Solving the State Elimination Case Study using Epsilon

Mohammadreza Sharbaf
m.sharbaf@eng.ui.ac.ir

Shekoufeh Kolahdouz-Rahimi
sh.rahimi@eng.ui.ac.ir

Bahman Zamani
zamani@eng.ui.ac.ir

MDSE Research Group
Department of Software Engineering
University of Isfahan, Iran

Abstract

The transformation of a finite state automaton into an equivalent regular expression is a challenging topic which is presented in TTC 2017. This paper presents a solution to State Elimination case using the Epsilon framework.

1 Introduction

This case study includes both a model to model and a model to text transformation, which aims to transform Finite State Automata (FSA) or Finite State Machines (FSM) into equivalent regular expressions. This is a challenging and expensive transformation. In this paper, we provide a solution to the transformation problem which eliminates states in an iterative manner. Our solution is based on random selection of a state and eliminating it from FSA. The solution is available as a Github repository[1].

Our solution is implemented using Epsilon[2] invoked from a Java application. Epsilon is an extensible set of languages and tools for model management which is built atop the Eclipse Modeling Framework (EMF) [1]. Epsilon can be used to perform all model management tasks, including in-place and out-place model transformations. Epsilon is an appropriate tool for solving the above mentioned case study that involves model modification, before generating equivalent regular expressions based on finite stated automata.

The remainder of this paper is structured as follows. Section 2 provides an introduction to the fundamental parts of Epsilon which are used for solving this problem. Section 3 provides our solution to the case study. Evaluation of the proposed solution is presented in section 4. Finally, section 5 summarizes our findings.

2 Epsilon Overview

Epsilon is a Java-based comprehensive framework which includes several languages for model management tasks such as model transformation, code generation, model refactoring and validation [2]. Following are the Epsilon languages which are used in our solution:

- **Epsilon Object Language (EOL):** EOL is an imperative language which is the core of Epsilon and can be used as a standalone generic language or used within other Epsilon model management languages. For tackling the state elimination problem, we have benefited from EOL in creation and deletion of states and transitions.

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

[1] https://github.com/MSharbaf/TTC2017-StateElimination

[2] https://www.eclipse.org/epsilon

- **Epsilon Transformation Language (ETL):** ETL is a declarative model-to-model transformation language, which inherits the imperative feature of EOL to perform complex transformations. It accepts multiple input models and is able to generate multiple target models. However, for this solution, we only used a single source and target model. Each ETL program consists of several ETL modules. A module can contain any number of transformation rules, operations and optional *pre* and *post* blocks, which are executed before and after the transformation rules, respectively.

- **Epsilon Generation Language (EGL):** EGL is a template-based language for text generation, which facilitates the construction of model-to-text transformations. Each EGL template consists of static and dynamic parts that reuses the EOL mechanism for defining declarative operations.

The aforementioned languages are used to create scripts (Epsilon codes) that take one or more models and transform them into another model or text. The ETL and EGL scripts can be executed with launch configuration facilities or alternatively invoked directly from a Java program.

3 Solving the State Elimination Case Study using Epsilon

In TTC 2017 a state elimination case study [3] based on the state elimination algorithm for FSA has announced. This case study includes a main task and two extensions. The main task is to convert a uniform FSA to the equivalent RE. An FSA is uniformed if it has a unique initial state with no incoming transitions and a unique final state with no outgoing transitions. A regular expression is a mathematical expression which specifies the language generated or accepted by a uniform FSA. The first extension is to transform a non-uniform FSA into a uniform one. The second extension is to convert a probabilistic FSA to a stochastic RE.

The state elimination algorithm for the main task takes a uniform FSA and generates an equivalent Regular Expression (RE). In this paper the Epsilon framework is used to solve the main task and the first extension. In the following sections the description of the solutions is provided in more details.

3.1 Overview of the Main Task Solution

In order to solve the main task problem, the transformation specification takes the uniform FSA as input model, and generates the equivalent regular expression as output model. The transformation eliminates each state with corresponding transitions and adds a new equivalent transition for predecessor and successor states in each iteration, until there is only one initial and one final state. In this solution the transformation chain includes two main phases:

- **Phase 1:** Transformation of a uniform FSA to a GTG

- **Phase 2:** Transformation of a GTG to a final regular expression

In the following section each phase is explained in more detail.

