M.EIC037: Formal Methods For Critical Systems
2024/2025

Project 1: Alloy

Version 1

Due: Tuesday, 1^{st} of April, at 09:00am

This is a team mini-project to be done in groups of three people.

Download the accompanying file `proj1.als`, type your names and your solution in that file as indicated there. You must document all your code with comments, including preconditions, post-conditions, and frame conditions. **Poor documentation will lead to a penalty.**

You will submit your project via the Moodle assignment "MFS TP1". *Only one member of the group should submit it*, but make sure you write down the name of all team members in the model file. Your submission should contain the following files:

- The Alloy file `proj1.als` with your initial solution (16 points);
- The Alloy file `proj1-improved.als` with your improved solution (2.5 points);
- A short report (maximum four pages – limit, not a goal) (1.5 points).

On April 1st, from 14h, you will present and discuss your project with the lecturer.

The grade for the assignment can be given on an individual basis.

Please remember that all group members must be able to explain every detail of the solution.

Note: This is a modeling project based on the specifications given below.

Note: This project requires you to use the Version 6 of Alloy. Make sure you use that version.

1 Mail Application (16 points)

You are to build an Alloy model of the high-level functionality of a typical email app. The model focuses on mailboxes, messages and operations that can be performed on them. All necessary signatures are provided in file `proj1.als`. The main ones are also reported below. The file contains additional code, such as auxiliary functions and predicates and run commands, that you might find useful in completing the model — and checking it with the Alloy Analyzer.

```
enum Status {External, Fresh, Active, Purged}

abstract sig Object {}

sig Message extends Object {
  var status: lone Status
}

sig Mailbox extends Object {
  var messages: set Message
}

-- Mail application
one sig Mail {
  -- system mailboxes
  inbox: Mailbox,
  drafts: Mailbox,
  trash: Mailbox,
  sent: Mailbox,
  -- user mailboxes
  var uboxes: set Mailbox,

  var op: lone Operator -- added for tracking purposes only
}
```

You are to model static and dynamic aspects of a single email app (`Mail`) that has zero or more user-created mailboxes and 4 predefined system mailboxes:

- an inbox, for incoming messages;
- a mailbox for drafts of messages being written and not yet sent;
- a trash mailbox for deleted, but not yet purged, messages;
- a mailbox for sent messages.

Users can create new mailboxes and delete them but they cannot delete the predefined ones. Every mailbox contains a set of messages that can vary over time. Messages can be created, moved from one mailbox to another and *purged*, that is, completely removed from the system. All messages in the trash mailbox are purged when that mailbox is emptied. All messages in a user-created mailbox are purged immediately when that mailbox is deleted.

Messages have an associated time-dependent status which, when present, indicates whether the message is currently external to the system (`External` status), new to the system (`Fresh`), active (`Active`), or purged (`Purged`).

We will abstract away the details of the operations required to create a new message, receive a message from the outside environment, and send a message out. We will do that by modeling the

creation of a new message as the addition of a fresh message in the drafts mail box; the receipt of a message as the addition of an external message to the inbox; and the dispatch of a message as the move of a message from the drafts mailbox to the sent mailbox.

You are to define the operations above as state transformer predicates by specifying each operator's preconditions, postconditions, and any relevant frame conditions. You will test the model by running several scenarios, as specified below. After that, you will assess the correctness and completeness of your model by formulating and checking a number of assertions about the expected behavior of the mail app.

1.1 Modeling the operators (4 points)

Model the functionality of the mail app by defining as predicates a number of two-state operators that model the various operations offered by the app. Each operator corresponds to one of the following operations.

Create message Create a new message and put it in the drafts mailbox. Since this is a new message, in terms of the Alloy model, the message can be drawn only from the set of messages that are currently fresh.

Move message Move a given message from its current mailbox to a given, different mailbox other than the trash mailbox.

Delete message Move a given, not yet deleted, message from its current mailbox to the trash mailbox.

