Algorithms Support Individual Fairness Analysis based on System Design Models

Supplementary martial of submission to the FAT20 conference

August 23, 2019

This report provides description for the algorithms that support our proposed framework in the submitted paper to the FAT20 conference.

A. The automatic generation of temporal logic claims. Algorithm 1 illustrates the generation of temporal logic claims. Our algorithm takes as input a UML model m annotated with the UMLfair profile and a databased db contains historical data. The algorithm returns B the set of all batches of claims with respect to sm, where sm is an «individual-Fairness»-annotated state machine in m. In the following, we explain Algorithm 1.

First, in line 2 of the algorithm, the name of the state machine that is annotated with the *«individual-Fairness»* stereotype will be retrieved. In the lines from 3-10 of the algorithm, the followings are declared and initialized:

1. the $protSet$. A set contains all the data that are defined as protected data in *{protectedData}* tags of the UML model m.
2. the $senSet$. A set contains all the data that are defined as sensitive decisions in *{sensitiveDecisions}* of sm.
3. the $gaurdsSet$. A set contains all atomic guards conditions in the UML model m.
4. the $threshold$. A variable of type double that takes as a value the specified value in the {threshold} tag of sm.
5. the $metric$. A variable of type string that takes as a value the specified value in the *{metric}* tag of sm.

In line 11 of Algorithm 1, an empty set B is declared. This set will be used in the algorithm to store all batches of claims that can be generated. In line from 12-16 of the algorithm, for each sensitive decision s belongs to the $senSet$, the followings will be performed:

1. in the lines from 13-14 of Algorithm 1, the explanatory data with respect to s that are defined in the {explanatroyData} tag of sm are will be retrieved and stored in the $expSet$.
2. in the lines from 15-16 of of Algorithm 1, all the used conditions in m with respect to s will be retrieved and stored into the $usedCondSet$ by calling the $getCond(m, db, protSet, expSet, gaurdsSet, metric, threshold)$ function. This function takes as input a UML model m, the database db, the set of protected data $protSet$, the specified $metric$ and $threshold$ in the model. The function returns all used conditions in m with respect to the sensitive decision s. Details on how the used conditions can be generated are provided in a by Algorithm 2 in the Appendix (C).

For each sensitive decision s that is defined in the m as a data attribute, the following will be performed:

Algorithm 1: Generating batches of claims from a UML model annotated with the UMLfair profile

```
1  generateBatchesOfClaims (m, db);
   Inputs : a UML model m annotated with fairness information and a
            database db
   Output: a set of batches of claims B
2  sm ← getIndividualFairnessStateMachines(m);
3  protSet ← ∅;
4  senSet ← ∅;
5  gaurdSet ← ∅;
6  threshold ← getThreshold(sm);
7  metric ← getMetric(sm);
8  protSet ← getProtected(m);
9  senSet ← getSensitive(sm);
10 gaurdsSet ← getGaurds(m);
11 B ← ∅;
12 foreach s ∈ senSet do
13     expSet ← ∅;
14     expSet ← getExplanatory(sm, s);
15     usedCondSet ← ∅;
16     usedCondSet ←
         getCond(m, db, protSet, expSet, gaurdsSet, metric, threshold);
17     if s is a data attribute then
18         valuesSet ← ∅;
19         valuesSet ← getValues(s, m);
20         foreach g ∈ usedCondSet do
21             batch ← ∅;
22             foreach v ∈ rangeSpaceSet do
23                 batch.add(
24                 {g →<> s == v}, {!g →<> s == v});
25             end
26             B.add(batch);
27         end
28     end
29     else
30         foreach g ∈ usedCondSet do
31             batch ← ∅;
32             batch.add(
33             {g →<> (event_queues?[call_s])},
                 {!g →<> (event_queues?[call_s])});
34             B.add(batch);
35         end
36     end
37 end
38 return B
```

1. in the lines from 18-19 of Algorithm 1, all the possible values that can be assigned to s will be retrieved from the UML model m and stored in the $valuesSet$.

2. in the lines from 20-26 of Algorithm 1, for each used condition g with respect to the sensitive decisions s, a batch of temporal claims will be defined as follows: for each value belongs to the $valuesSet$, a pair of claims will be defined and added to the batch. Each pair should has the format of the temporal logic pair in line 24 of Algorithm 1. After the iteration over all values in the $valuesSet$ is finished, the batch will be added to the set of all batches of claims B, as shown by line 26 of Algorithm 1.

For each sensitive decision s that is defined in the m as a call-operation event, the following will be performed:

1. in the lines from 30-33 of Algorithm 1, for each used condition g with respect to the sensitive decisions s, a batch of temporal claims will be defined. Each batch must consists of two claims conforms to the format that is given in line 33 of Algorithm 1.

