

Chapter 1

Evaluation

1.1 Experiment DSL Formal Specification

1.1.1 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 depicted 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 applied 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 applied to a filter function to filter the results related to the argument. Next, the analysis function is applied using two sets of data corresponding to each argument. The result of analysis is a set containing the results of each hypothesis.

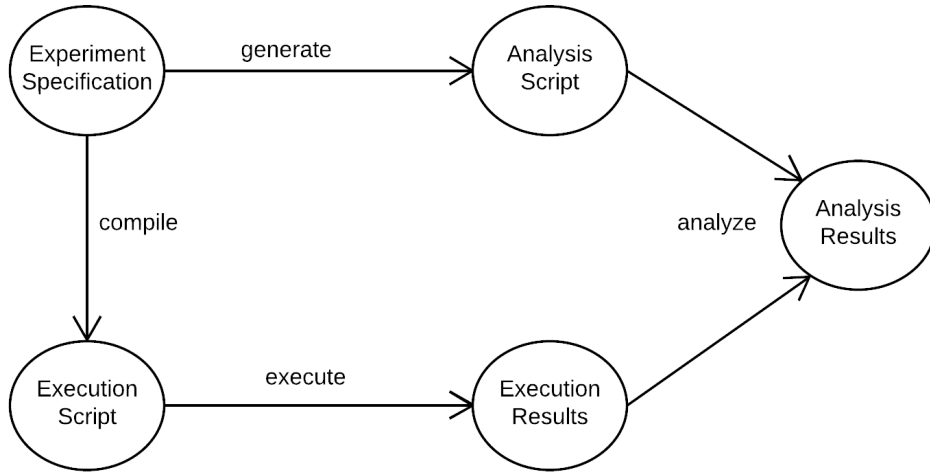


Figure 1.1: DSL semantics

1.1.2 DSL Model

DSL Model is listed in Listing 1. We represent types as records and we use `e.hypotheses` to access the field hypotheses of a given experiment $e \in E$, for example. In addition, we use overline to represent lists. For instance, hypotheses are represented by type `H`. So, \overline{H} represents a list of hypotheses. Experiments contain a list of hypotheses, a design, a list of treatments, a list of objects, and a list of dependent variables. We also assume the existence of primitive types, such as `String`, `Int`, and `Float`.

Listing 1 DSL Model

$E ::= \{hypotheses : \overline{H}, design : D, treatments : \overline{T}, objects : \overline{O}, dependentVariables : \overline{DV}\}$
 $H ::= \{name : String, dependentVariable : DV, treatment1 : T, operator : String, treatment2 : T\}$
 $D ::= \{runs : Int, designFunction : \overline{T} \times \overline{O} \rightarrow \overline{(T, O)}\}$
 $T ::= \{name : String, command : String\}$
 $O ::= \{name : String, argument : String\}$
 $DV ::= \{name : String, instrument : String\}$
 $ES ::= \{applications : \overline{A}\}$
 $A ::= \{instrument : String, command : String, argument : String\}$
 $ER ::= \{dependentVariableName : String, treatmentName : String, objectName : String, value : Float\}$
 $AS ::= \{hypothesesTests : \overline{HT}\}$
 $HT ::= \{hypothesisName : String, analysisTests : \overline{AT}\}$
 $AT ::= \{analysisFunction : \overline{ER} \times \overline{ER} \rightarrow AR, argument1 : ARG, argument2 : ARG\}$
 $ARG ::= \{dependentVariableName : String, treatmentName : String, objectName : String\}$
 $HR ::= \{hypothesisName : String, testResults : \overline{TR}\}$
 $TR ::= \{objectName : String, analysisResult : AR\}$
 $AR ::= \{result : String\}$
 $compileExecutionScript : E \rightarrow ES$
 $generateAnalysisScript : E \rightarrow AS$
 $execute : ES \rightarrow \overline{Float}$
 $generateListOfExecutions : E \rightarrow \overline{(T, O, DV)}$
 $zip : (\overline{T, O, DV}) \times \overline{Float} \rightarrow \overline{ER}$
 $compileApplication : (T, O, DV) \rightarrow A$
 $analyze : \overline{ER} \times AS \rightarrow \overline{HR}$
 $suitableFunction : H \rightarrow (\overline{ER} \times \overline{ER} \rightarrow AR)$

1.1.3 Definitions

Definition 1. Experiment semantics:

- $\forall e : E \cdot wf(e) \implies$
 $\llbracket e \rrbracket = analyze(zip(generateListOfExecutions(e), \llbracket compileExecutionScript(e) \rrbracket),$
 $generateAnalysisScript(e))$

Definition 2. Reproducibility: an experiment $e : E$ is reproducible if and only if $\llbracket e \rrbracket$ is a function.

