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 an execution script and to generate an analysis script. Next, the infrastructure uses the execution script to execute all its applications producing a series of data points. Finally, the data points are analyzed by the analysis script to confirm or refute the hypotheses specified in the experiment specification.

Execution script compilation takes an experiment specification and produces an execution script according to the hypotheses and the experimental design. An execution script 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 the infrastructure 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 the execution script.

Analysis script generation takes an experiment specification and generates an analysis script according to the hypotheses and the experimental design. An analysis 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 analysis script. 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.

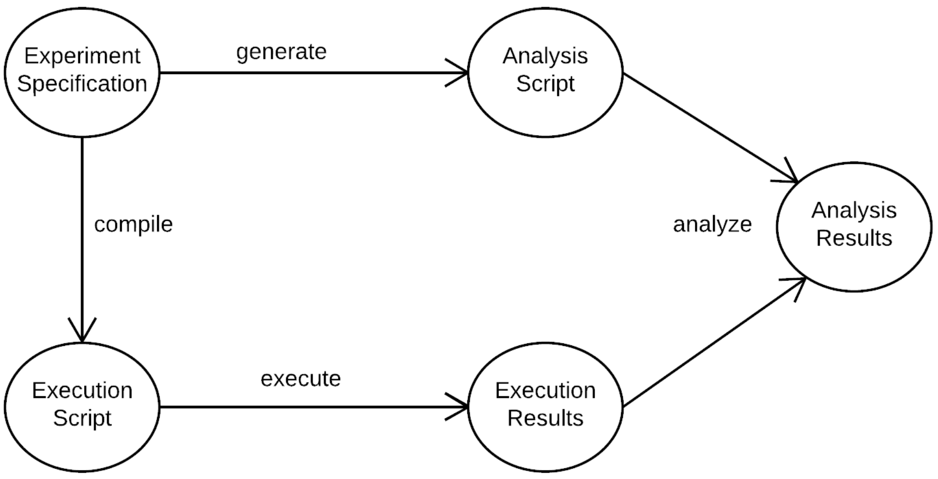

Figure 1.1: DSL semantics

1.2 DSL Model

Listing 1 Experiment.hs

```haskell
module Experiment where
import Infrastructure
import Analysis
import ExecutionResult

data Experiment = Experiment {researchHypotheses :: [ResearchHypothesis], design :: ExperimentalDesign, treatments:: [Treatment], objects:: [ExperimentalObject], dependentVariables :: [DependentVariable]}
data ResearchHypothesis = ResearchHypothesis {hypothesisName :: String, dependentVariable :: DependentVariable, treatment1 :: Treatment,
operator::String, 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 -> [HypothesisResult]
experiment exp= analyze  executionResults (generateAnalysisScript exp)
  where executionResults = map createExecutionResult (zip (generateListOfExecutions exp) (execute (compileExecutionScript exp)))

compileExecutionScript :: Experiment -> ExecutionScript
compileExecutionScript experiment = ExecutionScript (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))

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]

generateAnalysisScript :: Experiment -> AnalysisScript
generateAnalysisScript experiment = AnalysisScript (map (generateHypothesisTests
 ↪  (objects experiment) (design experiment)) (researchHypotheses experiment))

generateHypothesisTests :: [ExperimentalObject] -> ExperimentalDesign ->
 ↪  ResearchHypothesis -> HypothesisTest
generateHypothesisTests objects design hypothesis = HypothesisTest (hypothesisName
 ↪  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 (suitableFunction rh) argument1 argument2
  where argument1 = Argument (dvName (dependentVariable rh)) (treatmentName
   ↪  (treatment1 rh)) (objectName object)
        argument2 = Argument (dvName (dependentVariable rh)) (treatmentName
         ↪  (treatment2 rh)) (objectName object)

suitableFunction :: ResearchHypothesis ->
 ↪  ([ExecutionResult]->[ExecutionResult]->AnalysisResult)
suitableFunction rh = wilcoxTest
```