Phase 1: Transformation of a uniform FSA to a GTG

This phase is a model-to-model transformation, which is divided into three steps as follows:

- **Step 1:** Creating a GTG, with initial and final states, and an unlabeled transition between them

- **Step 2:** Selection and deletion of intermediate states (i.e., states other than initial and final) of FSA and their incoming and outgoing transitions

- **Step 3:** Labelled transition between initial and final states in GTG, equivalent to all transitions of the input FSA

These steps are implemented in an ETL module, which consists of ETL rules and sets of operations. In the following more details of this transformation is provided.

Step 1: Creation of GTG

In this step a GTG with a single initial and final state, and a single outgoing and incoming transitions is generated. Listing 1 illustrates the implementation of this rule in ETL. In this rule the initial and final states of input model is given to the transformation, and it then generates the model in the output with initial state, final state and a transition between them. Calculation of transition label is performed in the second and third steps.

```
rule StateAddition
  transform S1: input!State
  to S2: output!State{
  guard : S1.isInitial or S1.isFinal

  S2.id = S1.id ;
  S2.isInitial = S1.isInitial ;
  S2.isFinal = S1.isFinal ;

  if(S1.isInitial){
    trans.source = S2 ;
    S2.outgoing ::= trans ;
  }
  if(S1.isFinal){
    trans.target = S2 ;
    S2.incoming ::= trans ;
  }
}
```

Listing 1: ETL Rule demonstrating the *State Addition* operation

Step 2: Deletion of Intermediate states and transitions

In this step all the intermediate states and related transitions are eliminated and new labeled transitions corresponding to them are generated. The implementation of this step is provided in Listing 2. The EOL operations and expressions are used here for identification and deletion of the elements in the source model. In order to calculate the label of transition it is required to delete each intermediate state of FSA (for instance K), and transform its transitions into a new one with respect to its predecessor (P) and successor states (Q). In addition it is required to check the existence of loop, i.e., a transition with an identical initial and final state, in the selected state (K) and to identify the direct transitions between predecessor (P) and successor (Q) states. Following that all the transitions between predecessor (P) and successor states are eliminated and new transitions are generated. Finally a transition is labeled according to the $(\alpha_{pk}{\alpha^*}_{kk}\alpha_{kq})$ formula [3].

```
var lbl_PKQ = "" ;
var flag_lbl_PKQ = false ;
for(tp in P.outgoing.select(tr|tr.target == K)){
  for(tq in Q.incoming.select(tr|tr.source == K)){
    if(flag_lbl_PKQ == true)
      lbl_PKQ += "+" ;
    lbl_PKQ += tp.label ;
    lbl_PKQ += lbl_K_Self_Loop ;
    lbl_PKQ += tq.label ;
    flag_lbl_PKQ = true ;
  }
  delete tp ;
}
```

Listing 2: EOL expression to calculate the label of new transitions corresponding to eliminated states and related transitions

Step 3: Labelling the unlabeled transition in the GTG

In this step a single initial and final state with one or more transitions between them are remained in FSA. Additionally, it may be possible for each state to have loop. The transformation in this step generates a label equivalent to all the labels of transitions existed or generated in the previous step based on the state

elimination rule. This label is then attached to the new transition between initial and final states in the GTG. The EOL statements are used for implementation of this part in the post condition of ETL module.

Phase 2: Transformation of a GTG to a final regular expression

In this phase the generated GTG is transformed to a textual file that only contains a final regular expression, which is implemented with an EGL script as indicated in listing 3.

```
[%
  var sb := new Native("java.lang.StringBuilder");
  sb.append(Transition.allInstances.selectOne(s|s.label.isDefined()).label);
%]
[%=sb.toString()%]
```

Listing 3: EGL template for transforming a GTG into an equivalent RE

3.2 Overview of the First Extension Solution

For the main task it was assumed that the input FSA is uniformed. However, in the first extension it is possible to have more than one initial and final states. The extension is a model-to-model transformation. In our solution we have used ETL and EOL to solve this extension. In order to apply the state elimination algorithm to this cases, we should add a new initial state and change its *isInitial* property to true. Following that, a transition with a null label should be inserted from the new initial state to each original initial states and their *isInitial* property is changed to false. For the elimination of final states into a single state, we have followed the similar procedure.

In the proposed solution an operation is added to the ETL transformation for transforming non-uniform to uniform FSA, which generates a single initial and final state with its corresponding transitions. This operation is called in the *pre*-condition of the ETL module written for this extension.

