# Generalizing counterexamples: certification of temporal properties with parity games

Lehr- und Forschungseinheit für Theoretische Informatik
Institut für Informatik
Ludwig-Maximilians-Universität München

FSV 2 · Jan 30, 2018

# Roadmap

- Certified decision procedures
- μ-calculus and parity games
- Certificates for model checking and μ-calculus
- Computing certificates by fixpoint instrumentation
- Certificate checking
- Evaluation

# What is certified model checking?

**(Namjoshi 2001)**

> *[I]f it is determined that a property holds, model checkers produce only the answer "yes"! This does not inspire the same confidence as a counterexample; one is forced to assume that the model checker implementation is correct.*
>
> *— Kedar S. Namjoshi (2001)*

- Not only check whether a temporal property holds…
- …but also give an argument *why*.

# Certified decision procedures

- Decide a property, and also compute a *certificate* of it.
    - Checking the certificate only succeeds if the result is correct
- Requirements for useful certificates:
    - Checkable by a standalone, “simpler” routine
    - Checkable efficiently (in a lower complexity class)
    - Low overhead to generate
    - Compact representation
- Examples from other areas:
    - Certified UNSAT
    - Primality tests
    - Polyhedral Array-Bound Analysis

# Benefits of this approach

- More confidence that the result is right
- Separation between computation and correctness
    - Can optimize decision procedure independently
- Viable for formal verification
    - Only need to verify the certificate checker
    - Trade *completeness* for implementation efficiency
    - Possible to interpret certificate as proof object
        - Possible to use in a theorem prover

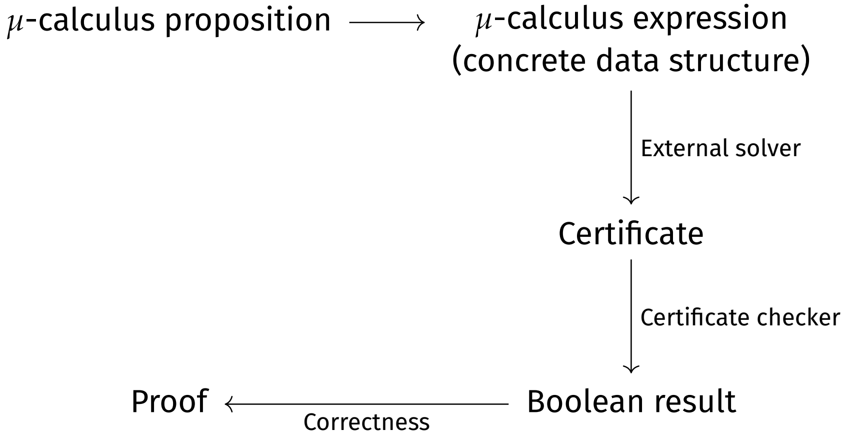
# Making use of external decision procedures

- Don’t want to do complex computations inside Coq
- Don’t want to prove correctness of the *whole* solver
- Want to make use of existing solvers
- Boolean results of external solvers are not enough

# Formally verified certified decision procedures

- Elegant approach: proof by reflection
- Given a temporal formula and a model in your proof logic
- Construct a concrete representation
- Use external decision procedure to compute certificate
- Interpret certificate as object of the proof logic
- Apply the correctness theorem of your certificate checker to gain a proof object of the original formula

# Reflecting decision procedures with certificates



# Consequences

- External program can do anything it wants:
  no completeness guaranteed. But errors will be noticed!
- Only certificate checker needs (weaker) correctness proof.
  $\forall s \in S, \phi \in \Phi : check(\phi, cert(\phi), s) = \textbf{\textit{true}} \rightarrow s \in [\![\phi]\!]$.
- Only certificate checker runs as a Coq function.
  - Coq function evaluation is quite speedy, can make use of a virtual machine.
  - Standalone checker `coqchk` is not so fast.
- Small proof objects, essentially the certificate + call to the checker.

# Certificates for model checking

- Familar case: counterexamples (“negative certificate”)
  - E.g. for LTL, a *lasso* where $\phi$ doesn’t hold to refute a $G\phi$ formula
- For simple CTL formulas, positive certificates could be:
  - for a $EF\phi$ formula, a finite path where at the end $\phi$ holds
  - for a $EG\phi$ formula, a lasso where $\phi$ holds in the loop
  - for a $AG\phi$ formula, an argument that $\phi$ holds for all reachable states, and all moves stay in that area
- Not obvious how certificates for general CTL or CTL* formulas look like (Shankar and Sorea 2003; Sorea 2005)
  - How to certify $AG\,EF\,\phi$?

