

DynAlloy: Upgrading Alloy with Actions (Extended Abstract)

Marcelo F. Frias^{*1}, Juan P. Galeotti², Carlos G. López Pombo², and
Nazareno M. Aguirre³

¹ Department of Computer Science, School of Exact and Natural Sciences, University of Buenos Aires, Argentina, and CONICET. mfrias@dc.uba.ar.

² Department of Computer Science, School of Exact and Natural Sciences, University of Buenos Aires, Argentina. {jgaleotti, clpombo}@dc.uba.ar.

³ Department of Computer Science, FCEFQyN - Universidad Nacional de Río Cuarto, Argentina. naguirre@dc.exa.unrc.edu.ar

Abstract. We present DynAlloy, an extension to the Alloy specification language to describe dynamic properties of systems using actions. Actions allow us to appropriately specify dynamic properties, particularly, properties regarding execution traces, in the style of dynamic logic specifications.

We extend Alloy’s syntax with a notation for partial correctness assertions, whose semantics relies on an adaptation of Dijkstra’s weakest liberal precondition. These assertions, defined in terms of actions, allow us to easily express properties regarding executions, favoring the separation of concerns between the static and dynamic aspects of a system specification.

We also extend the Alloy tool in such a way that DynAlloy specifications are also automatically analyzable, as standard Alloy specifications. We present the foundations, two case-studies, and empirical results evidencing that the analysis of DynAlloy specifications can be performed efficiently.

1 Introduction

Alloy [1, 2] is a formal specification language, which belongs to the class of the so-called model-oriented formal methods. Alloy is defined in terms of a simple relational semantics, its syntax includes constructs ubiquitous in object-oriented notations, and it features automated analysis capabilities [3]; these characteristics have made Alloy an appealing formal method.

Alloy has its roots in the Z specification language [4], and, as Z , is appropriate for describing structural properties of systems. However, in contrast with Z , Alloy has been designed with the goal of making specifications automatically analyzable.

Alloy’s representations of systems are based on abstract models. These models are defined essentially in terms of *data domains*, and *operations* between these

* Research partially funded by Antorchas foundation and project UBACYT X094.

domains. In particular, one can use data domains to specify the state space of a system or a component, and employ operations as a means for the specification of state change. Semantically, operations correspond to *predicates*, in which certain variables are assumed to be *output* variables, or, more precisely, are meant to describe the system state *after* the operation is executed. By looking into Alloy’s semantics, it is easy to verify that “output” and “after” are *intentional concepts*, i.e., the notions of output or temporal precedence are not reflected in the semantics and, therefore, understanding variables this way is just a (reasonable) convention. Variable naming conventions are a useful mechanism, which might lead to a simpler semantics of specifications. However, as we advocate in this paper, the inclusion of *actions*, with a well defined input/output semantics, in order to specify properties of executions, might provide a significant improvement to Alloy’s expressiveness and analyzability. Moreover, actions enable us to characterise properties regarding execution traces in a convenient way.

In order to see how actions might improve Alloy’s expressiveness, suppose, for instance, that we need to define the combination of certain operations describing a system. Some combinations are representable in Alloy; for instance, if we have two operations $Oper_1$ and $Oper_2$, and denote by $Oper_1; Oper_2$ and $Oper_1 + Oper_2$ the sequential composition and nondeterministic choice of these operations, respectively, then these can be easily defined in Alloy as follows:

$$Oper_1; Oper_2(x, y) = \text{some } z \mid (Oper_1(x, z) \text{ and } Oper_2(z, y)),$$

$$Oper_1 + Oper_2(x, y) = Oper_1(x, y) \text{ or } Oper_2(x, y) .$$

However, if we aim at specifying properties of executions, then it is reasonable to think that we will need to predicate at least about all terminating executions of the system. This demands some kind of iteration of operations. While it is possible to define sequential composition or nondeterministic choice, as we showed before, finite (unbounded) iteration of operations cannot be defined in Alloy.

Some effort has been put toward representing the iteration of operations, in order to analyze properties of executions in Alloy. By enriching models with the inclusion of a new signature (type) for execution traces [2], and constraints that indicate how these traces are constructed from the operations of the system, it is possible to *simulate* operation iteration. Essentially, traces are defined as being composed of all intermediate states visited along specific runs. While adding traces to specifications provides indeed a mechanism for dealing with executions (and even specifications involving execution traces can be automatically analyzed), this approach requires the specifier to explicitly take care of the definition of traces (an *ad hoc* task which depends on the properties of traces one wants to validate). Furthermore, the resulting specifications are cumbersome, since they mix together two clearly separated aspects of systems, the *static* definition of domains and operations that constitute the system, and the *dynamic* specification of traces of executions of these operations. We consider that actions, if appropriately used, constitute a better candidate for specifying assertions re-

garding the dynamics of a system (i.e., assertions regarding execution traces), leading to cleaner specifications, with clearer separation of concerns.

