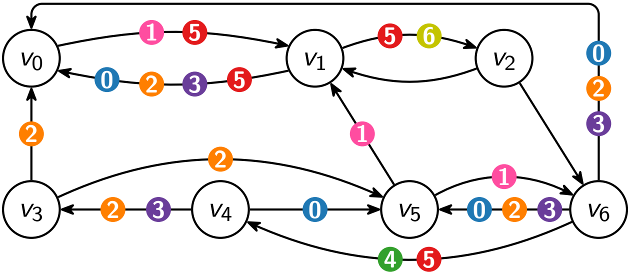
Generic Emptiness Check for Fun and Profit

Christel Baier František Blahoudek Alexandre Duret-Lutz
Joachim Klein David Müller Jan Strejček

ATVA 2019

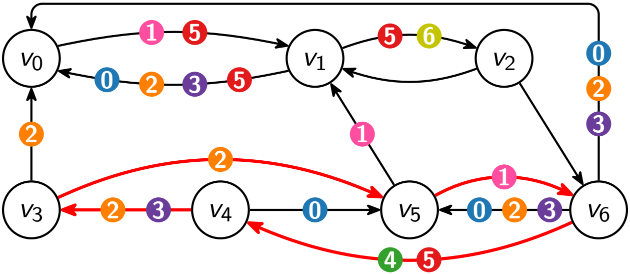
Puzzle



Is there a cycle whose set of marks satisfies this formula?

$$\Big((\neg \textcircled{0} \land \textcircled{1}) \lor (\neg \textcircled{2} \land \textcircled{3})\Big) \land (\neg \textcircled{4} \lor \textcircled{5}) \land (\neg \textcircled{6} \lor \textcircled{7})$$

Puzzle



Is there a cycle whose set of marks satisfies this formula?

$$\big((\neg 0 \wedge 1) \vee (\neg 2 \wedge 3)\big) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7)$$

Outline

- generic emptiness problem
- the algorithm
- solution of the puzzle
- applications and experimental results
    1. emptiness check for ω-automata
    2. probabilistic model checking

Generic emptiness problem

acceptance marks: ⓿, ❶, ❷, . . .

acceptance formulae: $\varphi ::= \mathsf{t} \mid \mathsf{f} \mid \mathsf{Inf}(●) \mid \mathsf{Fin}(●) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$

Let $G = (V, E)$ be a finite directed graph where edges are labelled with finite sets of acceptance marks, and let φ be an acceptance formula. The graph is empty iff there is no cycle satisfying φ.

a cycle satisfies Fin(●) $\iff$ ● is not on any edge of the cycle

a cycle satisfies Inf(●) $\iff$ ● is on some edge of the cycle

Generic emptiness problem

acceptance marks: ⓿, ❶, ❷, ...

acceptance formulae: $\varphi ::= \mathsf{t} \mid \mathsf{f} \mid \mathsf{Inf}(●) \mid \mathsf{Fin}(●) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$

Let $G = (V, E)$ be a finite directed graph where edges are labelled with finite sets of acceptance marks, and let φ be an acceptance formula. The graph is empty iff there is no cycle satisfying φ.

a cycle satisfies	Fin(●)	$\iff$	● is not on any	edge of the cycle
	Inf(●)		● is on some	

The problem whether G is not empty for φ is NP-complete

- given a cycle, one can check in P that it satisfies φ
- NP-hardness by reduction from SAT

Naïve solutions

Idea 1

- enumerate all cycles of G and evaluate φ on each
- runs in $\mathcal{O}(2^{|E|} \cdot |\varphi|)$

Naïve solutions

Idea 1

- enumerate all cycles of G and evaluate φ on each
- runs in $\mathcal{O}(2^{|E|} \cdot |\varphi|)$

