A Configurable CEGAR Framework with Interpolation-Based Refinements

Ákos Hajdu[1,2], Tamás Tóth[2], András Vörös[1,2], István Majzik[2]

[1]MTA-BME Lendület Cyber-Physical Systems Research Group,
Budapest, Hungary

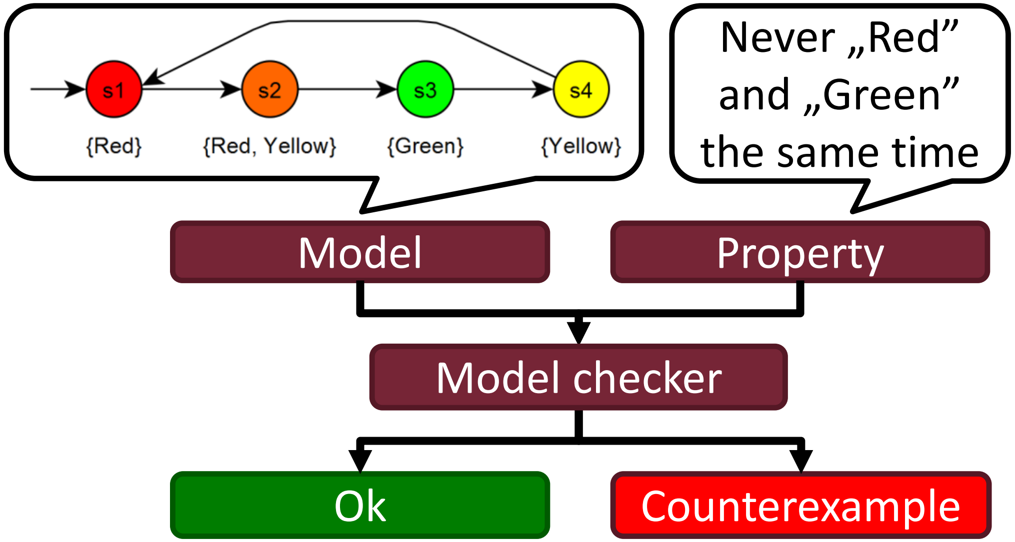
[2]Fault Tolerant Systems Research Group
Department of Measurement and Information Systems,
Budapest University of Technology and Economics

FORTE 2016, Heraklion, Greece, 08.06.2016.

