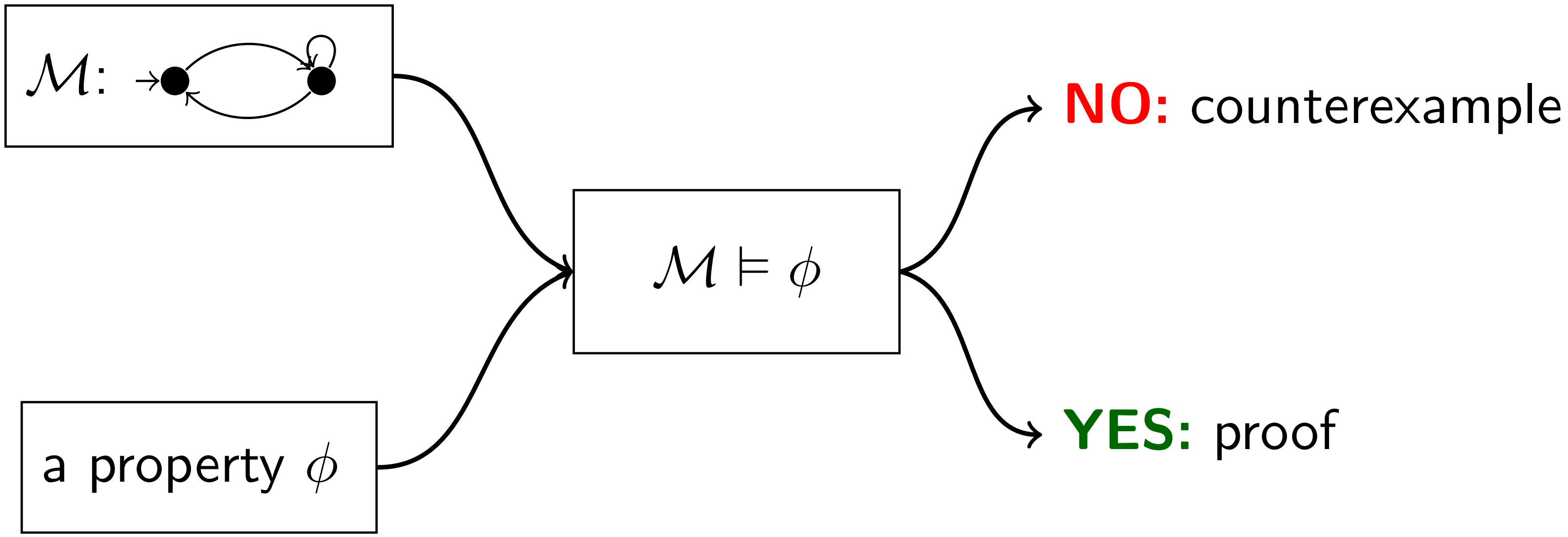
Certifying rlive: A New Proof Strategy for Liveness Model Checking

Giulia Sindoni[1], Alberto Griggio[1], and Stefano Tonetta[1]

[1]Fondazione Bruno Kessler, Trento, IT

FroCoS 2025, 1 October 2025

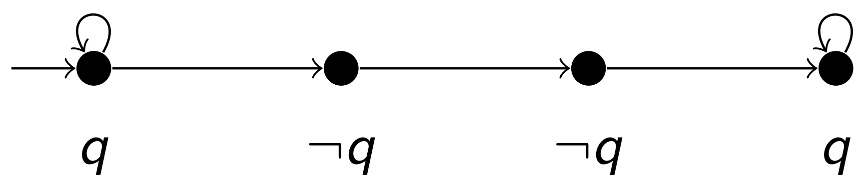
Certifying Model Checking (CMC)



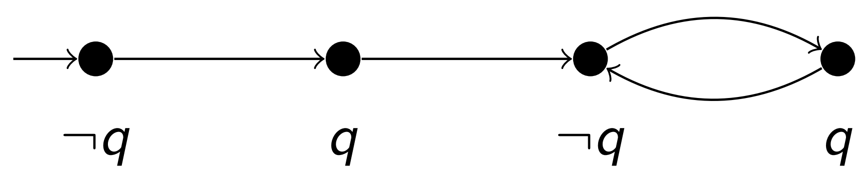
- Trust in verification results for safety-critical systems.
- Certification: extend standard MC with certificates: evidence validating the (yes) answer.
- What serves as evidence? A deductive proof.

