Finding Liveness Errors with ACO

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD DE MÁLAGA

Francisco Chicano and Enrique Alba

Motivation

- Nowadays software is very complex

- An error in a software system can imply the loss of lot of money …

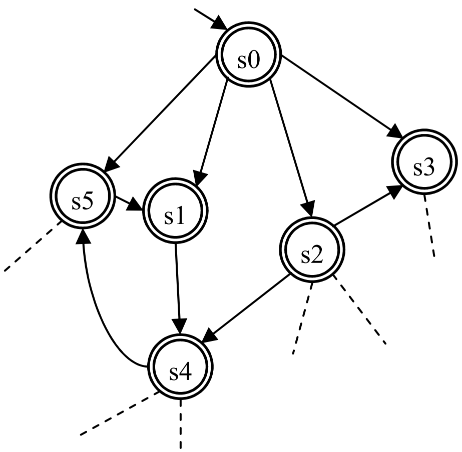
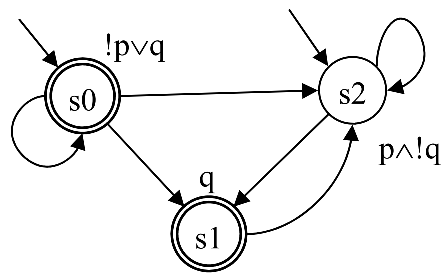
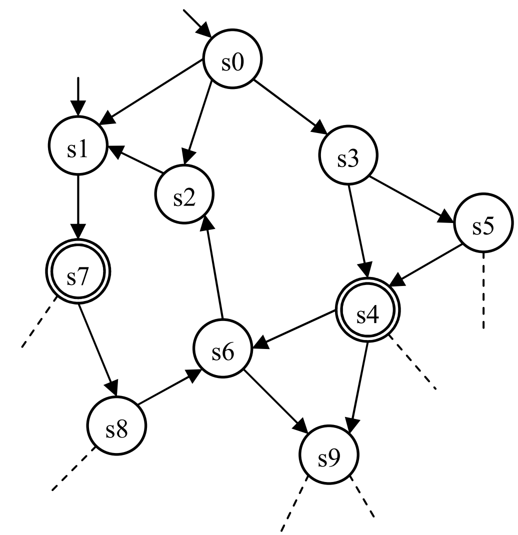
… and even human lifes

- Techniques for proving the correctness of the software are required

- Model checking → fully automatic

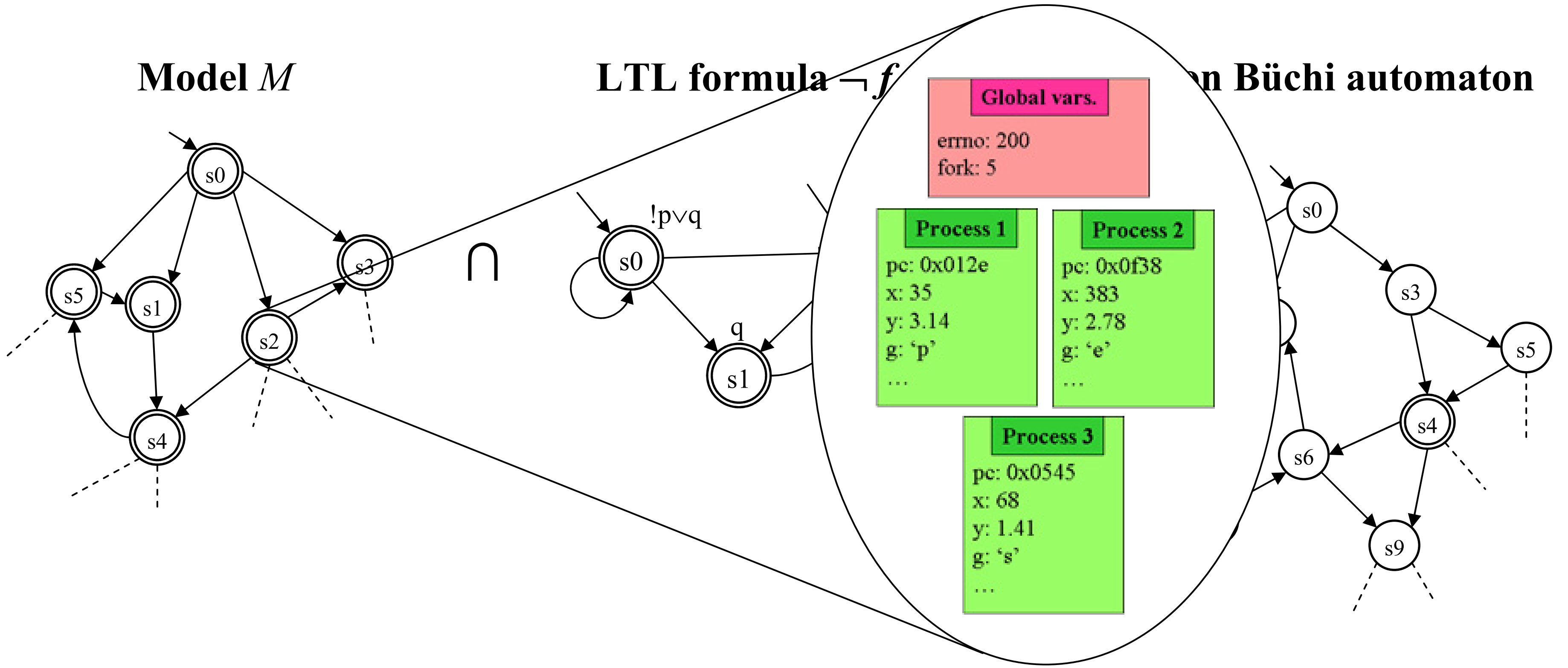
Explicit State Model Checking

- **Objective: Prove that model M satisfies the property f: $M \models f$**
- **HSF-SPIN: the property *f* is an LTL formula**

