MBFair: A Model-based Verification Methodology for Detecting Violations of Individual Fairness (Long Version)

***Abstract*—Decision-making systems are prone to discrimination against individuals with regard to protected characteristics such as gender and ethnicity. Detecting and explaining the discriminatory behavior of implemented software is difficult. We propose MBFair, a software model-based verification methodology, to uncover violations of individual fairness during the software design phase. Thereby, avoiding the possibility of discrimination from the onset of software development. Specifically, we propose a UML extension, UMLfair, to annotate a UML model with fairness-related specifications. From such models, we generate temporal logic formulas, which can be automatically verified to detect violations of individual fairness.**

We study the general applicability of MBFair using three case studies in real-world settings including: a bank services system, a delivery system, and a loan system. We empirically evaluate the necessity of the MBFair in a user study and compare it against a baseline scenario in which no modeling and tool support is offered. Empirical evaluation indicates that analyzing the UML models manually produces unreliable results with a high chance of 54% that analysts overlook true-positive discrimination. We conclude that analysts require support for fairness-related analysis, such as our MBFair methodology.

***Index Terms*—Software fairness, Individual fairness, Model-based verification, UML**

I. Introduction

Automated decision-making software are responsible for sensitive decisions affecting many areas of our lives. However, a falsely developed decision-making software may lead to unlawful discrimination against persons raising public awareness and legal concerns on the fairness of software [1, 2, 3]. For instance, Recital 71 of the European General Data Protection Regulation (GDPR, [4]) prescribes to *"implement technical and organizational measures appropriate to [...], and prevent, inter alia, discriminatory effects on natural persons on the basis of racial or ethnic origin, [...]"*. Furthermore, software fairness is stipulated by Article 22, which forbids decisions based on special categories of data as defined in Article 9, such as ethnicity and gender. These categories are known as protected characteristics [5].

A distinguished type of fairness is called *individual fairness*. A decision-making software preserves the individual fairness if it produces the same decision for every two individuals whose data that are given as input to the decision-making software are identical except for the protected characteristics [6, 7]. Only avoiding protected characteristics in a decision-making software does not prevent discrimination. Due to data correlations, other data may act as proxies for protected characteristics, thereby causing *indirect discrimination* [5]. For example, in 2016, a delivery decision-making software by Amazon excluded some neighborhoods with African American communities in the United States from being able to participate in a free-delivery service, although the software did not explicitly use the ethnicity of the customers for making the decisions. There are two main explanations for data correlations [8]: (i) societal fact: For instance, it turns out that the discriminatory behavior of the Amazon's delivery software results from the use of the customers' living zip-code which is, in the United States, highly correlated with the ethnicity. (ii) data flow: The actual input of a decision-making software may contain data resulting from processed protected characteristics. The latter issue is typical for the problem that Dwork et al. [9] observed: *"fairness is not a property of software in isolation, fairness is a property of the software composed into its system"*.

Furthermore, existing approaches do not analyze the software fairness *ex ante* but *ex post*, namely during the testing phase of software development life cycle (e.g, [5, 10]) or at the run-time of the software (e.g., [11, 12]). To reduce difficulties in detecting the discriminatory behavior of the software, it is important to deal with fairness from the early phases of software design. According to Brun et al. [13] "*as with software security [...], fairness needs to be a first-class entity in the software engineering process*". Therefore, we propose *MBFair* an ex ante methodology that verifies a software model, with respect to the overall behavior of its system, to detect violations of individual fairness.

The need for engineering fairness-aware systems based on software models has first been motivated by Ramadan et al. [14] and Brun et al. [13]. But the work in [13, 14] do not provide an approach that analyses software models for detecting discrimination. Hence, reasoning on fairness at the software design phase has been impossible.

We presents *MBFair*, a model-based methodology that supports the analysis of UML-based software designs with regard to individual fairness. The analysis in MBFair is established by generating temporal logic formulas, whose verification enables reporting on the individual fairness of the targeted software. This paper includes the following contributions:

(i) a semi-automated methodology *MBFair* that applies model checking techniques for analyzing UML models of software with respect to individual fairness (Sect. IV),
(ii) a UML profile *UMLfair* for annotating UML models with fairness-specific information (Sect. V),

(iii) a *fairness reporter* for reporting on the individual fairness of annotated UML models (Sect. VI to Sect. VII),
(iv) *three real case studies*, that show the general applicability of MBFair (Sect. IX), and
(v) a *user experiment*, in which we studied the necessity of our MBFair (Sect. X).

It is worth noting that MBFair is not concerned with analyzing machine learning-based software. Even if-then-else rule-based software can be unfair and such software still make up a vast percentage of software, especially when a sufficient amount of data for training machine learning algorithms is not available. MBFair is novel with regard to analyzing UML-based software designs with respect to fairness.

This paper is organized as follows. In Sect. II, the necessary background is provided. Sect. III provides a running example. Sect. IV is an overview of the MBFair methodology. Sect. V to Sect. VII provide detailed descriptions of the individual phases of MBFair. Sect. VIII presents tool support that we developed. Sect. IX and Sect. X study the applicability and necessity of MBFair using case studies and a user experiment, respectively. In Sect. XI we survey related work. Finally, in Sect. XII, we conclude and present future work.

II. Background

In this section, we provide the definitions of fairness-related concepts and an overview of formal verification.

Fairness-related concepts. In the following, we provide the definitions of four fundamental fairness-related concepts.