Idea 2

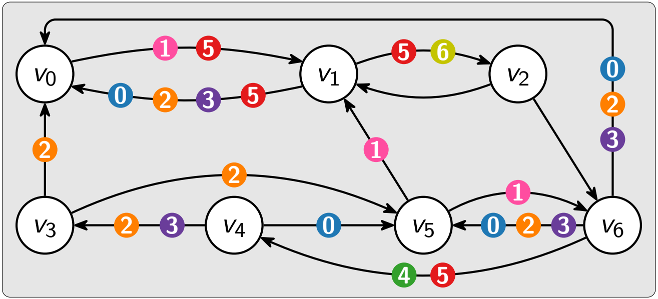
- enumerate all models m of φ and check whether G has a cycle satisfying m
- given a model $m = \{\mathrm{Fin}(⓿), \mathrm{Fin}(❶), \mathrm{Inf}(❷), \mathrm{Inf}(❸)\}$, we remove edges marked with ⓿ and ❶, decompose the graph to SCCs, and check for an SCC containing both ❷ and ❸
- runs in $\mathcal{O}(2^{|\varphi|} \cdot n \cdot |E|)$, n is the number of distinct marks

The algorithm

IS_EMPTY(graph G, acceptance condition φ)
    foreach non-trivial $S \in$ SCCS_OF(G) **do** IS_SCC_EMPTY(S, φ)

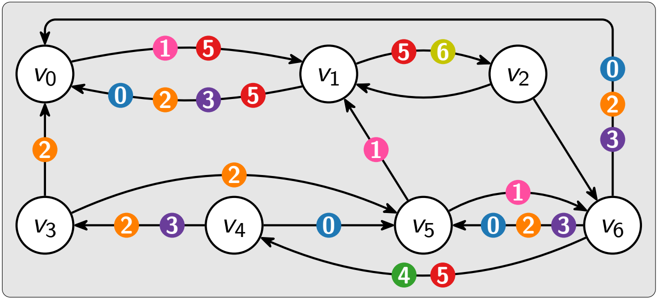
IS_SCC_EMPTY(SCC S, acceptance condition φ)
    $M_{\text{occur}} \longleftarrow$ MARKS_OF(S)
    $\varphi \longleftarrow \varphi[\forall \bullet \notin M_{\text{occur}} : \text{Inf}(\bullet) \leftarrow \text{f}, \text{Fin}(\bullet) \leftarrow \text{t}]$
    if $\varphi \equiv \text{f}$ **then return**
    if $\varphi[\forall \bullet \in M_{\text{occur}} : \text{Inf}(\bullet) \leftarrow \text{t}] \equiv \text{t}$ **then raise** NonEmpty
    foreach disjunct φ_j of φ **do**
        if $\varphi_j \equiv \varphi' \wedge \bigwedge_{\bullet \in M'} \text{Fin}(\bullet)$ **then**
            IS_EMPTY(REMOVE$(S, M'), \varphi')$
        else
            pick some $\bullet$ such that $\text{Fin}(\bullet)$ occurs in φ_j
            IS_EMPTY(REMOVE$(S, \{\bullet\}), \varphi_j[\text{Fin}(\bullet) \leftarrow \text{t}])$
            IS_SCC_EMPTY$(S, \varphi_j[\text{Fin}(\bullet) \leftarrow \text{f}])$

Solving the puzzle



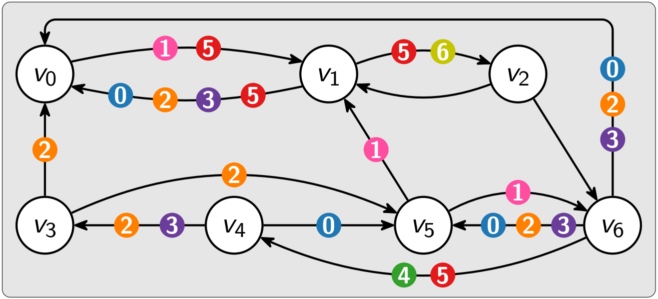
$$\Big(\big(\mathrm{Fin}(0)\wedge\mathrm{Inf}(1)\big)\vee\big(\mathrm{Fin}(2)\wedge\mathrm{Inf}(3)\big)\Big)\wedge\big(\mathrm{Fin}(4)\vee\mathrm{Inf}(5)\big)\wedge\big(\mathrm{Fin}(6)\vee\mathrm{Inf}(7)\big)$$