Model M		**LTL formula $\neg f$**		**Intersection Büchi automaton**
	$\cap$		=	

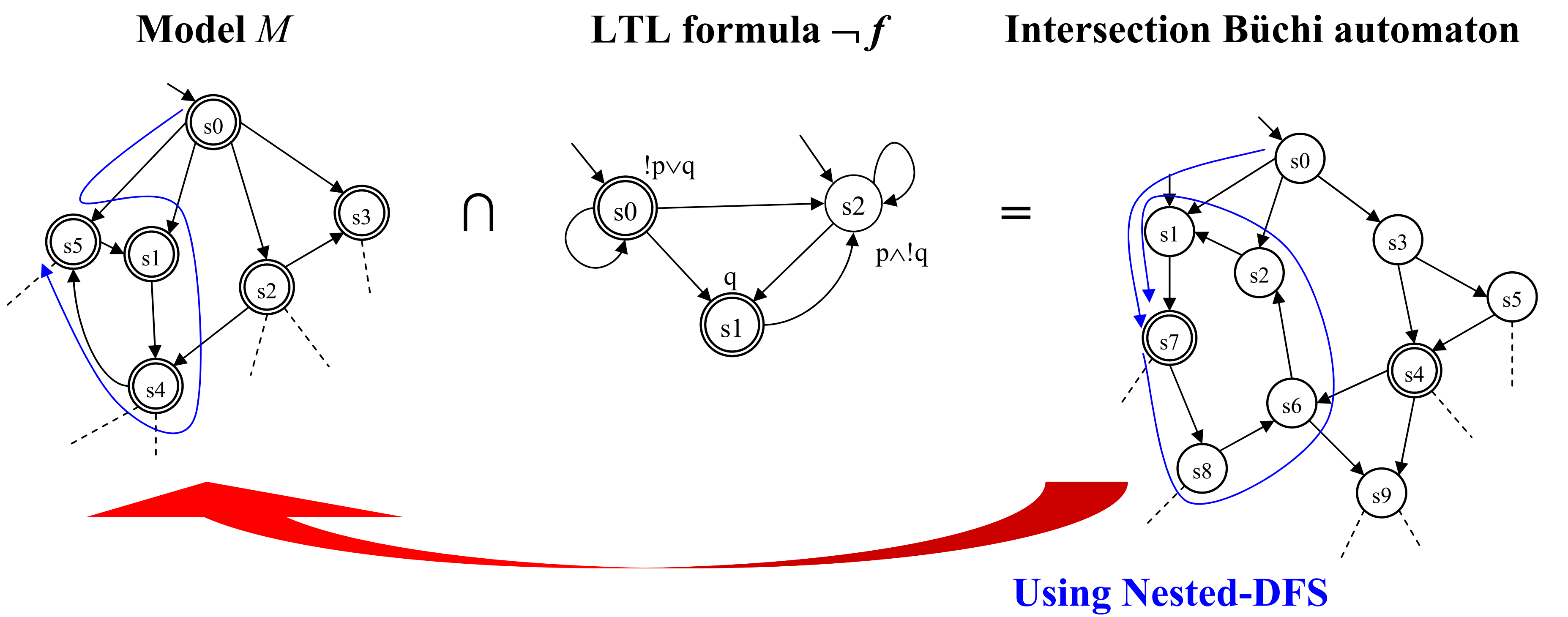
Explicit State Model Checking

- **Objective: Prove that model M satisfies the property f: $M \models f$**
- **HSF-SPIN: the property f is an LTL formula**



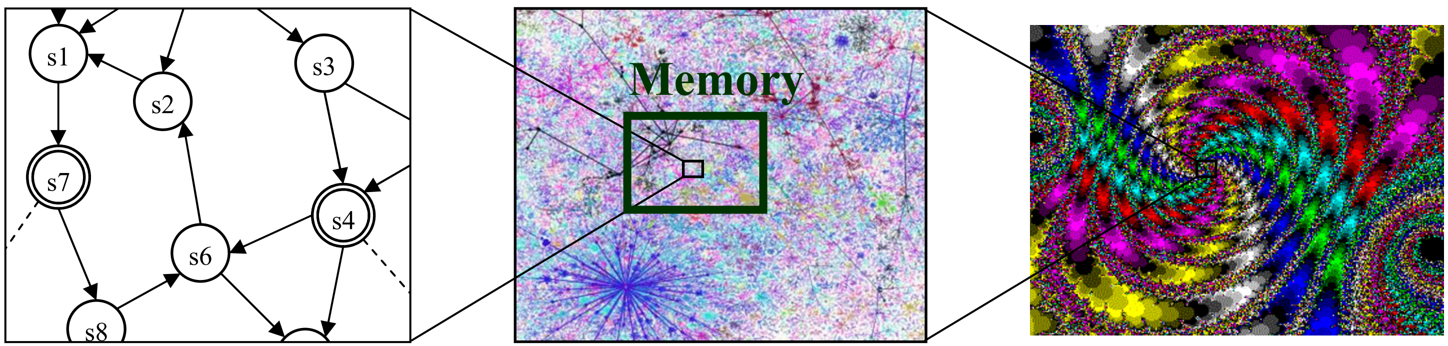
Explicit State Model Checking

- **Objective: Prove that model M satisfies the property f: $M \models f$**
- **HSF-SPIN: the property f is an LTL formula**