Protected characteristics refer to data that should not be used to differentiate between people in a decision-making process [7] such as gender, ethnicity or age.

Proxy information refers to data that are highly correlated with protected characteristics [10, 12]. For example, if young people are more likely to be healthy than older people, then the health status is a proxy for age. Proxy information can be identified manually by domain experts or automatically using a database of personal information.

Explanatory knowledge refers to knowledge for which a reasonable justification exists to be used in a given decision-making software [15]. Discrimination on the basis of protected characteristics or proxy information is not always forbidden. For example, non-admitting blind applicants to a driving license is allowed. The explanatory knowledge may differ from one domain application to another. Therefore, domain experts, based on organizational policies and equality acts such as the German General Act on Equal Treatment [16], can identify the circumstances in which discrimination on the basis of a protected characteristic is allowed.

Sensitive decisions is a list of data attributes and call operations that should not discriminate on the basis of protected characteristics, such decisions can be provided by domain experts.

Formal verification. A software model preserves individual fairness if it produces the same decision for any two feasible traces of events, whose configuration data are identical except for the protected characteristics or their proxy information. We derive this definition in analogy to the *no down-flow* policy applied in information security [17]. No down-flow means that a public output of a system does not depend on secure inputs. Generating all possible traces of events is a difficult task as the number of the traces may be relatively large. Therefore, we introduce the use of model checking techniques.

Model checking techniques are best applied in the early stages of software design when the costs are relatively low and the benefits are high [18]. Model checkers take as input a software model described in formal languages like the Process Meta-Language (PROMELA) for the model checker SPIN [19]. PROMELA permits describing the software as an interleaving composition of several automata that are running in parallel. The second input to the model checker Spin is a property which is checked against the PROMELA specification of the software. One of the main goals of model checker Spin is to check *liveness* properties. A liveness property asserts that something eventually happens [20]. For example, eventually a data attribute y takes on the value 1.

To specify complex liveness properties, one can make use of temporal logic. Temporal logic is a symbolic logic, which permits specifying formulas that have truth values [18]. One may use temporal logic formulas to require that if one condition holds, in the future a second condition will hold. For example, assume that x and y are two data attributes of type Boolean in a PROMELA model M. Formula (1) is a Linear Temporal Logic (LTL, [21]) formula, which asserts that given x is $true$, eventually ($\Diamond$) y is always ($\Box$) $true$. Specifically, Formula (1) asserts that there is some instantiation of (non-derived) data attributes in the PROMELA model and some execution such that if x is $true$, later at some point ($\Diamond$) y is always ($\Box$) $true$.

$$x{=}{=}true \rightarrow \Diamond\Box(y{=}{=}true) \quad (1)$$

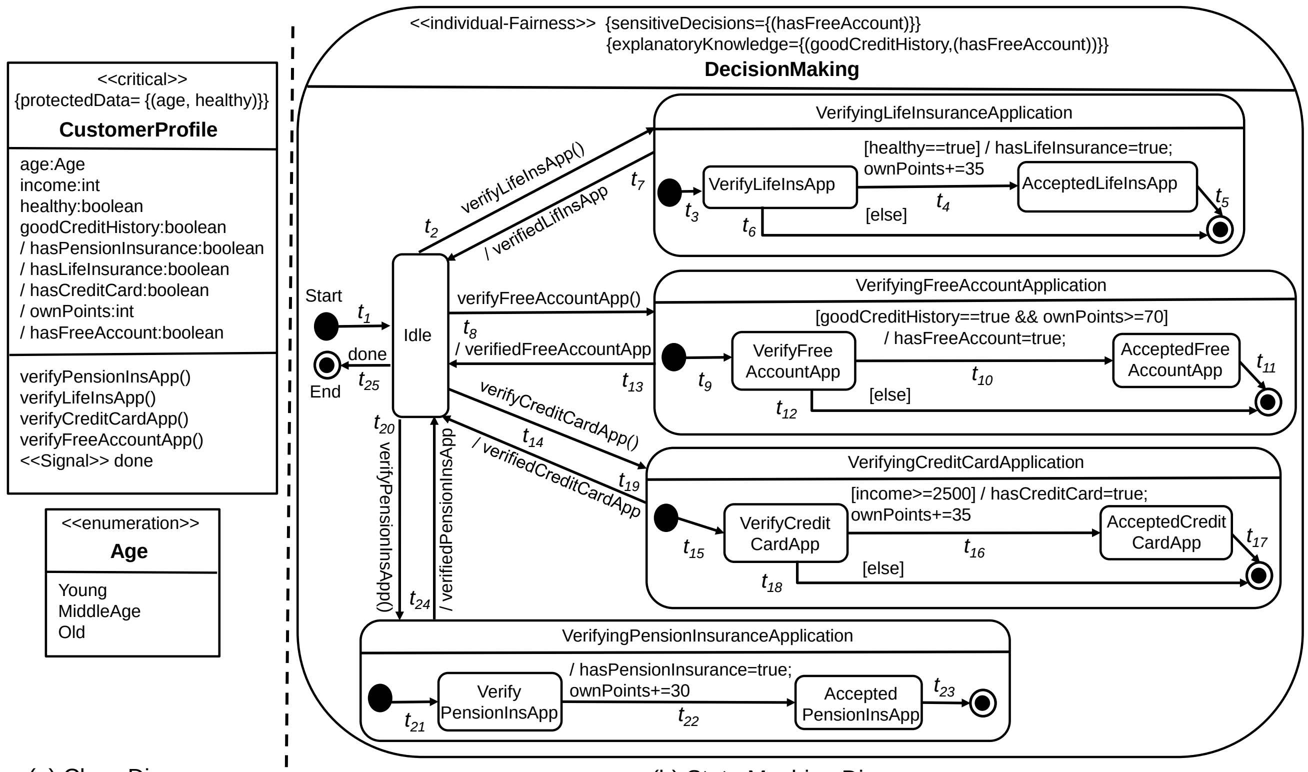
Given a PROMELA model M and a LTL formula ϕ, the aim of the model checker Spin is to check if $M \vDash \phi$ (i.e., M satisfies ϕ). The model checker Spin concludes that $M \vDash \phi$ by refutation, i.e. by showing that there is no model for $\neg\phi$ in M. For this, Spin represents the PROMELA model M and the LTL formula $\neg\phi$ in Büchi automata [22]. If the intersection between the automaton of the PROMELA model and the automaton of the $\neg\phi$ is empty (i.e., no acceptance cycle), then Spin concludes that ϕ is unsatisfied formula. Otherwise, ϕ is satisfied.