Send message Send a given message from the drafts inbox.

Get message Receive a new message in the inbox.

Empty trash Permanently purge all messages currently in the trash mailbox.

Create mailbox Create a new, empty mailbox and add it to the set of user-created mailboxes.

Delete mailbox Delete a given user-created mailbox. The mailbox is removed from the app's database and all of its messages are immediately purged.

Do nothing Preserve the current state of the system.

The interface of each predicate above is already provided in `proj.als`. Fill in the body of each predicate *without modifying its interface.* Provide preconditions, post-conditions and frame conditions as needed, and explain each of them in a comment.

The description of the various operations above is intentionally terse. Think carefully about each operation to ensure that your conditions are neither stronger nor weaker than necessary. In case of ambiguity or incompleteness in the specs above, ask for clarification during class or Slack (preferably in the #general channel so that all other groups can have access to any clarification — however, if you prefer, you can still ask any questions privately). If you make assumptions, then you must document them in comments.

Several of the operations above share a lot of functionality. Consider factoring that out and put it in auxiliary predicates. Make sure that you still properly annotate preconditions, postconditions, and frame conditions.

1.2 Modeling the initial-state predicate (0.5 points)

Fill in the body of the given `Init` predicate which is meant to model the initial state condition of the app. The predicate should impose the following constraints.

1. There exist no active or purged messages anywhere.
2. The system mailboxes are all distinct.
3. All mailboxes are empty.
4. There are no mailboxes except for the system mailboxes.

1.3 Modeling test scenarios (4.5 points)

The model already contains a definition of the mail app's transition relation in terms of the operators mentioned earlier. It also defines a predicate `System` representing all possible executions of the mail app. Model the following test scenarios in the provided `p1` through `p10` predicates. Use Alloy's temporal operators as needed.

These scenarios are meant as sanity checks against your definition of the various operators. They describe expected possible scenarios for the system. If the Alloy Analyzer is not able to find a satisfying run with a sufficiently high number of steps, debug your definitions of the various operators.

1. Eventually some message becomes active.
2. The inbox contains more than one message at some point.
3. The trash mailbox eventually contains messages and becomes empty some time **later**.
4. Eventually some message in the drafts mailbox moves to the sent mailbox.
5. Eventually there is a user mailbox with messages in it.
6. Eventually the inbox gets two messages in a row from outside.
7. Eventually some user mailbox gets deleted.
8. Eventually the inbox has messages. Every message in the inbox at any point is eventually removed.
9. The trash mail box is emptied of its messages eventually.
10. Eventually an external message arrives and after that nothing happens anymore.

Use the predicates defined as above in the provided corresponding `run` commands to make sure your definition of the operators and the initial state condition permit the given scenario. Each test scenario should be feasible within the given scope bounds. If Alloy finds it inconsistent with the model, amend as needed your formalization of the scenario, the operators and the init state predicate in order to eliminate the inconsistency.

If you do get an instance, make sure that you inspect it with the graphical visualizer.to double check that the trace found by the analyzer does what is expected. The reason is that, analogously to static models, it is possible for the trace to be bogus if your test or operator definitions are underconstrained.

1.4 Checking Expected Properties (5 points)

We expect the temporal properties below to hold for the mail app.

1. Every active message in the system is in one of the app's mailboxes.
2. Inactive messages are in no mailboxes at all.
3. Each of the user-created mailboxes differs from the predefined mailboxes.
4. Every active message was once external or fresh.
5. Every user-created mailbox starts empty.
6. User-created mailboxes stay in the system indefinitely or until they are deleted.
7. Every sent message is sent from the draft mailbox.
8. The app's mailboxes contain only active messages.
9. Every received message passes through the inbox.
10. A purged message is purged forever.
11. No messages in the system can ever (re)acquire `External` status.
12. The trash mailbox starts empty and stays so until a message is deleted, if any.
13. To purge an active message one must first delete the message or delete the mailbox it is in.
14. Every message in the trash mailbox had been previously deleted.
15. Every message in a user-created mailbox ultimately comes from a system mailbox.
16. A purged message that was never in the trash mailbox must have been in a user mailbox that was later deleted.

