Course 02263 Mandatory Asssignment 1, 2020

Anne E. Haxthausen
DTU Compute

September 18, 2020

1 Practical Information

- The assignment must be solved in the registered groups. (The deadline for registration was 15 September.)
- The solution to the assignment should be made in the form of a group report.
  - Full names and study numbers of all persons should appear on the front page.
  - The report must contain RSL specifications of the problems described in the assignment and informal explanations elaborating on the RSL specification.
  - It is strongly recommended to use LaTeX to produce your report.
    A LaTeX skeleton for the report is provided: `reportskeleton.tex`. It should be used together with a modified version of the listings package: `rsllisting.sty` that is also provided. You can find more documentation about LaTeX e.g. on `http://tex.loria.fr/english/general.html`
- It is a requirement that all specifications, types and values have names as required in the assignment text. (We need that when evaluating your solution, as we plan to run an automatic test on your specifications.)
- It is a requirement that all specifications have been type checked successfully with the RAISE tools, and that your test specification has been translated successfully with the RAISE tools. Solutions that do not fulfill these requirements will not be considered.
- One of the group members must upload (a) a zip file containing all your specifications (i.e. the files *GroupingBasics.rsl*, *GroupingReq.rsl*, *GroupingEx.rsl*, and *testGroupingEx.rsl*) and (b) a pdf-file containing the group report on DTU Inside under Assignment/Opgaver **not later than** Friday 9 October 2020 at 15:00 and remember to invite all group members to the handin group and set the group name to your group number.

2 Problem

This assignment concerns the problem of dividing a collection of persons into groups in such a way that people that are friends are put in different groups.

In this assignment you are asked first to make a requirement specification for a function *divide* that given a set of persons and a "friends relation" produces a grouping of the given persons such that friends are not put in the same group, and then you are asked to give an explicit (algorithmic) definition of this function such that it satisfies the requirement specification. It should be noted that for a given problem instance there are usually many solutions to this problem and consequently many algorithms for producing a solution. You should use one such algorithm for your function definition.

The specifications should be expressed without using imperative language constructs (i.e. without using variable assignments, loops etc).

On DTU Inside, among the assignment files, you will find skeletons for some of the specifications.

2.1 Requirements Specification

1. Complete the scheme *GroupingBasics* from DTU Inside (stored in a file named *GroupingBasics.rsl*):

```
scheme GroupingBasics =
class
  type
    Person = Text,
    Relation = (Person × Person)-set,
    Group = Person-set,
    Grouping = Group-set

  value /∗ auxilliary functions ∗/
    isFriendsRelation : Relation → Bool
    isFriendsRelation(r) ≡ ...,

    isCorrectGrouping : Grouping × Person-set × Relation → Bool
    isCorrectGrouping(gs,ps,r) ≡ ...,

    areFriends : Person × Person × Relation → Bool
    areFriends(p1, p2, r) ≡ ...,
    ...
end
```

   (a) About the types: *GroupingBasics* includes four type declarations. The idea is that *Person* should be used to represent identifiers of persons, *Relation* to represent friends relations, and *Grouping* to represent groupings of persons. A grouping is modelled as a set of groups, and a group is modelled as a set of persons.

   Example
   Assume that Anna, Henrik and Frederik are friends with each other, Hans, Mette and Joachim are friends with each other, Anna and Mette are friends, and Peter has no friends. This relation can for instance be represented by the RSL value of type *Relation*:

```
{ ("Anna", "Henrik"), ("Anna", "Frederik"), ("Henrik", "Frederik"),
  ("Anna", "Mette"),
  ("Hans", "Mette"), ("Hans", "Joachim"), ("Mette", "Joachim")
}
```

   One of the possible groupings of these persons consists of three groups: one consisting of Hans and Henrik, and one consisting of Mette and Frederik, and one consisting of Joachim, Peter, and Anna. This grouping can be represented by the RSL value of type *Grouping*:

```
{ {"Hans","Henrik"},
  {"Mette","Frederik"},
  {"Joachim","Peter","Anna"}
}
```

Note that there are many other possible groupings for these persons.

(b) *GroupingBasics* should also include explicit definitions of auxiliary functions that can be applied in the specification of the *divide* function, e.g. a function, $isFriendsRelation$, that can be used to test whether a value in the type *Relation* is wellformed (i.e. represents a friends relation), and a function, $areFriends$, that can be used to test whether two persons are friends. Complete the definition of these two functions. You will in a later step be asked to complete the definition of the function $isCorrectGrouping$. Add any other auxiliary functions that you would like to use in later steps[1].

In the report you must informally explain the functionality (purpose) of the auxiliary functions (in the same style as $areFriends$ was explained above). In the report you must also informally explain what the requirements you have formalised in $isFriendsRelation(r)$ are.

2. Consider the scheme *GroupingReq* from DTU Inside.

```
scheme GroupingReq =
extend GroupingBasics with
class
  value /* requirement spec */
    divide : Person-set × Relation ~→ Grouping
    divide(ps, r) as gs
    post isCorrectGrouping(gs,ps,r)
    pre isFriendsRelation(r)
 end
```

It extends the *GroupingBasics* scheme with a formal requirement specification of a *divide* function that is intended to take a set of persons and a friends relation as arguments and to return a grouping of the given persons such that friends are not in the same group.

