Generic Model Checking for Modal Fixpoint Logics in COOL-MC

(System Description)

D. Hausmann[1] M. Humml[2] S. Prucker[2] L. Schröder[2] A. Strahlberger[2]

VMCAI, London, January 15, 2024

[1]Gothenburg University, Sweden

[2]Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

COOL - A Generic Reasoner and Model Checker

COOL (Coalgebraic Ontology Logic Reasoner)

Toolsuite for verification and reasoning with the coalgebraic μ-calculus

- COOL-SAT: satisfiability checking (ExpTime-complete)
- COOL-MC: model checking (in NP $\cap$ co-NP, in QP)

Works by constructing and solving (general variants of) parity games

\+ Generic algorithms and implementations, framework easily extensible

\+ COOL is lazy: tries to solve games before they have been fully built

Model checking, supported logics

Logics currently supported by COOL-MC:

μ-calculus	models	examples
standard	Kripke frames	CTL, $\nu X.\, \mu Y.\, (p \wedge \Diamond X) \vee \Diamond Y$
monotone	neighbourhood models	game logic
probabilistic	Markov chains	$\nu X.\, \mu Y.\, (p \wedge \langle 0.3 \rangle X) \vee \langle 0.3 \rangle Y$
graded	weighted Kripke frames	$\nu X.\, \mu Y.\, (p \wedge \langle 2 \rangle X) \vee \langle 2 \rangle Y$
alternating-time	concurrent game frames	ATL, $\nu X.\, \mu Y.\, (p \wedge \langle C \rangle X) \vee \langle C \rangle Y$

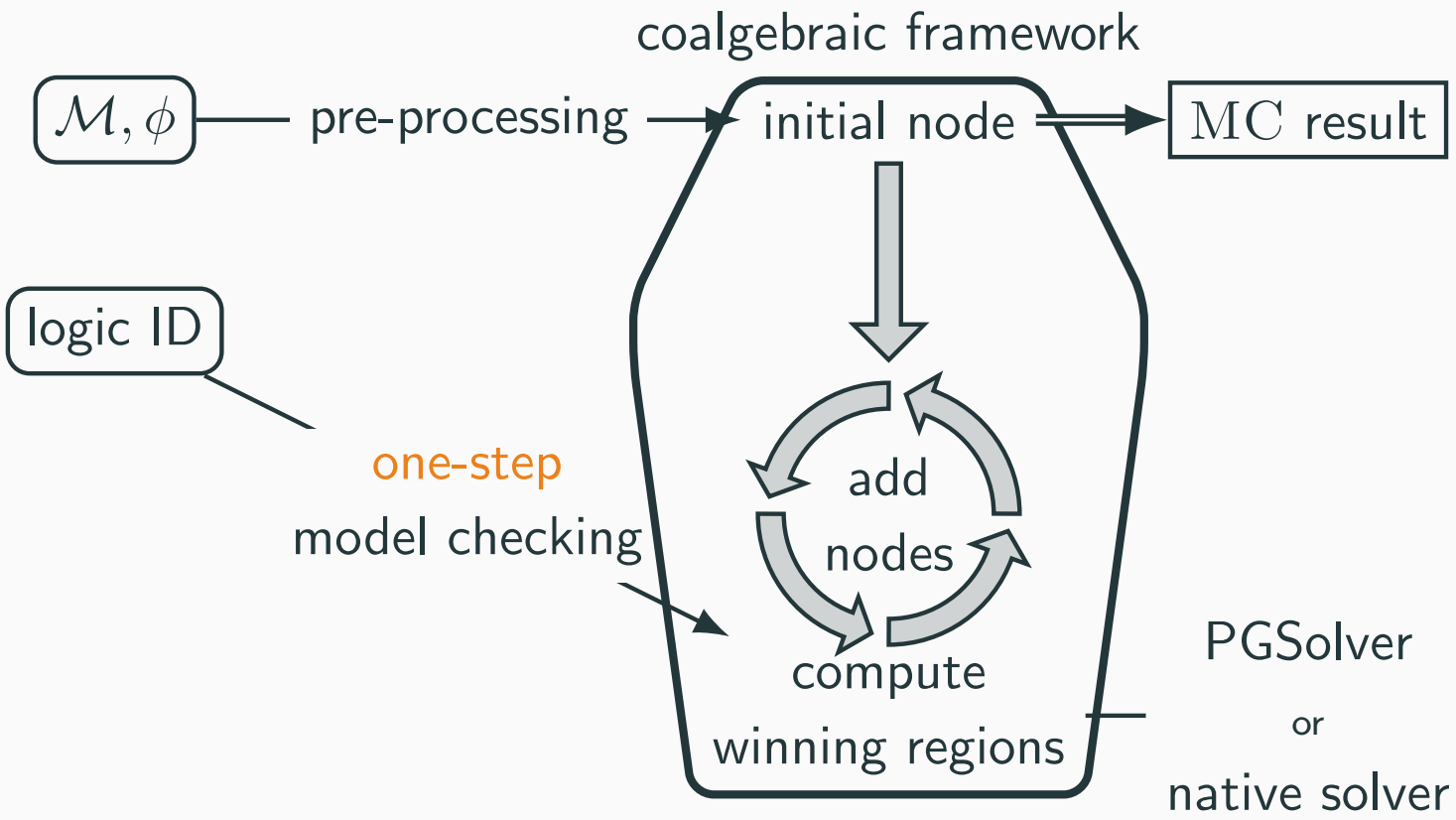
Fact:

Parity game solving and μ-calculus model checking are equivalent.

$\rightsquigarrow$ COOL-MC is solver for (mon., prob., graded, alt.-time) parity games!

COOL-MC schematics

Construct model checking game and solve it (on-the-fly / lazy):



Model Checking in COOL

COOL-MC implements two approaches from [H,Schröder, CONCUR 2019]:

Local model checking algorithm (I)

Build the game graph step-by-step, directly evaluate logical operators

- \+ enables lazy model checking
- \+ evades construction of complex subgames for modalities
- – native solver in COOL is unoptimized (fixpoint iteration)

Model Checking in COOL

COOL-MC implements two approaches from [H,Schröder, CONCUR 2019]:

Local model checking algorithm (l)

Build the game graph step-by-step, directly evaluate logical operators

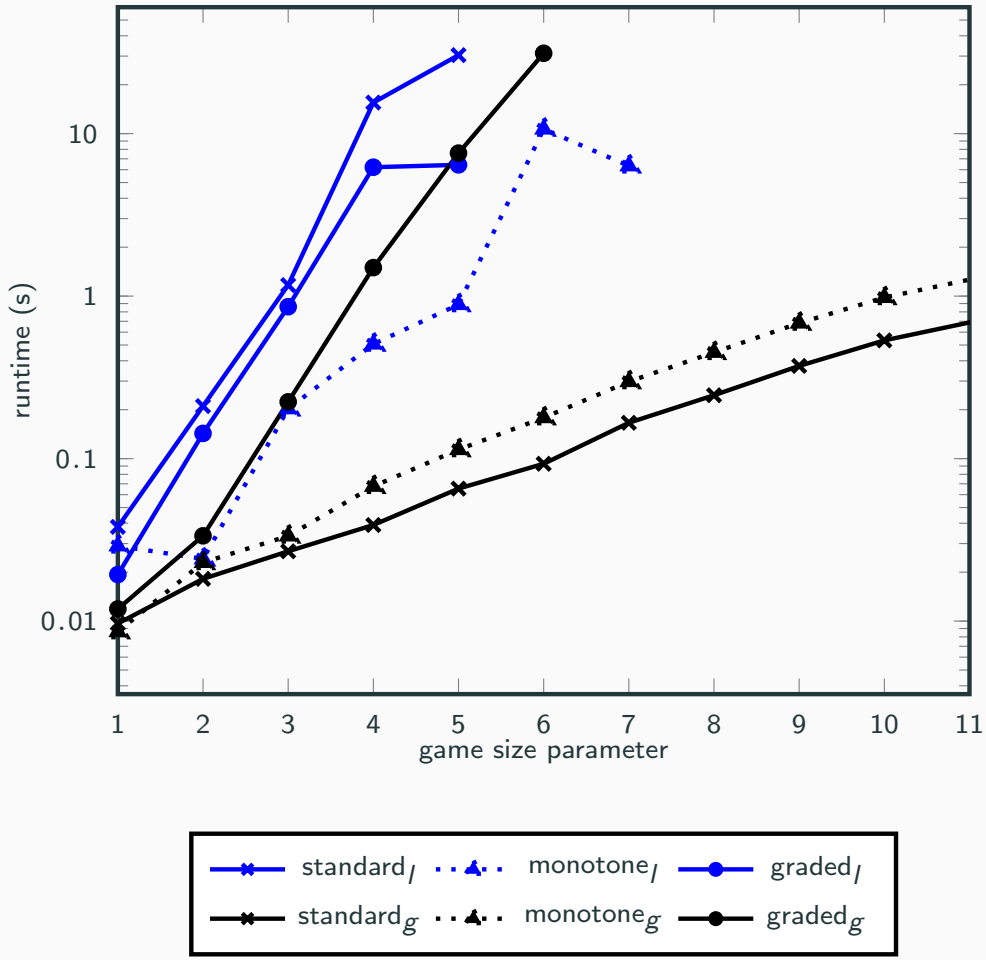
- \+ enables lazy model checking
- \+ evades construction of complex subgames for modalities
- – native solver in COOL is unoptimized (fixpoint iteration)