In this paper, we show how a set of temporal logic formulas can uncover whether a sensitive decision depends on a protected characteristic or proxy information.

III. Running Example

The following example is inspired from real decision-making processes in the financial sector.

Consider a bank interested in leveraging automatic decision-making to decide about different kinds of applications that can be submitted by customers for supplementary services offered by the bank, such as: (i) application for a life insurance, (ii) application for a pension insurance, or (iii) application for a



(a) Class Diagram (b) State Machine Diagram

Fig. 1: Excerpt from the class and the state machine diagrams of the bank's software.

credit card. Customers who subscribe to a threshold number of supplementary services can apply for a free bank account, while others must pay for the account.

The software model. Fig. 1 shows a simplified excerpt of the bank's UML model. We focus on the parts that are related to life insurance, pension insurance, credit card and free bank-account applications. Fig. 1(a) is a UML *class diagram* [23]. The "CustomerProfile" class shows several personal data attributes of customers. The "age" is an attribute of type *enumeration*. The possible values of "age" are restricted to one of the following values "Young", "MiddleAge" and "Old". The "healthy" attribute is "true" if a customer does not suffer from chronic diseases, else "false". The "income" is of type integer and it shows the monthly income of a customer in Euro. The "goodCreditHistory" is "true" if a customer pays his/her financial dues regularly, else "false".

The attributes that are marked with "/" are *derived*. A derived attribute is an attribute that is not obtained directly from a customer, but that results from data processing. The derived attributes in the "CustomerProfile" class are described as follows: The "hasPensionInsurance" is "true" if a customer has a pension insurance with the bank, else "false". The "hasLifeInsurance" is "true" if a customer has a life insurance with the bank, else "false". The "hasCreditCard" is "true" if a customer has a credit card, else "false". The "ownPoints" is an integer attribute based on the services that a customer subscribed to in the bank. The "hasFreeAccount" is "false" if a customer needs to pay for the account, else "true".

The "CustomerProfile" class shows also several operations. An example of an operation is the "verifyFreeAccountApp()". The *«Signal»*-annotated "done" expresses a received signal that an object of the "CustomerProfile" class may receive during its life-cycle ([23], p.169).

Fig. 1(b) is a UML *state machine diagram* [23] that describes the behavior of the "CustomerProfile" class. States are denoted by boxes. An example of a state is "VerifyFreeAccountApp". Transitions are denoted by arrows. A transition with label $e[g]/a$ indicates that an object in the first state will perform the action a and enter the target state when the event e occurs and the guard condition g is satisfied ([23], p.314). For example, in Fig. 1(b), if an object is in the "VerifyFreeAccountApp" state and the condition "[goodCreditHistory==true && ownPoints>=70]" is true, first the transition "t_{10}" will be fired, then the "hasFreeAccount" will be set to true and the "AcceptedToFreeAccount" state will be entered.

Proxy Information. In our running example, using a database based on real statistics [24, 25, 26], we identify "goodCreditHistory", "income" and "healthy" as proxies for "age", because people in the middle age are more likely to have higher income than younger or older people [25]. In addition, older people are more likely to have chronic diseases other than younger people and people in the middle age [24].

The challenge. Given the UML model in Fig. 1 and the proxy information, our question as computer scientists is: Does the behavior of the "DecisionMaking" state machine violate individual fairness with respect to protected characteristics?