INTRODUCTION

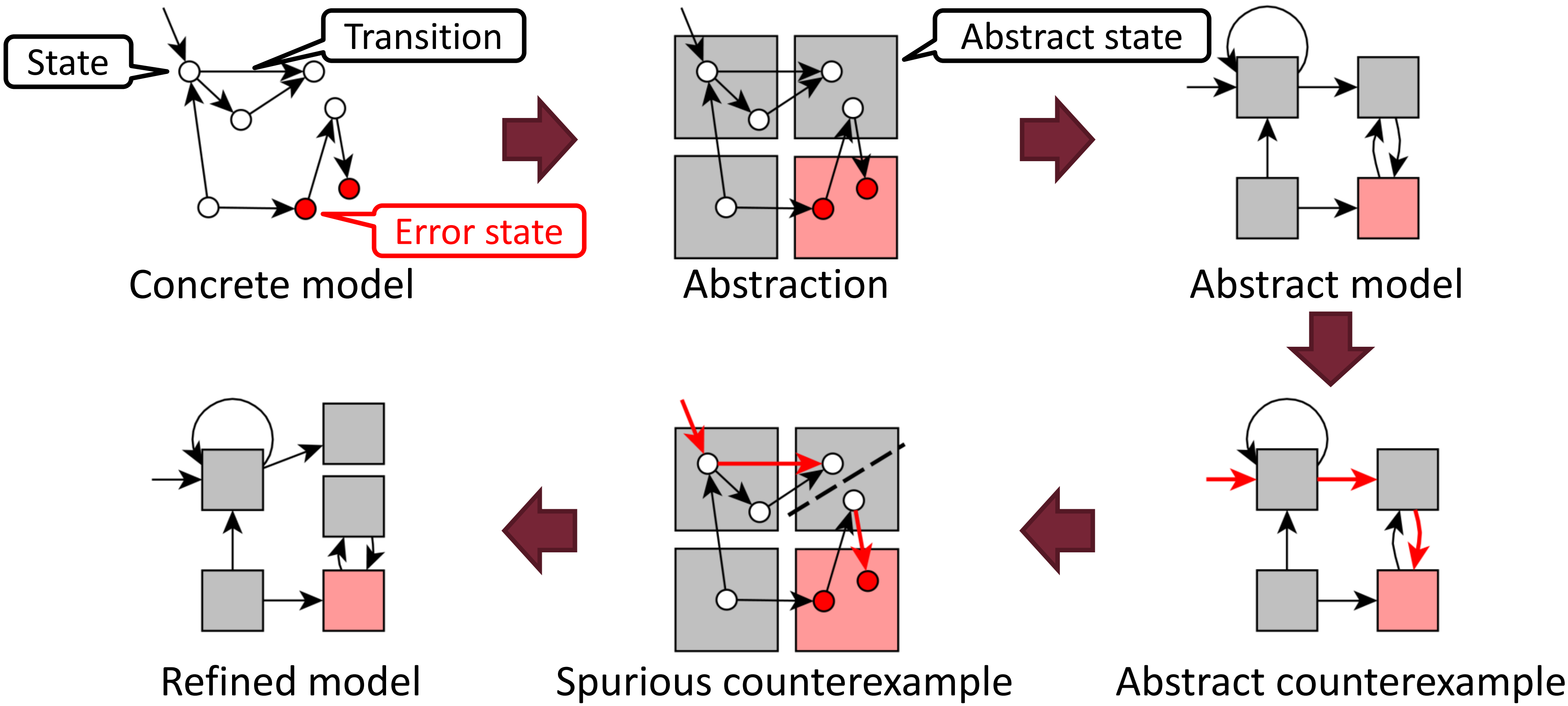
Introduction – Formal methods

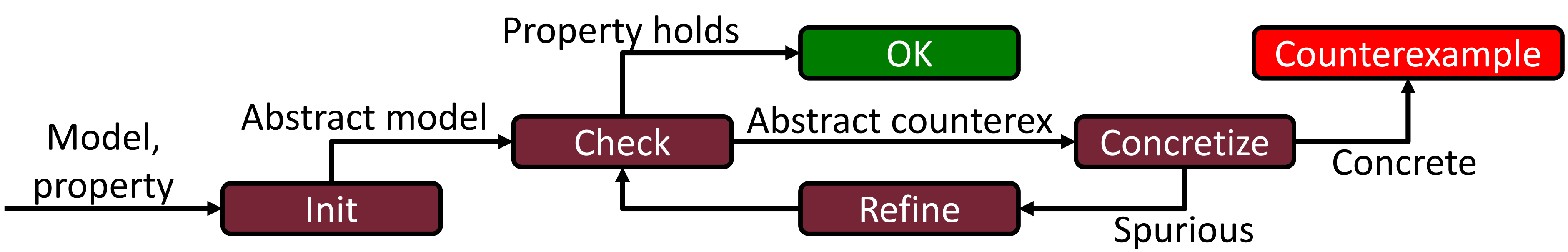
- Proving correctness
- Model checking
  - State space explosion



- Abstraction-based methods
  - Over-approximation
  - Problem: proper precision (coarse ↔ fine)
  - **C**ount**e**rexample-**G**uided **A**bstraction **R**efinement [Clarke et al.'03]
    - Start with a coarse abstraction
    - Refine until sufficient precision is reached

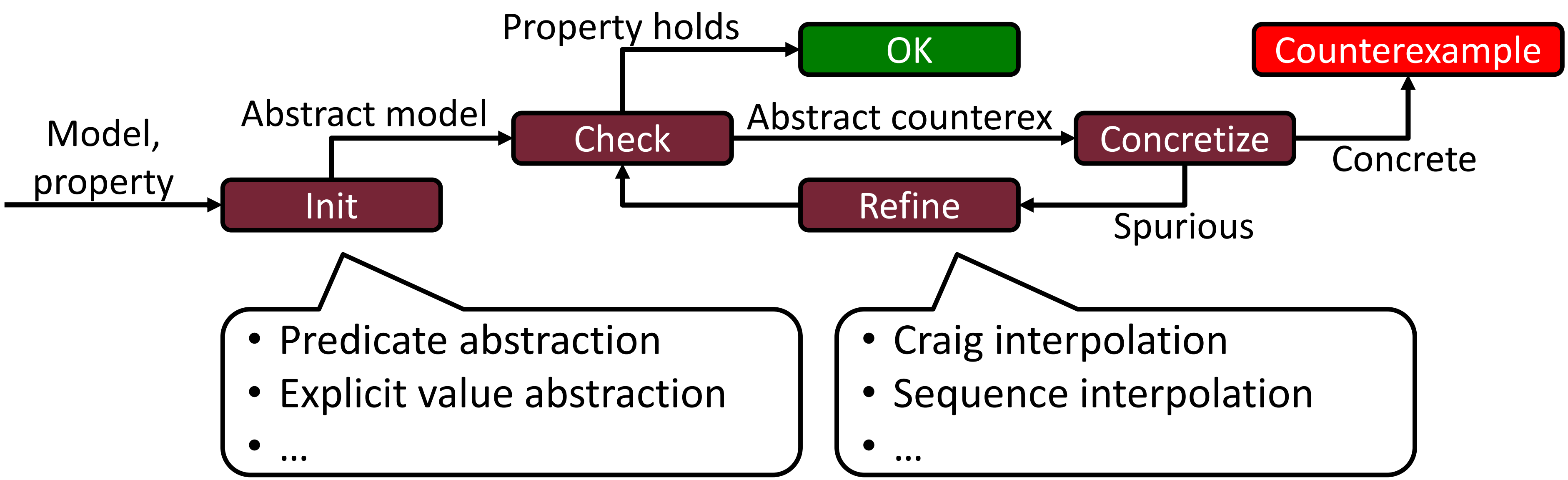
Introduction – CEGAR





Motivation

- Generic framework with interchangeable parts
  - Different abstraction methods
    - Based on symbolic representation of abstract states
  - Different refinement strategies
    - Based on splitting abstract states



A CONFIGURABLE CEGAR FRAMEWORK

Formal model and property

- Symbolic Transition System (STS)

```
var loc : integer
var   x : integer

invariant 0 <= loc and loc <= 3

initial loc = 0

transition
      (loc = 0             and loc' = 1 and x' = 0) or
      (loc = 1 and x  < 5 and loc' = 2 and x' = x) or
      (loc = 1 and x >= 5 and loc' = 3 and x' = x) or
      (loc = 2             and loc' = 1 and x' = x + 1)

models x <= 5
```

Variables

Invariant formula

Initial formula

Transition formula

Safety property to be checked:
Is *x <= 5* for all reachable states?