Definition 3. Infrastructure semantics:

- $\forall es : ES \cdot wf(es) \implies \llbracket es \rrbracket = execute(es)$

Definition 4. 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.

- An experiment specification is well-formed if and only if 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.
- $\forall e : E \cdot wf(e) \iff (\forall h \in e.hypotheses \cdot$
 $h.dependentVariable \in e.dependentVariables \wedge h.treatment1 \in e.treatments \wedge$
 $h.treatment2 \in e.treatments \wedge$
 $h.treatment1 \neq h.treatment2) \wedge$
 $(\forall rh1, rh2 \in e.hypotheses \cdot rh1 \neq rh2 \implies$
 $rh1.name \neq rh2.name) \wedge$
 $(\forall tr1, tr2 \in e.treatments \cdot tr1 \neq tr2 \implies$
 $tr1.name \neq tr2.name) \wedge$
 $(\forall o1, o2 \in e.objects \cdot o1 \neq o2 \implies$
 $o1.name \neq o2.name) \wedge$
 $(\forall dv1, dv2 \in e.dependentVariables \cdot dv1 \neq dv2 \implies$
 $dv1.name \neq dv2.name)$

Definition 5. Execution script well-formedness: Every execution script is well-formed.

- $\forall es : ES \cdot wf(es) = true$

Definition 6. Analysis script well-formedness: An analysis script is well-formed if and only if each distinct hypothesis test refers to a distinct hypothesis; each analysis test compares distinct treatments but the same object and dependent variable; and, for each hypothesis, each analysis test is related to a distinct object.

- $\forall as : AS \cdot wf(as) \iff (\forall ht1, ht2 \in as.hypothesesTests \cdot$
 $ht1 \neq ht2 \implies ht1.hypothesisName \neq ht2.hypothesisName) \wedge$
 $(\forall ht \in as.hypothesesTests \cdot$
 $(\forall at \in ht \cdot$
 $at.argument1.dependentVariableName =$
 $at.argument2.dependentVariableName \wedge$
 $at.argument1.objectName = at.argument2.objectName \wedge$

$$\begin{aligned}
& at.argument1.treatmentName \neq at.argument2.treatmentName) \wedge \\
& (\forall at1, at2 \in ht \cdot at1 \neq at2 \implies \\
& at1.argument1.objectName \neq at2.argument1.objectName))
\end{aligned}$$

1.1.4 Properties

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

$$\bullet \forall e : E \cdot wf(e) \implies wf(compileExecutionScript(e))$$

2. Execution script compilation soundness: The infrastructure runs required commands to evaluate all the hypotheses according to the design of the experiment.

- 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.

$$\begin{aligned}
& \bullet \forall e : E \cdot wf(e) \implies \forall h \in e.hypotheses \cdot \\
& \quad \forall (t, o) \in e.design.designFunction(\{t1, t2\}, e.objects) \cdot \\
& \quad \exists_{=n} a \in compileExecutionScript(e).applications \mid \\
& \quad a.instrument = h.dependentVariable.instrument \wedge \\
& \quad a.command = t.command \wedge \\
& \quad a.argument = o.argument
\end{aligned}$$

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

- Each application executed by the infrastructure 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.

$$\begin{aligned}
& \bullet \forall e : E \cdot wf(e) \implies \forall a \in compileExecutionScript(e).applications \cdot \\
& \quad \exists h \in e.hypotheses, \{dv, t\} \subseteq h, o \in e.objects \mid \\
& \quad a.instrument = dv.instrument \wedge \\
& \quad a.command = t.command \wedge
\end{aligned}$$

$$\begin{aligned}
& a.argument = o.argument \wedge \\
& (t, o) \in e.design.designFunction(\{h.treatment1, h2.treatment2\}, e.objects) \\
& \wedge (\forall \{dv', t'\} \subseteq h, o' \in e.objects \cdot \\
& a.instrument = dv'.instrument \wedge \\
& a.command = t'.command \wedge \\
& a.argument = o'.argument \wedge \\
& (t', o') \in e.design.designFunction(\{h.treatment1, h.treatment2\}, e.objects) \\
& \implies (dv, t, o) = (dv', t', o'))
\end{aligned}$$

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

$$\bullet \forall e : E \cdot wf(e) \implies wf(generateAnalysisScript(e))$$

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.