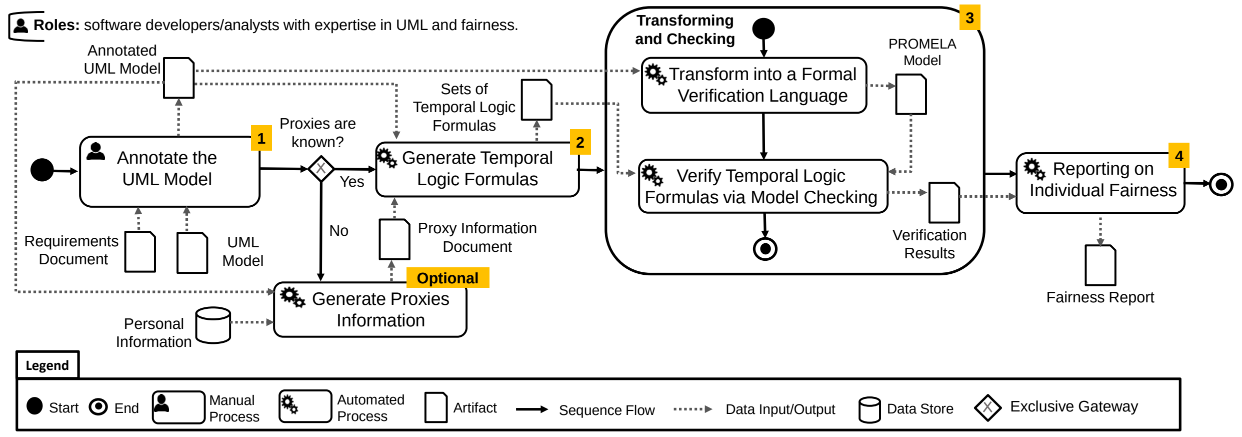

Roles: software developers/analysts with expertise in UML and fairness.
Annotated UML Model
Annotate the UML Model
1
Proxies are known?
Yes
No
Sets of Temporal Logic Formulas
Generate Temporal Logic Formulas
2
Requirements Document
UML Model
Proxy Information Document
Optional
Generate Proxies Information
Personal Information
Transforming and Checking
3
Transform into a Formal Verification Language
PROMELA Model
Verify Temporal Logic Formulas via Model Checking
Verification Results
Reporting on Individual Fairness
4
Fairness Report
Legend
Start
End
Manual Process
Automated Process
Artifact
Sequence Flow
Data Input/Output
Data Store
Exclusive Gateway

Fig. 2: The semi-automated process of the MBFair methodology.

If it does, then what is the cause of the discrimination? In the following, using our MBFair methodology, we show that it is possible to discriminate between two free account applicants whose data are identical except for “healthy” or “income”, which are proxies for “age”. The goal of MBFair is to report formal violation of individual fairness. Domain and legal experts can decide whether the detected violation is indeed an illegal one. In case a formal violation is legal, the model should be enriched with explanatory knowledge to avoid overlooking for formal violations that are acceptable.

IV. MBFair Methodology: Overview

Manually verifying a UML model to uncover violations of individual fairness is an error-prone task because the data attributes used in the UML model are distributed in multiple diagrams. To address this challenge, we propose *MBFair*. MBFair is a semi-automated model-based verification methodology that applies model checking to UML models for detecting violations of individual fairness. An overview of MBFair is shown in Fig. 2. In the following we survey the stakeholder roles, inputs/output and the phases of MBFair regarding the phases' numbers depicted in Fig. 2. Individual phases are explained in detail in Sect. V to Sect. VII.

Stakeholder Roles. MBFair requires the participation of *domain experts* and software *developers/analysts*. During the requirements elicitation, domain experts identify fairness-specific information such as protected characteristics and proxy information. Software developers design the UML model and annotate it with fairness-specific information.

Inputs. Three inputs are required: (1) a *Software Model* described in UML. Structural aspects of a software system are specified by class diagrams. Behavioral aspects are specified by state machine diagrams. (2) a *Requirements Document* in which domain experts have identified the protected characteristics and the circumstances in which discrimination on the basis of a protected characteristic is allowed. (3) *a Proxy Information Document* in which domain experts provide information about proxies of protected characteristics. In the absence of knowledge about proxies, historical personal data can be used to identify proxies. Appendix (A), in Sect. XIII, describes how we identify proxy information using a database.

Output. The output of MBFair is a *Fairness Report*. For each detected violation of individual fairness, the report shows: (1) the affected sensitive decision. (2) the source of the violation (i.e., the data attribute that caused the discrimination). (3) the effects of the source of the violation on the sensitive decision. The effects represent all the possible changes on the sensitive decision with respect to all possible values that can be assigned to the data attribute, which represents the source of the violation.

Phase 1. *Annotating the UML Model.* Software developers annotate the UML model manually with fairness information, based on the requirements document. We propose a UML profile called *UMLfair*. UMLfair is an extension of UML and permits software developers to specify fairness information in a UML model. Our UMLfair profile is described in Sect. V.

Phase 2. *Generating LTL Formulas.* This phase is concerned with generating sets of Linear Temporal Logic (LTL) formulas, where each set of LTL formulas analyzes the effects of a protected characteristic or a proxy on a sensitive decision. To automate the generation of LTL formulas, we developed an LTL generator. Our LTL generator takes as inputs a UML model annotated with fairness information and a proxy information document. The output is sets of LTL formulas. Sect. VI describes our LTL generator.

Phase 3. *Transforming and Checking.* The inputs to this phase are a UML model annotated with fairness information and sets of LTL formulas. Developers automatically check the assertions that are stated by the LTL formulas using model checking. For this, the UML model has to be transformed into a formal language using a transformation tool.

Phase 4. *Reporting on Individual Fairness.* This phase is concerned with analyzing the results of checking the assertions that are stated by the generated LTL formulas to report violations of individual fairness. To automate the reporting, we developed a fairness reporter. The fairness reporter takes as an input sets of LTL formulas with their checking results. Sect. VII is a detailed description of the fairness reporter.

V. Annotating the UML Model

We propose a UML profile called *UMLfair*. UMLfair is an extension of UML and permits software developers to annotate the targeted UML model with fairness information. Table I provides the *«stereotypes»* and the *{tags}* of UMLfair profile.