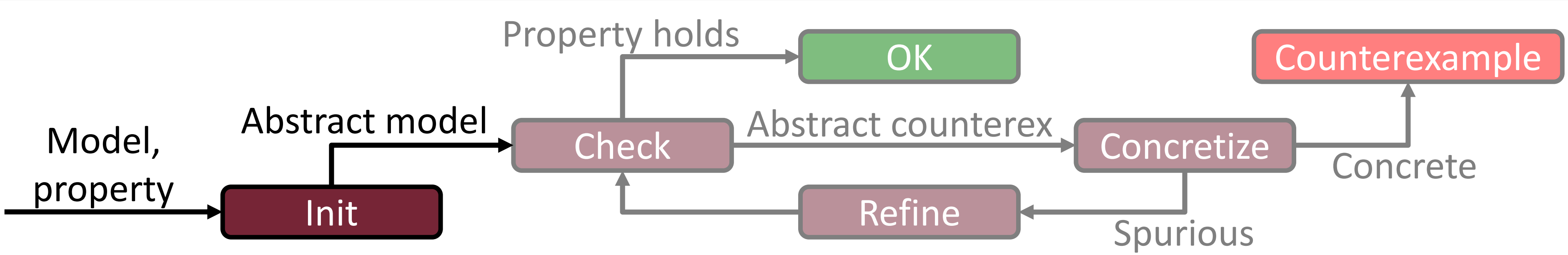
```
   int x

0: x = 0
1: while (x < 5) {
2:     x = x + 1
3: }
```

Initial abstraction

1. **Predicate abstraction** [Graf & Saidi'97]
   - Track predicates instead of concrete values
   - $|P|$ predicates → $2^{|P|}$ potential abstract states
   - Label of a state: predicates, e.g. $\neg(x > y) \land (y = 3)$

Variables:
x, y; $D_x = D_y = \{1, 2, 3\}$
Predicates:
$(x > y)$, $(y = 3)$

	$(x > y)$	$\neg(x > y)$
$(y = 3)$		(x=1, y=3) (x=2, y=3) (x=3, y=3)
$\neg(y = 3)$	(x=2, y=1) (x=3, y=1) (x=3, y=2)	(x=1, y=1) (x=1, y=2) (x=2, y=2)



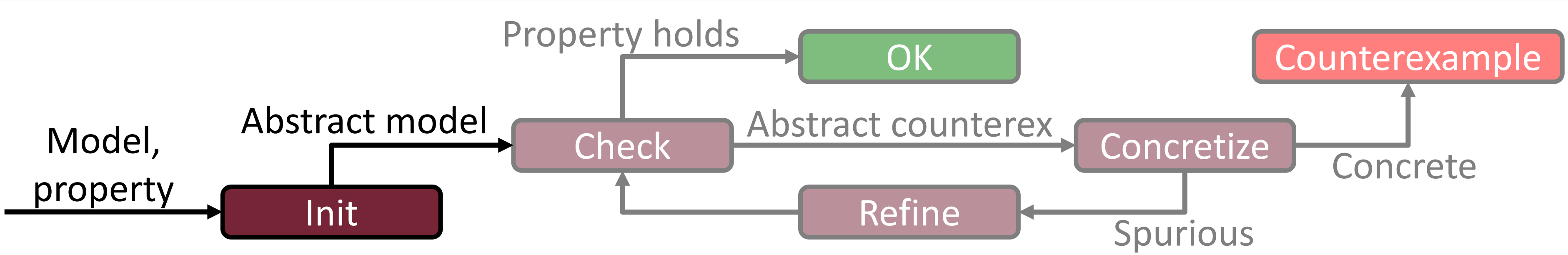
Initial abstraction

2. Explicit value abstraction [Clarke et al.'04]

- Partition variables: visible / invisible
- Track values for visible variables only
- Label of a state: assignment, e.g. $(x = 1) \wedge (y = 2)$

Variables: x, y, z
$D_x = \{0, 1\}$, $D_y = \{0, 1, 2\}$, $D_z = \{0, 1\}$
Visible = {x, y}

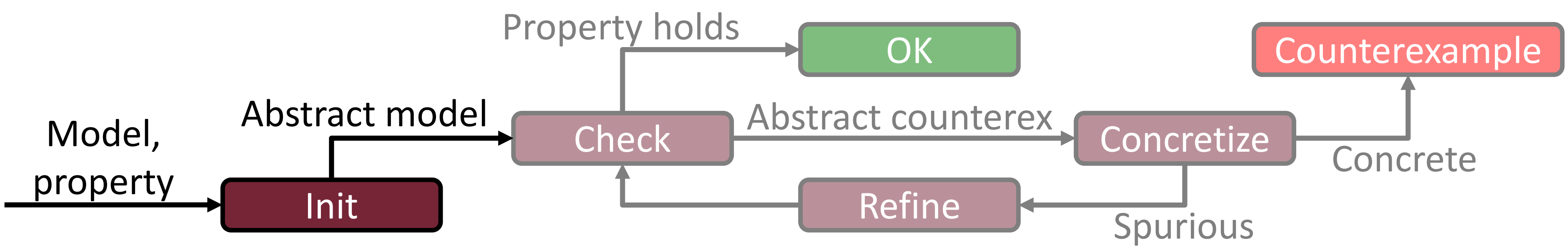
	x=0	x=1
y=0	(x=0, y=0, **z=0**) (x=0, y=0, **z=1**)	(x=1, y=0, **z=0**) (x=1, y=0, **z=1**)
y=1	(x=0, y=1, **z=0**) (x=0, y=1, **z=1**)	(x=1, y=1, **z=0**) (x=1, y=1, **z=1**)
y=2	(x=0, y=2, **z=0**) (x=0, y=2, **z=1**)	(x=1, y=2, **z=0**) (x=1, y=2, **z=1**)



