Symbolic Finite- and Infinite-state Synthesis

A CEGAR Approach with Liveness Refinements

Shaun Azzopardi, **Nir Piterman**, Luca di Stefano, Gerardo Schneider

University of Gothenburg, TU Wien

July 2024

UNIVERSITY OF GOTHENBURG

CHALMERS UNIVERSITY OF TECHNOLOGY

D-SynMA No.772459

European Research Council
Established by the European Commission

Reactive Synthesis from LTL Specifications

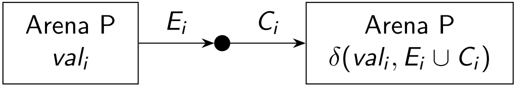
- We have seen:
  - 2EXPTIME-complete.
  - LTL $\Longrightarrow$ DPW $\Longrightarrow$ parity Game.
  - Competition!
- Arena + LTL?
  - GR[1].
  - Early work in robotics.
  - Work by Maoz, Somenzi, Holzmann, ...
  - Synthesis in SE (Uchitel, Braberman, ...)
- What if the Arena is infinite?
  - Decidable classes: Pushdown and beyond (e.g. [Wal'01]).
  - Undecidable classes: Finkbeiner, Piskac, Farzan, Dimitrova, ...

Problem - Infinite-state Arenas with LTL objectives

- Arena/Program over infinite-state variables $\mathbb{V}$: $P = \langle V, \mathbb{E}, \mathbb{C}, val_0, \delta \rangle$
  - V is a finite set of variables, possibly with an infinite domain (e.g., integers),
  - $\mathbb{E}$ is a finite set of Boolean variables controlled by the environment,
  - $\mathbb{C}$ is a finite set of Boolean variables controlled by the controller,
  - $val_0 \in Val(V)$ is initial valuation of $\mathbb{V}$,
  - $\delta : Val(V) \times 2^{\mathbb{E} \cup \mathbb{C}} \mapsto Val(V)$ is the transition function.
- We focus on finitely representable arenas, using a finite set of predicates and updates (more in a couple slides).

- Game $= \langle P, \phi \rangle$, where ϕ is an LTL objective over $\mathbb{E} \cup \mathbb{C} \cup \mathcal{PR}$, and $\mathcal{PR}$ is the set of predicates over V.

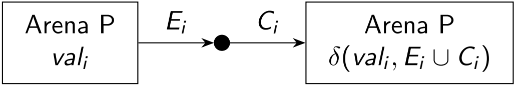
- **In each move**: environment (E) moves first, then controller (C), and finally the arena/program transition updates the variable valuation.



- Trace: $(val_0, E_0 \cup C_0), (\delta(val_0, E_0 \cup C_0), E_1 \cup C_1), ...$

Problem - Realisability and Unrealisability

- Game $= \langle P, \phi \rangle$, where ϕ is an LTL objective over $\mathbb{E} \cup \mathbb{C} \cup \mathcal{PR}$, and $\mathcal{PR}$ is the set of predicates over V.



- ϕ **is realisable modulo** P **iff**:
  there is a Mealy Machine C with input $\Sigma_{in} = 2^{\mathbb{E} \cup Pr}$ and output $\Sigma_{out} = 2^{\mathbb{C}}$
  s.t. every trace t of C that is concretisable on P also satisfies ϕ.

- ϕ **is unrealisable modulo** P **iff**:
  there is a Moore Machine Cs with output $\Sigma_{out} = 2^{\mathbb{E} \cup Pr}$ and input $\Sigma_{in} = 2^{\mathbb{C}}$
  s.t. every trace t of Cs is concretisable on P and violates ϕ.

- Set of predicates Pr includes those in ϕ.
- Symbolic trace **concretisable** on P, **if** it makes **correct predicate guesses** about the induced variable valuation in P: if for each step i $\mathbf{val_i} \vDash \mathbf{Pr_i}$.
- Undecidable in general.