The pre condition is meant to express requirements to the input of the *divide* function. In this case it is chosen to be expressed in terms of the $isFriendsRelation$.

The post condition is meant to express the requirements to *divide* in the form of a relation $isCorrectGrouping$ that must hold between input (ps and r) and output (gs). Any solution to the grouping problem should satisfy these requirements, so the requirements must not be biased towards a particular solution.

In the next step you are going to define $isCorrectGrouping$.

3. In the *GroupingBasics* scheme, complete the definition of the function

   isCorrectGrouping : Grouping × Person-**set** × Relation → **Bool**

   such that $isCorrectGrouping(gs, ps, r)$ formalises the required relation between input (ps and r) and output (gs) of the *divide* function, i.e. it expresses whether gs is a legal grouping for ps and r.

   In the report you must also informally explain what the requirements you have formalised in $isCorrectGrouping(gs, ps, r)$ are.

[1] You might wait making these additions until later steps where you find out what you need.

4. Type check the *GroupingBasics* scheme. If there are any error messages, you must fix the errors.

5. Ensure that your specification *GroupingBasics* becomes within the translatable subset of SML as explained in appendix A. Check that this is the case by translating the scheme to SML using the SML translator of rsltc.

6. Type check the *GroupingReq* scheme. If there are any error messages, you must fix the errors. Note that this scheme is not translatable to SML and should not be.

2.2 Explicit Specification

1. Now complete the scheme *GroupingEx* from DTU Inside.

   scheme GroupingEx =
   extend GroupingBasics **with**
   class
     value
       divide : Person-**set** × Relation $\xrightarrow{\sim}$ Grouping
       divide(ps, r) ≡ ...
       pre isFriendsRelation(r)
   end

   It should be just like *GroupingReq* except that the *divide* function should now be defined explicitly, i.e. you have to select an algorithm that produces one of the possible groupings solution. You may need to define auxiliary functions in *GroupingEx* in order to define the *divide* function. Ensure that your spec becomes within the translatable subset of SML as explained in appendix A. In the report you must also informally explain the idea behind your algorithm.

   Hint: There is an extension to RSL according to which the **hd** operation can be applied to a non-empty set and then it will return some (arbitrary) value from that set. You may need this in some of your function definitions.

2. Type check the *GroupingEx* scheme. If there are any error messages, you must fix the errors.

3. Translate *GroupingEx* to SML. If there are any error messages, you must fix the errors.

2.3 Testing by Translation to SML

1. Complete the scheme *testGroupingEx* from DTU Inside. It should extend *GroupingEx* with some test cases that test your *divide* function and in particular test that *divide* creates output that satisfies the requirements that you stated in the requirement specification. One of the tests should create a grouping of the example shown above (under item 1 (b) in section 2.1). Your grouping might be different from the grouping shown there. Also test the *isCorrectGrouping* function and other auxiliary functions. You will be evaluated on how thoroughly you test your specification.

2. Type check *testGroupingEx*. If there are any error messages, you must fix the errors.

3. Translate *testGroupingEx* to SML. If there are any error messages, you must fix the errors.
4. Execute the test cases.
5. In the report, you must (1) include your test specification and explain your test strategy and purpose of individual test cases, (2) show the results of executing the test cases, and (3) inform whether the results are as expected.

A Rewriting expressions into a translatable form

The tool `rsltc` can translate RSL specifications into SML programs provided that they are in a certain form as explained in the UNU/IIST user guide which can be found on DTU Inside in the Tools folder of the Filesharing. Below, it is explained the required form for quantified expressions and comprehended set expressions. It is your task to look in the UNU/IIST user guide to see the required form for other syntactic categories of RSL.

A.1 Universal quantification

Universal quantification can only be translated by `rsltc` if they take one of the forms:

$\forall$ x : type_expr • x $\in$ set_expr $\Rightarrow$ logical_expr
$\forall$ x : type_expr • x $\in$ set_expr $\wedge$ logical_expr_1 $\Rightarrow$ logical_expr_2

where set_expr contains a subset of the values in type_expr.

A.2 Existential quantification

Existential quantification can only be translated if they take one of the forms:

$\exists$ x : type_expr • x $\in$ set_expr
$\exists$ x : type_expr • x $\in$ set_expr $\wedge$ logical_expr
$\exists!$ x : type_expr • x $\in$ set_expr
$\exists!$ x : type_expr • x $\in$ set_expr $\wedge$ logical_expr

A.3 Nesting quantification

A quantified expression of the form

($\forall$ x, x$'$: type_expr • x $\in$ set_expr $\wedge$ x$'$ $\in$ set_expr$'$ $\Rightarrow$ logical_expr)

cannot be translated by `rsltc`. However, the expression can be rewritten into a translatable form:

```
(∀ x : type_expr •  x ∈ set_expr ⇒
   (∀ x′ : type_expr • x′ ∈ set_expr′ ⇒
      logical_expr
   )
)
```

A.4 Comprehended set expressions

Comprehended set expressions can only be translated if they take one of the forms:

$\{ \text{expr} \mid \text{x} : \text{type\_expr} \bullet \text{x} \in \text{set\_expr} \}$
$\{ \text{expr} \mid \text{x} : \text{type\_expr} \bullet \text{x} \in \text{set\_expr} \land \text{logical\_expr} \}$