Chapter 1

Experiment DSL Formal Specification

1.1 Introduction

An experiment comprises a set of research hypotheses, each of which is a statement on the measured effects of treatments. To determine the effect of treatments, a research design defines how to apply them to experimental objects. When such application is performed during experiment execution, the effect on dependent variables is measured by the corresponding instrumentation. This generates a series of data points that are analyzed to confirm or refute the hypotheses according to statistical tests corresponding to the type of statement on the research hypotheses. The semantics of an experiment consists of the confirmation/dismissal of its hypotheses.

The overall experiment semantics is despicted in Figure 1.1. In order to test the research hypotheses, first of all an experiment must be specified. Then this specification is used to compile a Dohko Application Descriptor and to generate a R script. Next, Dohko uses the Application Descriptor to execute all its applications producing a series of data points. Finally, the data points are analyzed by the R script to confirm or refute the hypotheses specified in the experiment specification.

Dohko compilation takes an experiment specification and produces an Application Descriptor according to the hypotheses and the experimental design. An Application Descriptor contains a series of applications in which each application contains an instrument, related to the dependent variable; an execution command, related to the treatment; and an argument, related to the experimental object.

During execution, each application command is executed by Dohko using the related argument. In addition, the instrument is applyed to collect the value for the dependent variable. The outcome is a series of values in which each element corresponds to the result of the execution of an application described in Application Descriptor.

R script generation takes an experiment specification and generates a R script according to the hypotheses and the experimental design. A R script contains a series of statistical tests. Each test contains an analysis function and two arguments. The arguments contain a dependent variable name, a treatment name, and an object name, which are used filter execution results.

Finally, execution results are analyzed by the generated RScript. During analysis, each argument is applyed to a filter function to filter the results related to the argument. Next, the analysis function is applyed using two sets of data corresponding to each argument. The result of analysis is a set containing the results of each hypothesis.

