Metodos Formais em Engenharia de Software (2026/27)

Software Design with Alloy & its Analyzer

Alloy is a formal specification language based on relational logic (an extension of first-order logic with relational and transitive closure operators). The structure of an Alloy model is introduced through *signatures* and *fields*, and both can be annotated with multiplicity constraints. Additional restrictions can be imposed on models through the declaration of *facts*. You can also use *predicates* and *functions* as auxiliary formulas and expressions. Besides the system specification, Alloy models can also contain *analysis commands*, either *run* commands to generate instances that obey certain properties, or *check* commands to test whether a property is guaranteed by the model, returning a counter-example otherwise.

Alloy is accompanied by an Analyzer that automatically executes these commands. To keep the problem decidable, these analyzes consider the universe to be *bounded*, which allows the model to be expanded to propositional logic and solved by off-the-shelf SAT solvers. The size of this universe is defined through *scopes* assigned to each command. When a command returns an instance (valid run commands or invalid check commands), it can be graphically visualized in the Analyzer. This visualization can be customized by configuring a *theme* to improve readability.

When writing a formal model from (informal) requirements, we need to perform both *validation* (are we building the *right thing*?) and verification (are we building the *thing right*?). This is an iterative process with several (more or less well-defined) phases, which in Alloy could be roughly implemented as follows:

1. Encode the system structure through signatures and fields with proper multiplicities
2. Write simple run commands to animate this (severely under-constrained) model
3. Customize a theme to improve the visualization of the returned instances
4. For each requirement of the system:
   (a) Write a few positive and negative test cases as run commands
   (b) Execute the commands and validate instances
   (c) Formalize the requirement and impose it in a fact
   (d) Re-run the test cases and validate the expected outcome
   (e) Fix the requirement formalization if needed
5. For each desirable property:
   (a) Formalize the property and specify it as an assertion
   (b) Write a check command to verify whether the assertion holds
   (c) If the check is invalid, inspect the counter-example:
      i. If it should not be allowed by the system, fix the imposed requirements
      ii. If it should be allowed by the system, fix the desirable property
   (d) If the check is valid, increase the scope of the analysis until sufficient confidence

The goal of this class is to revisit the domain models formalized in first-order logic in a previous class, and encode them in Alloy following the validation and verification steps defined above. That is, for each domain model X:

1. Encode the domain model in Alloy as indicated by exercise $X.1$
2. Encode and validate the requirements given in exercise $X.2$
3. Encode and verify the properties given in exercise $X.3$ as suitable Alloy commands

You can consult the file system model developed in the lectures as an example.