The Liveness Checking Problem

- Finite state transition systems $\mathcal{M} = \langle I, T, V \rangle$.
- Liveness checking problem: $\mathcal{M} \models \mathbf{FG}q$: for all paths of $\mathcal{M}$, q eventually holds in all the future states



- Counterexample: an infinite path where $\neg q$ is visited infinitely often ($\mathbf{GF}\neg q$). Finite states: if the property is violated, there always exists a lasso-shaped counterexample.

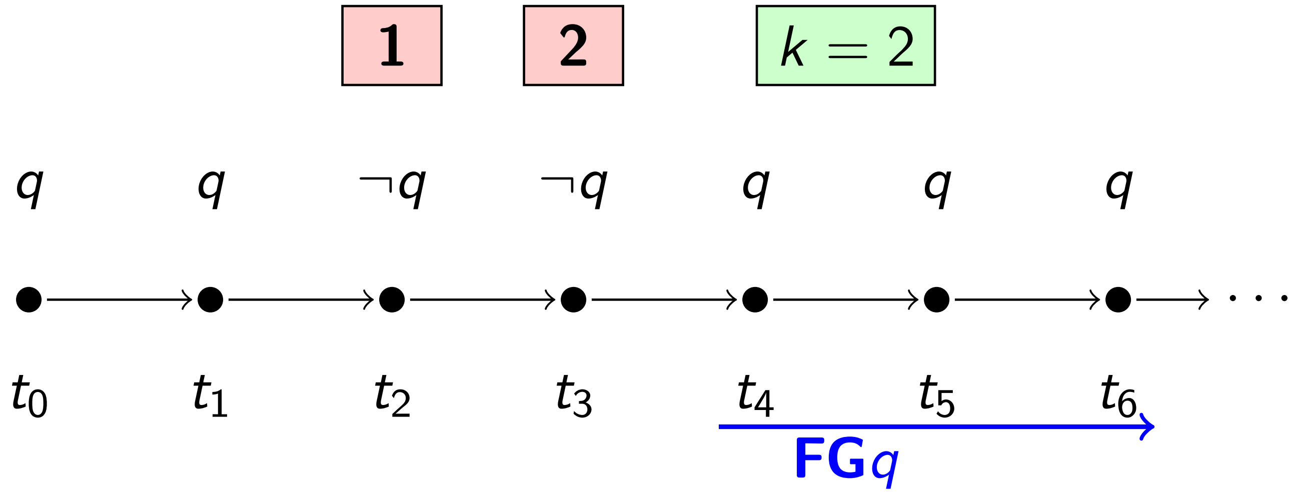


K-liveness: a Liveness Algorithm that Counts (Claessen and Sörensson, 2012)

Key Insight

For any valid liveness property **FG**q in a finite-state system, there exists a bound k such that $\neg q$ can become true at most k times in any trace.