- 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.

$$\bullet \forall e : E \cdot wf(e) \implies \forall hr \in \llbracket e \rrbracket \cdot$$

$$\forall tr \in hr \cdot tr = suitableFunction(h)(argument1, argument2)$$

where

$$argument1 = \llbracket compileApplication(h.dependentVariable, h.treatment1, o) \rrbracket$$

$$argument2 = \llbracket compileApplication(h.dependentVariable, h.treatment2, o) \rrbracket$$

$$h = b_{HR \leftrightarrow H} hr$$

$$o = b_{TR \leftrightarrow hObjects} tr$$

$$hObjects = e.design.designFunction(\{h.treatment1, h.treatment2\}, e.objects).objects$$

$b_{HR \leftrightarrow H}$ is a bijection between hypotheses results HR and hypotheses H. With $b_{HR \leftrightarrow H} hr$, we get the $h : H$ corresponding to a $hr : HR$.

Likewise, $b_{TR \leftrightarrow hObjects}$ is bijection between test results TR and the objects resulting of applying the design function to the treatments of a given hypothesis and the objects. We also use a helper function $objects : \overline{(T, O)} \rightarrow \overline{O}$.

Appendices

A Haskell Model

```
1 module Experiment where
2 import Infrastructure
3 import Analysis
4 import ExecutionResult
5
6 data Experiment = Experiment {researchHypotheses ::
    [ResearchHypothesis], design :: ExperimentalDesign, treatments::
    [Treatment], objects:: [ExperimentalObject], dependentVariables ::
    [DependentVariable]}
7 data ResearchHypothesis = ResearchHypothesis {hypothesisName :: String,
    dependentVariable :: DependentVariable, treatment1 :: Treatment,
8 operator::String, treatment2 :: Treatment} deriving (Show, Eq, Ord)
9 data DependentVariable = DependentVariable {dvName :: String,
    instrumentCommand :: String} deriving (Show, Eq, Ord)
10 data ExperimentalDesign = ExperimentalDesign {runs :: Int,
    designFunction :: [Treatment]->[ExperimentalObject]
    ->[(Treatment,ExperimentalObject)] }
11 data Treatment = Treatment {treatmentName :: String, treatmentCommand
    :: String} deriving (Show, Eq, Ord)
12 data ExperimentalObject = ExperimentalObject {objectName :: String,
    argument:: String} deriving (Show, Eq, Ord)
13
14 experiment :: Experiment -> [HypothesisResult]
15 experiment exp= analyze executionResults (generateAnalysisScript exp)
16 where executionResults = map createExecutionResult (zip
    (generateListOfExecutions exp) (execute (compileExecutionScript
    exp)))
17
18 compileExecutionScript :: Experiment -> ExecutionScript
19 compileExecutionScript experiment = ExecutionScript (map
    compileApplication (generateListOfExecutions experiment))
20
21 compileApplication ::(Treatment,ExperimentalObject, DependentVariable)
    -> Application
22 compileApplication (treatment,object, depVariable) = Application
    (instrumentCommand depVariable) (treatmentCommand treatment)
    (argument object)
23
24 generateListOfExecutions :: Experiment ->
    [(Treatment,ExperimentalObject,DependentVariable)]
```



```

25 generateListOfExecutions experiment = concatMap (replicate (runs
    (design experiment))) (removeDuplicates
    treatmentApplicationDependentVariable)
26 where treatmentApplicationDependentVariable = (concatMap (applyDesign
    (design experiment) (objects experiment)) (researchHypotheses
    experiment))
27
28 applyDesign :: ExperimentalDesign -> [ExperimentalObject]->
    ResearchHypothesis ->
    [(Treatment, ExperimentalObject, DependentVariable)]
29 applyDesign design objects hypothesis = map ((\a (b,c) -> (b,c,a))
    (dependentVariable hypothesis)) treatmentApplication
30 where treatmentApplication = (designFunction design) [treatment1
    hypothesis, treatment2 hypothesis] objects
31
32 removeDuplicates :: (Eq a) => [a] -> [a]
33 removeDuplicates [] = []
34 removeDuplicates (x:xs) | x `elem` xs = removeDuplicates xs
35                          | otherwise = x : removeDuplicates xs
36
37
38 createExecutionResult :: ((Treatment, ExperimentalObject,
    DependentVariable), IO Float) -> ExecutionResult
39 createExecutionResult ((treat, obj, depVariable), value) =
    ExecutionResult (dvName depVariable) (treatmentName treat)
    (objectName obj) value
40
41 cartesianProductDesign :: [Treatment] -> [ExperimentalObject] ->
    [(Treatment, ExperimentalObject)]
42 cartesianProductDesign treatments objects = [(treatment, object) |
    treatment <- treatments, object <- objects]
43
44 generateAnalysisScript :: Experiment -> AnalysisScript
45 generateAnalysisScript experiment = AnalysisScript (map
    (generateHypothesisTests (objects experiment) (design experiment))
    (researchHypotheses experiment))
46
47 generateHypothesisTests :: [ExperimentalObject] -> ExperimentalDesign
    -> ResearchHypothesis -> HypothesisTest
48 generateHypothesisTests objects design hypothesis = HypothesisTest
    (hypothesisName hypothesis) (map (createAnalysisTest hypothesis)
    commonObjects)
49 where treatmentApplication = (designFunction design) [treatment1
    hypothesis, treatment2 hypothesis] objects