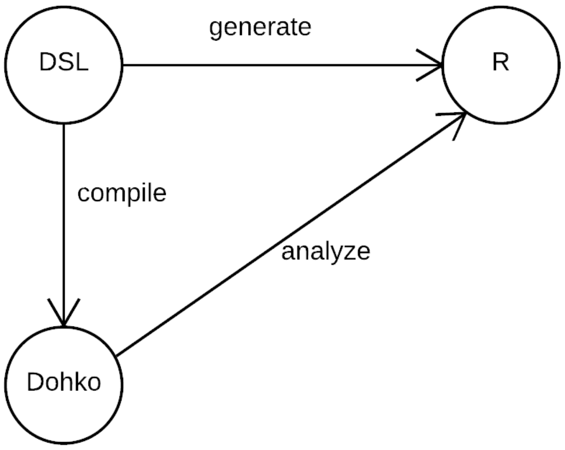

Figure 1.1: DSL semantics

1.2 DSL Model

Listing 1 Experiment.hs

```
module Experiment where
import Dohko
import RScript
import ExecutionResult

data Experiment = Experiment {researchHypotheses :: [ResearchHypothesis], design ::
 ↪  ExperimentalDesign, treatments:: [Treatment], objects:: [ExperimentalObject],
 ↪  dependentVariables :: [DependentVariable]}
data ResearchHypothesis = ResearchHypothesis {hypothesisName :: String,
 ↪  dependentVariable :: DependentVariable, treatment1 :: Treatment, treatment2 ::
 ↪  Treatment} deriving (Show, Eq, Ord)
data DependentVariable = DependentVariable {dvName :: String, instrumentCommand ::
 ↪  String} deriving (Show, Eq, Ord)
data ExperimentalDesign =  ExperimentalDesign {runs :: Int, designFunction ::
 ↪  [Treatment]->[ExperimentalObject] ->[(Treatment,ExperimentalObject)] }
data Treatment = Treatment {treatmentName :: String, treatmentCommand :: String}
 ↪  deriving (Show, Eq, Ord)
data ExperimentalObject = ExperimentalObject {objectName :: String, argument:: String}
 ↪  deriving (Show, Eq, Ord)

experiment :: Experiment -> [TestResult]
experiment exp= analyze  executionResults (generateRScript exp)
  where executionResults = map createExecutionResult (zip (generateListOfExecutions
   ↪  exp) (dohko (compileDohko exp)))

compileDohko :: Experiment -> ApplicationDescriptor
compileDohko experiment = ApplicationDescriptor (map compileApplication
 ↪  (generateListOfExecutions experiment))

compileApplication ::(Treatment,ExperimentalObject, DependentVariable) -> Application
compileApplication (treatment,object, depVariable) = Application (instrumentCommand
 ↪  depVariable) (treatmentCommand treatment) (argument object)

generateListOfExecutions :: Experiment ->
 ↪  [(Treatment,ExperimentalObject,DependentVariable)]
generateListOfExecutions experiment = concatMap (replicate (runs (design experiment)))
 ↪  (removeDuplicates treatmentApplicationDependentVariable)
  where treatmentApplicationDependentVariable = (concatMap (applyDesign (design
   ↪  experiment) (objects experiment)) (researchHypotheses experiment))
```

```haskell

applyDesign :: ExperimentalDesign -> [ExperimentalObject]-> ResearchHypothesis ->
 ↪  [(Treatment,ExperimentalObject,DependentVariable)]
applyDesign design objects hypothesis = map ((\a (b,c) -> (b,c,a)) (dependentVariable
 ↪  hypothesis)) treatmentApplication
  where treatmentApplication = (designFunction design) [treatment1 hypothesis,
   ↪  treatment2 hypothesis] objects

removeDuplicates :: (Eq a) => [a] -> [a]
removeDuplicates [] = []
removeDuplicates (x:xs) | x `elem` xs    = removeDuplicates xs
                        | otherwise      = x : removeDuplicates xs


createExecutionResult :: ((Treatment,ExperimentalObject, DependentVariable),IO
 ↪  Float)->ExecutionResult
createExecutionResult ((treat, obj, depVariable), value) = ExecutionResult (dvName
 ↪  depVariable) (treatmentName treat) (objectName obj) value

cartesianProductDesign::
 ↪  [Treatment]->[ExperimentalObject]->[(Treatment,ExperimentalObject)]
cartesianProductDesign treatments objects =[(treatment,object) | treatment <-
 ↪  treatments, object<-objects]

generateRScript :: Experiment -> RScript
generateRScript experiment = RScript (concatMap (generateHypothesisTests (objects
 ↪  experiment) (design experiment)) (researchHypotheses experiment))

generateHypothesisTests :: [ExperimentalObject] -> ExperimentalDesign ->
 ↪  ResearchHypothesis -> [AnalysisTest]
generateHypothesisTests objects design hypothesis = map (createAnalysisTest
 ↪  hypothesis) commonObjects
  where treatmentApplication = (designFunction design) [treatment1 hypothesis,
   ↪  treatment2 hypothesis] objects
        commonObjects =[object | object<-objects, elem ((treatment1
         ↪  hypothesis),object) treatmentApplication && elem ((treatment2
         ↪  hypothesis),object) treatmentApplication]

createAnalysisTest :: ResearchHypothesis -> ExperimentalObject -> AnalysisTest
createAnalysisTest  rh object = AnalysisTest wilcoxTest argument1 argument2
  where argument1 = Argument (dvName (dependentVariable rh)) (treatmentName
   ↪  (treatment1 rh)) (objectName object)
        argument2 = Argument (dvName (dependentVariable rh)) (treatmentName
         ↪  (treatment2 rh)) (objectName object)

wilcoxTest :: [ExecutionResult]->[ExecutionResult]->TestResult
wilcoxTest sample1 sample2 = TestResult "some result"
```

Listing 2 ExecutionResult.hs

```haskell
module ExecutionResult where
data ExecutionResult = ExecutionResult {resDependentVariableName :: String,
 ↪  resTreatmentName :: String, resObjectName :: String, value :: IO Float}
```

Listing 3 Dohko.hs

```haskell
module Dohko where
import System.Random
data ApplicationDescriptor = ApplicationDescriptor {applications :: [Application]}
 ↪  deriving (Show, Eq, Ord)
data Application = Application {appInstrument :: String, appCommand :: String,
 ↪  appArgument:: String} deriving (Show, Eq, Ord)

dohko :: ApplicationDescriptor -> [IO Float]
dohko applicationDescriptor =map execute (applications applicationDescriptor)
execute :: Application -> IO Float
execute application = getStdRandom (randomR (1,1000))
```

Listing 4 RScript.hs

```haskell
module RScript where
import ExecutionResult
data RScript = RScript {analysisTests :: [AnalysisTest]}
data AnalysisTest = AnalysisTest {analysisFunction ::
    [ExecutionResult]->[ExecutionResult]->TestResult, argument1 :: Argument,
    argument2::Argument}
data Argument = Argument {argDvName :: String, argTreatmentName :: String,
    argObjectName ::String} deriving (Show, Eq, Ord)
data TestResult = TestResult {testResult :: String}  deriving (Show, Eq, Ord)


analyze :: [ExecutionResult] -> RScript -> [TestResult]
analyze executionResults rscript = map ((\results test ->(analysisFunction test)
    (filterResults results (argument1 test)) (filterResults results (argument2 test)
    )) executionResults) (analysisTests rscript)

filterResults :: [ExecutionResult] -> Argument -> [ExecutionResult]
filterResults results (Argument dependentVariableName treatmentName objectName) = ([
    result | result <- results, (resDependentVariableName result) ==
    dependentVariableName && (resTreatmentName result)==treatmentName &&
    (resObjectName result)==objectName])
```

1.3 Properties

Definition 1. $\forall e : Experiment \cdot [\![e]\!] = analyze([\![compileDohko(e)]\!], \text{generateRScript}(\text{e}))$

Definition 2. $\forall ad : ApplicationDescriptor \cdot [\![ad]\!] = execute(ad)$

1. Experiment specification well-formedness: An experiment specification is well-formed if all elements used in its hypotheses are defined in the experiment specification and each hypothesis compares distinct treatments.