2. Each generated batch will be added to the set of all batches of claims B, as shown by line 33 of Algorithm 1.

B. The automatic individual-fairness-check. Algorithm 2 shows the proposed individual-fairness-check. Algorithm 3 takes as inputs: (1) a set of generated batches of claims B from Algorithm 1, (2) a set R contains the results of verifying the claims in B against the targeted system model. Algorithm 2 returns $fairnessReport$ contains the violations for the individual fairness, if any. In the following, we explain how Alg. 3 works.

Algorithm 2: Individual-fairness-check

```
1  indFairness (m, B, R);
   Output: fairnessReport a report shows the result of the individual
           fairness analysis
2  create fairnessReport;
3  foreach batch ∈ B do
4  |  flag ← 0;
5  |  foreach pair ∈ B do
6  |  |  claim1 ← getFirst(pair);
7  |  |  claim2 ← getSecond(pair);
8  |  |  if claim1 && Claim2 are satisfied in R then
9  |  |  |  flag++;
10 |  |  end
11 |  end
12 |  if flag!=1 then
13 |  |  create explainReport;
14 |  |  explainReport.add(getResutls(batch, R));
15 |  |  fairnessReport.add("a violation for individual fairness in the
   |  |   model is detected"+ explainReport);
16 |  end
17 end
18 if fairnessRepor.isEmpty() then
19 |  fairnessReport.add("the analyzed state machine preserves individual
   |   fairness");
20 end
21 return fairnessReport;
```

First, in line 2 of Algorithm 2 an empty $fairnessReport$ will be created. From line 3-17 for each batch of claims belongs to the set of all batches B the followings will be performed:

1. set a $flag$ to 0 as shown by line 4 of Algorithm 2. This flag will be used later to reason on individual fairness.
2. from line 5-10 for each pair of claims in a batch check the algorithm check if the claims are satisfied or not. If both claims of a pair are satisfied the $flag$ will be incremented by 1.
3. after iterating over all pairs in a batch, in lines 12, the algorithm checks if the flag is not equals to 1. If the flag is not equal to 1, the satisfaction

results of claims in the batch will be retrieved and stored into a report called *explainReport*.

4. in line 15 the *explainsReport* together with the following text *"a violation for individual fairness in the model is detected"* will be added to the created $fairnessReported$.

In line 18 of Algorithm 2, after iterating over all batches of claims in the set of all batches B, the algorithm check if the $fairnessReport$ is empty or not. In case the $fairnessReport$ is empty the following text will be added to the $fairnessReport$: *"the analyzed state machine preserves individual fairness"*.

C. The automatic generation of used conditions. Algorithm 3 illustrates how the used conditions with respect to a sensitive decision s can be automatically generated. Our algorithm takes as inputs: (1) a UML model m, (2) a database *db*. (3) a *protSet* that contains a set of protected data in m. (4) a *expSet* contains all the explanatory data with respect to a sensitive decision s in m. (5) a *gaurdsSet* contains atomic guard conditions in m. (6) a string *metric* shows the correlation metric that have be used for uncovering correlations. (7) ϵ a threshold value.

Algorithm 3: Algorithm for generating potential proxies from a UML model annotated with fairness specific information

```
1  getCond (m, db, protSet, expSet, gaurdsSet, metric, ϵ);
   Output: usedConditionsSet a set of used conditions
2  usedConditionsSet ← ∅;
3  protectedNotExp ← protSet \ expSet;
4  foreach con ∈ gaurdsSet do
5  |  attr ← getAttribute(con);
6  |  if attr ∈ expSet || attr is a derived attribute then
7  |  |  do nothing;
8  |  else
9  |  |  proxyOfProtSet ← ∅;
10 |  |  proxyOfProtSet ← f(m, db, con, protectedNotExp, metric, ϵ);
11 |  |  if (proxyOfSet!∅) then
12 |  |  |  usedConditionsSet.add(con);
13 |  |  end
14 |  end
15 end
16 return usedCondition
```

Algorithm 3 returns *usedConditionsSet* that contains all used conditions with respect to a sensitive decision s. In the following, we explain how Algorithm 3 works.

First, in line 2 of Algorithm 3 an empty *usedConditionsSet* is declared. In line 3 a set *protectedNotExp* is declared. This set contains all the data in the *protSet* except those in the `expSet` in order to avoid proxies for protected data

that is defined as explanatory data. In the lines from 4-15 of Algorithm 3, for each a atomic guard condition *con* in the set of all atomic guards conditions $gaurdsSet$, the followings will be performed:

1. in line 5 of Algorithm 3, the attribute of the guard condition *con* will be retrieved and stored in the *attr* variable.

2. in the lines from 6-15 of of Algorithm 3, if the data attribute *attr* does not belongs to the explanatory dataset $expSet$ and does not defined as derived data attribute in the UML model m: first: the function f will be called. This function returns the set of protected data $proxyOfProtSet$ that contains all the protected data attributes that the data attribute *attr* can be a proxy of them. Second, if the returned $proxyOfSet$ is not empty, the atomic guard *con* will be added to the $usedConditionsSet$.