Initial abstraction

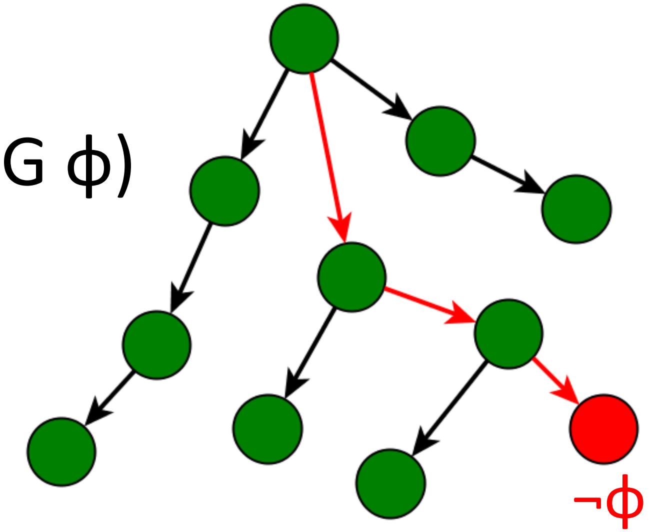
- Predicates vs. explicit values
  - Variable with large domain → predicates
  - Variable appearing in many predicates → explicit

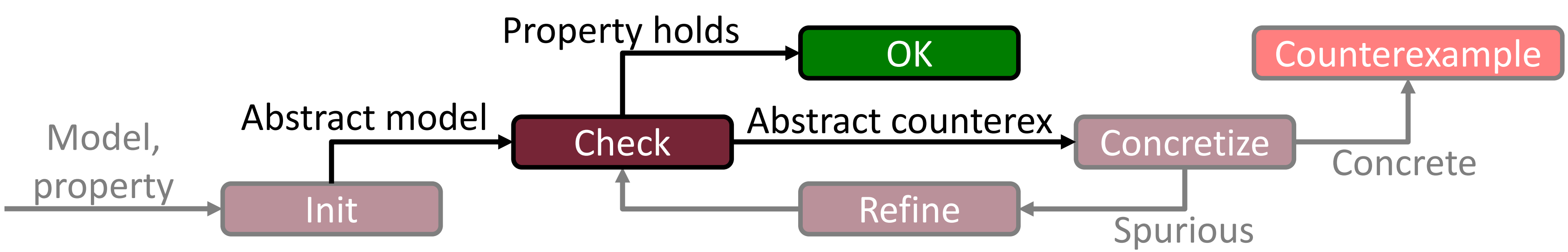
3. Combined abstraction
   - Predicates + explicit values for a set of variables
   - Explicit variables
     - User input
     - Heuristics (e.g., location variable)



Model checking

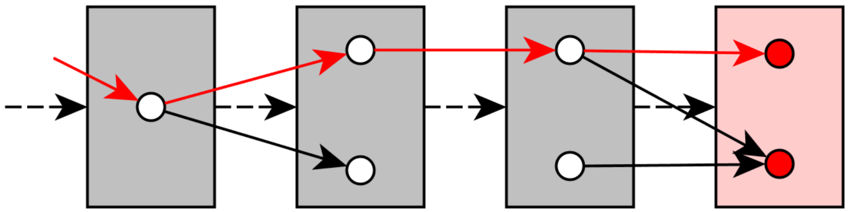
- Explicitly traverse abstract state space
- Safety properties
  - ϕ holds for each reachable state (AG ϕ)
  - Counterexample: loop-free path
- Optimizations
  - Explicit values: on-the-fly
  - Predicates: incremental