3.3 Execution of the Solution

The proposed solution in this paper requires Epsilon Core for execution of EOL, ETL and EGL scripts. Therefore, we use the Eclipse Distribution which contains most of the required prerequisites of Epsilon. The complete solution uses the ANT Epsilon tasks to execute transformation chains in the specific workflow and enables the user to run it from the Eclipse toolbar.

4 Evaluation

In the case study description [3] a set of quality characteristics with its measurable attributes are defined for systematic evaluation of each solution. In the following sections the evaluation of our solution according to the correctness, suitability, performance and scalability are provided.

4.1 Correctness

According to the evaluation criteria, a solution for the main task is correct when the RE obtained as a final result of the State Elimination passes all sets of positive and negative test cases. Each test case is a set of strings to which the produced Regular Expression should match or should not match, which can be checked by Evaluation Framework provided in the case description. Regarding the result of Evaluation Framework, our main task solution is correct and passed all the positive and negative test cases.

Additionally, the correctness of solution for the first extension task generates a uniform FSA which is equivalent to the input FSA with some initial and final states. The Epsilon solution in this paper generates correct output model for each input model in the extension part. Table 1 presents correctness evaluation for each individual task of proposed solution in this paper.

4.2 Suitability

The suitability in this case study is evaluated by the level of abstraction of transformation language, which is High for primarily declarative language and Low for primarily imperative language. Although our solution combines imperative EOL statements with ETL and uses EGL script which are primarily declarative, the overall

Table 1: Correctness Evaluation for each task

Task	Correctness
Main task	11
Extension 1	3
Extension 2	0

Table 2: Abstraction level for our solution

Element		Abstraction level
Main Task		
Phase 1	Transformation of a Uniform FSA to a GTG	Medium
Phase 2	Transformation of a GTG to a final regular expression	High
Overall solution		Medium
Extension 1		
Overall solution		Medium

level of abstraction in this solution is *Medium*.Table 2 shows the abstraction level for each phase of main task, and an overall value for solutions of main task and first extension.

4.3 Performance

The solution performance should be measured as the seconds spent for executing two transformation phases with the provided input models. This includes the loading of input FSA models and generating a text file as the equivalent regular expression output.

The following table shows the execution time for our solution in second. We measure execution time automatically by Ant builder. This measures is the average execution time of ten consecutive executions of Ant builder with the specified input model. All tests were carried out on a standard Windows 7 PC using an Intel® Core™ i7 with 3.6 GHz processor and 8GB RAM.

Table 3: Evaluation results for our solution includes execution times and scalability level

Model name (FSA uniform)	Correct	Execution Time (s)	Scalability
leader3_2 (26)	yes	0.1476	
leader4_2 (61)	yes	0.1617	
leader3_3 (69)	yes	0.1769	
leader5_2 (141)	yes	0.2252	
leader3_4 (147)	yes	0.2398	
leader3_5 (273)	yes	0.3877	
leader4_3 (274)	yes	0.3639	
leader6_2 (335)	yes	0.4528	
leader3_6 (459)	yes	0.71	
leader4_4 (812)	yes	1	
leader5_3 (1050)	yes	2	
leader3_8 (1059)	yes	2	
leader4_5 (1933)	yes	7	
leader6_3 (3759)	yes	23	
leader4_6 (3962)	yes	29	
leader5_4 (4244)	yes	31	
leader5_5 (12709)	yes	341.6	
leader6_4 (20884)	yes	920.8	
leader6_5 (78784)	yes	9834.34	
leader6_6 (234210)	yes	102602	Leader6_6

4.4 Scalability

According to the evaluation criteria, scalability is a model with maximum number of states which is correctly converted to equivalent regular expression. According to the table 3 the transformation generates result for all the test cases specially for model leader6_6 with 234210 states.

5 Conclusion

In this paper we used Epsilon languages to transform a finite state automaton into an equivalent regular expression. The suitability of transformation languages in this solution is medium for implementation of the main task and first extension. Transformation generates correct results for all the test cases and the result of execution is extensively better than the results provided in the case description. Additionally, the scalability of transformation is high as it managed to execute the largest test case with 234210 states.

References

[1] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008.

[2] D. Kolovos, L. Rose, A. Garcia-Dominguez, and R. Paige in *The Epsilon Book*, pp. 1–213, 2017.

[3] S. Getir, D. Vu, F. Peverali, and T. Kehrer, "State Elimination as Model Transformation Problem," in *Teransformation Tool Contest (TTC) 2017*, pp. 1–9, 2017.