For specifying protected characteristics, our profile has a stereotype called *«critical»*. We reused this stereotype from the UML security profile UMLsec [27]. In UMLsec, this stereotype annotates classes that may contain confidential data. In our work, we use this stereotype to annotate classes that may contain protected characteristics or proxies to protected characteristics. We extended the *«critical»* stereotype with a *{protectedData}* tag to define protected characteristics with respect to a class annotated *«critical»*. For example, in Fig. 1(a), the *{protectedData}* tag of the annotation *«critical»* on class "CustomerProfile" specifies the "age" and the "healthy" attributes as protected characteristics.

TABLE I: The UMLfair profile.

Stereotype	**Tag**
«critical»	{protectedData={(string[1...*])}} *//required*
«individual-Fairness»	{sensitiveDecisions={(string[1...*])}} *//required* {explanatoryKnowledge= {(string[1],(string[1...*]))*}} *//optional*

The *«individual-Fairness»* stereotype annotates a state machine that describes the behavior of a class, that is annotated *«critical»*. This stereotype has several tags. The *{sensitiveDecisions}* is a required tag and it allows defining decisions that should not discriminate on the basis of protected characteristics or proxy information. For example, the state machine annotated with *«individual-Fairness»* in Fig. 1(b), specifies "hasFreeAccount" data attribute as a sensitive decision.

The *{explanatoryKnowledge}* is an optional tag and it allows for defining that it is acceptable to use these data attributes to differentiate in sensitive decisions. The goal is to avoid overlooking on discriminatory cases that are acceptable. For example, financial institutes offer benefits to their customers who have good credit histories to make them permanent customers. Therefore, in Fig. 1(b) we defined "goodCreditHistory" to be an explanatory attribute for "hasFreeAccount", although "goodCreditHistory" is identified to be a proxy for age.

VI. Generating and Checking LTL Formulas

This sections presents our LTL generator and the model checking process.

Generating LTL formulas. Our LTL generator generates sets of LTL formulas, where each set of formulas analyzes the effects of a protected characteristic or a proxy on a sensitive decision. For example, Table IIa provides a set of LTL formulas that analyzes the effects that *income*, which is identified to be a proxy for age, has on *hasFreeAccount*. The formulas in the table are structured in term of pairs namely, p1 and p2. Each pair analyses one possible effect that *income* has on *hasFreeAccount*. Each formula in Table IIa asserts that an effect later at some point ($\Diamond$) holds always ($\Box$). That is to avoid the situation where an effect may happen at some intermediate stage of the model execution but not hold until the end of the execution.

Concerning the $(income{>=}2500)$ condition in Fig. 1(b), there might be two effects for *income* on *hasFreeAccount*. The formulas of p1 in Table IIa assert that *hasFreeAccount* is true, independently of *income*, while the formulas of p2 assert that *hasFreeAccount* is false, independently of *income*.

Specifically, the first formula of p1 asserts that there is some instantiation of (non-derived) data attributes and some execution, such that if *income* is $>=2500$, later at some point ($\Diamond$) *hasFreeAccount* is always ($\Box$) $true$. The second formula of p1 asserts that there is some instantiation of (non-derived) data attributes and some execution, such that if *income* is <2500, later at some point ($\Diamond$) *hasFreeAccount* is always ($\Box$) $true$. The first and second formulas of p2 are similar to the first and the second formulas of p1, respectively, but with asserting that *hasFreeAccount* equals false instead of true.

For each defined sensitive decision in the UML model, our LTL formulas generator generates a set of LTL formulas for (a) each protected characteristic (in the *{protectedData}* tag) and (b) for each proxy of those. Appendix (B), in Sect. XIII, provides technical details about the algorithm of our LTL generator.

Checking LTL formulas. The used model checker in this paper is SPIN [19]. The model checker SPIN accepts a model described by PROMELA.The transformation of a UML model to PROMELA can be performed automatically using the transformation tool Hugo/RT [28]. Given a PROMELA model M and a generated set of LTL formulas Z, we check: whether for each formula f in Z, $M \vDash f$ independent from the other formulas in the set Z. In other words, we do not look for a single model interpretation for all formulas in the set Z. For example, Table IIb shows the results of checking the formulas in Table IIa, by using the model checker SPIN. The result column shows satisfied if the corresponding PROMELA model to the UML model in Fig. 1 entails the corresponding formula in the table, else unsatisfied.

The results in Table IIb shows the $hasFreeAccount$ depends on the $income$. The table shows that the first formula of p1 is satisfied, which means that there is some instantiation of (non-derived) data attributes in the model and some execution, such that if $income$ is >=2500, later at some point ($\Diamond$) the $hasFreeAccount$ is always ($\Box$) $true$. Based on the specifications of the state machine in Fig. 1(b), this can happen when also the $goodCreditHistory$ and the $healthy$ equal $true$. In contrast, the second formula of p1 is unsatisfied. This is because when the $income$ is less than 2500, the $ownPoints$ will never be greater than or equal to 70 (see transition t_{10} in Fig. 1). In this case, it is impossible that the $hasFreeAccount$ will at some point ($\Diamond$) equal $true$ always ($\Box$). Therefore, the $hasFreeAccount$ can not be true independent of the $income$.