Solving the puzzle



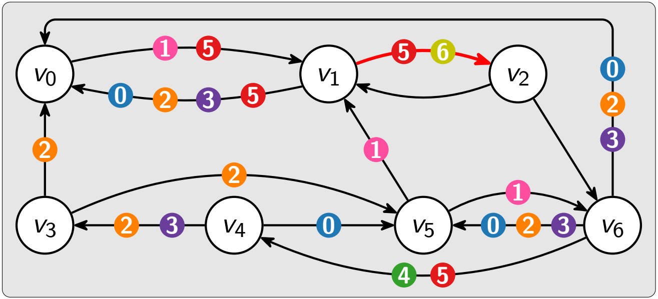
$$\Big(\big(\mathsf{Fin}(0)\wedge\mathsf{Inf}(1)\big)\vee\big(\mathsf{Fin}(2)\wedge\mathsf{Inf}(3)\big)\Big)\wedge\big(\mathsf{Fin}(4)\vee\mathsf{Inf}(5)\big)\wedge\big(\mathsf{Fin}(6)\vee\mathsf{Inf}(7)\big)$$

$7 \notin M_{\mathsf{occur}} \implies$ replacing $\mathsf{Inf}(7)$ by f



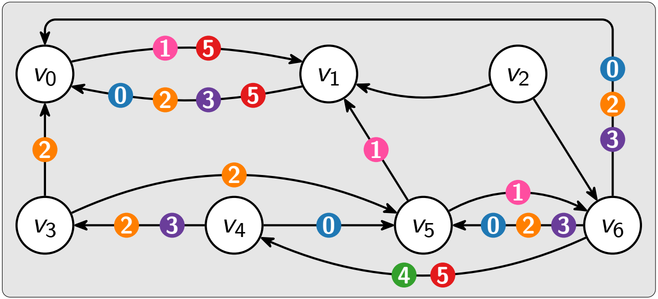
$$\Big(\big(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)\big) \vee \big(\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)\big)\Big) \wedge \big(\mathrm{Fin}(4) \vee \mathrm{Inf}(5)\big) \wedge \ \mathrm{Fin}(6)$$

Solving the puzzle



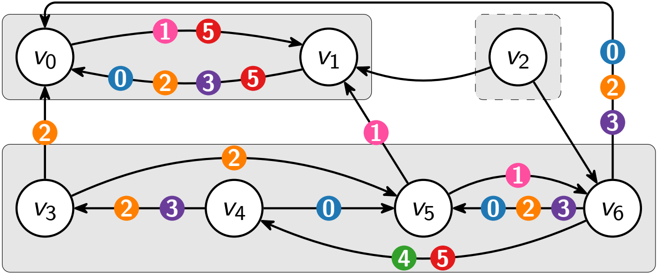
$$\Big(\big(\mathrm{Fin}(0)\wedge\mathrm{Inf}(1)\big)\vee\big(\mathrm{Fin}(2)\wedge\mathrm{Inf}(3)\big)\Big)\wedge\big(\mathrm{Fin}(4)\vee\mathrm{Inf}(5)\big)\wedge\ \mathrm{Fin}(6)$$

Solving the puzzle



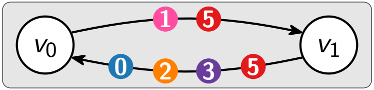
$$\Big(\big(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)\big) \vee \big(\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)\big)\Big) \wedge \big(\mathrm{Fin}(4) \vee \mathrm{Inf}(5)\big)$$

Solving the puzzle

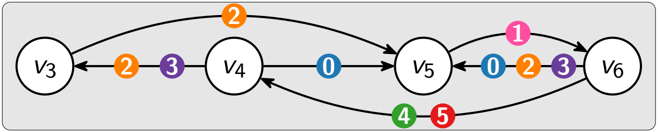


$$\Big(\big(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)\big) \vee \big(\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)\big)\Big) \wedge \big(\mathrm{Fin}(4) \vee \mathrm{Inf}(5)\big)$$