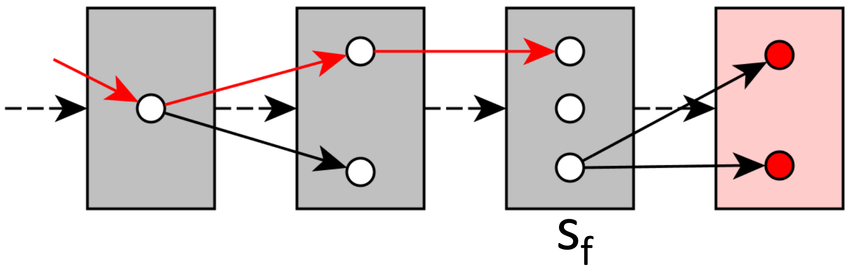


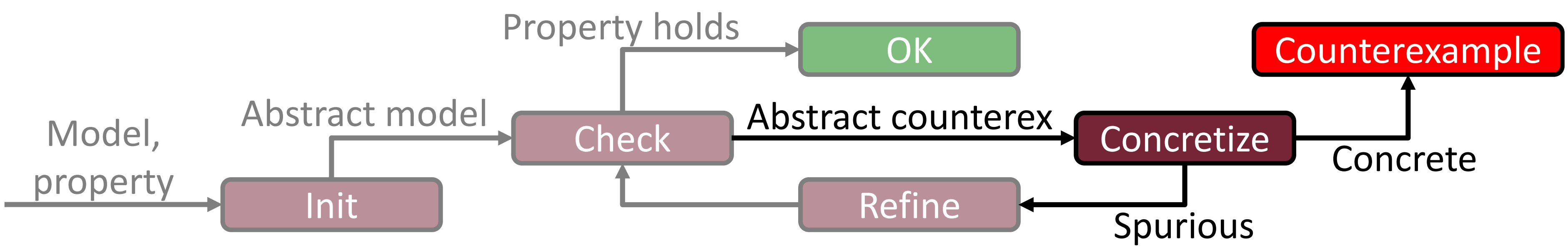
Counterexample concretization

- Traverse subset of concrete state space
  - Similar to bounded model checking
    - $Init_1 \land Label_1 \land Trans_1 \land Label_2 \land Trans_2 \land \ldots \land Trans_{n-1} \land Label_n$
  - Concrete counterexample



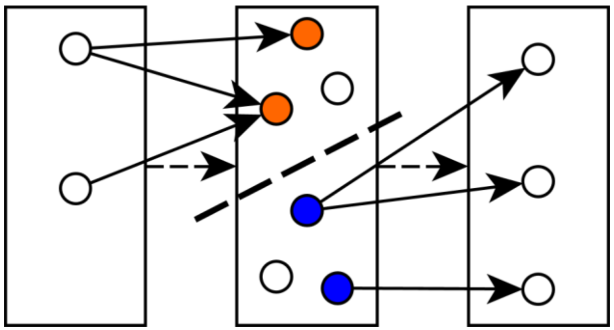
  - Spurious counterexample
    - Failure state (s_f)



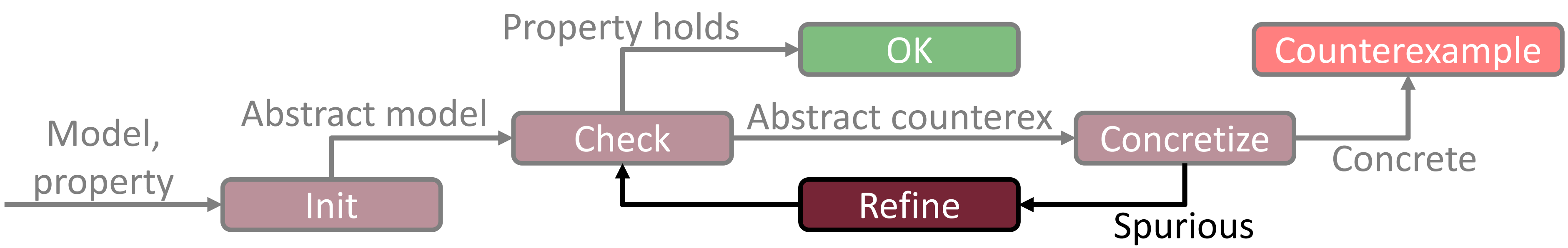


Abstraction refinement

- Classify states mapped to the failure state
  - $\color{orange}{D}$ = Dead-end: reachable
    - $Init_1 \land Label_1 \land Trans_1 \land \ldots \land Trans_{f-1} \land Label_f$
  - $\color{blue}{B}$ = Bad: transition to next state
    - $Label_f \land Trans_f \land Label_{f+1}$
  - IR = Irrelevant: others

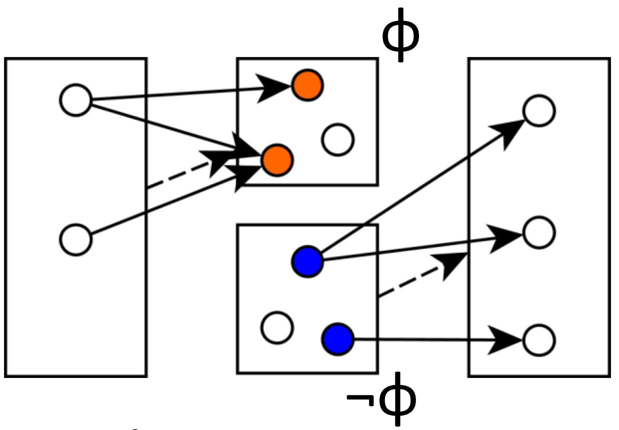


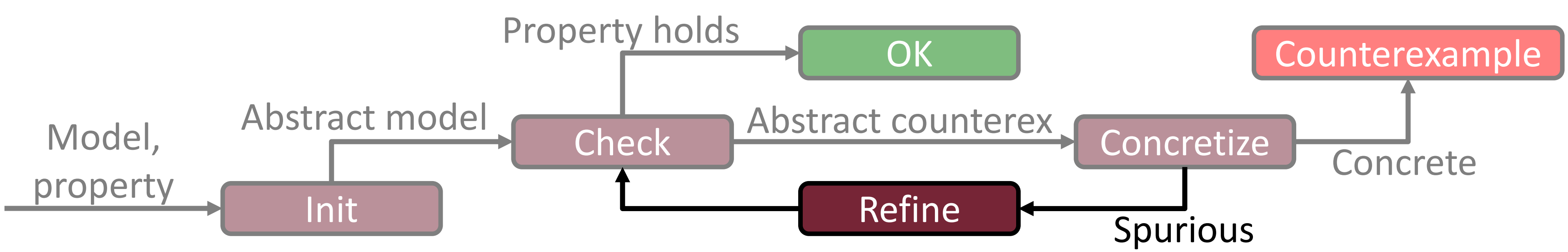
- Goal: finer abstraction mapping D and B to separate abstract states