# Certificates for µ-calculus

- µ-calculus is a powerful temporal logic with arbitrary nested greatest and least fixpoint operators
    - strictly more powerful than LTL, CTL and CTL* (which are embeddable using low quantifier alternation)
    - strictly more powerful with each quantifier alternation
- Well known relationship between µ-calculus and parity games (Emerson and Jutla 1991)
- Can leverage winning strategies for parity games as certificates for µ-calculus
- Winning stategies for the dual formula $\phi^*$ are *generalized counterexamples*

# µ-calculus

- highly expressive temporal logic, subsumes LTL, CTL, CTL*
- *model checking problem*: on which states of a given finite labelled transition system is a given formula true?

$$
\begin{aligned}
\phi ::= {} & X && \textit{(variables)} \\
\mid {} & p \mid \neg p && \textit{(atomic propositions)} \\
\mid {} & [a]\phi && \textit{(for all } a\textit{-transitions)} \\
\mid {} & \langle a\rangle\phi && \textit{(} a\textit{-transition exists)} \\
\mid {} & \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 && \\
\mid {} & \mu X.\phi && \textit{(least fixpoint)} \\
\mid {} & \nu X.\phi && \textit{(greatest fixpoint)}
\end{aligned}
$$

# μ-calculus

$$\phi ::= X \mid p \mid \neg p \mid [a]\phi \mid \langle a\rangle\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \mu X.\phi \mid \nu X.\phi$$

“for every path, q holds at every position”

$$\mu X.q \wedge [a]X$$

“there are paths where q is true infinitely often”

$$\nu X.\mu Y.(q \wedge \langle a\rangle X) \vee \langle a\rangle Y$$

“there are paths where q holds at every even position”

$$\nu X.q \wedge \langle a\rangle\langle a\rangle X$$

*(more powerful than CTL*)*

# μ-calculus: Set semantics

$$
\begin{aligned}
sem(X, \eta) &= \eta(X) \\
sem(\phi_1 \wedge \phi_2, \eta) &= sem(\phi_1, \eta) \cap sem(\phi_2, \eta) \\
sem(\phi_1 \vee \phi_2, \eta) &= sem(\phi_1, \eta) \cup sem(\phi_2, \eta) \\
sem([a]\phi, \eta) &= \widetilde{pre}(\xrightarrow{a})(sem(\phi, \eta)) \\
sem(\langle a \rangle \phi, \eta) &= pre(\xrightarrow{a})(sem(\phi, \eta)) \\
sem(\mu X.\phi, \eta) &= iter_X(\phi, \eta, \emptyset) \\
sem(\nu X.\phi, \eta) &= iter_X(\phi, \eta, S) \\
s \in \widetilde{pre}(\xrightarrow{a})(U) &\Leftrightarrow \forall t \in S.\, s \xrightarrow{a} t \implies t \in U \qquad \textit{(weakest precondition)} \\
s \in pre(\xrightarrow{a})(U) &\Leftrightarrow \exists t \in S.\, s \xrightarrow{a} t \wedge t \in U \qquad \textit{(preimage)} \\
iter_X(\phi, \eta, U) &= \text{let } U' := sem(\phi, \eta[X := U]) \text{ in} \\
&\qquad \text{if } U = U' \text{ then } U \text{ else } iter_X(\phi, \eta, U')
\end{aligned}
$$

# Parity games

A *parity game* consists of a disjoint sum of positions $\mathsf{Pos} = \mathsf{Pos}_0 \cup \mathsf{Pos}_1$, a total edge relation $\rightarrow \subseteq \mathsf{Pos} \times \mathsf{Pos}$ and a priority function $\Omega : \mathsf{Pos} \rightarrow \mathbb{N}$.

Moves happen along the edge relation. The destination decides who moves next.

The game is *won* if the largest priority that occurs infinitely often is even, the opponent wins if it is odd.

# Strategies for parity games

A *strategy* $\rho$ is a function that tells the player how to move next.

A *positional strategy* only takes the the current position into account.

A position is in a *winning set* $W_i$ if there exists a strategy $\rho$ such that player $i$ wins, starting at a position in $W_i$.