State Explosion Problem

- **Number of states very large even for small models**



- **Example: Dining philosophers with *n* philosophers → 3^n states**

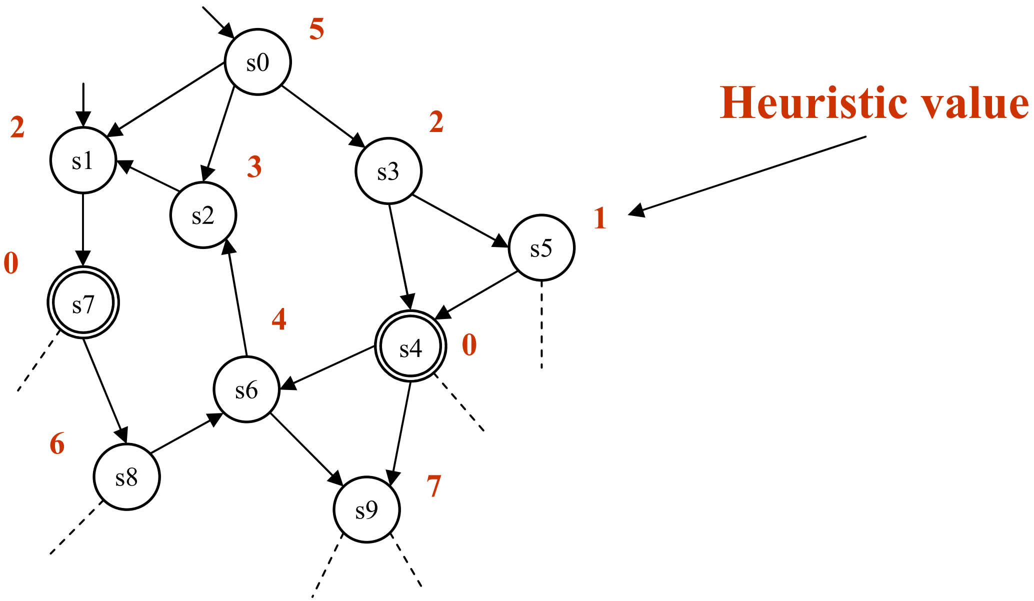
  20 philosophers → 1039 GB for storing the states

- **Solutions: collapse compression, minimized automaton representation, bitstate hashing, partial order reduction, symmetry reduction**

- **Large models cannot be verified but errors can be found**

Heuristic Model Checking

- The search for errors can be directed by using heuristic information



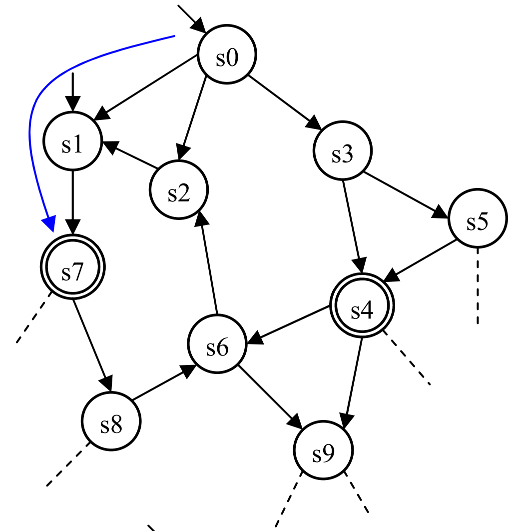
- Different kinds of heuristic functions have been proposed in the past:
  - Formula-based heuristics
  - Structural heuristics
  - Deadlock-detection heuristics
  - State-dependent heuristics

Safety and Liveness Properties

Safety property

$$\forall \sigma \in S^\omega : \sigma \nvdash \mathcal{P} \Rightarrow (\exists i \geq 0 : \forall \beta \in S^\omega : \sigma_i \beta \nvdash \mathcal{P})$$

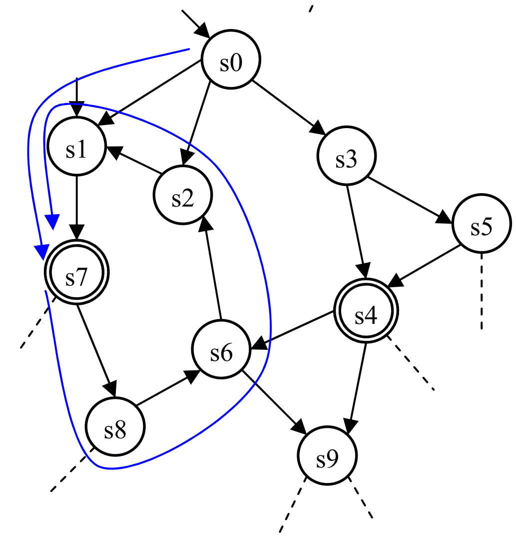
- **Counterexample ≡ path to accepting state**
- **Graph exploration algorithms can be used: DFS and BFS**