Running Example - Elevator

$\mathbb{V} = \{target : \mathbb{N} = 0, floor : \mathbb{N} = 0\}$
$\mathbb{E} = \{env\_inc, door\_open\}$
$\mathbb{C} = \{up, down\}$

Assumptions:

A1. $GF\, door\_open$
A2. $GF \neg door\_open$

Guarantees:

G1. $GF\, floor = target$
G2. $G(door\_open \implies (up \iff down))$

Figure 1: LTL objective.

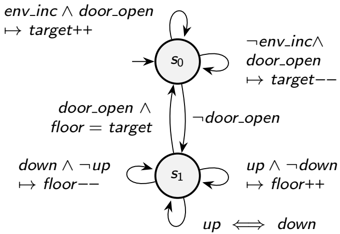
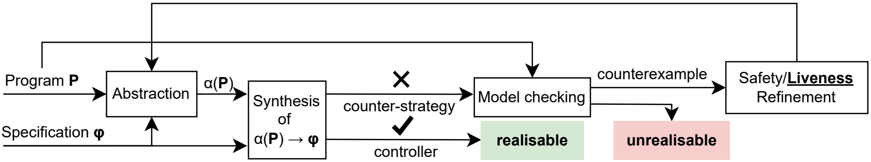


Figure 2: Symbolic arena.

Our Approach



(Predicate) Abstraction

$\mathbb{V} = \{target : \mathbb{N} = 0, floor : \mathbb{N} = 0\}$
$\mathbb{E} = \{env\_inc, door\_open\}$
$\mathbb{C} = \{up, down\}$

Assumptions:

A1. $GF\, door\_open$
A2. $GF \neg door\_open$

Guarantees:

G1. $GF\, floor = target$
G2. $G(door\_open \implies (up \iff down))$

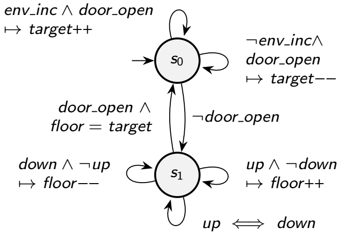
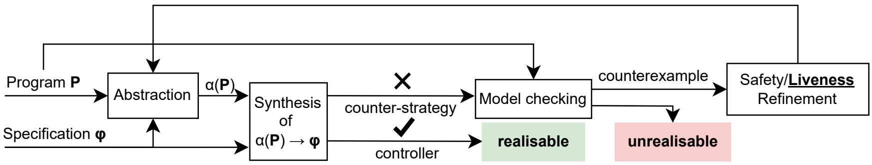


Figure 3: Program P

$$\alpha_0(P, \{floor \leq target, target \leq floor\}) \stackrel{\text{def}}{=}$$
$$floor = target$$
$$\wedge\ G((s_0 \wedge floor = target \wedge env\_inc \wedge door\_open) \implies X(s_0 \wedge floor < target))$$
$$\wedge\ G((s_1 \wedge floor < target \wedge up \wedge \neg down) \implies X(s_1 \wedge (floor < target \vee floor = target))$$
$$\wedge\ \cdots$$

Abstract Synthesis Problem



- Note $\alpha(P)$ always soundly abstracts the concrete behaviour of P.
- Abstraction refined incrementally when new predicates added.
- Each predicate p is replaced by a fresh Boolean variable v_p, controlled by the environment; we denote this set of variables by V_{Pr}.

Theorem (Reduction to Boolean LTL realisability)

For ϕ in $LTL(\mathbb{E} \cup \mathbb{C} \cup Pr)$ and an abstraction $\alpha(P)$ of P in $LTL(\mathbb{E} \cup \mathbb{C} \cup V_{Pr})$, if $\alpha(P) \implies \phi$ is realisable over inputs $\mathbb{E} \cup V_{Pr}$ and outputs $\mathbb{C}$, then ϕ is realisable modulo P.

- What about when the abstract problem is unrealisable?

