Exercise Session 10: Old Exam Question

Ingmar Dasseville *ingmar.dasseville@cs.kuleuven.be*
Thomas Vanstrydonck *thomas.vanstrydonck@cs.kuleuven.be*
Simon Marynissen *simon.marynissen@cs.kuleuven.be*

December 12, 2017

The exercise below is representative of what the exercise part of an exam could look like.

Package Management

In this exercise we will model a package system for a computer. Every package has dependencies on other packages. At each point in time, a number of packages are installed on the computer. A user will execute exactly one action at every point in time. He can choose one of 3 actions:

- install: install a package, this action is disabled when the direct dependencies for the package are not met
- remove: remove a package, this action is disabled when some other installed package depends directly on this package
- recursively install: install a package together with all its (indirect) dependencies

Installation actions can only be executed on a package which is not yet installed. Removal actions can only be executed on packages which are installed at that timepoint. Initially the system is empty.

Question A Linear Time Calculus:

A.1 the package management system as an LTC theory. Use at least the following types and predicates:

```
type Time isa int
type Package

//depends(a,b) means that package b
//depends directly on package a
depends(Package,Package)
//indicates which packages are installed at certain timepoints
installed(Time,Package)
//The following predicates represent the actions
install(Time,Package)
remove(Time,Package)
recInstall(Time,Package)
```

A.2 Now write an additional theory where we hold the sizes of the packages into account. Given the following functions `size(Package):int` and `totalDiskSpace:int`. Write a new theory, which can be merged with your theory of question A so that the new fluent `availableSpace(Time):int` is always be equal to the unused disk space. Installation/recursive installation actions can only be executed when there is sufficient disk space.

Hint: Be sure to take into account the sizes of all recursively added packages when calculating the diskspace cost of a recursive installation.

Question B Inferences:

B.1 I found a log file describing the installation history of the software on my pc. How can I use the theories from question A to check if there were any policy violations?

B.2 Consider the following LTL/CTL sentences. Are they true in the package management system A.1? Explain.

Suppose “IDP”, “ProB”, “Rodin” are packages and “Rodin” depends on “ProB”.

1. AG EF installed(“Rodin”)
2. G (¬installed(“Rodin”) U installed(“ProB”))
3. AF EX (¬installed(“Rodin”) ∧¬installed(“ProB”))

B.3 Could you check the validity of these LTL/CTL sentences for the theory you wrote in A.1 using IDP? Explain how / why not?

1. EF installed(“Rodin”)
2. G installed(“Rodin”)
3. AG EF installed(“Rodin”)

Question C Refinement:

C.1 merging the theory of question A.1 and A.2 constitute a refinement of the theory of question A.1? Explain.

C.2 In the appendix you can see an Event-B modelling of the `install` and `remove` actions. When implement the `recursive install` event, would it be possible to do this as a refinement of the `install` event? Given an implementation for a `recursive install` event. Would it be possible to define `install` as a refinement of this `recursive install` event?

```
context Packages

constants packages depends

axioms
  @pool packages ⊆ ℕ
  @depends depends ∈ packages ↔ packages
end


machine PackageManager sees Packages

variables installed

invariants
  @installed_type installed ⊆ packages

events
  event INITIALISATION
    then
      @init_installed installed ≔ ∅
  end

  event install
    any package
    where
      @package package ∈ packages
      @not_installed package ∉ installed
      @dependencies ∀ x · x ∈ packages ∧ x ↦ package ∈ depends
          ⇒ x ∈ installed
    then
      @install installed ≔ installed ∪ {package}
  end

  event remove
    any package
    where
      @package package ∈ packages
      @installed package ∈ installed
      @dependencies ∀ x · x ∈ packages ∧ package ↦ x ∈ depends
          ⇒ x ∉ installed
    then
      @remove installed ≔ installed \ {package}
  end
end
```