Count $\neg q$ occurrences

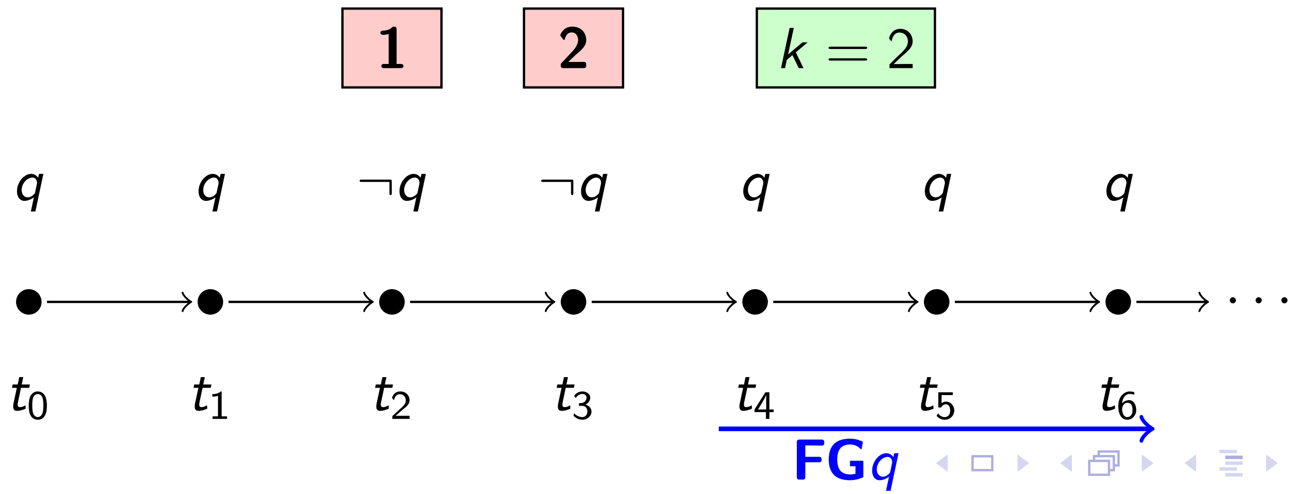


K-liveness: a Liveness Algorithm that Counts

Algorithm Idea

1. Start with $k = 0$
2. Try to prove: $\neg q$ occurs at most k times
3. If successful $\Rightarrow$ property holds
4. If failed, increment k and repeat
5. Each iteration is a safety check

Count $\neg q$ occurrences



rlive: Avoiding the Shoals (Xia et al., 2024)

Key Insight

Builds counterexamples to $\mathbf{FG}q$ incrementally through a recursive, depth-first search process.

Shoals

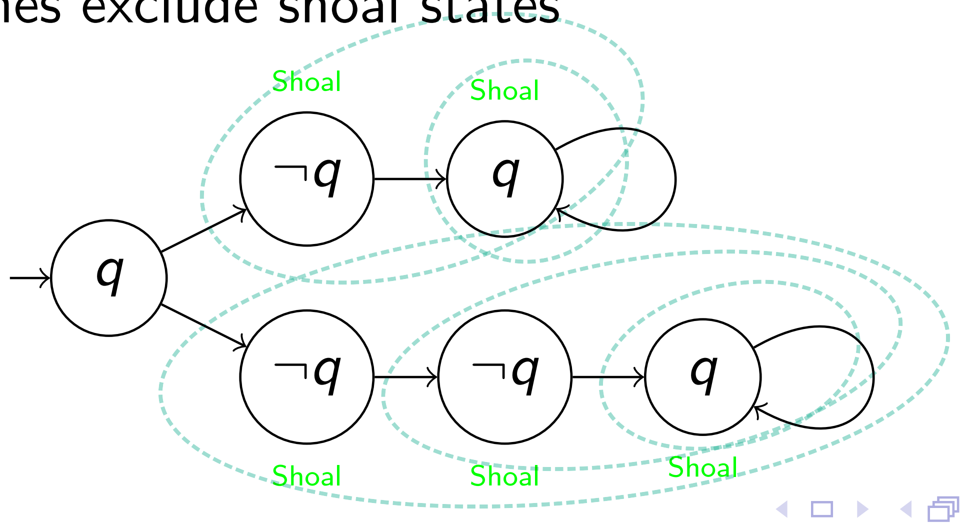
Sets of states that can reach $\neg q$ only finitely many times. When search reaches a dead-end, safety checker provides inductive invariant C representing a newly discovered shoal.

rlive: Shoals Block Future Searches

Algorithm:

1. Find path from initial states to $\neg q$ states
2. Continue searching from successors of each $\neg q$ state
3. Either a previously visited $\neg q$-state is met again, creating a lasso-shaped counterexample. Or no more $\neg q$-states can be reached: a shoal is obtained.