Liveness property

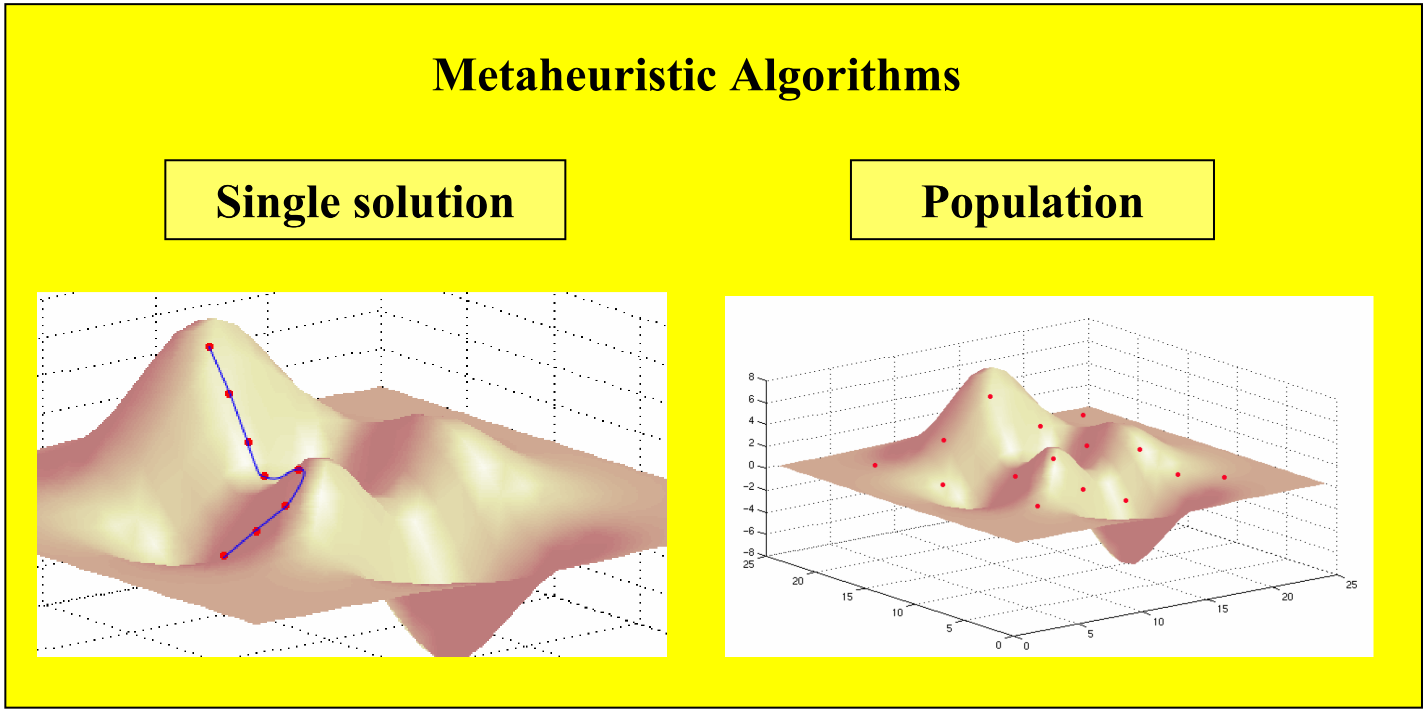
$$\forall \alpha \in S^* : \exists \beta \in S^\omega , \alpha\beta \vdash \mathcal{P}$$

- **Counterexample ≡ path to accepting cycle**
- **It is not possible to apply DFS or BFS**



Metaheuristic Algorithms

- **Designed to solve optimization problems**
  - ➢ **Maximize or minimize a given function: the fitness function**
- **They can find "good" solutions with a "reasonable" amount of resources**



Metaheuristics Classification

Single solution

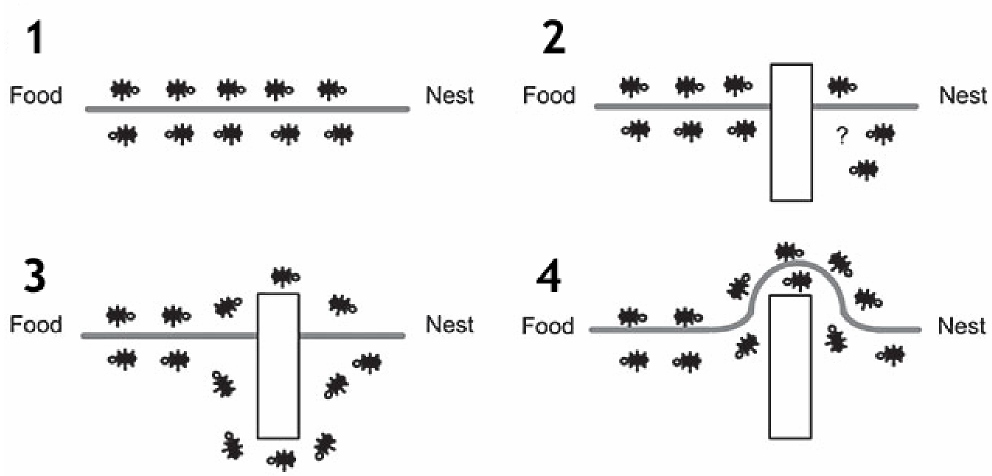
- Greedy Randomized Adaptive Search Procedure
- Iterated Local Search
- Variable Neighborhood Search
- Tabu Search
- Simulated Annealing
- Iterative Improvement
- Guided Local Search

