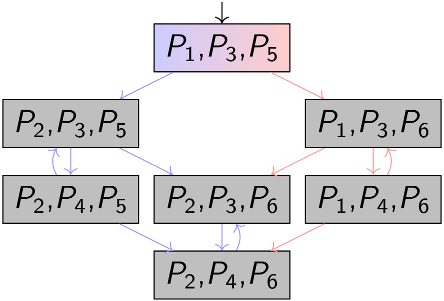
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



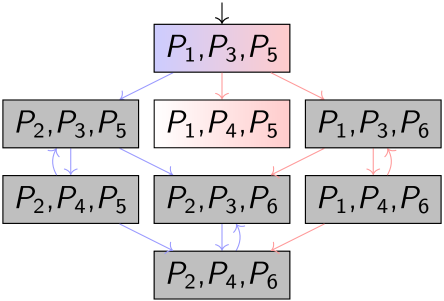
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



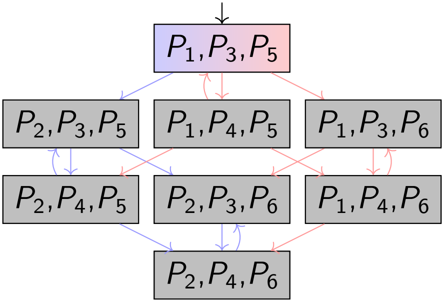
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



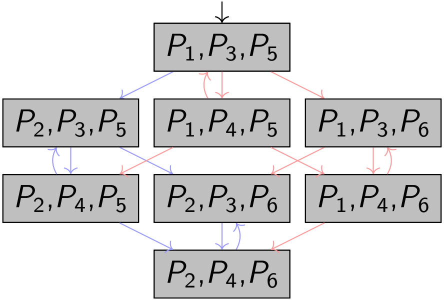
On-the-fly & Swarming for Deadlock

On-the-fly (Gerth et al. [1996])

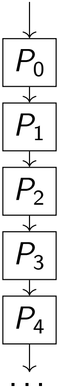
Only build the part of the state-space required to find a counterexample.

Swarming (Holzmann et al. [2011])

Run multiple parallel DFS and share information about states not participating to build a counterexample.



Problem Statement

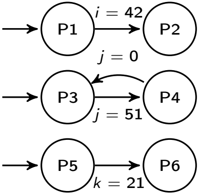


Using more than one thread couldn't bring any speedup
Using partial order reduction (Valmari [1991]) can raise this problem

If only we could start the second thread from $P3$ ...

Generation of Artificial Initial States

- More details about the system:



- Every state of the state space can be seen as :

shared variables	local var. proc 1	...	local var. proc n

How can we generate artificial initial states?

Crossover and Mutation

Perform a bounded exploration to have a pool of valid states

Crossover and Mutation

Perform a bounded exploration to have a pool of valid states

- **Crossover**: mix two states to build a third one

state_1	00000001	01001001
state_2	01001000	11001011
result	00000001	11001011

Crossover and Mutation

Perform a bounded exploration to have a pool of valid states

- **Crossover**: mix two states to build a third one

state_1	00000001	01001001
state_2	01001000	11001011
result	00000001	11001011

- **Mutation**: perform variations on a state

state_1	00000001	01001001
result	00000001	01011001

Various Fitness Function

How to detect if an artificial initial state is a good candidate?

Consider the following fitness functions and T_{avg} the average number of outgoing transition from the pool:

- **Equality**: the number of successors of a good state is equal T_{avg}. (independent processes)
- **LessThan**: the number of successors of a good state is less or equal to T_{avg}. (synchronized processes)
- **GreaterThan**: the number of successors of a good state is greater or equal T_{avg}. (non-deterministic processes)

Problems with Artificial State Generation

- Consider the effect of a mutation on a 8 integer tabular:
  `tab [i] = ...` with $i = 9$

We have to patch the transition relation to avoid such problems.