Abstraction refinement

1. Predicate refinement
   - Characterize D and B with formulas, such that D ∧ B is unsatisfiable
   - Craig interpolation [Henzinger et al.'04]
     - A predicate ϕ exists corresponding to variables of s_f
     - Generalizing D, contradicting B
   - P ∪ {ϕ} eliminates the spurious counterexample
     - Lazy abstraction: only split s_f





Abstraction refinement

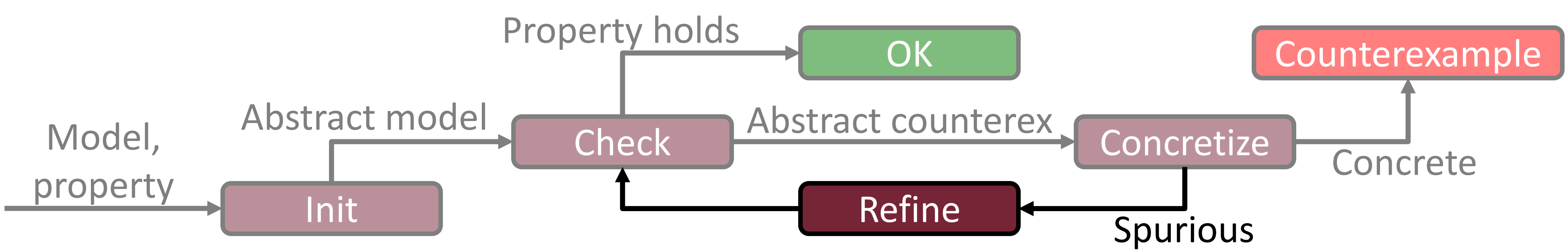
2. Explicit value refinement
   - Make some invisible variables visible
     - Variables that can distinguish D and B
   - Craig interpolation: generate ϕ as in predicate refinement
     - Visible := Visible + variables in ϕ

Visible = {x}
Invisible = {y, …}
S_f: (x=0)

Dead-end	(x=0, y=1, …) (x=0, y=1, …)
Irrelevant	(x=0, y=1, …) (x=0, y=0, …)
Bad	(x=0, y=0, …) (x=0, y=0, …)

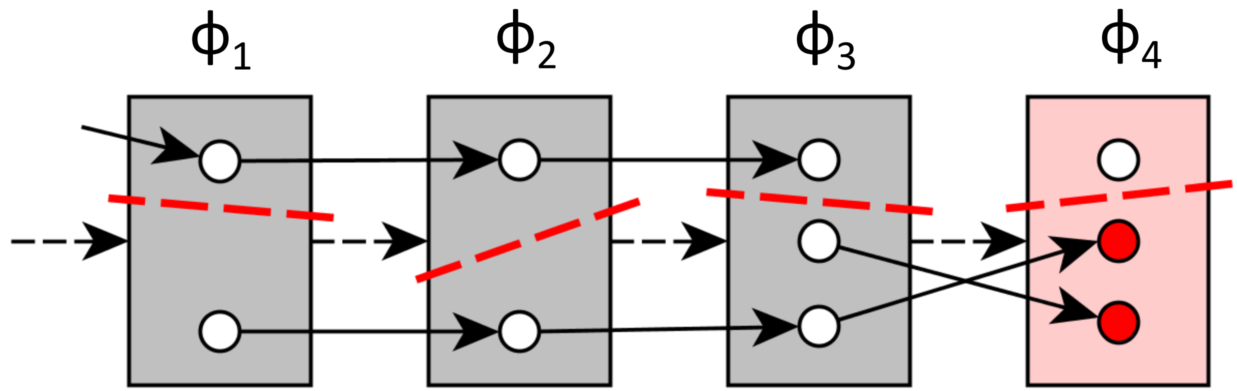
ϕ: (y < 1)

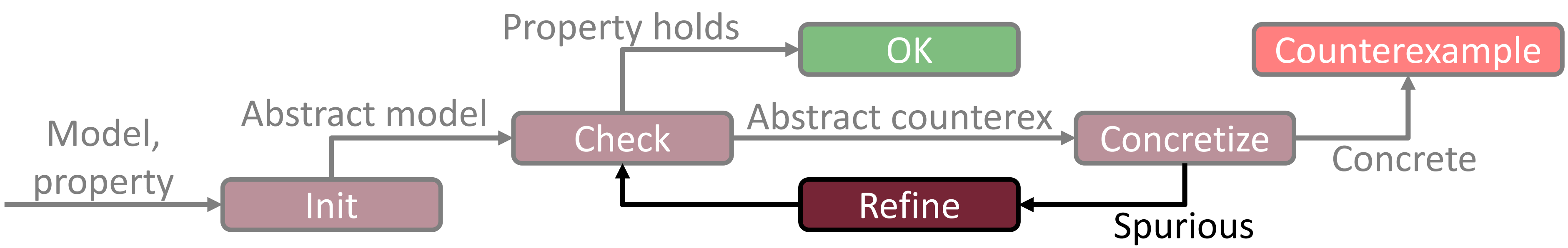
Visible = {x, y}
Invisible = {…}



Abstraction refinement

- Generalization: sequence interpolation [McMillan'06]
  - Sequence of interpolants, one for each state (of the counterex.)
  - Predicate refinement: split a sequence of states
  - Explicit value refinement: extract variables from sequence