Population

- Estimation of Distribution Algorithms
- Evolutionary Computation
- Scatter Search
- Ant Colony Optimization
- Particle Swarm Optimization

ACO: Introduction

- **Ant Colony Optimization (ACO) metaheuristic is inspired by the foraging behaviour of real ants**

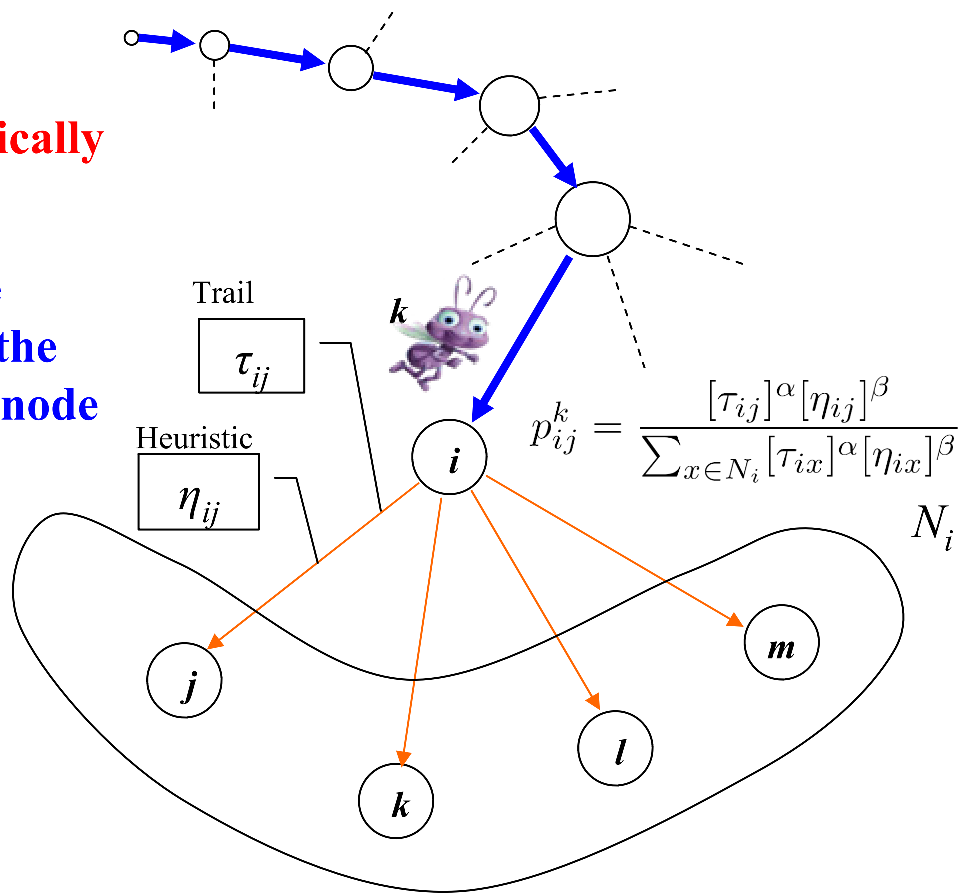


- **ACO Pseudo-code**

```
procedure ACOMetaheuristic
    ScheduleActivities
        ConstructAntsSolutions
        UpdatePheromones
        DaemonActions // optional
    end ScheduleActivities
end procedure
```

ACO: Construction Phase

- **The ant selects its next node stochastically**
- **The probability of selecting one node depends on the pheromone trail and the heuristic value (optional) of the edge/node**
- **The ant stops when a complete solution is built**



ACO: Pheromone Update

- **Pheromone update**
  - **During the construction phase**

$$\tau_{ij} \leftarrow (1-\xi)\tau_{ij} \quad \text{with} \quad 0 \leq \xi \leq 1$$