Game-based model checking (g)

Polynomial reduction to parity games for all supported logics

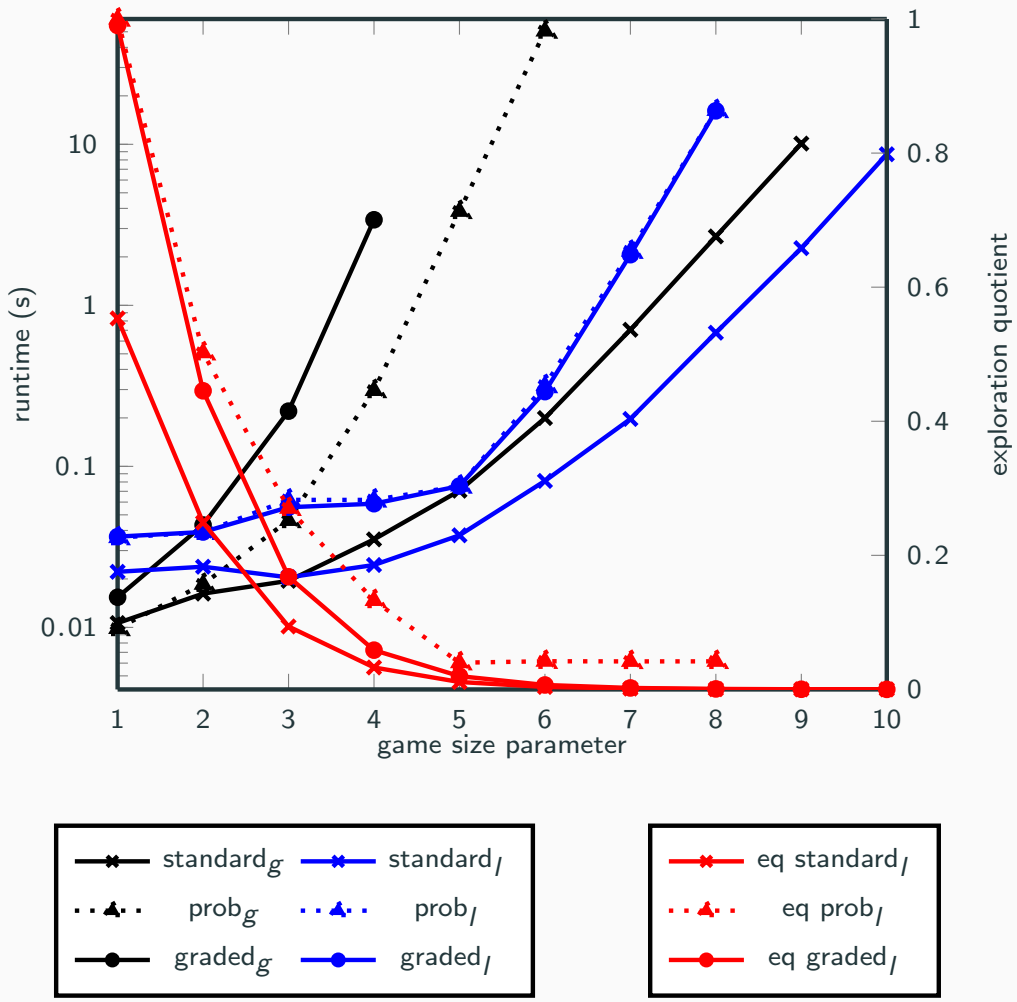
- \+ enables usage of parity game solvers (currently: PGSolver)
- – subgames for modal steps tend to be large (graded, probabilistic)
- – currently no support for lazy solving

Benchmarking: Generalized Parity Games



Language inclusion games (standard, monotone and graded μ-calculi)