Solving the puzzle

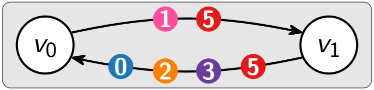


$$\Big((\mathrm{Fin}(❶) \wedge \mathrm{Inf}(❶)) \vee (\mathrm{Fin}(❷) \wedge \mathrm{Inf}(❸))\Big) \wedge (\mathrm{Fin}(❹) \vee \mathrm{Inf}(❺))$$



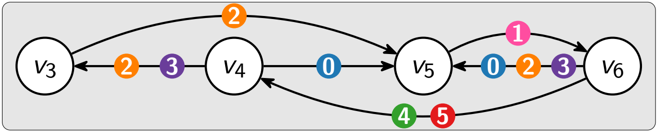
$$\Big((\mathrm{Fin}(⓿) \wedge \mathrm{Inf}(❶)) \vee (\mathrm{Fin}(❷) \wedge \mathrm{Inf}(❸))\Big) \wedge (\mathrm{Fin}(❹) \vee \mathrm{Inf}(❺))$$

Solving the puzzle



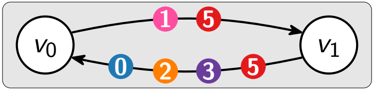
$$\Big((\mathsf{Fin}(0) \wedge \mathsf{Inf}(1)) \vee (\mathsf{Fin}(2) \wedge \mathsf{Inf}(3))\Big) \wedge (\mathsf{Fin}(4) \vee \mathsf{Inf}(5))$$

$4 \notin M_{\mathit{occur}} \implies$
replacing Fin(4) by t

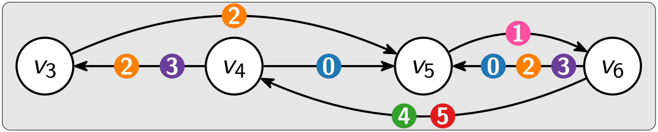


$$\Big((\mathsf{Fin}(0) \wedge \mathsf{Inf}(1)) \vee (\mathsf{Fin}(2) \wedge \mathsf{Inf}(3))\Big) \wedge (\mathsf{Fin}(4) \vee \mathsf{Inf}(5))$$

Solving the puzzle

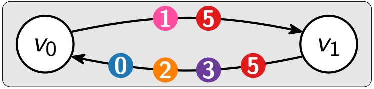


$$\Big((\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)) \vee (\mathrm{Fin}(2) \wedge \mathrm{Inf}(3))\Big)$$

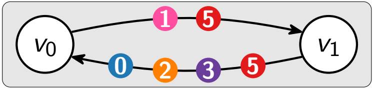


$$\Big((\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)) \vee (\mathrm{Fin}(2) \wedge \mathrm{Inf}(3))\Big) \wedge (\mathrm{Fin}(4) \vee \mathrm{Inf}(5))$$

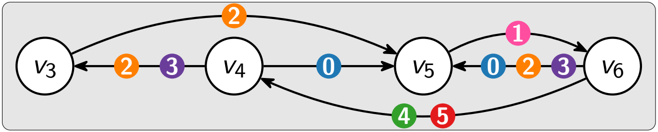
Solving the puzzle



$\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)$

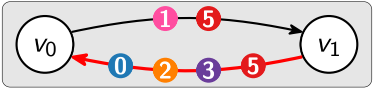


$\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)$

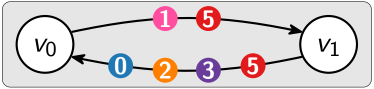


$$\Big(\big(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)\big) \vee \big(\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)\big)\Big) \wedge \big(\mathrm{Fin}(4) \vee \mathrm{Inf}(5)\big)$$

