# Ampersand
# Event-Condition-Action Rules

Test Plan

Yuriy Toporovskyy, Yash Sapra, Jaeden Guo

CS 4ZP6
October 30th, 2015
Fall 2015 / Winter 2016

Table 1: Revision History

| **Author** | **Date** | **Comment** |
|---|---|---|
| Yuriy Toporovskyy | 27 / 10 / 2015 | Reorganized document |
| Yuriy Toporovskyy | 27 / 10 / 2015 | Initial version - template |

# Contents

# Chapter 1

# General Information

## 1.1 Purpose

This document outlines the test plan for ECA for Ampersand, including our general approach to testing, system test cases, and a specification of methodology and constraints. This test plan specifically targets our contribution to Ampersand, namely ECA – elements of Ampersand, such as design artifact generation, will not be tested.

## 1.2 Objectives

### Preparation for testing

The primary objective of this test plan is to collect all relevant information in preperation of the actual testing process, in order to facilitate this process.

### Communication

This test plan intends to clearly communicate to all developers of ECA for Ampersand their intended role in the testing process.

### Motivation

The testing approach is motivated by constraints and requirements outlined in the Software Requirements Specification. This document seeks to clearly demonstrate this motivation.

**Environment**

This test plans outlines the resources, tools, and software required for the testing process. This includes any resources needed to perform automated testing.

**Scope**

This test plan intends to better describe the scope of our contribution, ECA, within Ampersand.

## 1.3 Definitions

**SRS**

Software Requirements Specification. Document regarding requirements, constraints, and project objectives.

**RA**

Relation algebra. The mathematical language used in ADL files to specify business rules.

**PA**

Process algebra. The mathematical language used by ECA rules to describe the action to be taken to fix violations. A "PA clause" (also written as "PAclause"), or process algebra clause, is an imperative-style language which is taken to represent *mathematical* process algebra in Ampersand. The syntax of PA clauses, in EBNF notation, is as follows:

```
PAclause ::= 'One' '(' PAclause { ',' PAclause } ')' ;
           | 'Choice' '(' GPAclause { ',' GPAclause } ')' ;
           | 'All' '(' PAclause { ',' PAclause } ')' ;
           | ('Ins' | 'Del') '(' RExpr ',' RAtom ')' ;
           | 'Nop'
           | 'Blk'
GPAclause ::= RExpr '->' PAclause ;
```

where "RExpr" represents RA expressions, and "RAtom" (RA atom) represents *atomic* RA expressions (that is, terms with no operators).

Table 1.1: Semantics of PAclause terminals

| | |
|---|---|
| One($p_0 \ldots p_n$) | Execute exactly one of $p_0 \ldots p_n$. |
| Choice($g_0 \rightarrow p_0 \ldots g_n \rightarrow p_n$) | Execute exactly one of $p_i$, such that $g_i$ is a non-empty RA term. |
| All($p_0 \ldots p_n$) | Execute all of $p_0 \ldots p_n$. |
| <Ins/Del> $(e, r)$ | Insert or delete the expression $e$ from the relation $r$. |
| Nop | Do nothing. |
| Blk | The null command, which blocks forever. |

The semantics of process algebra says that the "choice" operators (One and Choice) execute their subclauses concurrently; if *any* subclause completes, the PA clause has restored the violation.

### ECA Rule

Event-Condition-Action Rule. A rule which describes how to handle a constraint violation in a database. The syntax of ECA rules is as follows:

```
ECArule ::= 'On' ('Ins' | 'Del')
            '(' RExpr ',' RAtom ')'
            'Do' PAclause
```

### HUnit

A Haskell library for unit testing. See section 3.2.

### QuickCheck

A Haskell library for running automated, randomized tests. See section 3.2.

# Chapter 2

# Plan

## 2.1 Software Description

Ampersand is a software tool which converts a formal specification of business entities and rules, and compiles it into different design artifacts, as well as a prototype web application.

This prototype implements the business logic in the original specification, in the form of a relational database with a simple web-app front-end.