Model Checking for Unconcretisability Checking

- Given:
  - an abstract counterstrategy as a Moore Machine Cs, and
  - a program/arena P.
- We define a simulation relation that allows to ask whether Cs simulates P, i.e. Cs guesses predicates correctly for every execution it induces in P.
- Practically encoded as the **invariant checking problem** $Cs \| P \vDash G(invar)$
  - Cs chooses the original environment inputs, driving program P.
  - $invar \stackrel{\text{def}}{=} \bigwedge_{p \in Pr} v_p \iff p$: checks correctness of Cs' predicate guesses.
- If $Cs \| P \vDash G(invar)$ then the abstract counterstrategy works also for the concrete problem (but checking it is undecidable).
- Otherwise we are guaranteed to find a finite counterexample which we can use to refine the abstraction.

Theorem (Semi-Decision Procedure for Unconcretisability Checking)

This is a semi-decision procedure for determining unconcretisability of the counterstrategy, and a decision procedure when the program is finite.

Refinements

Two kinds of refinements:

- **Safety refinement (interpolation of counterexample)**
  - Adds state predicates to abstraction.
- Liveness refinements:
  - **Structural loop refinement (find terminating loops in counterstrategy, encode their termination in LTL)**
    Adds:
    - State predicates,
    - Transition predicates,
    - Boolean variables marking points in loop body execution, and
    - LTL constraints.
  - **Ranking refinement (find well-founded relations relevant to the program, encode their well-foundedness in LTL)**
    Adds:
    - State predicates,
    - Transition predicates, and
    - LTL constraints.

Safety Refinement

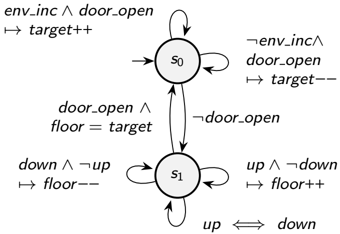
$\mathbb{V} = \{target : \mathbb{N} = 0, floor : \mathbb{N} = 0\}$
$\mathbb{E} = \{env\_inc, door\_open\}$
$\mathbb{C} = \{up, down\}$

Assumptions:

A1. $GF\,door\_open$
A2. $GF\neg door\_open$

Guarantees:

G1. $GF\,floor = target$
G2. $G(door\_open \implies (up \iff down))$



- Safety refinement applies interpolation to the counterexample.
- Initial counterexample:
  - $s_0 \wedge env\_inc \wedge door\_open \wedge floor = 0 \wedge target = 0 \wedge v_{floor \leq target} \wedge v_{target \leq floor}$
  - $s_0 \wedge \neg door\_open \wedge floor = 0 \wedge target = 1 \wedge v_{floor \leq target} \wedge \neg v_{target \leq floor}$
  - $s_1 \wedge up \wedge \neg down \wedge floor = 0 \wedge target = 1 \wedge v_{floor \leq target} \wedge \neg v_{target \leq floor}$
  - $s_1 \wedge floor = 1 \wedge target = 1 \wedge v_{floor \leq target} \wedge$ $\boxed{\neg v_{target \leq floor}}$
- **Interpolation**: gives us $floor - target \leq 1$ and $floor - target \geq 1$, add to abstraction and retry.
- **More safety refinement → enumeration → non-termination**
- Our liveness refinements come to the rescue!

Liveness refinements - Structural Loop Refinement

- Counterexample exposes failed execution of a lasso in counterstrategy? Yes!
  $floor = 0; target = 0; target := target + 1; \textbf{while}(\neg target \leq floor) floor := floor + 1$
- Heuristically generalise precondition (maintaining termination), *true* suffices:

$$\boxed{\textbf{while}(\neg target \leq floor) floor := floor + 1}$$

Liveness refinements - Structural Loop Refinement

- Counterexample exposes failed execution of a lasso in counterstrategy? Yes!
  $floor = 0; target = 0; target := target + 1; \textbf{while}(\neg target \leq floor) floor := floor + 1$
- Heuristically generalise precondition (maintaining termination), *true* suffices:

$$\boxed{\textbf{while}(\neg target \leq floor) floor := floor + 1}$$

- Create an LTL monitor that detects when loop entered and exited:
  - **Initially not in loop**: $\neg in\_loop$
  - **In loop iff (loop iteration or (in loop and stutter))**:

$$G\left(\begin{array}{c} \neg target \leq floor \wedge floor := floor + 1 \wedge target := target \\ \vee \\ in\_loop \wedge floor := floor \wedge target := target \end{array} \iff X\ in\_loop\right)$$

