

Ampersand Event-Condition-Action Rules

Test Plan

Yuriy Toporovskyy, Yash Sapra, Jaeden Guo

Table 1: Revision History

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

Contents

1	General Information	1
1.1	Purpose	1
1.2	Objectives	1
1.3	Definitions	2
2	Plan	5
2.1	Software Description	5
2.2	Test Team	5
2.3	Test Schedule	6
3	Methods and constraints	7
3.1	Methodology	7
3.2	Test tools	8
3.2.1	Static Typing	8
3.2.2	Formal verification	8
3.2.3	Random Testing	8
3.2.4	Unit Testing	9
3.3	Requirements	9
3.3.1	Functional requirements	9
3.3.2	Non-Functional requirements	9
3.4	Data recording	10
3.5	Constraints	10
3.6	Evaluation	10
4	System Test Descriptions	11
T1	ECA rule executing “All” subclauses	11
T2	ECA rule executing “Choice” subclauses	12
T3	ECA rule with empty PA clause	12
T4	ECA rule inserting into Identity relation	13
T5	ASQL is valid	14
T6	EFA System Compatibility	14
T7	EFA Pure Function	16
T8	EFA User Feed Back	16
T9	EFA Code Walk-through	18

T10	Degradation Test	18
T11	EFA Annotated Code	19
T12	EFA domain	19

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 is centered around our contribution to Ampersand and ignores other elements of the Ampersand process such as the artifacts that are generated.

1.2 Objectives

Preparation for testing

The primary objective of this test plan is to gather all relevant information that could aid in creating effective tests for EFA. EFA in this regard is modularized and tested as an individual component before it is tested within the Ampersand system.

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 based on the constraints and requirements presented in the Software Requirements Specification (i.e. SRS). This document focuses on how the

functional and non-functional requirements provided in the SRS will be tested for this project.

Environment

This test plan 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 the Ampersand system.

1.3 Definitions

SRS

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

EBNF

Extended BackusNaur Form. A notation for specifying the syntax of languages, see [14910].

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 represents the *mathematical* process which Ampersand uses. 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 “RExp” represents RA expressions, and “RAtom” (RA atom) represents *atomic* RA expressions (i.e. terms with no operators).

Table 1.1: Semantics of PAclause terminals

One($p_0 \dots p_n$)	Execute exactly one of $p_0 \dots p_n$.
Choice($g_0 \rightarrow p_0 \dots g_n \rightarrow p_n$)	Execute exactly one of p_i , such that g_i is a non-empty RA term.
All($p_0 \dots p_n$)	Execute all of $p_0 \dots p_n$.
<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 (e.g. One and Choice) may execute any one of their subclauses; if *any* of the subclauses can be completed, the PA clause has restored the violation. One choice may be considered better in some ways, for example, different alternatives could have vastly different execution costs. For the purpose of this document, however, we will make the simplest “choice” possible, which generally means an arbitrary choice.

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.

Sentinel

A test server accessible through the Ampersand website ([sena]) which executes a set of randomly generated tests on Ampersand on a daily basis, see [senb] for details.

Chapter 2

Plan

2.1 Software Description

Ampersand is a software tool that converts the formal specifications of business entities and rules, compiles it into various design artifacts (e.g. latex documents), and produces a prototype web application.

The business prototype implements business logic according to specifications provided by the user, it uses the entities and relationships that the user provides to form a relational database with a simple web application front-end.

ECA for Ampersand focuses on automatically restoring a particular class of database violations using an algorithm called AMMBR [Joo07]. This class of violations is realized within Ampersand as ECA rules – our contribution to Ampersand will add support for ECA rules, that will affect Ampersand’s back-end as well as the generated prototype it provides.

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

Dates	Tests to be performed
12-20-2015 - 01-07-2016	Unit testing, black box testing, white box testing with heavy use of QuickCheck
01-08-2016 - 01-20-2016	System testing EFA with Ampersand
01-20-2016-02-01-2016	Testing for abnormality and performance.

Note: More details for specific test will be provided in the future.