Formulate each of the properties above as Alloy assertions and check them with the Alloy Analyzer with the suggested scopes. Pay particular attention to which properties are implicitly meant to be invariant for the system (that is, true in every state) and which are implicitly meant to be true in the initial state, and formulate them accordingly.

If the Analyzer is able to falsify any of the properties, they may be invalid because of a number of reasons:

1. The property, as stated above, does not actually hold given the rest of the requirements on the system. In other words, the *customer* had wrong expectations about the behavior of the system.
2. The property should indeed hold but your formulation of it in Alloy is incorrect.
3. The property should indeed hold and is formulated correctly but your definition of the transition operators is incorrect.

4. Other combinations where more than one thing is incorrect.

It is up to you to figure out which case is which. Case 1 is unlikely unless we made a mistake in formulating the property. Nevertheless, if you are sure the English statement does not hold post a note on the #general channel on Slack, where you argue why, so that we can all discuss it. In the other cases, analyze the counterexample provided by the Analyzer to see how to modify the model and/or the property until you convince yourselves that they are both correct.

Note: Recall that Alloy can only *disprove* assertions, not prove them. The fact that Alloy cannot disprove an assertion does not necessarily mean that the assertion captures the English statement correctly (maybe all conterexample traces are longer than the set limit on the number of steps; or maybe you have written a trivially valid property, say, like `Message in Object`). This means that you need at some point to be confident in your own judgement that your model and properties are correct. The tool can help you but the final call is yours. In general, *your model will be graded manually and not just based on the Analyzer's result.*

1.5 Checking invalid properties (2 points)

We expect the properties below to be satisfied by at least one execution of the mail app.

1. It is possible for messages to stay in the inbox indefinitely.
2. A message that was removed from the inbox may later reappear there.
3. A deleted message may go back to the mailbox it was deleted from.
4. Some external messages may never be received.

One way to validate each of these properties is to express its negation as an assertion and verify that the Alloy Analyzer is able to falsify the assertion. The returned counterexample trace is an execution confirming our expectation.

For each of the properties above, formulate its negation as an Alloy assertion and check it with the Analyzer. *Also, provide the assertion in English in a comment.*

When the Analyzer is able to falsify the assertion, check that the provided counterexample trace is in fact a witness for the original (non-negated) property. If the counterexample trace is not a witness or the Analyzer is unable to falsify the assertion, try to understand once again if the problem is with the way you formulated it or with the model, and amend them as needed.

Hint: Formulate the negation of each property in English first. Then, translate that to Alloy. Be careful in formulating the negation of a temporal statement so as to not make it stronger than necessary. For instance, the negation of "it eventually stays constant" is "it repeatedly changes" (i.e., from time to time), not "it continuously changes" (i.e., each time).

2 Improved Mail App (2.5 points)

The goal of this part of the assignment is for you to improve the mail app. This can be done by refining its current behaviour and also by adding functionalities to it. As a group, you must decide on a new addition to the model and discuss it with the lecturer using your group's Slack channel for approval — this step is to make sure that the difficulty level of your suggestion is appropriate (not too simple nor too difficult). You will need to model the new behaviour and follow all needed steps to ensure that the model behaves as expected.

3 Report (1.5 points)

You should submit a short report (maximum four pages) describing and discussing the improvements you made to the model and how you validated them (first and second version). You need to report on the steps you followed to achieve your solution. You should not document code in the report – that must be done in the `.als` files.

You can, and perhaps should, include visualizations of instances of the model showing the behaviours you wished to avoid/add. Images do not count towards the maximum number of pages.

Happy Modelling!