**Theorem.** Every position $p$ is either in $W_0$ or $W_1$ and player $i$ wins positionally from every position in $W_i$.

# μ-calculus and parity games

- Positions of the parity game: states $\times$ subformulas
- Moves: according to the edges of the model resp. subformula relation
- Priorities: outer fixpoints have higher priority, $\mu$ odd, $\nu$ even, we win (= formula is true) when highest recurrent priority is even.
- Strategy: how to move at a given position
    - There are always memoryless winning strategies
    - Actual choices only on $\vee$ and $\langle a \rangle$
    - Representable in $O(|S|^2|\phi|)$

# Computing winning strategies for μ-calculus

- For finite models, μ-calculus validity is computed by fixpoint iteration in a straight forward manner.
    - As usual, computation complexity raises with quantifier alternation.
- We show that such a fixpoint computation can be *instrumented* to compute a winning strategy as well.

# Computing winning strategies by fixpoint iteration

- Instead of computing with sets, we use *partial winning strategies*, i.e. winning stategies defined on a subset of the states.
- Inductively defined by the structure of the formula, computed in a *compositional* manner.
- For fixpoints, we iteratively grow a partial winning strategy to its maximum domain.
    - Same time complexity as computing set semantics
    - Space complexity increases from $O(|S||\phi|)$ to $O(|S|^2|\phi|^2)$ to keep track of the strategy.

# Strategies for μ-calculus

We can interpret a $\mu$-calculus formula $\phi$ as a parity game. Moves can happen along the subformulae (example later). The priority of a position depends on the kind of formula and its nesting depth.

A partial winning strategy for $\mu$-calculus is a partial function

$$
\begin{aligned}
\Sigma : \Phi \times S \rightharpoonup\ & s && \textit{(move to state } s \in S\textit{)} \\
\mid\ & 1 && \textit{(take the left formula of disj.)} \\
\mid\ & 2 && \textit{(take the right formula of disj.)} \\
\mid\ & * && \textit{(take the only possible move)}
\end{aligned}
$$

# Strategy semantics

$$
\begin{aligned}
(\Sigma + \Sigma')(\phi, s) &= \text{ if } (\phi, s) \in dom(\Sigma) \text{ then } \Sigma(\phi, s) \text{ else } \Sigma'(\phi, s) \\
\mathsf{SEM}(X)_\eta &= \{(X, s) \mapsto * \mid s \in \eta(X)\} \\
\mathsf{SEM}(p)_\eta &= \{(p, s) \mapsto * \mid p \text{ holds at } s\} \\
\mathsf{SEM}(\neg p)_\eta &= \{(p, s) \mapsto * \mid p \text{ does not hold at } s\} \\
\mathsf{SEM}(\phi \wedge \psi)_\eta &= \mathsf{SEM}(\phi)_\eta + \mathsf{SEM}(\psi)_\eta \\
&\quad + \{(\phi \wedge \psi, s) \mapsto * \mid (\phi, s) \in dom(\mathsf{SEM}(\phi)_\eta) \\
&\qquad\qquad \wedge (\psi, s) \in dom(\mathsf{SEM}(\psi)_\eta)\} \\
\mathsf{SEM}(\phi \vee \psi)_\eta &= \mathsf{SEM}(\phi)_\eta + \mathsf{SEM}(\psi)_\eta \\
&\quad + \{(\phi \vee \psi, s) \mapsto 1 \mid (\phi, s) \in dom(\mathsf{SEM}(\phi)_\eta)\} \\
&\quad + \{(\phi \vee \psi, s) \mapsto 2 \mid (\psi, s) \in dom(\mathsf{SEM}(\psi)_\eta)\}
\end{aligned}
$$