Benchmarking: Lazy Solving



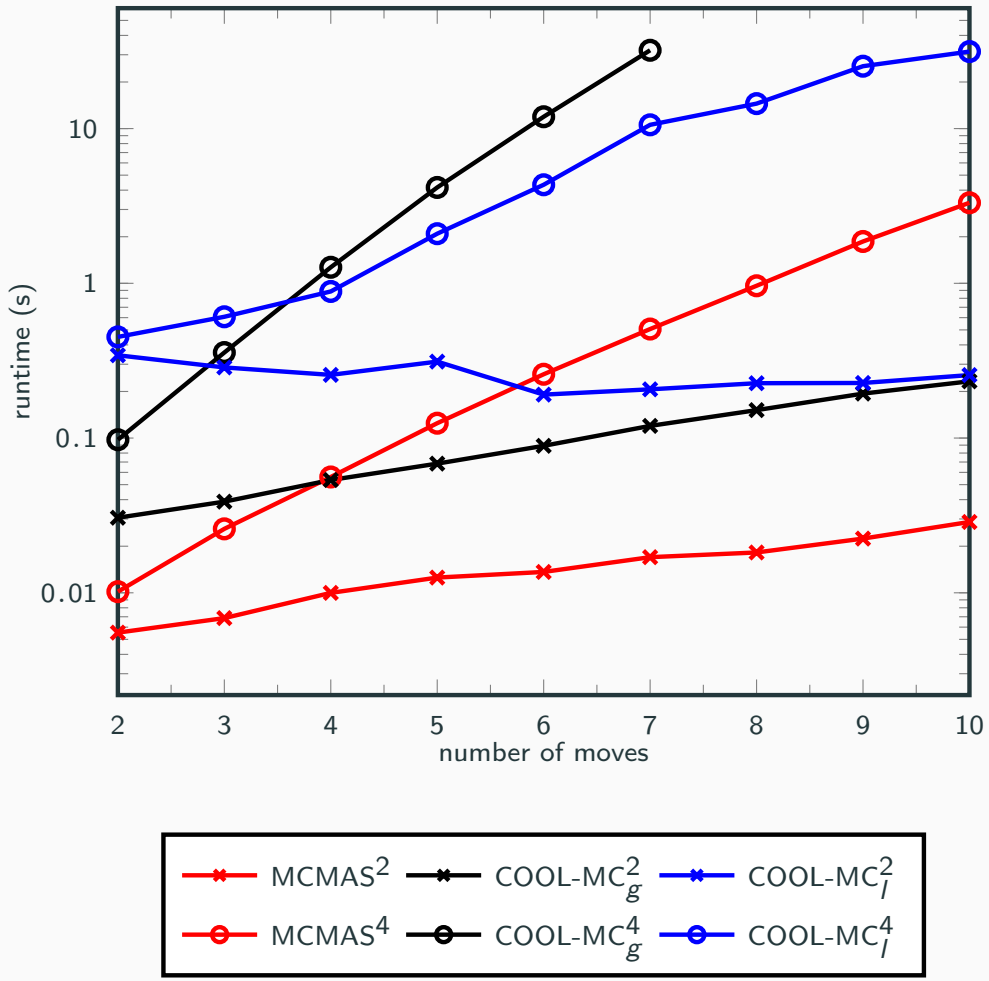
Lazy Tower of Hanoi games (standard, probabilistic and graded μ-calculi)

Benchmarking: Lazy Solving, ctd.

Sizes of (full and lazy) graphs and constructed parity games:

Experiment series	parameter	worlds	full graph	lazy graph	game size
Language incl., mon.	1	3	93	59	126
	7	313	9,703	937	13,146
	30	†	†	†	1,099,896
Lazy Hanoi, standard	1	5	103	57	133
	5	245	5,143	57	6,613
	9	19,685	413,383	53	531,493
	10	59,051	1,240,069	53	†
Lazy Hanoi, graded	1	5	103	102	523
	2	11	229	102	2,345
	4	83	1,741	102	126,222
	10	59,051	1,240,069	102	†

Benchmarking: ATL



ATL model checking: Modulo games (COOL-MC vs. MCMAS)

Summary and Future Work

Take-away:

- COOL-MC: a model checker for μ-calculi with complex modalities
  - Direct fixpoint computation (lazy)
  - Polynomial reduction to parity games
- Benchmarking shows advantages of both approaches

Future work:

- Add back-end (lazy) support for other parity game solvers (e.g. Oink)
- Add support for symbolic (e.g. BDD-based) model checking

Get COOL:

Artifact (functional and reusable): https://zenodo.org/records/10039210