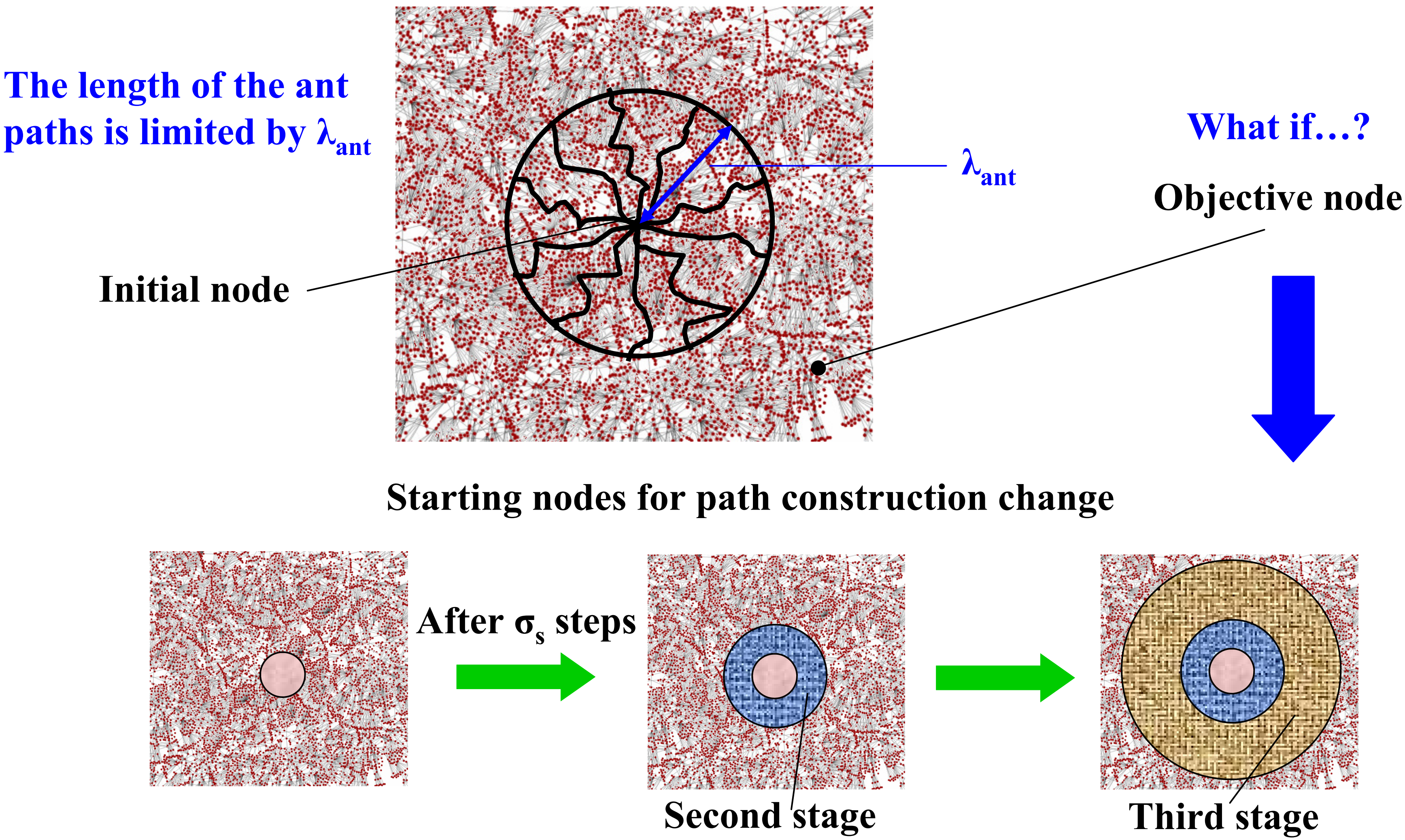
  - **After the construction phase**

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij}^{bs} \quad \text{with} \quad 0 \leq \rho \leq 1$$

- **Trail limits (particular of *MM*AS)**
  - **Pheromones are kept in the interval [τ_{min}, τ_{max}]**

$$\tau_{max} = \frac{Q}{\rho} \qquad \tau_{min} = \frac{\tau_{max}}{a}$$

ACOhg: Huge Graphs Exploration

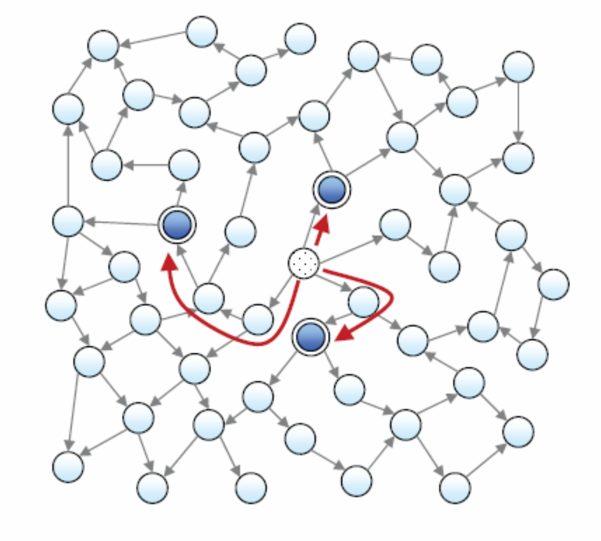


ACOhg-live

- **The search is an alternation of two phases**
  - **First phase: search for accepting states**
  - **Second phase: search for cycles from the accepting states**

ACOhg-live Pseudocode

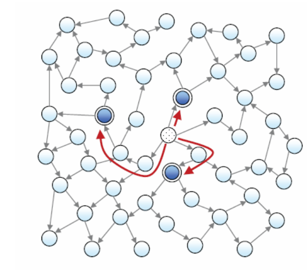
```
repeat
  accpt = acohg1.findAcceptingStates(); {First phase}
  for node in accpt do
    acohg2.findCycle(node); {Second phase}
    if acohg2.cycleFound() then
      return acohg2.acceptingPath();
    end if
  end for
  acohg1.insertTabu(accpt);
until empty(accpt)
return null;
```