Problems with Artificial State Generation

- Consider the effect of a mutation on a 8 integer tabular:
  `tab [i] = ...` with $i = 9$

We have to patch the transition relation to avoid such problems.

- The generated state may not belong to the state space

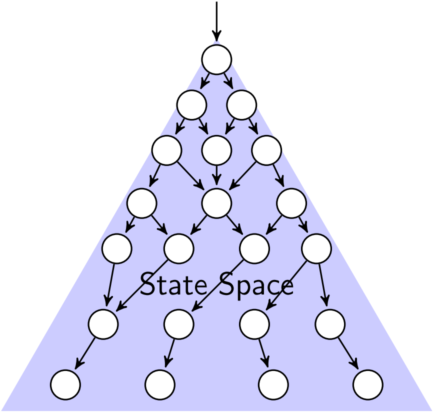
  - Spawn only one thread over two from an artificial initial state to ensure a minimum speedup
  - Once a *valid* thread stops, stops all threads

Checking Safety Properties (Deadlock Example)

Can we report a deadlock as soon as a thread detects it?

Checking Safety Properties (Deadlock Example)

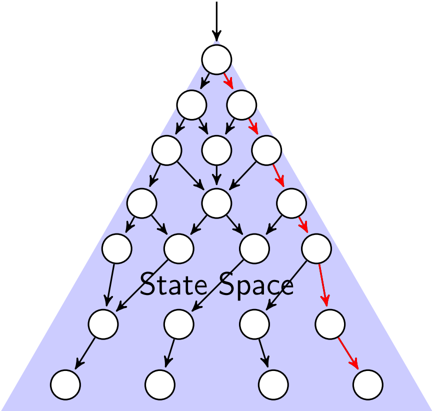
Can we report a deadlock as soon as a thread detects it?



Checking Safety Properties (Deadlock Example)

Can we report a deadlock as soon as a thread detects it?

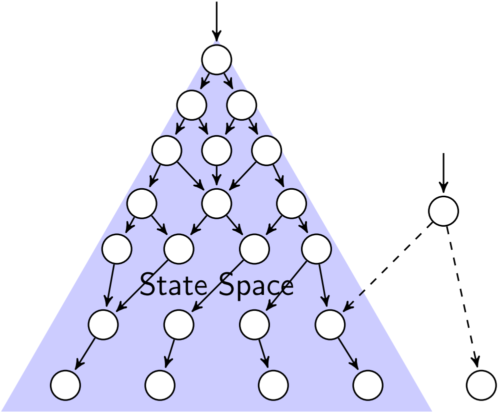
- YES if the thread started from the real initial state



Checking Safety Properties (Deadlock Example)

Can we report a deadlock as soon as a thread detects it?

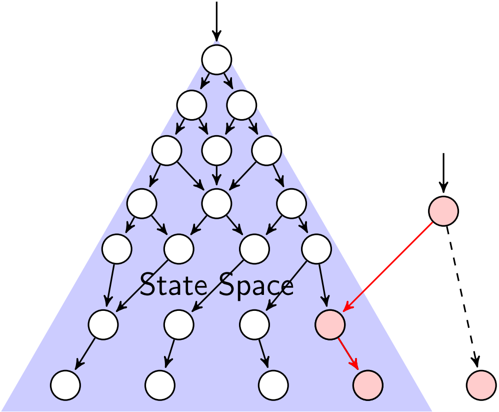
- YES if the thread started from the real initial state



Checking Safety Properties (Deadlock Example)

Can we report a deadlock as soon as a thread detects it?

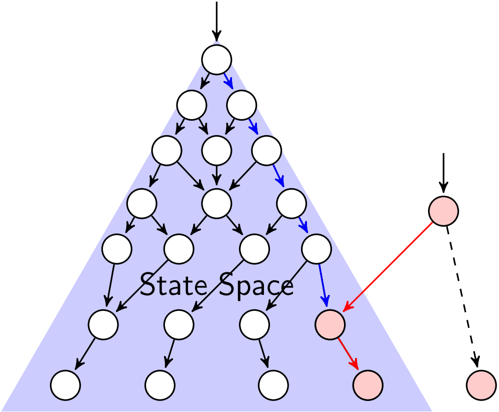
- YES if the thread started from the real initial state
- Otherwise information must be propagated at backtrack



Checking Safety Properties (Deadlock Example)

Can we report a deadlock as soon as a thread detects it?