EVALUATION

Evaluation

- Java implementation
  - Z3 solver
- Configurations

	Abstraction	Refinement
PC	**P**redicate	**C**raig interpolation
PS	**P**redicate	**S**equence interpolation
CC	**C**ombined	**C**raig interpolation
CS	**C**ombined	**S**equence interpolation
EC	**E**xplicit value	**C**raig interpolation
ES	**E**xplicit value	**S**equence interpolation

- $+/-$ property satisfied or violated

Evaluation

- Industrial PLC models *(runtime, s)*
  - L: locations, V: variables
  - CC, CS: location as explicit variable

Best performance for most models

No preprocessing reductions

Largest state space

	L	V	PC	PS	CC	CS	EC	ES
–PLC01	36	66	**22.5**	50.2	42.0	48.5	36.4	211.8
–PLC02	36	66	**22.7**	49.4	41.0	47.3	32.2	428.5
+PLC03	17	29	479.2	99.2	28.2	51.8	**5.2**	9.9
–PLC04	17	29	40.2	14.4	17.6	6.1	**3.3**	3.8
+PLC04	17	29	44.0	406.7	34.3	36.1	**7.6**	38.0
+PLC05	17	29	42.2	21.4	17.4	6.3	**3.5**	4.7
+PLC06	43	82	1512.8		333.1	**227.5**	1254.5	
+PLC07	43	82	190.8	462.2	164.8	164.8	78.1	**50.9**
–PLC08	43	82	86.1		46.7	**46.2**	65.1	123.0
+PLC09	14	23	87.4	94.6	61.3	35.7	**11.8**	14.5

Combined outperforms pure predicate

Combined outperforms pure predicate

Evaluation

- Fischer's protocol *(runtime, s)*
  - Mutual exclusion algorithm
  - #: number of participants
  - Clock variables → infinite state space
  - CC, CS: lock as explicit variable

#	PC	PS	CC	CS
+2	1.2	3.0	0.8	1.2
−2	0.6	1.1	0.8	1.2
+3	12.1	68.2	10.3	45.8
−3	1.4	1.5	1.7	2.9

Property holds

Property violated

Craig itp. is more efficient

Evaluation

- Hardware models *(runtime, s)*
  - Hardware Model Checking Competition
  - I: inputs, L: latches, A: and-gates

	I	L	A	PC	PS	EC	ES
+mutexp0	11	20	159	**10.3**	24.5	14.3	22.7
+mutexp0neg	11	20	159	6.1	**3.7**	8.8	6.7
−nusmv.syncarb52.B	5	10	52	1.3	3.1	0.7	**0.2**
−nusmv.syncarb102.B	10	20	157	31.6	117.9	239.8	**1.6**
−pdtpmsarbiter	3	46	209	**0.5**	4.6	5.3	7.8
+ringp0	15	25	145	16.4	25.6	16.1	**14.5**
+ringp0neg	15	25	145	**7.8**	35.7	187.5	108.2
+srg5ptimonegnv	30	47	304	**0.3**	0.5	1.7	1.3

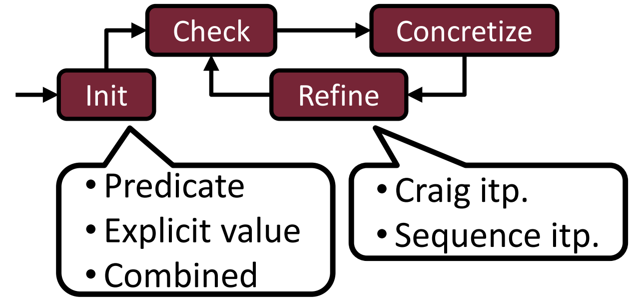
Predicate abs. is more efficient with Craig itp.

Expl. val. abs. is more efficient with sequence itp.

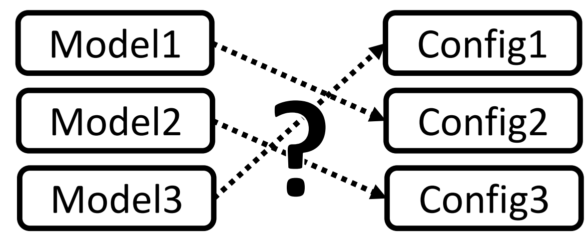
CONCLUSIONS

Conclusions

- **Results**
  - Configurable CEGAR framework
    - Different abstraction methods
    - Different refinement strategies
    - Initial abstraction: predicates with explicit values
  - Evaluation
    - Behavior of several configurations on different models
- **Future work**
  - Further configurations
  - Heuristics for configuration selection