  - **And enforce its termination, or eventual non-progress**:
    $(GF\neg in\_loop) \vee FG(floor := floor \wedge target := target \wedge in\_loop)$
- *Next counterexample*: gives us dual loop (while cond $\neg floor \leq target$)
- These refinements suffice to show the problem realisable.

- Can also handle more complicated loops, e.g., with multiple steps.

Liveness refinements - Ranking Refinement

- For a well-founded term w.r.t an invariant, add assumptions of the form $GF(\text{term\_decreases}) \implies GF(\text{term\_increases} \vee \neg\text{invariant})$.
- Find ranking functions corresponding to terminating loops in counterexample.

Liveness refinements - Ranking Refinement

- For a well-founded term w.r.t an invariant, add assumptions of the form $GF(\text{term\_decreases}) \implies GF(\text{term\_increases} \vee \neg\text{invariant})$.
- Find ranking functions corresponding to terminating loops in counterexample.

or

- **Acceleration**:
  - Before any synthesis attempts, identify predicates in synthesis problem: e.g., $floor \leq target$.
  - Massage: $floor \leq target \equiv 0 \leq target - floor$
  - If term is well-founded w.r.t. predicate (always true for LIA), add assumption:
    - $GF[target - floor]_{dec} \implies GF([target - floor]_{inc} \vee \neg(floor \leq target))$
  - $[target - floor]_{dec} = target_{prev} - floor_{prev} > target - floor$
  - $[target - floor]_{inc} = target_{prev} - floor_{prev} < target - floor$
  - Doing the same for $target \leq floor$ allows us to determine realisability immediately.

Results

Theorem (Correctness of Refinements)

The predicates, boolean variables, and LTL formulas added by each refinement maintain abstraction soundness.

Theorem (Progress of Refinements)

Given a counterexample, there is always a refinement that can be performed, and performing a refinement based on a counterexample ensures the same counterexample and refinement is not re-encountered in subsequent iterations.

Theorem (Sound and complete for finite programs)

The CEGAR algorithm terminates on finite programs.

Evaluation - Tools compared against

- Criteria for comparison:
  - Ability to handle (counter)strategy synthesis, not just realisability checking.
  - Handling at least Büchi objectives.

1. Raboniel (R)
   Maderbacher+Bloem FMCAD22
   - Problem as LTL modulo theory (TSL)
   - CEGAR approach, safety refinements
2. `temos` (T)
   Choi+Finbkeiner+Piskac+Santolucito PLDI22
   - Problem as LTL modulo theory (TSL)
   - One-shot approach (only realisabilility)
   - Safety and limited liveness refinements (SyGuS)
3. `rpgSolve` (RPG)
   Heim+Dimitrova POPL24
   - Problem as deterministic game, with at most Büchi objectives
   - Relies on computing attractors
   - Queries termination of loops
   - No synthesis of counterstrategies

- Other tools:
  - Raboniel and rpgSolve claim similar results to safety/reachability approaches.
  - Other safety-refinement-based approaches for Temporal Stream Logic (TSL) not publicly available or superseded by Raboniel and `temos`.
  - `rpgSolve` claims better results than other realisability checking tools.

Evaluation - Our prototype implementation `sweap`

- Handles LIA problems.
- Strix for LTL synthesis, nuXmv for model/invariant checking, CPAChecker for termination checking, MathSat for SMT solving.
- Two configurations:
  - $S_{acc} \rightarrow$ initially applies acceleration, i.e. performs ranking refinements based on predicates in problems, and
  - $S \rightarrow$ does not perform acceleration.
- Differences from other approaches:
  - `sweap` more initially aggressive abstraction-wise than others; e.g., Raboniel learns relevant transition constraints on-the-fly.
  - Other approaches allow the environment to directly control the value of some numeric variables; for LIA we can encode this with extra states allowing arbitrary inc/decrements.
  - The other approaches rely on quantifier elimination.

