1 Hotel Key Card System

Most hotels operate a digital key card system. Upon check-in, you receive a card with your own key on it (typically a pseudorandom number). The lock for each room reads your card and opens the door if the key is correct. For many reasons, these systems are decentralized, i.e., each lock is a stand-alone (typically battery-powered) device without connection to the computer at the reception or to any other device. So how does the lock know that your key is correct?

The trick is to have two keys on your card: the old key of the previous occupant of the room (“key1”), and your own new key (“key2”). The lock always holds one key, its “current” key. When you enter your room for the first time, the lock notices that its current key is key1 on your card and recodes itself, i.e., it replaces its own current key with key2 on your card. When you enter the next time, the lock finds that its current key is equal to your key2 and opens the door without recoding itself. Your card is never modified by the lock. Eventually, a new guest with a new key enters the room, recodes the lock, and you cannot enter anymore.

Specify the behaviour of this system, explain its limitations, and prove that it guarantees some security property like “Only the owner of a room can be in the room”.

Here are some hints:

Analysis The system as sketched above has a few problems. Both in terms of safety (nothing bad can happen) and liveness (the system cannot get stuck). List those problems. Your formal specification does not need to correct them, they are present in real systems, too.

Specification Specify the state of the system as a record or tuple that holds all the required information: who is the owner of a room, what is the current key in a lock, which guest has which cards, who is in a room, etc. Specify this information as
abstractly as possible, using functions, sets, lists, etc. Use the minimal number of concepts necessary for a realistic model.

Specify the reachable states of your system inductively. For each possible event in the system, specify a rule of the form

\[ s \in \text{reachable}; \ldots \implies s' \in \text{reachable} \]

to express how the system evolves: \( s \) and \( s' \) represent the state of the system before and after the event and may be arbitrary terms. There are many possible events you can model: checkin, checkout, enter normal, enter recode, exit room, lose card, new card, find card, forge card, break lock, . . . .

Include the essential aspects and leave out the inessential ones. A specification that says everything but about which you can prove nothing is as useless as one that says nothing and about which you can prove everything.

**Verification** The informal analysis of your system should already have told you that it is impossible to prove an unconditional safety property like “Only the owner of a room can be in the room.” You will have to include some (reasonable!) preconditions before this statement becomes true. You may even have to augment your state with auxiliary components to express the preconditions.

There are many possible solutions. It is less important that you rediscover the one I have in mind (although that is perfectly acceptable). It is more important that your design is abstract, realistic, interesting, and that you are aware of its shortcomings and can defend them.

This exercise is based on and inspired by the description of a hotel key card system by Daniel Jackson [Jac06]. I recommend this book to anybody interested in abstract models of software systems.

**Abgabe/Präsentation: 26. 7. 2006**

**Literatur**