Forge: Building a Pedagogic Solver Tool in Racket (Extended Abstract)

Tim Nelson[0000−0002−9377−9943] and
Shriram Krishnamurthi[0000−0001−5184−1975]

Brown University, Providence, RI, USA {tim_nelson,shriram}brown.edu

1 Pedagogic Context

Brown's Logic for Systems course teaches modeling and reasoning about systems via constraint solving. It relentlessly focuses on tools and applications—covering the necessary formalisms only as needed. The choice of tool used is therefore vital. Our starting point is the Alloy Analyzer [7], which is used in several textbooks and courses [1]. Alloy works especially well with students who are not instinctively inclined towards formal methods:

– it has an approachable syntax reminiscent of Java;
– it is completely automated, providing rapid feedback and allowing a user's "complexity budget" to be spent on modeling the domain and precisely expressing their goals, rather than on proof;
– it allows users to *explore* models without even writing properties, meaning that there is incremental value to creating a model of a system and the process itself can lead to eliciting properties—following the "lightweight formal methods" philosophy of Jackson and Wing [8].

Thus, while Alloy is not inherently useful in teaching students about *deductive* proof, it is very useful in getting them acquainted with specification, properties, and verification. Unfortunately, our years of experience have also revealed problems, which are exacerbated in an accessible, first formal-methods course.

2 Language Levels for Teaching Formal Methods

While Alloy has a familiar, Java-esque syntax, its underlying semantics is based on relations. For instance, Alloy makes clever use of the join operator, which is written as `.` (dot): it *looks* like a field access, but is actually a form of relational join. Early on this pun works well, and gets students comfortable writing specifications. However, at some point, every student confronts the fact that the semantics is not what they expected:[1] e.g., when an attempted join yields no tuples, the result is an empty relation rather than the error one expects from a field-access interpretation.

[1] It doesn't so much matter what they were *taught*; students form expectations based on syntactic recall.

Alloy is full of clever syntactic choices such as this. While they may be convenient to the expert modeler, over the years we have found that they are problematic for students not yet versed in the art of relational specification or in discrete mathematics. Instead, we want a series of *sub-languages* for students that grow with student instruction and accomplishment.

We adopt the solution proposed by multiple teams over the years [6, 3, 5], but most highlighted in Racket: *language levels*. That is, instead of presenting just a single language, present a family of growing sub-languages that match the learning progression.[2] This process has been extensively implemented for the book *How to Design Programs* [4] in the DrRacket programming environment.

3 The Forge Tool

This solution is realized in our pedagogic formal-methods tool, Forge[3]. Forge is built atop Racket, and supports Alloy's existing (infix) syntax and connects to the same existing solvers: we employ the same back-end [9], used by Alloy 6. Using Forge, students can begin with a language that supports their object-oriented intuitions before moving on to relational modeling, and finally to working with temporal logics.

References

1. Alloytools.org: Courses on Alloy. https://alloytools.org/citations/courses.html, accessed January 24th, 2021
2. Dennis, G., Chang, F., Jackson, D.: Modular verification of code with SAT. In: International Symposium on Software Testing and Analysis (2006)
3. du Boulay, B., O'Shea, T., Monk, J.: The black box inside the glass box. International Journal of Human-Computer Studies **51**(2), 265–277 (1999)
4. Felleisen, M., Findler, R.B., Flatt, M., Krishnamurthi, S.: How to Design Programs. MIT Press (2001), http://www.htdp.org/
5. Findler, R.B., Clements, J., Flanagan, C., Flatt, M., Krishnamurthi, S., Steckler, P., Felleisen, M.: DrScheme: A programming environment for Scheme. Journal of Functional Programming **12**(2), 159–182 (2002)
6. Holt, R.C., Wortman, D.B.: A sequence of structured subsets of PL/I. SIGCSE Bulletin **6**(1), 129–132 (1974)
7. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press, 2 edn. (2012). https://doi.org/10.5555/2141100
8. Jackson, D., Wing, J.: Lightweight formal methods. IEEE Computer (Apr 1996)
9. Macedo, N., Brunel, J., Chemouil, D., Cunha, A., Kuperberg, D.: Lightweight specification and analysis of dynamic systems with rich configurations. In: Foundations of Software Engineering (2016)

[2] It is not sufficient to focus only on the language or semantics. Critically, all *feedback*—such as error messages—must also use only vocabulary and concepts that have been introduced up to that point [5], otherwise the "epistemic closure" of the language is destroyed.

[3] Not to be confused with the Alloy-based but unrelated, defunct Forge tool [2].