CS:5810 Formal Methods in Software Engineering
Fall 2025

Mini Project 1

Due: Tuesday, October 14, at 11:59pm

This is a team mini-project to be done in teams 2 people.

Download the accompanying file `proj1.als`, type your names and your solution in that file as indicated there, and submit it on ICON, without renaming it. *Only one member of the team should submit*, but make sure you write down the name of both team members in the model file.

The grade for the assignment will be given on an individual basis. For every team, both students must also submit an evaluation of how well they and their teammate performed as team members. See Piazza for instructions on that. Each evaluation is confidential and will be incorporated in the calculation of the grade.

Cheating policy: Make sure to review the syllabus for details about this course's cheating policy.

Use of AI tools: Make sure you read the document on ICON explaining our policy on the use of generative AI tools in homework assignments, and the policy's reporting requirements.

Note: This is a modeling project based on the specifications given below. *The instructors might adjust them during the course of the project as a result of your enquiries.* This is intentional and is meant to mimic to some extent the interaction between customers and modelers in a real world situation.

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

1 Mail Application

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 structural and behavioral aspects of a single email app (`Mail`) that has zero or more user-created mailboxes and 4 predefined system mailboxes:

- an inbox, a mailbox storing incoming messages;
- a drafts mailbox storing drafts of messages being written and not yet sent;
- a trash mailbox storing deleted, but not yet purged, messages;
- a sent mailbox storing 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 mailbox; the receipt of a message as the addition of an external message to the inbox; and the dispatch of a message as the moving 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.

Problem 1 — Modeling the operators

Model the functionality of the mail app by defining as predicates a number two-state operators that model the various operations offered by the app, similarly to what seen in class and in the readings. 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, non yet deleted, message from its current mailbox to the trash mailbox.

Send message Send a given message from the drafts mailbox. In the model, the detectable effect of this operation is simply to move the message to the sent messages mailbox.

Get message Receive a new message in the inbox. In the model, the detectable effect of this operation is simply to add an external message to 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.

Note: 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, make suitable assumptions and document them in comments. Better yet, ask for clarifications about them on Piazza or during office hours.

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

Problem 2 — Modeling the initial-state predicate

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 (but not more).

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

Note In this problem and the next ones, think carefully about each statement made on the system and its behavior. Reflect on what the statement says, and what it does not, and discuss it within your team. If you have doubts you cannot resolve, ask for clarifications on Piazza.

Problem 3 — Modeling test scenarios

The model 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 `T1` through `T10` 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 a 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 mailbox 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 feasible within the given scope bounds. If Alloy finds it inconsistent with the model, amend your formalization of the scenario, the operators and the init state predicate as needed in order to eliminate the inconsistency.

If you do get an instance, make sure inspect it with the graphical visualizer[1] 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.

Optional (for fun, no credit)

Find and report the largest subset of tests in the `allTests` command for which Alloy can find an instance satisfying all the test predicates.

For this, you may need to replace SAT4J, the default constraint solver used by the Alloy Analyzer, with another, faster one. To do that go to the Option pull-down Menu and choose the Solver submenu. Then choose a solver listed before SAT4J. Even with the other solvers, however, be prepared to wait from tens of seconds to minutes for the `allTests` run to complete if you have a sufficiently high number of test predicates in it.

Problem 4 — Checking Expected Properties

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. Messages are sent exclusively from the drafts mailbox.
8. The app's mailboxes contain only active messages.
9. Every received message goes through the inbox.
10. Purged message are 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 delete, if any.

[1] We recommend using the Magic Layout.

13. To purge an active message one must first delete the message or delete the mailbox that contains it.

14. Every message in the trash mailbox is there because it had been previously deleted.

15. [**Optional, extra credit**] Every message in a user-created mailbox ultimately comes from a system mailbox.

16. [**Optional, extra credit**] 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 an Alloy assertion and check it with the Alloy Analyzer using the suggested scope. Pay particular attention to which properties are implicitly meant to be invariant for the system (that is, true in the initial state an every state after that) and which are not, 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 in English 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 Piazza where you argue why, so we can revise the statement for everybody as needed. 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. You do not need to report the response of the Analyzer for each assertion check.

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.*

Problem 5 — Checking possible execution scenarios

We expect the properties below to be satisfied by at least one execution of the mail app.[2]

[2]If you find some of them surprising reflect of why that is the case.

1. It is possible for messages to stay in the inbox indefinitely.
2. A message in the sent mailbox need not be there because it was sent.
3. A message that leaves the inbox may later reappear there.
4. A deleted message may go back to the mailbox it was deleted from.
5. Some external messages may never be received.
6. A deleted mailbox may reappear in the system.
7. It is possible for the system to reach a point after which none of the system mailboxes change content anymore.
8. Just deleting a message does not guarantee that it gets eventually purged.

One way to validate each of these conjectures 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 conjecture.

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 becomes constant" is "it repeatedly changes" (i.e., from time to time), not "it continuously changes" (i.e., each time).

Grading criteria

Your submission will be reviewed for:

- The quality of your model. *Keep the model simple and readable.* Submissions with complicated, lengthy, redundant, or unused constraints may be rejected.
- The correctness of the formalization of the various constraints and properties.