3.2 Test tools

Various tools are used by the existing Ampersand developers to test Ampersand. In order to integrate best with Ampersand, our code will adopt their 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 RA 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 results. 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

Requirements are the main motivator for tests and test methodology – testing should help ensure that requirements are met. To this end, our requirements are briefly summarized and labelled below.

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 ECARule 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

Not Available at this time.

3.5 Constraints

Not applicable at this time.

3.6 Evaluation

Due to the early stage of development, this is not available at this time.

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, black box, automated
Schedule	Term 2
Requirements	F1

Input

The input is an ECA rule of the form:

On $\text{Ins}(\Delta, r_0)$ **Do** **All**($\text{Ins}(e_1, r_1), \text{Ins}(e_2, r_2)$)
where $r_0, r_1, r_2 := \text{Atomic Relation}$
 $e_1, e_2, \Delta := \text{Expression}$

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, black box, automated
Schedule	Term 2
Requirements	F1

Input

The input is an ECA rule of the form:

On **Ins**(Δ, r_0) **Do Choice**(p_0, p_1)
where r_0 := Atomic Relation
 Δ := Expression
 p_0, p_1 := PA Clause

Output

The output is an abstract SQL function of the form:

f (delta, r_0):
 <p_i_statement>

where $i \in \{0, 1\}$.

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, black box, automated
Schedule	Term 2
Requirements	F1

Input

The input is an ECA rule of the form:

On {Ins/Del}(Δ, r_0) **Do** Nop
where C_0, C_1 := Concept
 r_0 := Atomic Relation

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, black box, automated
Schedule Term 2
Requirements F1

Input

The input is an ECA rule of the form:

On {Ins/Del}(Δ, r_0) **Do** {Del/Ins}(e_0, \mathbb{I}_{C_0})
where C_0 := Concept
 r_0 := Atomic Relation
 e_0 := Expression

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 ASQL is valid

Test type	Dynamic, black box, manual
Schedule	Term 2
Requirements	F3, F1

Input

The input is an ADL file which contains various entities, among them business process rules which produce ECA rules.

Output

The prototype generated by Ampersand, which should be syntactically and semantically valid, as determined by the software which runs the prototype.

Description

Ampersand generates a prototype from the input ADL file, which a developer will open on a web server running on their machine. If the file compiles (or more likely, is interpreted) successfully, this is an indication that the generated ASQL inside the prototype is correct, in the context of the entire prototype.

T6 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.

T7 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 : \forall
{(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

T8 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

User feed back if file has no errors:

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

T9 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.

T10 Degradation Test

Test type Dynamic, black box, partially automated
Schedule First Week of February 2016
Requirements F6

Input

The Ampersand test suite.

Output

The time taken to run the test suite, both with EFA enabled and EFA disabled.

Description

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. The comparison will be made between `time ampersand-test-suite` and `ampersand-test-suite --noEFA` for enabled and disabled, respectively.

T11 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.

T12 EFA domain

Test type Static, white box, manual
Schedule Term 2
Requirements T2

Description

Our contribution will take as an input the internal representation of the entire ADL file, which is essentially a large record, called **FSpec**. For the most part, the majority of components will not be used – we will mainly focus on one or two components of dozens. However, potentially the entire structure may be useful, so we say that our domain is this structure.

Therefore, the entry point to our code should be a function with type **FSpec** \rightarrow **a** \rightarrow **ASQL** for some **a** (representing optional or auxillary arguments), where **ASQL** is the type representing abstract SQL. If this code type checks, then the compiler has verified that our domain of input is indeed exactly **FSpec**.

Bibliography

- [14910] ISO/IEC 14977:1996. Syntactic metalanguage – Extended BNF. http://www.iso.org/iso/catalogue_detail?csnumber=26153, 2010.
- [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.
- [sena] Sentinel Automatic testing of Ampersand System. <http://sentinel.tarski.nl/>. Accessed: 2015-11-03.
- [senb] Sentinel Test ref. <https://ampersandtarski.gitbooks.io/the-tools-we-use-for-ampersand/content/sentinel.html>. Accessed: 11-03-2015.