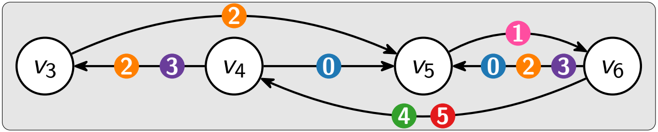
Solving the puzzle



$\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)$

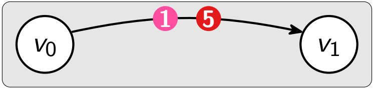


$\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)$

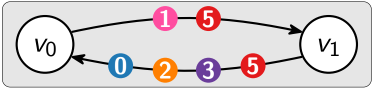


$$\Big(\big(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)\big) \vee \big(\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)\big) \Big) \wedge \big(\mathrm{Fin}(4) \vee \mathrm{Inf}(5)\big)$$

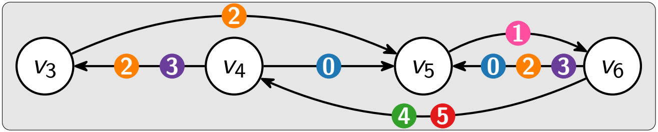
Solving the puzzle



$\mathrm{Inf}(1)$

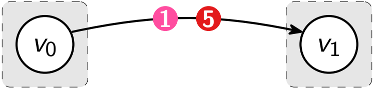


$\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)$

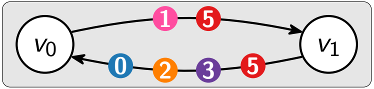


$$\Big((\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)) \vee (\mathrm{Fin}(2) \wedge \mathrm{Inf}(3))\Big) \wedge (\mathrm{Fin}(4) \vee \mathrm{Inf}(5))$$

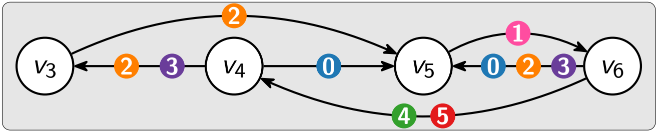
Solving the puzzle



Inf(❶)

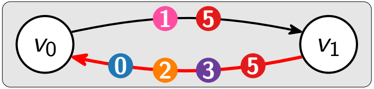


Fin(❷) ∧ Inf(❸)

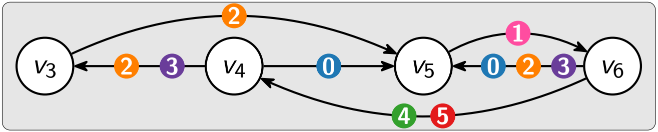


$\Big($(Fin(❶) ∧ Inf(❶)) ∨ (Fin(❷) ∧ Inf(❸))$\Big)$ ∧ (Fin(❹) ∨ Inf(❺))

Solving the puzzle

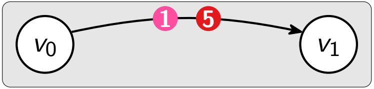


Fin(2) ∧ Inf(3)

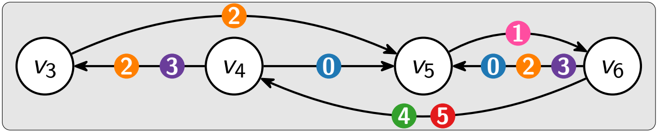


((Fin(0) ∧ Inf(1)) ∨ (Fin(2) ∧ Inf(3))) ∧ (Fin(4) ∨ Inf(5))

Solving the puzzle

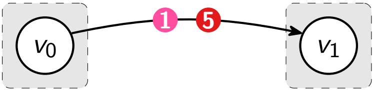


Inf(3)

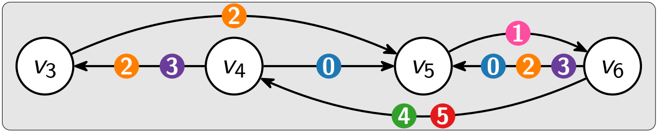


((Fin(0) ∧ Inf(1)) ∨ (Fin(2) ∧ Inf(3))) ∧ (Fin(4) ∨ Inf(5))