- YES if the thread started from the real initial state
- Otherwise information must be propagated at backtrack



Benchmark

- Implemented into a fork of Spot
- 38 models from the BEEM Benchmark
  - many kind of topologies represented
  - no longer than 40 minutes for a single-threaded DFS
- Up to 12 threads
- System generated On-The-Fly using Divine2.4 patched by the LTSmin team
- Xeon(R) @ 2.00GHz with 250GB of RAM

Impact on a Swarmed-DFS

	Threshold							
	0.7		0.8		0.9		0.999	
	nb	*Time*	*nb*	*Time*	*nb*	*Time*	*nb*	*Time*
gt	35	1 041	35	970	35	1 000	37	900
eq	35	3 217	35	965	35	934	38	907
lt	35	972	35	951	35	928	38	904
ls	35	970	35	983	35	935	38	894
	No threshold							
rnd	(trivial comparator)						32	5 079
DFS	(state-of-the-art with 1 threads)						38	2 960
DFS	(state-of-the-art with 4 threads)						38	1 186
DFS	(state-of-the-art with 8 threads)						38	981
DFS	(state-of-the-art with 12 threads)						38	978

12 threads, nb_generation=3, init=1000, pop_size=50, Time in seconds.

Benchmark for Safety Properties

	DFS		GPDFS			
	(state-of-the-art)		**lessthan**		**lessstrict**	
	Time	States	Time	States	Time	States
Deadlocks	2	7.01e^6	3	5.87e^6	3	5.47e^6
No deadlocks	516	5.79e^8	462	6.73e^8	468	6.82e^8

84% of the generated states belongs to the state space

Related work

- Godefroid and Khurshid [2002]: use genetic programming as an heuristic to help random walks

- Sivaraj and Gopalakrishnan [2003]: perform a bounded BFS to obtain a pool of initial states to maximize random walk coverage

- Verification and Genetic programming:
    - Katz and Peled [2013]: Synthesis of Parametric Programs
    - Ammann et al. [1998]: the automatic generation of mutants that can be seen as particular "tests cases"

Conclusion & Perspectives

- 84% of generated states are valid
- 10 % Faster than State-Of-The-Art (12 threads)
- Improve Swarming where the topology cap the speedup
- Easily adaptable for checking liveness properties
- Combination with POR than tends to have *linear* topology
- Combination with machine learning for the generation of *better* states

Bibliography I

P. E. Ammann, P. E. Black, and W. Majurski. Using model checking to generate tests from specifications. In ICFEM'98, pp. 46–54, december 1998.

R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In PSTV'95, pp. 3–18, 1996. Chapman & Hall. URL `http://citeseer.nj.nec.com/gerth95simple.html`.

P. Godefroid and S. Khurshid. Exploring Very Large State Spaces Using Genetic Algorithms, pp. 266–280. Springer, Berlin, Heidelberg, 2002.

G. J. Holzmann, R. Joshi, and A. Groce. Swarm verification techniques. IEEE Transaction on Software Engineering, 37(6):845–857, 2011.

G. Katz and D. A. Peled. Synthesis of parametric programs using genetic programming and model checking. In INFINITY'13, pp. 70–84, 2013.

H. Sivaraj and G. Gopalakrishnan. Random walk based heuristic algorithms for distributed memory model checking. Electronic Notes in Theoretical Computer Science, 89(1):51 – 67, 2003.

A. Valmari. Stubborn sets for reduced state space generation. In ICATPN'91, vol. 618 of LNCS, pp. 491–515, 1991. Springer.