Concerning the formulas of p2 in Table IIb, they are both satisfied because independent of the $income$ there is the possibility that the $hasFreeAccount$ will at some point ($\Diamond$) equal $false$ always ($\Box$). This can happen when $goodCreditHistory$ and $healthy$ equal $false$. In fact, having the second formula of p1 is unsatisfied, there is no need to check the second

TABLE II: An example of the generated LTL formulas and their checking results

(a) A set of LTL formulas (b) The checking results

Pair	LTL Formula	Result
p1	$income{>=}2500 \rightarrow \Diamond\Box(hasFreeAccount{==}true)$	satisfied
	$\neg(income{>=}2500) \rightarrow \Diamond\Box(hasFreeAccount{==}true)$	unsatisfied
p2	$income{>=}2500 \rightarrow \Diamond\Box(hasFreeAccount{==}false)$	satisfied
	$\neg(income{>=}2500) \rightarrow \Diamond\Box(hasFreeAccount{==}false)$	satisfied

formula of p2. This is because having the second formula of p1 is unsatisfied means that for every instantiation of (non-derived) data attributes in the model and for every execution, such that if $income$ is less than 2500, later at some point ($\Diamond$) $hasFreeAccount$ is always ($\Box$) $false$.

VII. Reporting on Individual Fairness

This section presents our developed fairness reporter. The fairness reporter analyzes the results of checking the generated LTL formulas to decide whether a decision is independent of a protected characteristic/proxy.

As discussed in Sect. VI, each generated set of LTL formulas is structured in term of pairs, where each pair of formulas analyzes one possible effect of a protected characteristic/proxy on the sensitive decision. Therefore, given a set of LTL formulas Z that analyzes the effects of a protected characteristic/proxy p on a sensitive decision s in a PROMELA model M, our reporter returns that s independent of p if one of the following cases is true:

Case 1. for each formula f in the set Z, $M \vDash f$. This case means that the sensitive decision s can take a value independent from the protected characteristic/proxy p.

Case 2. There exists pair of formulas W in the set Z such that for each formula f in the pair W, $M \vDash f$ while each formula f in $\{set_i \backslash pair_k\}$ is unsatisfied. This case means that the sensitive decision s *always* has the same value independent of the protected characteristic/proxy p.

Case 2 is trivial because it does not make sense that a decision has the same value always. However, if case 2 holds, it means that the sensitive decision is independent of the protected characteristic/proxy.

Our fairness reporter returns "*the state machine sm preserves the individual fairness*" if *for each* generated set of LTL formulas one of the cases above is true. Otherwise, the individual fairness is violated. For example, regarding the set of LTL formulas and the model checking results in Table II, one can notice that neither all the LTL formulas of the set are satisfied nor the LTL formulas of one pair are satisfied while the formulas of the other pair are unsatisfied. Therefore, the $hasFreeAccount$ depends on the $income$. Hence, it is possible to discriminate between two free-account applicants whose data are identical except for "income", which is a proxy of "age". The algorithm of our fairness reporter is provided in Appendix (C), in Sect. XIII.

VIII. Tool Support

We developed prototypical implementations[1] to support several phases of the MBFair methodology. We developed:

First, a Papyrus editor extension that permits annotating UML models manually using the UMLfair profile.

Second, a LTL generator that takes as input a UML model, in XML format, annotated with fairness information and proxies information document in Excel format. Our LTL generator uses the DOM Java library to parse the XML file and generate LTL formulas. In the worst case, the number of the generated formulas from a model will belong to $\mathcal{O}(ns*ng*nv)$, where: ns is the number of the sensitive decisions, ng is the number of the guard conditions that process protected characteristics and proxy information, and nv is the number of values in the range of a sensitive decision. Although this may be relatively large, we assume that the value that can be assigned to a decision data-attribute belongs to a discrete set.

Third, a proxy generator that can be invoked in the absence of proxy information. The proxy generator parses the models and returns a text file that contains all the data attributes that are annotated as protected characteristics. Then the proxy generator identifies proxy information for the returned protected characteristics using correlation metrics (e.g., conditional entropy) and a database that contains personal information.

Forth, a launch configuration that calls the automatic transformation tool Hugo/RT. The launch configuration takes as an input a UML model in XML format, which will passed to Hugo/RT. The output is a (.pml) file contains the PROMELA specifications of the transformed model.

Fifth, a fairness reporter that takes as input a (.xlsx) file that contains the generated LTL formulas with the checking results. The reporter returns a (.html) report that describes the detected violations of individual fairness.

IX. The Applicability of MBFair

We studied the applicability of MBFair to assess its ability in detecting discriminatory cases by verifying software models. For this purpose, we applied MBFair to three case studies.

The Bank Services Management System is a case study inspired by a decision process in a banking system. The case study describes situations where bank customers can apply for services organized by the bank. We designed the UML model of the bank services system based on the available description for getting a free-account in the bank. An excerpt of the model is shown in Fig. 1. We also created a data set of personal

[1]Our implementations are anonymously available online at https://github.com/mbfairness/MBfair/blob/main/README.md

information based on real statistics [24, 25, 26], that show high correlations between age, income and health status.

The Delivery Management System presents an incident based on Amazon's delivery management system. The UML model of the delivery system is designed based on the incident's description [29]. Amazon's software offered a free-delivery service for prime customers whose orders exceeds $35 and who live in zip-codes that are near to the locations of Amazon's stores. To uncover proxies information, we created a data set of artificial personal information. Our goal is to check if the model preserves the individual fairness with respect to the ethnicity when deciding about the free-delivery service.