# Strategy semantics, cont'd

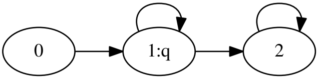
$$
\begin{aligned}
\mathsf{SEM}([a]\phi)_\eta &= \mathsf{SEM}(\phi)_\eta \\
&\quad + \{([a]\phi, s) \mapsto * \mid (\phi, s) \in dom(\mathsf{SEM}(\phi))_\eta\} \\
\mathsf{SEM}(\langle a\rangle\phi)_\eta &= \mathsf{SEM}(\phi)_\eta \\
&\quad + \{(\langle a\rangle\phi, s) \mapsto s' \mid s \xrightarrow{a} s' \wedge (\phi, s') \in dom(\mathsf{SEM}(\phi))_\eta\} \\
\mathsf{SEM}(\nu X.\phi)_\eta &= \mathsf{SEM}(\phi)_{\eta[X:=sem(\phi,\eta)]} \\
\mathsf{SEM}(\mu X.\phi)_\eta &= \mathsf{ITER}_X(\phi, \eta, \{\})
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{ITER}_X(\phi, \eta, \Sigma) = {} & \mathsf{let}\ \Sigma' := \mathsf{SEM}(\phi)_{\eta[X:=dom(\Sigma)]}\ \mathsf{in} \\
& \mathsf{if}\ \Sigma = \Sigma'\ \mathsf{then}\ \Sigma\ \mathsf{else}\ \mathsf{ITER}_X(\phi, \eta, \Sigma')
\end{aligned}
$$

# Checking certificates

- Naive approach: play according to the supposed winning strategy, and branch for all possible adversarial moves. (Easily runs into exponentially many cycles.)
- Better approach: We can reduce strategy checking to the problem of determining *emptiness of a Streett automaton*. Streett criterion in this case: for every recurrent odd priority there is a recurrent and higher even priority. Can reuse linear time algorithms for checking Streett automata (Duret-Lutz, Poitrenaud, and Couvreur 2009; Duret-Lutz 2007). Checking each SCC of the play is enough!

# A small example

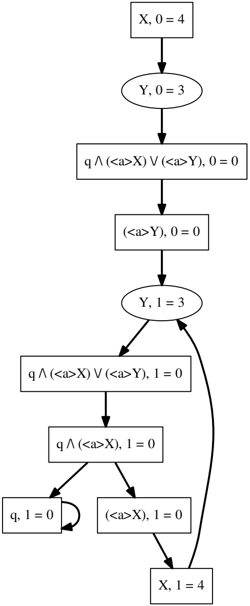



"there is a path along which $q$ holds infinitely often"

$$\nu X.\mu Y.(q \wedge \langle a \rangle X) \vee \langle a \rangle Y$$

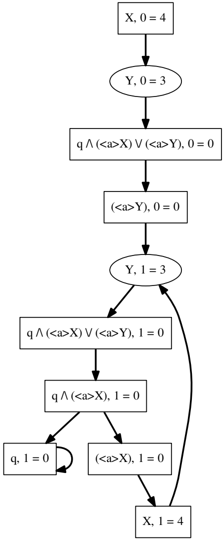
$$X \stackrel{\nu}{=} Y$$

$$Y \stackrel{\mu}{=} (q \wedge \langle a \rangle X) \vee \langle a \rangle Y$$



# Checking the example strategy

```
X 0 -> * |
X 1 -> * |
Y 0 -> * |
Y 1 -> * |
q 1 -> * |
(<a>X) 0 -> X 1 |
(<a>X) 1 -> X 1 |
(<a>Y) 0 -> Y 1 |
(<a>Y) 1 -> Y 1 |
q /\ (<a>X) 1 -> * |
(q /\ (<a>X)) \/ (<a>Y) 0 -> #2 |
(q /\ (<a>X)) \/ (<a>Y) 1 -> #1
```

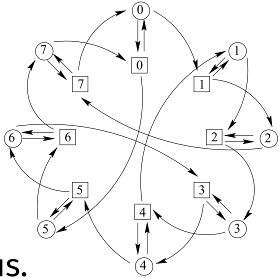


# Evaluation

- Experimental implementation `micromu` in OCaml.
- Hard to find good benchmarks for μ-calculus, created three rather synthethic problems:
  - A parity game translated into μ-calculus
  - A simple reachability property to measure overhead
  - A worst-case example for checking complexity

# Flower benchmark

**(Buhrke, Lescow, and Vöge 1999)**

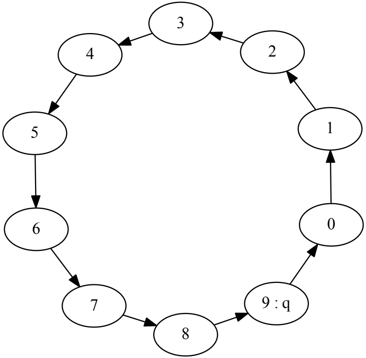



A parity game translated into μ-calculus.