Evaluation - Benchmarks

- Only LIA problems (our tool, `sweap` only handles these currently).
- We collect infinite-state benchmarks from literature (19), and contribute our own (9).
- We cast these benchmarks to finite-state domain, to measure scaling against domain size.
- We follow Raboniel and `rpgSolve`, and ignore benchmarks from literature where the LTL objective is immediately realisable (no knowledge of the underlying theory needed).

Evaluation

- –: timeout (10 mins)
- n/a: unsupported goal
- unk: inconclusive result
- ~~x~~: synthesis timeout or unsupported, correct verdict in x sec
- Column **W** indicates winner.
- Best times are set in bold
- *: solvable with only safety refinements

(a) Infinite-state experiment results.

G.	Name	W	R	T	RPG	S_{acc}	S
Safety	box*	S	1.1	unk	**0.5**	7.7	2.9
	box-limited*	S	2.7	–	**0.7**	22.2	5.0
	diagonal*	S	9.8	unk	**0.5**	47.7	3.9
	evasion*	S	5.8	–	**0.8**	–	15.4
	follow*	S	–	–	**1.1**	–	229.0
	square*	S	136.9	–	**0.7**	83.3	48.9
Reachability	robot-cat-r-1d	S	–	–	~~79.5~~	**41.3**	–
	robot-cat-u-1d*	E	–	–	~~75.5~~	48.6	**4.7**
	robot-cat-r-2d	S	–	–	–	–	–
	robot-cat-u-2d*	E	–	–	–	–	**22.6**
	robot-grid-reach-1d	S	–	unk	**1.4**	2.9	8.6
	robot-grid-reach-2d	S	–	unk	~~0.8~~	**7.0**	146.1
	heim-double-x*	S	–	unk	~~1.1~~	–	–
	xyloop	S	–	unk	–	**3.0**	–
Det. Büchi	robot-grid-commute-1d	S	–	unk	~~2.0~~	**12.8**	–
	robot-grid-commute-2d	S	–	–	~~12.1~~	–	–
	robot-resource-1d	E	–	unk	~~25.0~~	**48.8**	122.0
	robot-resource-2d	E	–	–	~~4.8~~	–	–
	heim-buechi	S	unk	–	**4.1**	437.7	–
	heim-fig7*	E	**1.5**	unk	–	3.3	2.7
	batch-arbiter-u*	E	unk	–	~~6.7~~	**3.7**	4.0
Full LTL	reversible-lane-r	S	–	–	n/a	**9.5**	48.5
	reversible-lane-u*	E	–	–	n/a	30.9	**5.7**
	batch-arbiter-r	S	–	–	n/a	**3.2**	9.5
	elevator-w-door	S	–	–	n/a	**4.2**	47.8
	rep-reach-obst.-1d	S	unk	unk	n/a	**3.2**	9.5
	rep-reach-obst.-2d	S	unk	unk	n/a	**16.8**	74.5
	taxi-service	S	–	–	n/a	–	**228.1**
num. solved (out of 28)			6	0	8	20	20

(b) Finite-state experiment results.