- Each shoal blocks part of state space
- Add constraint $\neg C \wedge \neg C'$ to transition relation
- Future searches exclude shoal states



Certifying rlive: From the Algorithm to the Temporal Deductive rule

rlive Algorithm

```
Procedure rlive(X, I, T, FGq) begin
    C := ⊥
    B := empty stack of states
    while check-invariant(X, I, T ∧ (¬C ∧ ¬C'), T^{-1}(¬C) → q) is Unsafe do
        s := final state of get-counterexample()
        B.push(s)
        while B is not empty do
            s := B.top()
            if check-invariant(X, T(s), T ∧ (¬C ∧ ¬C'), T^{-1}(¬C) → q) is
              Unsafe then
                t := final state of get-counterexample()
                if t ∈ B then
                    return Unsafe
                B.push(t)
            else
                inv := get-inductive-invariant()
                C := C ∨ inv
                B.pop()
    return Safe
```

$\Longrightarrow$

RL rule

$$
\dfrac{\begin{array}{l}
P_{\texttt{init}} := (\mathcal{I} \land \mathbf{G}\mathcal{T}) \to \mathbf{F}C \lor \mathbf{G}q \\
P_0 := \mathbf{G}(C_0 \leftrightarrow \bot) \\
Pk_1 := \mathbf{G}((C_0 \lor C_1) \land \mathcal{T} \to \mathbf{X}(C_0 \lor C_1)) \\
Pp_1 := \mathbf{G}((C_0 \lor C_1) \land \mathcal{T} \land \neg q \to \mathbf{X}(C_0)) \\
\vdots \\
Pk_n := \mathbf{G}((C_0 \lor \ldots \lor C_n) \land \mathcal{T} \to \mathbf{X}(C_0 \lor \ldots \lor C_n)) \\
Pp_n := \mathbf{G}((C_0 \lor \ldots \lor C_n) \land \mathcal{T} \land \neg q \to \\
\mathbf{X}(C_0 \lor \ldots \lor C_{n-1}))
\end{array}}{\boxed{\mathcal{I} \land \mathbf{G}(\mathcal{T}) \to \mathbf{FG}q}}\;\text{RL}
$$

$= \mathcal{M} \models FG\,q$

where $C := C_0 \lor C_1 \lor \ldots \lor C_n$ is the final set of discovered shoals.

Certifying rlive: the Temporal Deductive rule

- $P_{\texttt{init}} := (\mathcal{I} \wedge \mathbf{G}\mathcal{T}) \rightarrow \mathbf{F} C \vee \mathbf{G} q$: on any trace either the shoal is eventually entered, or q is an invariant.
- $P_0 := \mathbf{G}(C_0 \leftrightarrow \bot)$: the shoal is empty initially.
- $Pk_i := \mathbf{G}((C_0 \vee \ldots \vee C_i) \wedge \mathcal{T} \rightarrow \mathbf{X}(C_0 \vee \ldots \vee C_i))$ the invariant C incrementally built is inductive.
- $Pp_i := \mathbf{G}((C_0 \vee \ldots \vee C_i) \wedge \mathcal{T} \wedge \neg q \rightarrow \mathbf{X}(C_0 \vee \ldots \vee C_{i-1}))$ the search space can be incrementally restricted, as long as we keep visiting a new $\neg q$-state.

Correctness and Completeness of the RL Rule

Proof of Correctness: by Contradiction

Correctness of rule RL **formally proven** in the theorem prover. The main step is:

$$\cfrac{\mathrm{RLB}\ \cfrac{\mathbf{F}C \qquad \Pi \qquad [\mathbf{GF}\neg q]}{\mathbf{F}C_0} \qquad\qquad P_0 := \mathbf{G}\neg C_0}{\cfrac{\bot}{\mathbf{FG}q}}$$

where $\Pi = \{Pk_1, Pp_1, \ldots, Pk_n, Pp_n\}$

Proof of Completeness

If the algorithm rlive succeeds in establishing the liveness property, then it generates shoals $C = C_0 \vee \ldots \vee C_n$ such that the premises of the temporal deductive rule RL are true. So the model will satisfy the necessary premises for RL to be applied successfully.

