CS4ZP6 Test Plan
Ampersand Tarski Event-Condition-Action Rules

Yuriy Toporovskyy, Yash Sapra, Jaeden Guo

October 23, 2015

The focus of the test plan will be to demonstrate the current state of the Ampersand model prior to the addition of our project and to provide an overview of where our project fits into the Ampersand system. Ampersand is a system with patches for its unfinished pieces and one of those missing pieces is the purpose of our project, EFA. Although Ampersand is functional, it is limited in the number of finished products and complete services it can provide to its users. An example script will be compiled using Ampersand and multiple switches (i.e. compilation options) in order to generate different user feedback and create various Ampersand artifacts.

This demonstration will be a black box test of the current Ampersand system, one designed to produce various kinds of errors and violations. The error messages inform the user of syntax and definition errors, such as a missing variable. Violations inform the user when the model they are attempting to build violates pre-set conditions within the Ampersand system. An example of this is would be if a theatre is hiring actors to play a role, then a rule could be put in place stating that only actors who have a sufficient amount of experience can play lead. The user can define a sufficient amount of experience to be anything they wish, from the number of plays in which the individual has acted to the number of years that the actor has worked in the industry. A violation of the rule would be an attempt to cast an actor in a leading role who has not met the conditions of a sufficient amount of experience.

The purpose of the EFA project is to automate the correction of a particular class of violations in order to restore invariants within the Ampersand system. These violations come from the misuse of PAClauses (Process Algebra Clauses); PAClauses are datatypes formed from ECA (Event-Condition Action) rules that are used to represent the structure of an active database system. In Ampersand, these ECA rules are deconstructed to their most basic forms which we call Atoms. Atoms are the simplest possible form a rule can take before losing its meaning. From here, a Haskell program (i.e., EFA) will automate the correction of violations by detecting contradictions between Atoms imposed on various datasets. For example, one Atom states that all entities in its set must be blue, another Atom states that all entities in its set must be red. The intersection between these two Atoms is red AND blue; if the user inserts something in the intersection of red and blue that is not red and blue, it is a violation of both Atoms. A quick fix to restore both rules would be to remove the entity that is

not red or blue from the intersection of the red and blue set.

EFA is in its initial stages of development, and at this time it is difficult to provide a rigorous guide as to how this project will proceed. The first step was to figure out the particular classes of violations and the most conventional ways they are triggered. From there the developers of Ampersand have provided a partial algorithm for how individual violations can be restored. Foreseeable issues include not having a complete algorithm to implement on complex system such as Ampersand, in addition to figuring out how to fix violations that may cycle within itself.