Solving the puzzle

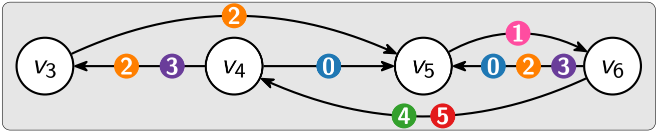


Inf(3)



$\big((\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)) \vee (\mathrm{Fin}(2) \wedge \mathrm{Inf}(3))\big) \wedge (\mathrm{Fin}(4) \vee \mathrm{Inf}(5))$

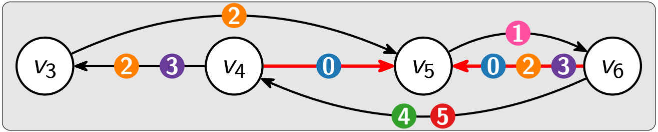
Solving the puzzle



$$\Big(\big(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)\big) \vee \big(\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)\big)\Big) \wedge \big(\mathrm{Fin}(4) \vee \mathrm{Inf}(5)\big)$$

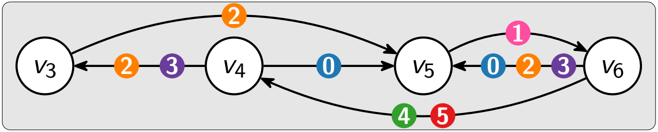
Fin(0) is either true or false

Solving the puzzle



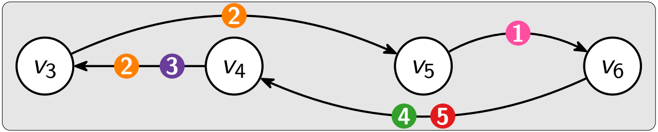
$$\Big(\big(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)\big) \vee \big(\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)\big)\Big) \wedge \big(\mathrm{Fin}(4) \vee \mathrm{Inf}(5)\big)$$

replace Fin(0) by t and remove edges with 0

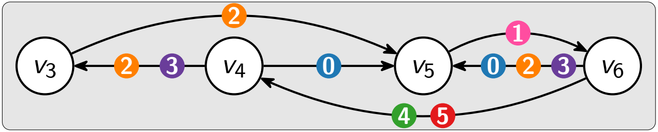


$$\Big(\big(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)\big) \vee \big(\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)\big)\Big) \wedge \big(\mathrm{Fin}(4) \vee \mathrm{Inf}(5)\big)$$

replace Fin(0) by f

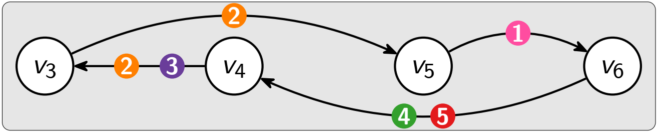


$$\Big(\mathrm{Inf}(1) \lor \big(\mathrm{Fin}(2) \land \mathrm{Inf}(3)\big)\Big) \land \big(\mathrm{Fin}(4) \lor \mathrm{Inf}(5)\big)$$



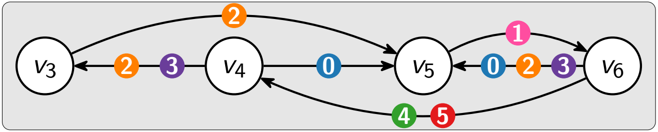
$$\mathrm{Fin}(2) \land \mathrm{Inf}(3) \quad \land \big(\mathrm{Fin}(4) \lor \mathrm{Inf}(5)\big)$$

Solving the puzzle



$$\Big(\mathrm{Inf}(1) \lor \big(\mathrm{Fin}(2) \land \mathrm{Inf}(3)\big)\Big) \land \big(\mathrm{Fin}(4) \lor \mathrm{Inf}(5)\big)$$

evaluates to t after replacing
$\mathrm{Inf}(1), \mathrm{Inf}(3), \mathrm{Inf}(5)$ by t $\implies$ NonEmpty



$$\mathrm{Fin}(2) \land \mathrm{Inf}(3) \quad \land \big(\mathrm{Fin}(4) \lor \mathrm{Inf}(5)\big)$$