A particular class of relational database violations can be automatically restored; the algorithm for computing the code to fix these violations is called AMMBR [Joo07]. This class of violations is realized within Ampersand as ECA rules – our contribution to Ampersand will add support for ECA rules, in both the Ampersand back-end and the generated prototype.

## 2.2 Test Team

The test team which will execute the strategy outline in this document is comprised of

- Yuriy Toporovskyy
- Yash Sapra
- Jaeden Guo

## 2.3 Test Schedule

# Chapter 3

# Methods and constraints

## 3.1 Methodology

## 3.2 Test tools

### 3.2.1 Static Typing

Programming languages can be classified by many criteria, one of which is their type systems. One such classification is static versus dynamic typing. Our implementation language, Haskell, has a static type system. Types will be checked at compile-time, allow us to catch errors even before the code is run, reducing the errors that need to be found and fixed using testing techniques.

### 3.2.2 Formal verification

A part of our project deals with generating source code annotated with the proof of derivation of that source code, which will act as a correctness proof for the system. In particular, when we generate code to restore a database violation using ECA rules, then the generated code will have a proof associated with it, which details how that code was derived from the original specification given by the user.

### 3.2.3 Random Testing

Random testing allows us to easily run a very large number of tests without writing them by hand, and also has the advantage of not producing biased test cases, like a programmer is likely to do.

We will be using QuickCheck [hac] for random testing. The existing Ampersand code base using QuickCheck for testing, therefore, using QuickCheck has the added benefit of easier integration with the existing Ampersand code base.

QuickCheck allows the programmers to provide a specification of the program, in the form of properties. A property is essentially a boolean valued Haskell function of any number of arguments. QuickCheck can test that these properties hold in a large number of randomly generated cases. QuickCheck also takes great care to produce a large variety of test cases, and generally produces good code coverage. QuickCheck will be used for individualized module testing and well as provide a fair array of random tests for the combination of all modules [hac].

### 3.2.4 Unit Testing

Unit testing is comprised of feeding some data to the functions being tested and compare the actual results returned to the expected resultd. We will be using HUnit for unit testing of the new source code in Ampersand. HUnit is a library providing unit testing capabilities in Haskell. It is an adaption of JUnit to Haskell that allows you to easily create, name, group tests, and execute them.

## 3.3 Requirements

### 3.3.1 Functional requirements

The functional requirements for ECA for Ampersand are detailed in the SRS; they are also briefly summarized here. Our implementation must

**F1** provably implement the desired algorithm.

**F2** accept its input in the existing PAClause format.

**F3** produce an output compatible with the existing pipeline.

**F4** annotated generated code with proofs of correctness or derivations, where appropriate.

**F5** automatically fix database violations in the mock database of the prototype.

**F6** not introduce appreciable performance degradation.

**F7** provide diagnostic information about the algorithm to the user, if the user asks for such information.

### 3.3.2 Non-Functional requirements

The functional requirements for ECA for Ampersand are detailed in the SRS; they are also briefly summarized here. Our implementation must

**N1** produce output which will be easily understood by the typical user, such as a requirements engineer, and will not be misleading or confusing.

**N2** be composed of easily maintainable, well documented code.

**N3** compile and run in the environment currently used to develop Ampersand.

**N4** be a pure function; it should not have side effects.

## 3.4 Data recording

## 3.5 Constraints

## 3.6 Evaluation

# Chapter 4

# System Test Descriptions

Many test cases use domain specific language to indicate inputs and outputs, for both clarity and brevity. This includes the syntax and semantics of ECA rules and Abstract SQL. For the full syntax and semantics of these , as well as related definitions, see section 1.3.

## T1 ECA rule executing "All" subclauses

| | |
|---|---|
| **Test type** | Dynamic, white box, automated |
| **Schedule** | Term 2 |
| **Requirements** | F1 |

### Input

The input is an ECA rule of the form:

$$\begin{array}{l}
\textbf{On Ins}(\Delta, r_0) \;\; \textbf{Do All}(\textbf{Ins}(e_1, r_1), \textbf{Ins}(e_2, r_2)) \\
\text{where} \;\; \begin{array}{rl} r_0, r_1, r_2 & := \text{Atomic Relation} \\ e_1, e_2, \Delta & := \text{Expression} \end{array}
\end{array}$$

### Output

The output is an abstract SQL function of the form:

```
f (delta, r_0):
  INSERT INTO <r_1_table> VALUES <e_1_query>;
  INSERT INTO <r_2_table> VALUES <e_2_query>;
```

### Description

ECA rules of the input format are generated using QuickCheck, converted to abstract SQL, then compared against the expected output format using HUnit.

