clintest: Efficient off-the-shelf unit testing for clingo programs

Extended abstract

Tobias Stolzmann[0000−0002−1436−0715]

University of Potsdam, Germany

Abstract. *clintest* is a test framework making it easy to write efficient unit tests for *clingo* programs. While equipping the user with numerous off-the-shelf tests for standard use cases, *clintest* is also easily extensible to support non-standard ones. In order to guarantee an efficient test execution, *clintest* monitors the test's outcome while steering the solving process.

Keywords: answer set programming · unit testing · framework.

1 Introduction

With the advent of Answer Set Programming (ASP; [6]) in industrial applications [3], the need to adapt classic software engineering methods to ASP is steadily increasing. Among these methods is unit testing, an easy but powerful approach to catch bugs in software systems. A unit test is a test conducted on a minimal unit of source code to ensure its correct implementation. It provides a carefully chosen input to the unit, executes it, and ensures that its output satisfies a set of assertions [2]. In ASP, the unit is an answer set program, the input is a set of facts, the output is the set of answer sets, and – of course – the unit is not executed but solved. Even though unit testing for ASP was previously discussed [5, 1, 7], there seems be no framework that make it easy to write tests for standard use cases, is extensible to support non-standard ones, and guarantees an efficient test execution. In this work, we present *clintest*, a test framework for answer set programs written in the dialect of *clingo* [4] that aims to be just that.

2 Design principles of *clintest*

In order to make it easy to write tests for standard use cases, *clintest* provides numerous off-the-shelf components that can be plugged together in various ways. E.g., in order to ensure that all models of a program contain the atom `a`, one needs to assemble the test `t1 = Assert(All(), Contains("a"))`. If, in addition to that, `b` should be in any model, one could to write `t2 = And(t1, Assert(Any(), Contains("b")))`. Even though compositions of standard components like that may support rather complex use cases, they will never support everything. To mitigate that, every component in *clintest* implements an abstract class that can be implemented by the user as well. `Assert` and `And` are `Test`s. `All` and `Any` are `Quantifier`s. `Contains` is an `Assertion`. There is a class `Solver` that allows to customize the solver used for a test. If a necessary component is not there, the user may just implement it and it will work along the others just fine.

To ensure an efficient test execution, *clintest* makes sure the solving process is terminated once the outcome of a test is certain. This is implemented as follows. Before searching for the next model, a `Solver` must query the `Test` for the current outcome using the `outcome`-method. If the outcome is not certain, the `Solver` must either notify the `Test` that the search was exhaustive using `on_finish` or provide the next model using `on_model`. If the outcome is certain, the solver must abort the search.

Efficiency considerations are also built into tests like `And`. `And` will not provide further models to underlying tests once their outcome is certain. This is possible because a certain outcome must (obviously) not change. Beyond that, the outcome of `And` will become certainly false once the outcome of a single underlying becomes certainly false. This is possible because $x \wedge \bot = \bot$.

3 More features and future work

It goes without saying that the off-the-shelf components presented in the previous section are not exhaustive. There are more, even though the addition of components is still ongoing. Among the most requested features is support for optimization, e.g., it should be possible to write a test like `Implies(Optimal(), Contains("a"))`. Beside the ability to review stable models, *clintest* is capable to assess the brace and cautious consequences. However, all artifacts must currently be emitted by *clingo*. In the future, we would like to add an abstract class `Model` that allows to reason about all kinds of sets of facts. Beyond adding features that are directly necessary to assemble and run certain unit tests, we try to make *clintest* as user-friendly as possible. It is already the case that each `Test` must have a method providing human-readable string representation of their current state.

clintest is freely available at `https://github.com/potassco/clintest`. The current version is 0.1.0.

References

[1] G. Amendola, T. Berei, and F. Ricca. "Testing in ASP: Revisited Language and Programming Environment". In: *Logics in Artificial Intelligence - 17th European Conference, JELIA 2021, Virtual Event, May 17-20, 2021, Proceedings.* Ed. by W. Faber et al. Vol. 12678. Lecture Notes in Computer Science. Springer, 2021, pp. 362–376. DOI: `10.1007/978-3-030-75775-5\_24`.

[2] P. Ammann and J. Offutt. *Introduction to Software Testing.* Cambridge University Press, 2008. ISBN: 978-0-521-88038-1. DOI: `10.1017/CBO9780511809163`.

[3] E. Erdem, M. Gelfond, and N. Leone. “Applications of Answer Set Programming”. In: *AI Magazine* 37.3 (2016), pp. 53–68. DOI: `10.1609/aimag.v37i3.2678`.

[4] M. Gebser et al. “Multi-shot ASP solving with clingo”. In: *Theory and Practice of Logic Programming* 19.1 (2019), pp. 27–82. URL: `http://arxiv.org/abs/1705.09811`.

[5] A. Greßler, J. Oetsch, and H. Tompits. “Harvey: A System for Random Testing in ASP”. In: *Proceedings of the Fourteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR’17)*. Ed. by M. Balduccini and T. Janhunen. Vol. 10377. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2017, pp. 229–235.

[6] V. Lifschitz. *Answer Set Programming*. Springer, 2019. ISBN: 978-3-030-24657-0. DOI: `10.1007/978-3-030-24658-7`.

[7] J. Oetsch. “Testing for ASP – ASP for Testing”. PhD thesis. Technische Universität Wien, 2022. DOI: `10.34726/hss.2022.102508`.