The Loan Management System is based on a business process model (BPMN) [30] which has been generated from an event log of a financial institute [31]. The BPMN model consists of two main processes, namely the *loan request management* and the *risk analysis*. The former checks whether a loan request will be accepted. The latter creates a loan proposal for each accepted loan request and performs a risk analysis to decide whether the proposal will be granted. For our analysis, we semi-automatically transformed the BPMN model in [30] to a UML model using the transformation approaches in [32, 33]. Due to the lack of details about the data used in the BPMN model, we assumed the following: First, a loan request will be accepted if the applicant's account has no critical credit history and if the saving amount exceeds 1000 Euro. Second, the risk analysis process is a black box decision-making component. Our goal is to check if the risk analysis component will be invoked for every two loan applicants whose data differ only in the protected characteristics or their proxies. To uncover proxy information, we derived data from the Statlog Credit data set [34].

An overview of the analyzed UML models. The used UML models in our applicability study are of a medium size and all the modeled procedures are if-then-else rules that are almost similar to the example model in Fig. 1. An overview of the analyzed UML models is provided in Table III. The first column of the table provides the name of the analyzed model. The second column provides the total number of the UML elements in the analyzed models. The number of the elements includes the number of the classes, data attributes, operations, state machines, transitions and other elements. To check if MBFair can detect discrimination in UML models that have concurrent executions, an orthogonal state machine is considered in the model of the bank services system.

The third column of Table III provides the number of the UML elements that are manually annotated with our UMLfair profile. Based on the entries in the third column, one can notice that the users of MBFair will not put a lot of manual-effort for annotating the models. This is because, in general, decision-making software that should produce fair decisions represent a small percentage of the modeled system as a whole. For example, 2 out of 487 elements in the UML model of the bank system were annotated with our UMLfair profile.

The fourth column of Table III provides the number of the data attributes in the models. The fifth column shows the defined protected characteristics. The sixth column provides the proxy information. The proxy information were automatically generated based on the data sets of the case studies. A data attribute may be a proxy for a protected characteristic that is not used in the model. For example, the delivery system has only one protected characteristic that is directly declared in its model; namely "age". Nevertheless, Table III shows that "billingAddress" can act as a proxy for "ethnicity", which is a protected characteristic that is not used in the model.

TABLE III: Overview of the analyzed UML models

Model	Element	Annot.	Att.	Protected	proxy
Bank Services System	486	2	51	age	income, goodCreditHistory, healthy
				gender	-
				healthy	income, goodCreditHistory, healthy
Delivery System	289	2	42	age	-
				ethnicity	billingAddress
Loan System	279	2	41	age	saving, creditHistoryStatus
				gender	-
				citizenship	-

The number of protected characteristics and the proxy information in the analyzed models are small comparing with the total number of data attributes. This is because most often software engineer avoid using protected characteristics and proxy information directly in a decision-making software. Hence, we believe that experts will not put much manual-effort in identifying protected characteristics and proxy information in a UML model. For instance, Table III shows that 3 out 51 data attributes in the bank system model represent protected characteristics; namely "age", "gender" and "healthy".

Applying MBFair. After annotating the UML models, we automatically generated LTL formulas using our LTL generator. The formulas were then automatically checked, using the SPIN model checker [19]. Since SPIN only accepts PROMELA models, we first automatically transformed the UML models into PROMELA, using Hugo/RT [28]. The results of checking the formulas were then analyzed, using our fairness reporter. All the artifacts of this section are provided online[2].

Table IV summarizes the analysis results. The first and the second column provides the name of the analyzed UML model and the number of elements in it, respectively. The third column provides the time in milliseconds (ms) that is taken by Hugo/RT [28] to transform the UML model into PROMELA. The fourth and the fifth columns provide the number of the generated LTL formulas and the time taken by our LTL generator, respectively. The sixth column provides the time taken by SPIN [19] to check the formulas. The tests were performed on a computer with a 2.2 Ghz processor and 8 GB of memory.

Table IV also provides the number of detected violations of individual fairness. For each detected violation, Table IV provides: First, the protected characteristics, against which the violations occurred. Second, the source of the violation. Third, whether the violation happened due to a data flow or a direct usage for the source of the violation. For example, the table

[2]The artifacts of our case studies are anonymously available at https://github.com/mbfairness/MBfair/blob/main/README.md

TABLE IV: The Detected Individual-Fairness Violations

Model	Elements	Trans. Time [ms]	LTL Formulas	Generation Time [ms]	Check Time [ms]	Sensitive Decision	Violations	Against	Source	Through
Bank Services System	486	2240	8	528	161430	hasFree Account	4	age (2 times)	income	Data Flow
									healthy	Data Flow
								healthy (2 times)	income	Data Flow
									healthy	Data Flow
Delivery System	289	2260	4	600	57910	freeDelivery Status	1	ethnicity	billing Address	Direct Usage
Loan System	279	2190	8	589	108300	riskAnalysis() call operation	2	age (2 times)	creditHistory Status	Data Flow
									saving	Data Flow

shows two violations of individual fairness against "age" in the loan system when deciding about triggering the "riskAnalysis()" call-operation. The violations happened due to a flow for the "creditHistoryStatus" and the "saving" which (as provided in Table III) are proxies for the "age".

Discussion and limitations. Our case studies demonstrate how MBFair methodology allows for detecting violations of individual fairness at the software design phase. Our evaluation based on case studies gives an impression of the performance of the automated parts of MBFair for models of medium size. Based on Table IV, one can observe that the time taken by the automated parts of MBFair is small. Due to the difference in the sizes of the analyzed models, we notice differences in the times that are taken by SPIN model checker. However, it is not necessary to always analyze a large model as a whole. There are techniques to analyze subsystems in separation [35]. In future, we plan to analyze the time-complexity of the automatic parts of MBFair to report on their scalability.