Name	W	range	R	T	RPG	S_{acc}	S
elevator simple	S	0..5	9.0	–	7.1	4.0	**3.7**
		0..10	164.2	–	32.8	5.3	**4.4**
		0..50	–	–	–	–	–
elevator signal	S	0..5	–	–	**5.4**	9.9	–
		0..10	–	–	11.0	**10.3**	–
		0..50	unk	–	–	**9.9**	–
rob-grid reach-1d	S	0..5	3.3	unk	**1.5**	3.3	5.7
		0..10	–	unk	**2.3**	3.3	5.8
		0..50	–	unk	10.9	**3.3**	5.8
rob-grid reach-2d	S	5×5	–	–	**3.8**	6.7	14.5
		10×10	–	–	13.6	**6.5**	15.1
		50×50	–	–	–	**7.0**	14.8
batch-arbiter-u	E	0..5	unk	–	~~2.8~~	**3.6**	6.7
		0..10	unk	–	~~3.9~~	**3.8**	6.6
		0..50	unk	–	~~20.8~~	**3.8**	6.7
batch-arbiter-r	S	0..5	–	–	n/a	**3.5**	6.5
		0..10	–	–	n/a	**3.4**	6.2
		0..50	–	–	n/a	**3.5**	6.3
reversible -lane-r	S	0..5	–	–	n/a	**19.8**	59.2
		0..10	–	–	n/a	**20.6**	59.1
		0..50	–	–	n/a	**20.9**	58.7
reversible -lane-u	E	0..5	–	–	n/a	31.3	**6.7**
		0..10	–	–	n/a	54.8	**6.4**
		0..50	–	–	n/a	60.6	**6.3**
elevator w. door	S	0..5	–	–	n/a	**7.6**	111.2
		0..10	–	–	n/a	**7.7**	148.5
		0..50	–	–	n/a	**7.8**	138.4
num. solved (out of 27)			3	0	9	26	23

Evaluation - Infinite-state results/summary

- Verdict (Synt) column indicates number of problems on which correct verdict (counter/strategy) given. Columns joined when the number is the same.

Tool	Solved Realisable		Solved Unrealisable		Unencodable	Total	
	Verdict	Synt	Verdict	Synt		Verdict	Synt
Raboniel	5		1		0	6	
temos	0		(unsupported)		0	0	
rpgSolve	13	8	4	(unsupported)	7	17	8
S_{acc}	15		5		0	20	
S	14		6		0	20	

- sweap outstrips others in LIA synthesis:
  - overkill for the small and simple safety benchmarks.
  - S_{acc} can solve 18 problems immediately, mostly faster than S.
  - only 4 problems not solved by a portfolio approach $S_{acc} \| S$.
- timeout cause:
  - sweap $\rightarrow$ LTL becomes too big.
  - other approaches $\rightarrow$ real timeout, or non-termination (e.g., xyloop for rpgSolve, and problems not marked by $*$ for Raboniel).
- rpgSolve is the only other competitor:
  - solves as many as $S_{acc} \| S$ in realisability if we focus on det. Büchi or simpler;
  - when it succeeds in synthesis, it wins in time taken.

Evaluation - Finite-state results

- `sweap` performance mostly independent of the benchmarks' domain size:
  - A class of problems has the property that:
    if the problem is cast into different finite or infinite domains essentially the same set of safety and liveness properties suffice to decide each variant problem.
  - `sweap` is well-behaved for this class of problems, unlike other approaches.

- The other approaches are too sensitive to the finite domain size, even when it is irrelevant:
  - time-taken increases significantly with the finite domain size increase.

- Our approach, `sweap`, performs best on our finite-state benchmarks
  - it only loses when the domain size is very small (between 0-10).

Future Work

- Turns out refinements based on ranking functions already used for a CEGAR approach to model checking (Balaban, Pnueli, and Zuck, 2005):
  - Predicate and ranking abstractions sound and complete for infinite-state model checking (identification of rankings remains undecidable, of course).
  - Gives hope for a similar relative completeness result for infinite-state synthesis and game solving.
- Game-theoretic view:
  - Liveness refinements rely on finding loops in the abstract game graph winning for the environment.
  - Missing: controller-winning loops (a simple extension, but reduces compositionality of LTL formula, may affect Strix optimisations).
  - We are working on applying this approach directly for infinite-state game solving.
- Extend for Linear Real Arithmetic, and other theories.
- Numeric variables set directly by environment and controller.
- Khalimov and Ehlers, TACAS 24, present a symbolic approach for (finite) synthesis: safety (GR[1]-like) arena with LTL objectives.
  - Can be faster than Strix, but no synthesis of strategies yet..