## T2 ECA rule executing "Choice" subclauses

| | |
|---|---|
| **Test type** | Dynamic, white box, automated |
| **Schedule** | Term 2 |
| **Requirements** | F1 |

### Input

The input is an ECA rule of the form:

$$\begin{array}{lll} \multicolumn{3}{l}{\textbf{On Ins}(\Delta, r_0)\ \ \textbf{Do Choice}(p_0, p_1)} \\ \text{where} & r_0 & := \text{Atomic Relation} \\ & \Delta & := \text{Expression} \\ & p_0, p_1 & := \text{PA Clause} \end{array}$$

### Output

The output is an abstract SQL function of the form:

```
no idea
```

### Description

ECA rules of the input format are generated using QuickCheck, converted to abstract SQL, then compared against the expected output format using HUnit.

## T3 ECA rule with empty PA clause

| | |
|---|---|
| **Test type** | Dynamic, white box, automated |
| **Schedule** | Term 2 |
| **Requirements** | F1 |

**Input**

The input is an ECA rule of the form:

$$\textbf{On}\ \{\textbf{Ins/Del}\}(\Delta, r_0)\ \ \textbf{Do Nop}$$
$$\begin{array}{lll} \text{where} & C_0, C_1 & := \text{Concept} \\ & r_0 & := \text{Atomic Relation} \end{array}$$

**Output**

The output is the empty abstract SQL statement; that is, a statement of the form:

```
f (delta, r_0): {} \\ Do nothing
```

**Input**

ECA rules of the input format are generated using QuickCheck, converted to abstract SQL, then compared against the expected output format using HUnit.

# T4 ECA rule inserting into Identity relation

| | |
|---|---|
| **Test type** | Dynamic, white box, automated |
| **Schedule** | Term 2 |
| **Requirements** | F1 |

**Input**

The input is an ECA rule of the form:

$$\textbf{On}\ \{\textbf{Ins/Del}\}(\Delta, r_0)\ \ \textbf{Do}\ \{\textbf{Del/Ins}\}(e_0, \mathbb{I}_{C_0})$$
$$\begin{array}{lll} \text{where} & C_0 & := \text{Concept} \\ & r_0 & := \text{Atomic Relation} \\ & e_0 & := \text{Expression} \end{array}$$

**Output**

The output is an abstract SQL statement of the form:

```
f (delta, r_0):
  {INSERT INTO/DELETE FROM} <C_0_Population> VALUES <e_0_query>;
```

where `C_0_Population` is the table corresponding to $\mathbb{I}_{C_0}$.

**Description**

ECA rules of the input format are generated using QuickCheck, converted to abstract SQL, then compared against the expected output format using HUnit.

# T5 EFA System Compatibility

| | |
|---|---|
| **Test type** | Functional/Black box/ |
| **Schedule** | Dec 2015 |
| **Requirements** | F3 |

**Input**

ADL File Input 1:
Based on the Rule: Only members who have relevant experience may apply for this job
Using Sets: JOBS-AVAIL, APPLICANTS, EMPLOYEES-WITH-RELEVANT-EXPERIENCE
With ECA rules:
ADL Files Input 2:
Based on the Rule: Only members who have relevant experience may apply for this job
Using Sets: JOBS-AVAIL, APPLICANTS, EMPLOYEES-WITH-RELEVANT-EXPERIENCE
With ECA rules: APPLICANTS must be a member of both EMPLOYEES-WITH-RELEVANT-EXPERIENCE AND have a relation to (i.e. applied for) JOBS-AVAIL

**Output**

EFA User Output for Input 1:
Reading <file>.adl..
Generating..
Rules Done..
Sets Done..
No Errors
No Violations
EFA User Output for Input 2:
Reading <file>.adl..
Generating..
Rules Done..
Sets Done..
ECA Rules Done..
No Errors

No Violations

**Description**

Two different version of the same script is given as input, the first is without ECA rules the second is with ECA rules that this project adds. Both of these scripts should pass through the Ampersand generator without causing errors or violations. If the second script which contains ECA rules successfully passes through each part of Ampersand then the new additions generated by EFA is compatible with the old Ampersand system.

