

Counterexample-guided Abstraction Refinement for Model Checking ATL

Preliminaries

Alternating-time temporal logic (ATL) [1] is designed to specify collaborative as well as adversarial interaction between different components of a distributed system. Checking the validity of an alternating-time property in an explicit model is cheap: linear in the size of the formula and the model.

Alternating transition systems (ATS) are used to model reactive components and their interactions. Given an ATS \mathcal{A} and an ATL-specification φ we can construct a weak two-player game $\mathcal{G}_{\mathcal{A}}^{\varphi}$. (A weak game is an infinite game over a finite arena with a particularly simple acceptance mechanism.) The two players represent a ‘verifier’ that tries to demonstrate that \mathcal{A} satisfies φ , and a ‘falsifier’, who tries to disprove this claim. Checking validity can be reduced to finding a winning strategy for either player in the model-checking game $\mathcal{G}_{\mathcal{A}}^{\varphi}$. This strategy also serves as a witness for the correctness of the model (if the ‘verifier’ wins) or as a counterexample (if the ‘falsifier’ wins). The size of the arena of this game is bilinear in the size of the ATS and the specification, while solving weak games takes only linear time.

However, this seemingly low complexity is misleading, since models are usually described symbolically rather than explicitly. (For the specification language used in MOCHA, for example, the ATL model-checking complexity is provably exponential in the size of the specification.)

The simplest method to model-check a symbolic model \mathcal{S} is to first transform it into an explicit representation \mathcal{A} , and then construct the model-checking game $\mathcal{G}_{\mathcal{A}}^{\varphi}$. This method is, however, intractable if the explicit state-space becomes infinite, and inefficient in most other cases.

Thus, methods for coping with large (and infinite) state-spaces are crucial for bringing model checking into practice. The common approach to treating large and infinite state-spaces in model checking is abstraction. That is, we construct a “simpler” model for which it is cheaper to check the desired property. Counterexample-guided refinement is a technique for the automatic construction of an abstraction, which is precise enough to prove (or disprove) the given property. One starts with a coarse abstraction; if the attempt to verify the abstract model fails, there is an abstract counterexample which is either genuine (that is, it has a concrete counterpart), or spurious. In the latter case, the spurious counterexample is exploited to guide the refinement of the abstract model.

Thesis

For alternating-time logics, it seems promising to postpone abstraction refinement to the model-checking game. That is, we construct a game $\mathcal{G}_{\mathcal{S}}^{\varphi}$ directly

from its small symbolic representation \mathcal{S} , and apply abstraction refinement on the game level. A counterexample-guided refinement algorithm for two-player games is presented in [2]. The composition of the reduction to a weak game and this algorithm results in a counterexample-guided abstraction refinement procedure for model checking ATL.

The algorithm from [2] is designed for turn-based two-player safety games. It is also shown how this approach can be generalized to all ω -regular winning conditions. Weak games are particularly simple ω -regular games, which form a true subset of Büchi and Co-Büchi games. (Weak conditions are merely boolean combinations of safety conditions.)

The thesis should comprise a tool for solving weak games over finite arenas, which generates a strategy for the player who wins the game. This tool should contain an abstraction-refinement-based core to solve abstract games, and a translator, which turns a symbolic model \mathcal{S} and an alternating-time system specification φ into such an abstract game. More precisely, the thesis work shall cover the following aspects:

- Understand the theory.
- Implement a component that solves abstract weak games (favoring the ‘falsifier’).
- Implement an abstraction-refinement component. The abstraction-refinement component takes as input the outcome from the game-solver applied to the abstract game. If the ‘falsifier’ wins, this component checks whether this strategy represents a genuine or spurious counterexample. If the counterexample-analysis procedure reports that the counterexample is spurious, it returns a set of refinement predicates derived from this analysis.
- Implement a component which takes a symbolic representation \mathcal{S} of an ATS, an ATL specification φ and a set of predicates, and construct the abstract weak game $\mathcal{G}_{\mathcal{S}}^{\varphi}$.

References

- [1] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Lecture Notes in Computer Science*, 1536:23–60, 1998.
- [2] T. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided control, 2003.