With increasing problem size, solving gets exponentially harder, but checking remains polynomial.

| Problem | States | sem [s] | SEM [s] | Check [s] | Check SCC [s] |
|---|---|---|---|---|---|
| Flower 8 | 16 | 0.179 | 0.203 | 0.009 | 0.040 |
| Flower 10 | 20 | 3.166 | 1.960 | 0.071 | 0.419 |
| Flower 12 | 24 | 32.269 | 11.688 | 0.287 | 2.061 |
| Flower 14 | 28 | 320.931 | 61.733 | 1.298 | 10.829 |
| Flower 16 | 32 | 3196.043 | 326.666 | 6.131 | 58.871 |

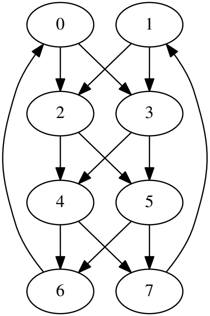
# Circle benchmark




A simple reachability property to measure overhead

| Problem | States | sem [s] | SEM [s] | Check [s] | Check SCC [s] |
|---|---|---|---|---|---|
| Circle 100 | 100 | 0.003 | 0.001 | 0.001 | 0.001 |
| Circle 1000 | 1000 | 0.109 | 0.018 | 0.005 | 0.006 |
| Circle 10000 | 10000 | 15.763 | 3.398 | 0.054 | 0.057 |
| Circle 100000 | 100000 | 2027.584 | 811.041 | 0.581 | 0.582 |

# Braid benchmark




A worst-case example for checking complexity.

Naive checking blows up, but with SCC its still fine.

| Problem | States | sem [s] | SEM [s] | Check [s] | Check SCC [s] |
|---|---|---|---|---|---|
| Braid 6 | 12 | 0.001 | 0.005 | 1.282 | 0.009 |
| Braid 8 | 16 | 0.002 | 0.003 | 31.062 | 0.013 |
| Braid 10 | 20 | 0.002 | 0.006 | 711.002 | 0.020 |
| Braid 100 | 200 | 0.663 | 0.993 | — | 3.674 |

# Perspectives

- Current implementation is pretty naive, does not use state of the art optimizations (BDDs, avoiding materialization...)
- Implementing certificate generation on existing model checkers
- Performance problems due to functional programming style
- Want to formalize certificate checking in Coq
  - ...yielding a formally verified certifying implementation of μ-calculus, usable inside a theorem prover

# Summary

- Use certificates to *split complexity* of a problem into a hard certificate generation and an easy certificate checking problem.
    - Parts can be tweaked independently.
    - Only checking needs to be formally verified.
- Leverage unverified algorithms and existing implementations in a formally verified setting.
- Benefit from fast computation and compact proofs.

**Questions?**

**Thank you.**

[1] Nils Buhrke, Helmut Lescow, and Jens Vöge. “Strategy construction in infinite games with Streett and Rabin chain winning conditions”. In: *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS ’96, Passau, Germany, March 27-29, 1996, Proceedings*. Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 1055. Lecture Notes in Computer Science. Springer, 1999, pp. 207–225. ISBN: 3-540-61042-1.

[2] Alexandre Duret-Lutz. “Contributions à l’approche automate pour la vérification de propriétés de systèmes concurrents”. PhD Thesis. Université Pierre et Marie Curie (Paris 6), July 2007.

[3] Alexandre Duret-Lutz, Denis Poitrenaud, and Jean-Michel Couvreur. "On-the-fly Emptiness Check of Transition-based Streett Automata". In: *ATVA'09*. Ed. by Zhiming Liu and Anders P. Ravn. Vol. 5799. Lecture Notes in Computer Science. Springer, 2009, pp. 213–227.

[4] EA Emerson and CS Jutla. "Tree automata, mu-calculus and determinacy". In: *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS'91)*. IEEE. 1991, pp. 368–377.

[5] K. Namjoshi. "Certifying model checkers". In: *Computer Aided Verification*. Springer, 2001, pp. 2–13.

[6] N. Shankar and M. Sorea. *Counterexample-Driven Model Checking (revisited version)*. Tech. rep. SRI-CSL-03-04. SRI International, 2003.

[7] M. Sorea. “Dubious Witnesses and Spurious Counterexamples”. UK Model Checking Days, York. 2005.