Correctness and complexity

Theorem

Given a graph $G = (V, E)$ and an acceptance condition φ, the algorithm is correct and runs in time $\mathcal{O}(2^f \cdot n \cdot |\varphi| \cdot |E|)$, where

- *f is the number of distinct marks in* Fin(●) *terms of φ,*
- *n is the number of distinct marks in φ.*

Application 1:
Emptiness check for ω-automata

Emptiness check for ω-automata

Transition-based Emerson-Lei automata (TELA)

- ω-automata with acceptance conditions as considered before
- a run satisfies Fin(●) iff it visits ● only finitely often
- a run satisfies Inf(●) iff it visits ● infinitely often
- a run is accepting iff it satisfies the acceptance condition

Emptiness check for ω-automata

Transition-based Emerson-Lei automata (TELA)

- ω-automata with acceptance conditions as considered before
- a run satisfies Fin(●) iff it visits ● only finitely often
- a run satisfies Inf(●) iff it visits ● infinitely often
- a run is accepting iff it satisfies the acceptance condition

- TELA represents a non-empty language iff it contains a reachable cycle satisfying the acceptance condition
- to decide emptiness, we remove unreachable states and run the algorithm

Complexity of the emptiness check on classical ω-automata

Emerson-Lei	arbitrary φ	$\mathcal{O}(2^f \cdot n \cdot \lvert\varphi\rvert \cdot \lvert E\rvert)$
Büchi	$\mathsf{Inf}(\bullet)$	
generalized Büchi	$\bigwedge_i \mathsf{Inf}(ⓘ)$	
Rabin	$\bigvee_i (\mathsf{Fin}(ⓘ) \wedge \mathsf{Inf}(ⓘ))$	
Streett	$\bigwedge_i (\mathsf{Inf}(ⓘ) \vee \mathsf{Fin}(ⓘ))$	

Complexity of the emptiness check on classical ω-automata

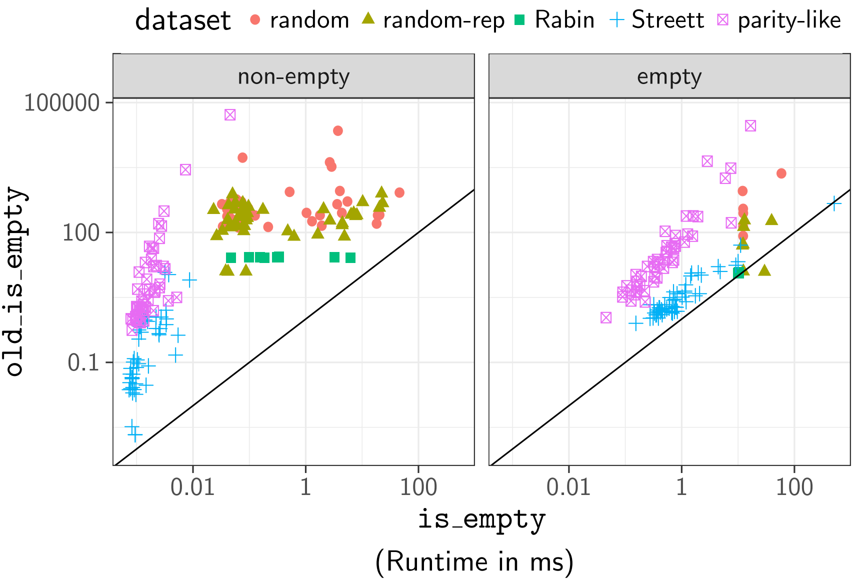
Emerson-Lei	arbitrary φ	$\mathcal{O}(2^f \cdot n \cdot \lvert\varphi\rvert \cdot \lvert E\rvert)$
Büchi	Inf(●)	$\mathcal{O}(\lvert E\rvert)$
generalized Büchi	$\bigwedge_i$ Inf(ⓘ)	$\mathcal{O}(n \cdot \lvert E\rvert + \lvert\varphi\rvert \cdot \lvert V\rvert)$
Rabin	$\bigvee_i$ (Fin(ⓘ) $\wedge$ Inf(ⓘ))	$\mathcal{O}(n \cdot \lvert\varphi\rvert \cdot \lvert E\rvert)$
Streett	$\bigwedge_i$ (Inf(ⓘ) $\vee$ Fin(ⓘ))	$\mathcal{O}(f \cdot (n \cdot \lvert E\rvert + \lvert\varphi\rvert \cdot \lvert V\rvert))$