First phase

ACOhg-live

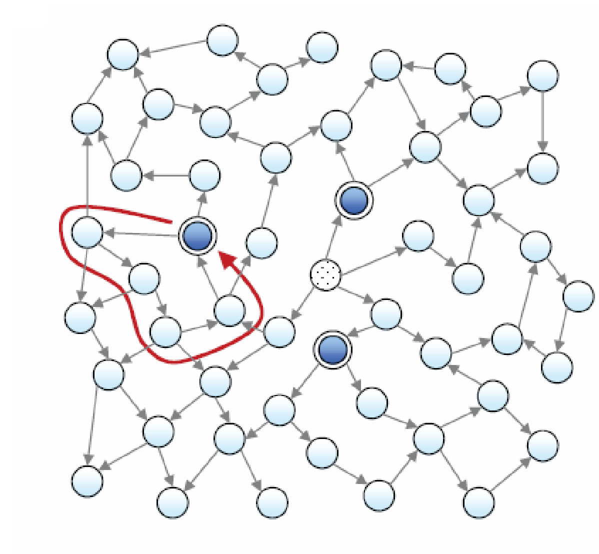
- **The search is an alternation of two phases**



  - **First phase: search for accepting states**
  - **Second phase: search for cycles from the accepting states**

ACOhg-live Pseudocode

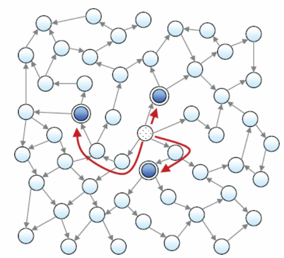
```
repeat
  accpt = acohg1.findAcceptingStates(); {First phase}
  for node in accpt do
    acohg2.findCycle(node); {Second phase}
    if acohg2.cycleFound() then
      return acohg2.acceptingPath();
    end if
  end for
  acohg1.insertTabu(accpt);
until empty(accpt)
return null;
```

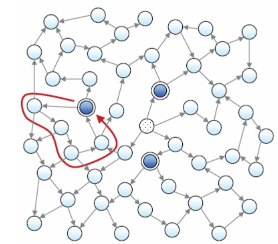


Second phase

ACOhg-live

- **The search is an alternation of two phases**





  - **First phase: search for accepting states**
  - **Second phase: search for cycles from the accepting states**

ACOhg-live Pseudocode

```
repeat
  accpt = acohg1.findAcceptingStates(); {First phase}
  for node in accpt do
    acohg2.findCycle(node); {Second phase}
    if acohg2.cycleFound() then
      return acohg2.acceptingPath();
    end if
  end for
  acohg1.insertTabu(accpt);
until empty(accpt)
return null;
```

Promela Models

- We selected 7 Promela models for the experiments

Model	LoC	Scalable	Processes	LTL formula (liveness)
alter	64	no	2	□(p → ◊q) ^ □(r → ◊s)
giopij	740	yes	i+3(j+1)	□(p → ◊q)
phij	57	yes	j+1	□(p → ◊q)

- Parameters for ACOhg-live

Parameter	*msteps*	*colsize*	λ_{ant}	σ_s	ξ	a	ρ	α	β
1st phase	100	10	20	4	0.7	5	0.2	1.0	2.0
2nd phase		20	4		0.5				

- ACOhg-live implemented in HSF-SPIN
- 100 independent executions

Promela Models

- **We selected 7 Promela models for the experiments**

Model	LoC		Processes	LTL formula (liveness)
`alter`	[illegible]	[illegible]	2	□(p → ◊q) ^ □(r → ◊s)
`giopij`	740	yes	i+3(j+1)	□(p → ◊q)
`phij`	57	[illegible]	j+1	□(p → ◊q)

giopij: i=2,6,10 j=2

phij: j=8,14,20

- **Parameters for ACOhg-live**

Parameter	*msteps*	*colsize*	λ_{ant}	σ_s	ξ	a	ρ	α	β
1st phase	100	10	20	4	0.7	5	0.2	1.0	2.0
2nd phase		20	4		0.5				