Listing 2 ExecutionResult.hs

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

Listing 3 Infrastructure.hs

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

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

Listing 4 Analysis.hs

```haskell
module Analysis where
import ExecutionResult
data AnalysisScript = AnalysisScript {hypothesesTests :: [HypothesisTest]}
data HypothesisTest = HypothesisTest {hypName::String, analysisTests :: [AnalysisTest]}
data AnalysisTest = AnalysisTest {analysisFunction :: [ExecutionResult]->[ExecutionResult]->AnalysisResult, argument1 :: Argument, argument2::Argument}
data Argument = Argument {argDvName :: String, argTreatmentName :: String, argObjectName ::String}
data HypothesisResult = HypothesisResult {hrHypothesisName::String, testResults::[TestResult]}  deriving (Show, Eq, Ord)
data TestResult = TestResult {trObjectName::String, analysisResult :: AnalysisResult} deriving (Show, Eq, Ord)
data AnalysisResult = AnalysisResult {result :: String}  deriving (Show, Eq, Ord)

analyze :: [ExecutionResult] -> AnalysisScript -> [HypothesisResult]
analyze results (AnalysisScript hypothesesTests) = map (analyzeHypothesis results) hypothesesTests

analyzeHypothesis :: [ExecutionResult] -> HypothesisTest -> HypothesisResult
analyzeHypothesis executionResults ht = HypothesisResult (hypName ht) (map (applyAnalysisFunction executionResults) (analysisTests ht))

applyAnalysisFunction :: [ExecutionResult] -> AnalysisTest -> TestResult
applyAnalysisFunction results test = TestResult (argObjectName (argument1 test)) ((analysisFunction test) (filterResults results (argument1 test)) (filterResults results (argument2 test) ))

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

1.3 Definitions

Definition 1. Experiment semantics:

$$\forall e : Experiment \cdot wf(e) \implies$$
$$[\![e]\!] = analyze([\![compileExecutionScript(e)]\!], generateAnalysisScript(e))$$

Definition 2. Infrastructure semantics:
$\forall es : ExecutionScript \cdot wf(es) \implies [\![es]\!] = execute(es)$

Definition 3. Experiment specification well-formedness: An experiment specification is well-formed if all elements used in its hypotheses are defined in the experiment specification with unique names and each hypothesis compares distinct treatments.

1. 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. In addition, each hypothesis, treatment, object, and dependent variable is specified with a unique name.

2. $\forall e : Experiment \cdot wf(e) \iff (\forall (dv, t1, t2) \in researchHypotheses(e) \cdot$
$dv \in dependentVariables(e) \land t1 \in treatments(e) \land t2 \in treatments(e) \land$
$t1 \neq t2) \land$
$(\forall rh1, rh2 \in researchHypotheses(e) \cdot rh1 \neq rh2 \implies$
$hypothesisName(rh1) \neq hypothesisName(rh2)) \land$
$(\forall tr1, tr2 \in treatments(e) \cdot tr1 \neq tr2 \implies$
$treatmentName(tr1) \neq treatmentName(tr2)) \land$
$(\forall o1, o2 \in objects(e) \cdot o1 \neq o2 \implies$
$objectName(o1) \neq objectName(o2)) \land$
$(\forall dv1, dv2 \in dependentVariables(e) \cdot dv1 \neq dv2 \implies$
$dvName(dv1) \neq dvName(dv2))$

Definition 4. Execution script well-formedness: An execution script is well-formed if and only if it is of type Execution Script.
$\forall es : ExecutionScript \cdot wf(es) = true$

Definition 5. Analysis script well-formedness: An analysis script is well-formed if and only if it is of type AnalysisScript.
$\forall as : AnalysisScript \cdot wf(r) = true$

1.4 Properties

1. Execution script compilation well-formedness: The result of compiling a well-formed experiment specification is a well-formed execution script.

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

2. Execution script 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) $\forall e : \mathit{Experiment} \cdot \mathit{wf}(e) \implies \forall (dv, t1, t2) \in \mathit{researchHypotheses}(e) \cdot$
   $\forall (t, o) \in \mathit{design}(\{t1, t2\}, \mathit{objects}(e)) \cdot$
   $\exists_{=n} (i, c, a) \in \mathit{compileExecutionScript}(e) \mid$
   $i = \mathit{instrument}(dv) \land c = \mathit{command}(t) \land a = \mathit{argument}(o)$

3. 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) $\forall e : \mathit{Experiment} \cdot \mathit{wf}(e) \implies \forall (i, c, a) \in \mathit{compileExecutionScript}(e) \cdot$
   $\exists h \in \mathit{researchHypotheses}(e), \{dv, t\} \subseteq h, o \in \mathit{objects}(e) \mid$
   $i = \mathit{instrument}(dv) \land c = \mathit{command}(t) \land a = \mathit{argument}(o) \land$
   $\land (t, o) \in \mathit{design}(\{\mathit{treatment}1(h), \mathit{treatment}2(h)\}, \mathit{objects}(e)) \land$
   $(\forall \{dv', t'\} \subseteq h, o' \in \mathit{objects}(e) \cdot$
   $i = \mathit{instrument}(dv') \land c = \mathit{command}(t') \land a = \mathit{argument}(o') \land$
   $\land (t', o') \in \mathit{design}(\{\mathit{treatment}1(h), \mathit{treatment}2(h)\}, \mathit{objects}(e)) \implies$
   $(dv, t, o) = (dv', t', o'))$

4. Analysis script generation well-formedness: The result of generating an analyis script from a well-formed experiment specification is a well-formed analysis script.

   (a) $\forall e : \mathit{Experiment} \cdot \mathit{wf}(e) \implies \mathit{wf}(\mathit{generateAnalysisScript}(e))$

5. 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) $\forall e : Experiment \cdot wf(e) \implies \forall hr \in [\![e]\!] \forall tr \in hr \cdot$
$tr = suitableFunction(h)(argument1, argument2)$
$where$
$argument1 = [\![compileApplication(dependentVariable(h), treatment1(h), o)]\!]$
$argument2 = [\![compileApplication(dependentVariable(h), treatment2(h), o)]\!]$
$h = b_{HypothesisResult \leftrightarrow Hypothesis} hr$
$o = b_{TestResult \leftrightarrow hObjects} tr$
$hObjects = objects(design(\{treatment1(h), treatment2(h)\}, objects(e)))$