- polynomial also for generalized Rabin, parity, hyper-Rabin, ...
- often the same complexity as the best known algorithms (does not hold for Streett automata)

Experimental evaluation

- implemented in Spot 2.7
- Spot 2.0–2.6 decides emptiness of TELA by transformation to automata with Fin-less acceptance and an SCC-decomposition of these automata (we call it `old_is_empty`)
- comparison on 5 sets of random automata and automata translated from random LTL formulae

Comparison with the old emptiness check



Application 2:
Probabilistic model checking

Probabilistic model checking

The problem: decide whether a given MDP P satisfies a path property given as an LTL formula φ with a positive probability.

Standard approach

1. translate φ into a deterministic Rabin or generalized Rabin automaton A
2. make a product of P and A
3. search for maximal end-components (SCCs closed under probabilistic choice) satisfying the accepting condition of A.

- implemented e.g. in PRISM
- we modified PRISM 4.4 to handle deterministic (state-based) Emerson-Lei automata using the generic emptiness algorithm (just instead SCCs, it considers maximal end-components)

Experimental evaluation

- model of mutual exclusion protocol (27600 st.) and 6 formulas
- deterministic automata produced by ltl2dstar (Rabin), Rabinizer 4 (generalized Rabin), and Spot (Emerson-Lei).
- time (in seconds) of generalized Rabin emptiness check (t_{Rabin})
- n is the number of acceptance marks

	generalized Rabin		Rabin	
Property	t_{Rabin}	n	t_{Rabin}	n
$\Pr^{\min}(\phi_1)$	130.7	4	–	14
$\Pr^{\max}(\phi_2)$	234.3	6	–	8
$\Pr^{\max}(\phi_3)$	100.1	5	–	6
$\Pr^{\min}(\phi_4)$	251.9	6	1.6	6
$\Pr^{\max}(\phi_5)$	–	12	–	–
$\Pr^{\min}(\phi_6)$	355.3	10	54.9	6

Experimental evaluation

- model of mutual exclusion protocol (27600 st.) and 6 formulas
- deterministic automata produced by ltl2dstar (Rabin), Rabinizer 4 (generalized Rabin), and Spot (Emerson-Lei).
- time (in seconds) of generalized Rabin emptiness check (t_{Rabin}) and our algorithm (t_{EL})
- n is the number of acceptance marks

	Em.-Lei		generalized Rabin			Rabin		
Property	t_{EL}	n	t_{Rabin}	t_{EL}	n	t_{Rabin}	t_{EL}	n
$\Pr^{\min}(\phi_1)$	109.8	4	130.7	121.1	4	–	–	14
$\Pr^{\max}(\phi_2)$	0.4	3	234.3	0.7	6	–	585.9	8
$\Pr^{\max}(\phi_3)$	0.4	3	100.1	0.6	5	–	855.1	6
$\Pr^{\min}(\phi_4)$	0.6	4	251.9	119.0	6	1.6	0.6	6
$\Pr^{\max}(\phi_5)$	–	4	–	–	12	–	–	–
$\Pr^{\min}(\phi_6)$	107.0	6	355.3	127.3	10	54.9	9.6	6

Conclusion

Contributions

- generic emptiness check that unifies various emptiness checks for simpler classes
  - polynomial on common acceptance conditions
  - exponential (in the number of Fin terms) in the worst case
- implemented in Spot and PRISM, with very clear improvements

Possible improvements

- parallelization
- heuristics for non-deterministic choices