# T6 EFA Pure Function

| | |
|---|---|
| **Test type** | Dec 2015 |
| **Schedule** | F4 |
| **Requirements** | Two conditions must hold for a function to be considered a pure function 1. The fu |

**Input**

The input is an ECA rule of the form:

ECA = {Condition that triggers action: Insertion of <new field into table>,
Change that initiated trigger: Insertion of <e2> into current data scheme,
Action to be done : ∀
{(take the difference of the previous result of the expressions
(take result of intersection of the returned result
(composition of the result
(Simple declaration of the result of the conversion
(convert expression e2 using the identity relation
of e1
)
)
with e1)
with e1)
with e1 where e1 is another expression)}
}

**Output**

asdfasdf

## T7 EFA User Feed Back

| | |
|---|---|
| **Test type** | Functional |
| **Schedule** | January 2016 |
| **Requirements** | F6,N1 |

**Brief Explanation Concerning Context**

Only those who are qualified can be cast into roles, the actor must have relevant experience.

**Input**

The input shall be ECA rules specifying invariants that must be maintained throughout the program.

Example concerning how roles are cast for a theater performance:

User Input:

RULE "who's cast in roles" : cast; instantiates —- qualifies;comprises MEANING "an Actor may appear in a Performance of the Play only if the Actor is skilled for a Role that the Play comprises "

EFA INPUT:

ECA = {
    ECA Trigger: if cast member does not have relevant experience or enough experience
    ECA Violation: lack of experience
    ECA Action: Remove actors without enough relevant experience
  }

**Output**

<u>User feed back if file has no errors:</u>
Reading file theatreCasting.adl..
Done.
Done.

User feed back if file has errors:
Reading file theater.adl
Error(s) found:
Type error, cannot match:
the concept "Role" (Tgt of qualifies)
and concept "Performance" (Src of instantiates)
if you think there is no type error, add an order between concepts "Role" and "Performance".
Error at symbol () in file theater.adl at line 26 : 44
================================
No declarations match the relation: actor
Error at symbol () in file theater.adl at line 26 : 62
================================
ECA Rule Violation:
Error at Rule declaration and structure in file theater.adl at line 33 : 41
Error: Structure and Meaning do not match

# T8 EFA Code Walk-through

| | |
|---|---|
| **Test type** | Non-functional |
| **Schedule** | January 2016 |
| **Requirements** | N2 |

**Brief Explanation**

Input and output are not available for this test, as it requires each member of the design team to walk through the code line by line to check if it is easy to understand by another programmer and well documented. If it is easy to read and understand but not only the individual who wrote it but those around them, then it should be easy to maintain.

# T9 Degradation Test

| | |
|---|---|
| **Test type** | Non-functional |
| **Schedule** | First Week of February 2016 |
| **Requirements** | F6 |

### Brief Explanation

Degradation shall be measured through a comparison of Ampersand before EFA and Ampersand after EFA. The amount of time Ampersand takes to compile a prototype will measure performance degradation; if Ampersand takes substantially longer to compile after the addition of EFA then it is an appreciable difference. A Linux distribution will be used to time multiple trials of Ampersand with and without EFA. The test materials used will be taken off of the Ampersand-models github.

# T10 EFA Annotated Code

| | |
|---|---|
| **Test type** | Non-functional |
| **Schedule** | January 2016 |
| **Requirements** | N4 |

### Description

There is no input, however there will be annotations available for output, this is used for debugging purposes and the user will never see this.

# Bibliography


[hac] QuickCheck: Automatic testing of Haskell programs. `https://hackage.haskell.org/package/QuickCheck`. Accessed: 2015-10-29.

[Joo07] Stef Joosten. AMMBR: A Method to Maintain Business Rules. 2007.