CS:5810 Formal Methods in Software Engineering
Fall 2015

Mini Project 2

Due: Tuesday, November 17, by 9pm

Students can work on this project in teams of up to 2 people. Each student is responsible for contacting other students and form a team. Teams of 1 are accepted but not encouraged. In particular, no reduction of work will be granted to them.

Even for teams of 2, the grade for the project will be given on an individual basis. All student in such teams will be asked to submit an evaluation (on a form provided by the instructor) of how well they and their teammate performed as team members. Each evaluation is confidential and will be be incorporated into the calculation of the project grade.

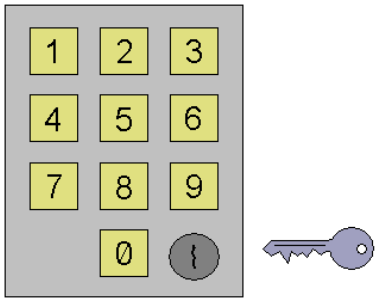
The Setting

In this project, you will use Lustre and Kind 2 to implement the control system of a simple door lock. You will then formally specify a few requirements for it, and verify them using Kind 2.

Feel free to use any temporal operator seen in class and define additional auxiliary nodes if needed.

The Door Lock

Consider a simple electronic door lock, as used in several apartment buildings. On the outside of the door there is a panel, on which one is supposed to enter a n-digit sequence, for some fixed n. This sequence is translated into a code that is then compared to one stored internally in the lock. If the two codes match, the door is unlocked for some amount of time, and one can go in. A picture of such a panel is shown below.



Every once in a while, the building superintendent changes the internal access code of the door lock. To do that he inserts a master key into the lock and then enters a new n-digit sequence. This changes the access code into the newly entered code—while keeping the door locked.

Lock Operation

The door lock checks an integer signal `Digit` and two Boolean signals `Press` and `MasterKey`, all coming from the keypad, and produces one Boolean signal `Unlocking`. The keypad will set the signal `Press` to `true` every time a key is pressed. Depending on what key is pressed, the integer stream `Digit` will get a value between 0 and 9. The signal `MasterKey` is `true` precisely when the master key is in the key lock.

When the signal `Unlocking` has value `true` the door will be unlocked and the door can be pushed open.

Modeling

You are to model the lock controller as a Lustre node called `Lock`. For simplicity, we assume that the lock works with 3-digit sequences. The model relies on two auxiliary nodes: `Keypad` and `Control`.

The node `Keypad` maps the most recent 3 digits that were entered on the keypad[1] to an integer number between 0 and 999 representing the entered code uniquely. The computed code is output via an integer stream called `Code`. A Boolean output signal `Request` is used to indicate whether there is currently a request to unlock the door or not. `Request` is true as soon as a new code has been entered whose digit sequence does not overlap with a sequence used for a previous code; it is false at all other times. For instance, if 5 3 8 1 9 0 4 are, in order, the first seven digits entered, `Request` is true only at the time 8 is entered and at the time 0 is entered, and false at all other times up to the time 4 is entered.

The node `Control` stores the current access code, if any, also as an integer value. It takes as input the output of `Keypad` and the signal `MasterKey`. Whenever there is an unlock request, it decides if the door should be unlocked or not depending on whether the provided code is the same as the current internal access code. This decision is communicated via a Boolean output signal called `Granted`. Once `Control` authorizes the unlocking of the door, `Lock` holds the door unlocked for 4 clock cycles.

Extra credit As given above, the lock system may be somewhat underspecified. To produce a more robust and realistic model you can add for extra credit further (reasonable) restrictions on the lock's behavior, relying on your understanding of how locks work in reality, with an eye on security concerns. Make sure though to *clearly state all additional assumptions* in your submitted solution.

Problem 1

1. Implement the node `Keypad` with the following interface:

```
node Keypad( Digit: int; Press: bool ) returns ( Request: bool; Code: int );
```

[1] Note that these digits are not necessarily entered at consecutive clock cycles.

2. Kind 2 allows you to check a property directly inside a system component, as opposed to inside an observer. This is usually done for properties that cannot be deduced from from a node's input-output behavior only and require instead knowledge of its internals. One such property for `Keypad` is that that if `Digit` is always in the [0..9] range, every time `Keypad` issues a request it outputs a code in the [0..999] range. Add a local Boolean variable `P` to this node and formalize this property in it.

3. Using a synchronous observer called `ReqKeypad`, verify that the following requirements hold for your `Keypad` node:

   R1 The component will not send an unlock requests before its third cycle.

   R2 Any two distinct unlock requests are separated by at least 2 clock cycles.

Problem 2

1. Implement a Lustre node `Control` with the following interface:

   ```
   node Control( Request : bool; Code : int; MasterKey : bool )
   returns ( CurrentCode : int; Granted : bool );
   ```

   The node stores internally the current access, updates it, and authorizes the unlocking of the door, through the output signal `Granted`, when the correct code has been entered. From the moment the internal access code is first set, the output stream `CurrentCode` contains the current internal access code. The internal access code is (re)set to the value of `Code` whenever `Request` and `MasterKey` are both true.

   For security, the control node will not unlock the door unless the internal access code has been set.

2. Using a synchronous observer called `ReqControl`, verify that the following requirements hold for your `Control` node:

   R1 Until the internal access code is first set, the door cannot be unlocked.

   R2 Unless the internal access code is being changed, an unlock request is granted whenever the provided code equals the internal access code.

   R3 An unlock request is granted only if the provided code equals the internal access code.

   R4 Once it has been set, the internal access code changes only when the master key is inserted.

For the requirements above to hold you may need to add suitable assumptions about the input streams. Otherwise, the requirements might have have spurious counterexamples, for instance, with illegal values for `Code`. Add these assumptions in the observer node in the form of `assert` statements as needed.

Problem 3

1. Put together the nodes `Keypad` and `Control` as appropriate in a node called `Lock`, with the following interface:

   ```
   node Lock( Digit : int; Press : bool; MasterKey : bool )
   returns ( Unlocking : bool );
   ```

   This node will hold the door unlocked for exactly 4 clock cycles once an unlock request is accepted.

2. Define a synchronous observer called `ReqLock`, with the following requirements for the `Lock` node :

   R1 The door will never be kept unlocked for more than 4 clock cycles at a time.

   R2 Once a door locks (after being unlocked) it remains locked as long as no one uses the keypad.

1 Submission Instructions

Your submission will consist of the following.

1. One Lustre source file called `proj2.lus` with *the team member names* in it and containing the 3 implementation nodes `Keypad`, `Control`, and `Lock` described above, plus the 3 synchronous observers `ReqKeypad`, `ReqControl`, and `ReqLock`, and any other auxiliary nodes you are using. Since we may use automatic checking procedures, *it is imperative that you write the interfaces of your* `Keypad`*,* `Control`*, and* `Lock` *nodes as specified.*

2. A short report within a multiline Lustre comment at the end of `proj2.lus` containing:

   (a) Any significant decisions you made to model the system based on the description above.

   (b) For *each requirement* above, a brief argument for why your formalization corresponds to the informal statement of that requirement. Make sure to document and motivate any significant assumptions your made to formalize the requirements.

You will be reviewed for:

- The quality of your implementation. *Keep the code short and simple.* Submissions with complicated, lengthy, redundant, or unused code may be rejected.
- Correctness of the formalization of the properties.
- The motivations for your decisions in your report.

One of the team members should submit the Lustre source file and the report through ICON. Make sure your report lists all the members of the team.

For team solutions, *each* team member must also individually submit a completed team evaluation form, which will be available for download before the submission deadline.