RL as a Generalization of k-liveness rule

Shared Intuition

Both algorithms prove the same fundamental property: $\neg q$ can occur at most finitely many times in any trace of a finite-state system.

The Generalization

Key insight: In rule for k-liveness (Griggio et al. 2021) we have formulae (inductive invariants) $\alpha_0, \ldots, \alpha_{k+1}$, that keep count of the number of times $\neg q$ is reached. There is a mapping $\alpha_i \mapsto C_{k-i+1}$ such that:

- RL rule it can be used to build proofs for k-liveness using this mapping.

The Proof Strategy for Liveness Checking

$$\frac{P_{\texttt{init}} \quad P_0 \quad Pk_1 \quad Pp_1 \quad \ldots \quad Pk_n \quad Pp_n}{\mathcal{I} \wedge \mathbf{G}(\mathcal{T}) \rightarrow \mathbf{FG}q} \text{ RL}$$

TP strategy

1. Assume $P_{\texttt{init}}, P_0$ and Π,
2. Apply RL rule: thus we can conclude the goal: $\vdash \mathcal{I} \wedge \mathbf{G}(\mathcal{T}) \rightarrow \mathbf{FG}q$
3. Discharge of proof obligations: P_0, Π discharged using SAT solver.
4. Discharge of proof obligations:
$P_{\texttt{init}} := \mathcal{I} \wedge \mathbf{G}\mathcal{T} \rightarrow \mathbf{F}C \vee \mathbf{G}q = \mathcal{I} \wedge \mathbf{G}(\mathcal{T} \wedge \neg C) \rightarrow \mathbf{G}q$ invariant claim discharged using a subroutine for proving invariants.

Technical foundation: Formalising LTL in PVS

- PVS: specification language with integrated theorem prover.
- Interactive but also supports strategies developments.

Shallow embedding of LTL

```
shallow_ltl[State: TYPE+]: THEORY
BEGIN
Trace: TYPE = ARRAY[nat -> State]
ltlformula: TYPE = [Trace -> [nat -> bool]]
...
NOT(P)(trace: Trace)(t: nat): bool = NOT P(trace)(t);
NEXT(P)(trace: Trace)(t: nat): bool = P(trace)(t+1);
...
valid(P): bool = FORALL (trace: Trace): P(trace)(0)
...
```

Certification Flow

$\mathcal{M} \models \mathbf{FG}q$

Model checking

NuXMV

$C_0, \ldots C_n, \psi$

Generation of formal theory

aigtopus.py

$\mathcal{I}, \mathcal{T} : \ldots$

$C, \psi : \ldots$

$thm : \vdash \mathcal{I} \wedge G(\mathcal{T}) \rightarrow \mathbf{FG}q$

Theorem Prover

PVS

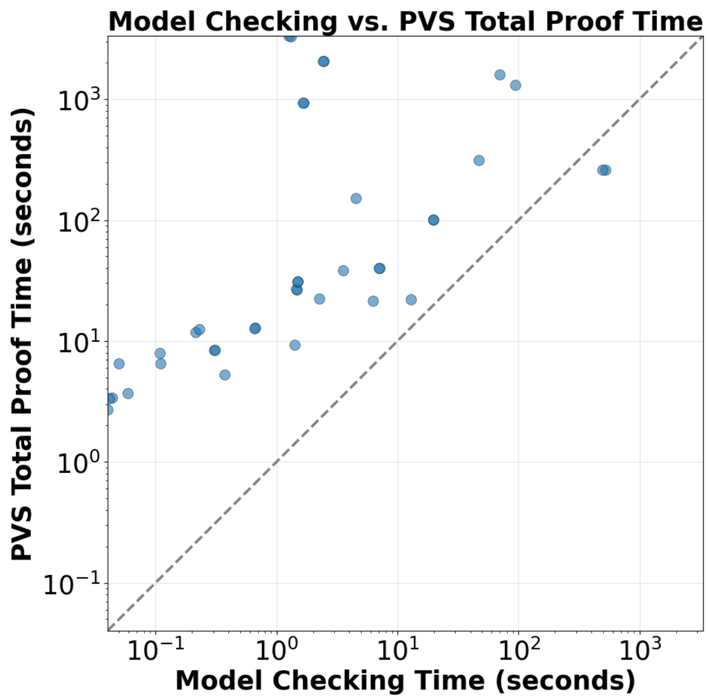
Proof:

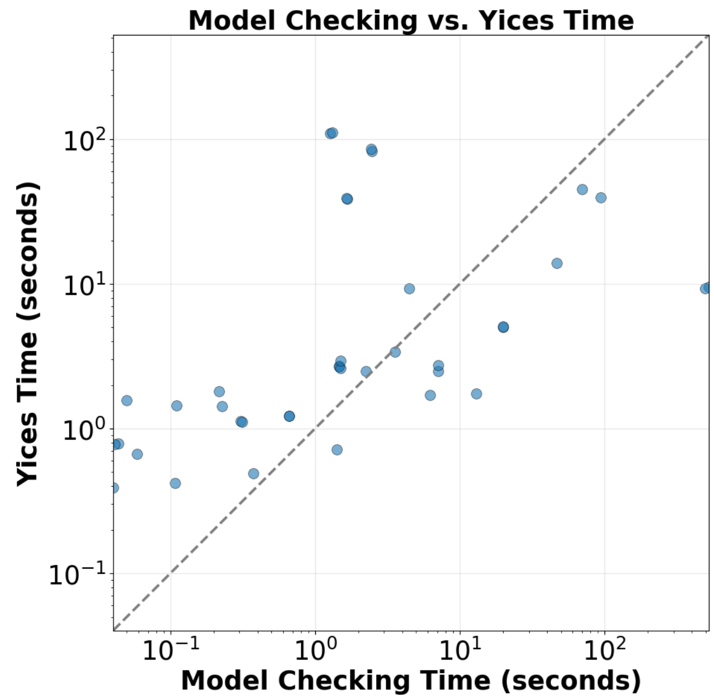
$\vdots$

$\vdash thm$

Apply RL Lemma: proven once & for all

Experimental Setup





- Benchmarks Source: Hardware Model Checking Competition.
- 53 problems tested, 41 successfully certified within time and memory limit.
- Demonstrates feasibility but highlights performance gap.
- Bottleneck: PVS internal bookkeeping and definition management.
- Insight: performance gap primarily due to theorem prover infrastructure

Achievements and Future Work

Key Contributions:

- Novel proof strategy for certifying liveness checking results.
- Despite the complexity of rlive, shoals provided by the model checker are sufficient to generate proofs.
- Minimal model checker modifications - only output shoal.
- Progress in CMC: distribute the trust across more fundamental principles and create redundancy that increases overall confidence.

Future Work:

- Extending certifying model checking approach to other liveness checking algorithms (liveness-to-safety, FAIR): a strategy that encompasses them all?
- Generalisations to the infinite-state transition systems and SMT.

Bibliography

1. Claessen Koen, and Niklas Sörensson. "A liveness checking algorithm that counts." Formal Methods in Computer-Aided Design (FMCAD). IEEE, 2012.
2. Xia Yechuan, et al. "Avoiding the shoals: a new approach to liveness checking." CAV24, 2024.
3. Alberto Griggio, Marco Roveri, and Stefano Tonetta. "Certifying proofs for SAT-based model checking." Formal Methods in System Design 57.2 (2021): 178-210.
4. Giulia Sindoni, et al. "A Theorem Prover Based Approach for SAT-Based Model Checking Certification." Automated Deduction–CADE 30, 2025.