In order to compare these two approaches, let us suppose that we need to specify that every terminating arbitrary execution of two operations $Oper_1$ and $Oper_2$ beginning in a state satisfying formula α terminates in a state satisfying β . Using the approach presented in [2], it is necessary to provide an explicit specification of execution traces complementing the specification of the system, as follows:

1. specify the initial state as a state satisfying α ,
2. specify that every pair of consecutive states is either related by $Oper_1$ or by $Oper_2$,
3. specify that the final state satisfies β .

Using the approach we propose, based on actions, execution traces are only implicitly used. The above specification can be written in a simple and elegant way, as follows:

$$\begin{array}{c} \{\alpha\} \\ (Oper_1 + Oper_2)^* \\ \{\beta\} \end{array}$$

This states, as we required, that every terminating execution of $(Oper_1 + Oper_2)^*$ (which represents an unbounded iteration of the nondeterministic choice between $Oper_1$ and $Oper_2$) starting in a state satisfying α , ends up in a state satisfying β . This notation corresponds to the traditional and well-known notation for partial correctness assertions. Notice that no explicit reference to traces is required. Nevertheless, traces exist and are well taken care of in the semantics of actions, far from the eyes of the software engineer writing a model. It seems clear then that pursuing our task of adding actions to Alloy might indeed contribute toward the usability of the language.

As we mentioned, one of the main features of Alloy is its analyzability. The Alloy tool allows us to automatically analyze specifications by searching for counterexamples of assertions with the help of the off-the-shelf SAT solvers MChaff, ZChaff [5] and Berkmin [6]. Therefore, extending the language with actions, while still an interesting intellectual puzzle, is not important if it cannot be complemented with efficient automatic analysis. So, we modify the Alloy tool in order to deal with the analysis of Alloy specifications involving actions and execution traces assertions.

The contributions that we will present are then summarized as follows.

- We add to Alloy the possibility of defining actions and assert properties using partial correctness assertions, as a mechanism for the specification of operations. We refer to this extension of Alloy as DynAlloy.
- We present a modification of the Alloy tool in order to allow for an efficient verification of DynAlloy specifications.
- We present two case-studies for which we compare the analysis running time when using actions and traces. We conclude that efficiency increases when using actions and partial correctness assertions.

2 Conclusions and Further Work

We believe that using actions within Alloy in order to represent state change is a methodological improvement. Effectively, using actions favors a better separation of concerns, since models do not need to be reworked in order to describe the adequate notion of trace modeling the desired behavior. Using actions the problem reduces to describing how actions are to be composed. This methodological improvement is supported by empirical results evidencing that analysis can be done more efficiently than resorting to traces.

The shape of the formulas obtained during the translation of partial correctness assertions into Alloy gives us the opportunity of parallelizing their analysis process, allowing larger models to be analyzed.

Different SAT solvers react differently to the formulas result of the translation. While all of them behave satisfactorily, we can still generate different translations depending on the chosen SAT solver, in order to improve analysis time.

Finally, a new version of Alloy (Alloy 3.0) has been made recently available. All our developments will be ported to this new version as soon as its source code is released.

References

1. Jackson, D.: Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology* (2002)
2. Jackson, D., Shlyakhter, I., Sridharan, M.: A micromodularity mechanism. In: *Proceedings of the 8th European software engineering conference held together with the 9th ACM SIGSOFT international symposium on Foundations of software engineering*, Vienna, Austria, Association for the Computer Machinery, ACM Press (2001) 62–73
3. Jackson, D.: *A micromodels of software: Lightweight modelling and analysis with Alloy*. MIT Laboratory for Computer Science, Cambridge, MA. (2002)
4. Spivey, J.M.: *Understanding Z: a specification language and its formal semantics*. Cambridge University Press (1988)
5. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: *Proceedings of the 38th conference on Design automation*, Las Vegas, Nevada, United States, ACM Press (2001) 530–535
6. Goldberg, E., Novikov, Y.: BerkMin: A fast and robust sat-solver. In: *Proceedings of the conference on Design, automation and test in Europe*, IEEE Computer Society (2002) 142–149