- **ACOhg-live implemented in HSF-SPIN**
- **100 independent executions**

Results I: Comparison of Heuristic Information

- **Comparison of H_{ham} and H_{fsm}**

Models	Measure	H_{ham}		H_{fsm}		Test
alter	Hit rate	100/100		100/100		-
	Length	28.54	9.44	30.68	10.72	-
	Mem. (KB)	1925.00	0.00	1925.00	0.00	-
	Time (ms)	88.90	15.03	90.00	13.86	-
giop2	Hit rate	100/100		100/100		-
	Length	43.09	5.33	43.76	5.82	-
	Mem. (KB)	2834.48	369.42	2953.76	327.48	+
	Time (ms)	817.50	560.73	747.50	408.09	-
giop6	Hit rate	100/100		100/100		-
	Length	58.41	7.16	58.77	7.21	-
	Mem. (KB)	5418.32	1161.17	5588.04	631.36	-
	Time (ms)	16049.10	15256.93	8733.50	3304.90	-
giop10	Hit rate	60/100		86/100		+
	Length	61.10	6.42	62.85	7.03	-
	Mem. (KB)	9669.80	1597.14	9316.67	700.44	+
	Time (ms)	87236.00	69218.19	43059.07	21417.74	+
phi8	Hit rate	100/100		100/100		-
	Length	52.38	9.26	51.36	6.95	-
	Mem. (KB)	2097.52	21.74	2014.32	18.87	+
	Time (ms)	2271.40	573.56	2126.10	479.64	-
phi14	Hit rate	99/100		99/100		-
	Length	74.68	8.66	76.05	9.35	-
	Mem. (KB)	2593.37	179.03	2496.07	41.81	+
	Time (ms)	9369.90	3706.83	8070.30	1530.12	+
phi20	Hit rate	99/100		98/100		-
	Length	95.28	9.97	97.39	10.14	-
	Mem. (KB)	3324.26	104.27	3244.67	91.33	+
	Time (ms)	21323.54	10600.89	18064.90	5538.30	+

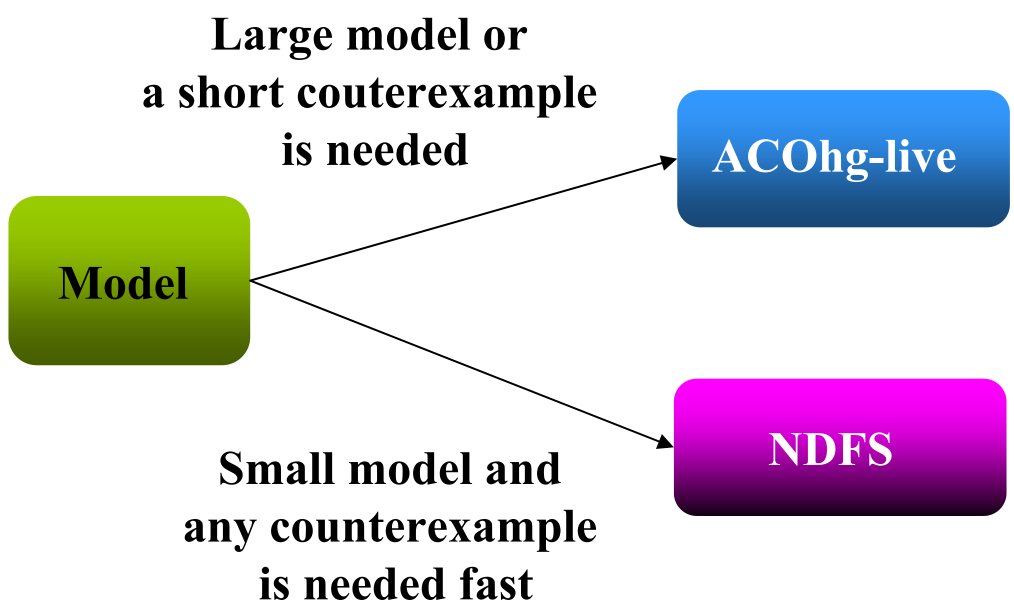
Results II: Comparison of ACOhg-live and NDFS

- **Comparison of ACOhg-live and NDFS**

Models	Measure	ACOhg-live	Nested-DFS	Test
alter	Hit rate	100/100	1/1	-
	Length	30.68	64.00	+
	Mem. (KB)	1925.00	1873.00	+
	Time (ms)	90.00	0.00	+
giop2	Hit rate	100/100	1/1	-
	Length	43.76	298.00	+
	Mem. (KB)	2953.76	7865.00	+
	Time (ms)	747.50	240.00	+
giop6	Hit rate	100/100	0/1	+
	Length	58.77	●	●
	Mem. (KB)	5588.04	●	●
	Time (ms)	8733.50	●	●
giop10	Hit rate	86/100	0/1	+
	Length	62.85	●	●
	Mem. (KB)	9316.67	●	●
	Time (ms)	43059.07	●	●
phi8	Hit rate	100/100	1/1	-
	Length	51.36	3405.00	+
	Mem. (KB)	2014.32	4005.00	+
	Time (ms)	2126.10	40.00	+
phi14	Hit rate	99/100	1/1	-
	Length	76.05	10001.00	+
	Mem. (KB)	2496.07	59392.00	+
	Time (ms)	8070.30	2300.00	+
phi20	Hit rate	98/100	1/1	-
	Length	97.39	10001.00	+
	Mem. (KB)	3244.67	392192.00	+
	Time (ms)	18064.90	17460.00	-

How to use ACOhg-live

- **ACOhg-live should be used in the first/middle stages of the software development, when software errors are expected**
- **ACOhg-live can also be used in other phases of the software development for testing concurrent software**



Conclusions & Future Work

Conclusions

- ACOhg-live is the first algorithm based on metaheuristics (to the best of our knowledge) applied to the search for liveness errors in concurrent models
- The heuristic function based on finite state machines is a better guide in the second phase of ACOhg-live
- ACOhg-live is able to outperform Nested-DFS in efficacy and efficiency in the search for liveness errors

Future Work

- Use of Strongly Connected Components of the never claim graph for improving the search (in progress)
- Analysis of parameterization for reducing the parameters
- Include ACOhg-live into JavaPathFinder for finding liveness errors in Java programs

Thanks for your attention !!!