	L	V	PC	PS	CC	CS	EC	ES
−PLC01	36	66	**22.5**	50.2	42.0	48.5	36.4	211.8
−PLC02	36	66	**22.7**	49.4	41.0	47.3	32.2	428.5
+PLC03	17	29	479.2	99.2	28.2	51.8	**5.2**	9.9
−PLC04	17	29	40.2	14.4	17.6	6.1	**3.3**	3.8
+PLC04	17	29	44.0	406.7	34.3	36.1	**7.6**	38.0
+PLC05	17	29	42.2	21.4	17.4	6.3	**3.5**	4.7
+PLC06	43	82	1512.8		333.1	**227.5**	1254.5	
+PLC07	43	82	190.8	462.2	164.8	164.8	78.1	**50.9**
−PLC08	43	82	86.1		46.7	**46.2**	65.1	123.0
+PLC09	14	23	87.4	94.6	61.3	35.7	**11.8**	14.5



hajdua@mit.bme.hu
inf.mit.bme.hu/en/members/hajdua

References

- [Clarke et al.’03] Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: **Counterexample-guided abstraction refinement for symbolic model checking.** J. ACM 50(5), 752–794 (2003)
- [Graf & Saidi’97] Graf, S., Saidi, H.: **Construction of abstract state graphs with PVS.** In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
- [Clarke et al.’04] Clarke, E.M., Gupta, A., Strichman, O.: **SAT-based counterexample-guided abstraction refinement.** IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 23(7), 1113–1123 (2004)
- [Henzinger et al.’04] Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: **Abstractions from proofs.** In: Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 232–244. ACM (2004)
- [McMillan’06] McMillan, K.L.: **Lazy abstraction with interpolants.** In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 123–136. Springer, Heidelberg (2006)

Details of measurements

- Industrial PLC models
  - L: locations, V: variables
  - T: runtime (s), #R: refinements, #S: total abstract states
  - CC, CS: location as explicit variable

			PC			PS			CC			CS			EC			ES		
	L	V	T	#R	#S	T	#R	#S	T	#R	#S	T	#R	#S	T	#R	#S	T	#R	#S
−PLC01	36	66	**22.5**	33	100	50.2	34	191	42	20	452	48.5	1	81	36.4	7	1640	211.8	3	758
−PLC02	36	66	**22.7**	33	100	49.4	34	191	41	20	452	47.3	1	81	32.2	7	1697	428.5	5	1439
+PLC03	17	29	479.2	195	6694	99.2	23	292	28.2	34	629	51.8	6	212	**5.2**	1	339	9.9	1	369
−PLC04	17	29	40.2	64	1076	14.4	16	82	17.6	21	353	6.1	2	47	**3.3**	1	165	3.8	1	165
+PLC04	17	29	44	65	1069	406.7	31	1198	34.3	35	650	36.1	5	192	**7.6**	2	274	38	1	209
+PLC05	17	29	42.2	63	1130	21.4	17	98	17.4	21	352	6.3	2	47	**3.5**	1	167	4.7	1	167
+PLC06	43	82	1512.8	159	4812				333.1	52	1369	**227.5**	2	120	1254.5	3	20956			
+PLC07	43	82	190.8	58	552	462.2	66	1057	164.8	26	657	164.8	1	70	78.1	2	1163	**50.9**	1	518
−PLC08	43	82	86.1	37	111				46.7	0	43	**46.2**	0	43	65.1	2	628	123	3	541
+PLC09	14	23	87.4	90	1716	94.6	32	633	61.3	94	1845	35.7	11	193	**11.8**	5	1261	14.5	4	833

Details of measurements

- Fischer's protocol
  - Mutual exclusion algorithm
  - #: number of participants
  - Clock variables → infinite state space
  - CC, CS: lock as explicit variable
  - T: runtime (s), #R: refinements, #S: total abstract states

	PC			PS			CC			CS		
#	T	#R	#S	T	#R	#S	T	#R	#S	T	#R	#S
+2	1.2	17	69	3	15	107	**0.8**	18	66	1.2	14	78
−2	**0.6**	11	41	1.1	9	45	0.8	18	62	1.2	12	58
+3	12.1	97	998	68.1	101	1584	**10.3**	93	1329	45.8	99	1334
−3	**1.4**	19	70	1.5	9	44	1.7	28	121	2.9	21	105

Details of measurements

- Hardware models
  - Hardware Model Checking Competition
  - I: inputs, L: latches, A: and-gates
  - T: runtime (s), #R: refinements, #S: total abstract states

				PC			PS			EC			ES		
	I	L	A	T	#R	#S	T	#R	#S	T	#R	#S	T	#R	#S
+mutexp0	11	20	159	**10.3**	63	494	24.5	43	420	14.3	8	742	22.7	7	806
+mutexp0neg	11	20	159	6.1	44	284	**3.7**	12	82	8.8	9	441	6.7	6	330
−nusmv.syncarb52.B	5	10	52	1.3	30	139	3.1	14	132	0.7	6	113	**0.2**	2	18
−nusmv.syncarb102.B	10	20	157	31.6	110	779	117.9	56	1491	239.8	11	5179	**1.6**	2	32
−pdtpmsarbiter	3	46	209	**0.5**	6	22	4.6	6	22	5.3	15	130	7.8	13	108
+ringp0	15	25	145	16.4	55	300	25.6	19	127	16.1	10	763	**14.5**	7	657
+ringp0neg	15	25	145	**7.8**	21	83	35.7	31	237	187.5	11	4870	108.2	7	2629
+srg5ptimonegnv	30	47	304	**0.3**	3	9	0.5	4	15	1.7	4	40	1.3	3	36