DCC831 Formal Methods
2021.1

Mini Project 1

Due: Friday, July 16, at 11:59pm

This assignment can be done in teams of up to 2 people. Each student is responsible for contacting other students and form a team. Individual submissions are accepted but not encouraged. In particular, no reduction of work will be granted to them.

Download the accompanying file `proj1.als`, type your name(s) and your solutions in that file as indicated there, and submit it on Moodle. For team solutions, *only one member of the team should submit*, but make sure to write down the name of both team members in the model files.

Note: This is a modelling 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 meant to mimic to some extent the interaction between customers and modelers in a real world situation.

Hint: Develop and test your Alloy model incrementally by commenting out selected parts of it.

1 Mail Application

You are to build an Alloy model of the high-level functionality of 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` and 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.

```
open util/ordering[Time] as T
sig Time {}

abstract sig ObjectStatus {}
one sig InUse, Purged extends ObjectStatus {}

abstract sig Object {
  status: ObjectStatus lone -> Time
}

sig Message extends Object {}
sig Mailbox extends Object {
  messages: Message set -> Time
}

one sig MailApp {
  inbox: Mailbox,
  drafts: Mailbox,
  trash: Mailbox,
  sent: Mailbox,
  userboxes: Mailbox set -> Time
}
```

You are to model static and dynamic aspects of a single email app (`MailApp`) that has zero or more user-created mailboxes and 4 predefined 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; and 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. Messages in the trash mailbox are purged when that mailbox is emptied. Messages in a user-created mailbox are purged immediately when that mailbox is deleted. A mailbox itself is purged when deleted.

Both messages and mailboxes have an associated time-dependent status which, when present, indicates whether that object is active (`InUse` status) or has been purged (`Purged` status).

Problem 1

Model the various functionalities of this app with two-state operators, as seen in class and in readings. The operators correspond to the following operations.

Create message Create a new message and put it in the drafts mailbox. Since this is a fresh message, in terms of the Alloy model, the message cannot be drawn from the set of messages that are currently active or purged.

Move message Move a given message from its current mailbox to a given, different mailbox.

Delete message Move a given, non yet deleted, message from its current mailbox to the trash mailbox.

Send message Send a new message. In terms of the Alloy model, the effect of this operation is simply to move a selected message from the draft mailbox to the sent messages mailbox.

Get message Receive a new message in the inbox. In the model, the effect of this operation is simply to add a message coming from *outside* of the mail app to the inbox. At the time of arrival, the message can be neither active or purged.

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.

The interface of each predicate above is already provided in `proj1.als`. Fill in the body of each predicate *without modifying its interface.* Provide preconditions, post-conditions and frame conditions, and identify them as such with comments.

Note: The description of the various operations above is intentionally succinct. 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. You are of course also welcome to ask questions and clarifications about them on Moodle.

Problem 2

Fill in the body of the given `init[t: Time]` predicate that models the initial state condition of the app. The predicate should impose the following constraints at the given time `t`.

1. There are no purged objects at all.
2. All mailboxes are empty.
3. The predefined mailboxes are all distinct.
4. The predefined mailboxes are the only active objects.
5. The app has no user-created mailboxes.

Problem 3

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. Fill in the body of the predicates `p1`, . . . , `p7` which represent properties we expect the system to satisfy *at all times.* Check whether each property holds in the system by checking the corresponding, already given, assertion. Check the assertion in a scope that is large enough for this model (at least 8).

1. Active mailboxes contain only active messages.
2. Every active message belongs to some active mailbox.
3. Every message is at most in one mailbox, that is, mailboxes do not share messages.
4. The system mailboxes are always active.
5. User-created mailboxes are different from the system mailboxes.
6. An object can be have `Purged` status only if it was once active.
7. Every sent message was once a draft message.

Properties 1–7 describe expected invariants for the system. When you write them and check the corresponding Alloy assertions, they may be invalid because of a number of reasons:

1. The property stated in English cannot actually be invariant 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 be invariant but your encoding of it in Alloy is incorrect.
3. The property should indeed be invariant and is encoded correctly but your Alloy model is incorrect.
4. Other combinations where more than one thing is incorrect.

It is up to you to figure out which case is which. In case 1, argue in a comment in plain English why the property cannot be invariant in the first place. In the other cases, 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 Alloy Analyzer for each assertion check.

Note: When checking an assertion, you should revise your model or the assertion if you get a counterexample, as that is a sure indication that something is wrong. Also, 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 (you could have written a trivially valid property, say, like `Message in Object`). This means that you need at some point to be confident on your own that your model and properties are correct. The tool can help you but the final call is yours. Finally, run some simulations on your model, as seen in class, to make sure that it is consistent. If your model is inconsistent to start with, all assertion checks will succeed trivially!

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.