   (a) An experiment specification is well-formed if and only if it is of type Experiment, all treatments and dependent variables referred in its hypotheses are specified, and each hypothesis compares distinct treatments.

   (b) $\forall e : Experiment \cdot wf(e) \iff \forall (dv, t1, t2) \in researchHypotheses(e) \cdot dv \in dependentVariables(e) \wedge t1 \in treatments(e) \wedge t2 \in treatments(e) \wedge t1 \neq t2$

2. Dohko well-formedness: A Dohko Application Descriptor is well-formed if and only if it is of type Application Descriptor.

   (a) $\forall ad : ApplicationDescriptor \cdot wf(ad) = true$

3. Dohko compilation well-formedness: The result of compiling a well-formed experiment specification is a well-formed Application Descriptor

   (a) $\forall e : Experiment \cdot wf(e) \implies wf(compileDohko(e))$

4. Dohko compilation soundness: The infrastructure runs required commands to evaluate all the hypotheses according to the design of the experiment.

   (a) For each hypothesis of the experiment, all of its treatments are applied n times to the experimental objects according to the experimental design and using the corresponding instrumentation. The number of repetitions n is specified in the experimental design.

   (b) $\forall e : Experiment \cdot wf(e) \implies \forall (dv, t1, t2) \in researchHypotheses(e) \cdot$
   $\forall (t, o) \in design(\{t1, t2\}, objects(e)) \cdot$
   $\exists_{=n}(i, c, a) \in compileDohko(e) \mid i = instrument(dv) \wedge c = command(t) \wedge a = argument(o)$

5. Execution resource optimization: The infrastructure only runs commands required to evaluate the hypotheses according to the design of the experiment and nothing else.

   (a) Each application executed by the infrasctructure applies a treatment to an object according to the design of the experiment. The treatment is related to one hypothesis specified in the experiment and the instrument used to measure the dependent variable is related to the same hypothesis. In addition, the experimental object is related to the treatment according to the experimental design.

   (b) $\forall e : Experiment \cdot wf(e) \implies \forall (i, c, a) \in compileDohko(e) \cdot$
   $\exists h \in researchHypotheses(e), \{dv, t\} \subseteq h, o \in objects(e) \mid$
   $i = instrument(dv) \wedge c = command(t) \wedge a = argument(o) \wedge$
   $\wedge\ (t, o) \in design(\{treatment1(h), treatment2(h)\}, objects(e)) \wedge$
   $(\forall \{dv', t'\} \subseteq h, o' \in objects(e) \cdot$
   $i = instrument(dv') \wedge c = command(t') \wedge a = argument(o') \wedge$
   $\wedge\ (t', o') \in design(\{treatment1(h), treatment2(h)\}, objects(e)) \implies$
   $(dv, t, o) = (dv', t', o'))$

6. RScript well-formedness: A R script is well-formed if and only if it is of type RScript.

(a) $\forall r : RScript \cdot wf(r) = true$

7. RScript Generation well-formedness: The result of generating a RScript from a well-formed experiment specification is a well-formed RScript.

   (a) $\forall e : Experiment \cdot wf(e) \implies wf(generateRScript(e))$

8. Experiment soundness: Analysis is performed by using a suitable analysis function for each hypothesis and using correct arguments in the correct order. In addition, execution data are produced by executing a sound execution script compiled from the experiment specification.

   (a) For each hypothesis, the analysis function is suitable to analyze it and each argument of the analysis function corresponds to a set of data resulting of applying each treatment to an object, according to the experimental design, and measured by the corresponding instrument. In addition, the arguments are provided to the analysis function in the correct order. Moreover, execution data are produced by executing a sound execution script compiled from a well-formed experiment specification.

   (b) $\forall e : Experiment \cdot wf(e) \implies \forall (dv, t1, t2) \in researchHypotheses(e) \cdot$
   $\forall o \in design(\{t1, t2\}, objects(e)) \cdot$
   $\exists (analysisFunction, arg1, arg2) \in$
   $analyze([\![compileDohko(e)]\!], generateRScript(e)) \mid$
   $arg1 = filterResults([\![compileDohko(e)]\!], (name(dv), name(t1), name(o))) \land$
   $arg2 = filterResults([\![compileDohko(e)]\!], (name(dv), name(t2), name(o)))$