```

```

50         commonObjects =[object | object<-objects, elem ((treatment1
            hypothesis),object) treatmentApplication && elem
                ((treatment2 hypothesis),object) treatmentApplication]
51
52 createAnalysisTest :: ResearchHypothesis -> ExperimentalObject ->
    AnalysisTest
53 createAnalysisTest rh object = AnalysisTest (suitableFunction rh)
    argument1 argument2
54 where argument1 = Argument (dvName (dependentVariable rh))
    (treatmentName (treatment1 rh)) (objectName object)
55         argument2 = Argument (dvName (dependentVariable rh))
    (treatmentName (treatment2 rh)) (objectName object)
56
57 suitableFunction :: ResearchHypothesis ->
    ([ExecutionResult]->[ExecutionResult]->AnalysisResult)
58 suitableFunction rh = wilcoxTest
59
60 wilcoxTest :: [ExecutionResult]->[ExecutionResult]->AnalysisResult
61 wilcoxTest sample1 sample2 = AnalysisResult "some result"

```

Listing 2: Experiment.hs

```

1 module ExecutionResult where
2 data ExecutionResult = ExecutionResult {resDependentVariableName ::
    String, resTreatmentName :: String, resObjectName :: String, value
    :: IO Float}

```

Listing 3: ExecutionResult.hs

```

1 module Infrastructure where
2 import System.Random
3 data ExecutionScript = ExecutionScript {applications :: [Application]}
    deriving (Show, Eq, Ord)
4 data Application = Application {appInstrument :: String, appCommand ::
    String, appArgument:: String} deriving (Show, Eq, Ord)
5
6 execute :: ExecutionScript -> [IO Float]
7 execute applicationDescriptor =map executeApplication (applications
    applicationDescriptor)
8 executeApplication :: Application -> IO Float
9 executeApplication application = getStdRandom (randomR (1,1000))

```

Listing 4: Infrastructure.hs

```

1 module Analysis where
2 import ExecutionResult

```

```

3 data AnalysisScript = AnalysisScript {hypothesesTests ::
    [HypothesisTest]}
4 data HypothesisTest = HypothesisTest {hypName::String, analysisTests ::
    [AnalysisTest]}
5 data AnalysisTest = AnalysisTest {analysisFunction ::
    [ExecutionResult]->[ExecutionResult]->AnalysisResult, argument1 ::
    Argument, argument2::Argument}
6 data Argument = Argument {argDvName :: String, argTreatmentName ::
    String, argObjectName ::String}
7 data HypothesisResult = HypothesisResult {hrHypothesisName::String,
    testResults::[TestResult]} deriving (Show, Eq, Ord)
8 data TestResult = TestResult {trObjectName::String, analysisResult ::
    AnalysisResult} deriving (Show, Eq, Ord)
9 data AnalysisResult = AnalysisResult {result :: String} deriving
    (Show, Eq, Ord)
10
11 analyze :: [ExecutionResult] -> AnalysisScript -> [HypothesisResult]
12 analyze results (AnalysisScript hypothesesTests) = map
    (analyzeHypothesis results) hypothesesTests
13
14 analyzeHypothesis :: [ExecutionResult] -> HypothesisTest ->
    HypothesisResult
15 analyzeHypothesis executionResults ht = HypothesisResult (hypName ht)
    (map (applyAnalysisFunction executionResults) (analysisTests ht))
16
17 applyAnalysisFunction :: [ExecutionResult] -> AnalysisTest -> TestResult
18 applyAnalysisFunction results test = TestResult (argObjectName
    (argument1 test)) ((analysisFunction test) (filterResults results
    (argument1 test)) (filterResults results (argument2 test) ))
19
20 filterResults :: [ExecutionResult] -> Argument -> [ExecutionResult]
21 filterResults results (Argument dependentVariableName treatmentName
    objectName) = ([ result | result <- results,
    (resDependentVariableName result) == dependentVariableName &&
    (resTreatmentName result)==treatmentName && (resObjectName
    result)==objectName])

```

Listing 5: Analysis.hs