Finally, our LTL generator considers only sensitive decisions of Boolean and Enumeration types. Decisions of other types, such as integer, are currently not supported. In future, we plan to integrate data discretization to convert sensitive decisions of continuous domains to discrete ones. In addition, MBFair is concerned only with analyzing rule-based software (if-then-else). A possibility for future work is to extend MBFair to support analyzing machine learning-based software with respect to the overall behavior of the system in question.

X. The Necessity of MBFair

We further aimed to study the necessity for an automated approach that analyses UML models to detect discrimination. We aimed at investigating the following hypotheses:

H1: Analysts overlook true-positive discrimination when they analyze UML models manually.

H2: The result of detecting discrimination manually has a low degree of reliability.

The set-up of the experiment. We studied the hypotheses H1 and H2 in a user experiment with 13 participants. specifically, 10 master students, and 3 researchers. Hence, we relied primarily on students as participants. We justify this choice with earlier research findings, according to which students perform nearly equally to experts when faced with a software development approach [36].

As a baseline, we assume the participants to be familiar with UML state-machine diagram. Therefore, we targeted participants who had relevant background in informatics and data science. To ensure understandability for the concepts and the models that are used in our experiment, we trained the participants by giving them a short online lecture in which the UML state machine and individual fairness were introduced.

After the lecture, we asked all participants to self-assess their expertise in UML, UML state machine, and software fairness on a 5-point Likert scale. For each background field, Table V reports the mean and average, response values, and the standard deviation. With expertise levels of 3.3/5 for UML, 2.69/5 for UML state machine, and 2.46/5 in software fairness (means of response values), the background of the participants is characterized by relevant knowledge, but not expert knowledge in UML and software fairness.

TABLE V: Experience levels of participants

	Mean	Median	St.dev
UML	3.3	3	0.6
UML state machine	2.69	3	0.99
Software fairness	2.46	3	0.63

We presented the participants with an online questionnaire and an auxiliary "cheatsheet". The cheatsheet consisted of definitions for fairness-related concepts that are used in the experiment. The questionnaire consisted of two tasks namely, manual discrimination detection and labeling cases of discrimination. In both tasks, we used a variant of the within-subject experimental design [37], in which all participants act as their own control. The tasks and their results are described below. The artifacts of our experiment are provided online[3].

Task 1. We showed the participants the state machine from the bank system in Fig.1. We also provided the participants with legal knowledge concerning the protected characteristics, proxy information, and sensitive decisions. Then we asked the participants to analyze the state machine manually to detect violations of individual fairness.

The results of Task 1: Only 54% of all participants agree that the bank system can discriminate between people on the basis of their protected characteristics when deciding on a free bank account application. Table VI gives an overview about the manually detected cases of discrimination. In total the participants detected 6 cases of discrimination. The first column of Table VI provides the detected cases of discrimination. Each case describes against which protected characteristic the

[3]The artifacts of our user experiment are anonymously available at https://github.com/mbfairness/userExp/blob/main/README.md

TABLE VI: The results of detecting discrimination manually

Manually Detected Case	Percentage of participants
Case1: against= "age", source= "income", through = Data Flow	54%
Case2: against = "age", source= "healthy", through= Data Flow	23%
Case3: against= "healthy", source="income", through= Data Flow	54%
Case4: against= "healthy", source= "healthy", through= Data Flow	31%
Case5: against= "age", source= "goodCreditHistory", through= Direct Usage	15%
Case6: against= "healthy", source= "goodCreditHistory", through= Direct Usage	8%

discrimination can happen, the source of discrimination and whether it can happen through a data flow or a direct usage for the source of discrimination. For example, Case1 describes a discrimination against "age" due to a data flow of "income", which is defined as a proxy for "age" in the proxy information of the experiment. The second column in Table VI provides the percentage of participants who detected the corresponding case. For example, 54% of all participants detected Case1.

Compared to the detected discrimination by our MBFair methodology when applied to the model (see Table IV), the cases 1-4 in Table VI represent true-positive discrimination. However, we noticed a variant-agreement between the participants in the true-positive discriminatory cases that can occur. For example, Case1 were detected by 54% of all participants while Case2 were detected by 23% of all participants. We also noticed that analysts reported false-positive discrimination. For instance, portions of 15% and 8% detected Case5 and Case6, respectively. But Case5 and Case6 represent false-positive discrimination because the "goodCreditHistory" does not belong to the proxy information of the experiment.

Task 2. We presented the UML state machine and legal knowledge from Task 1 to the participants. Different from Task 1, where we asked the participants to analyze the model manually and report the detected discriminatory cases, in Task 2 we presented the participants with 4 cases of individual fairness violations. For each case, the participant were asked to identify whether it actually occurs.

The results of Task 2: Table VII gives an overview about the results of Task 2. The cases of discrimination that we provided to the participants are provided in the first column. Each case describes against which protected characteristic the discrimination can happen, the source of discrimination and whether it can happen through a data flow or a direct usage for the source of discrimination. The second column of Table VII shows the percentage of participants who labeled the corresponding discriminatory case as true-positive discrimination.

The cases from 1-4 in Table VII are identical with the cases from 1-4 that are detected manually in Task 1. Comparing with the results of Task 1, we notice positive variants in the participants' performance in Task 2. For example, in Task 1, only 23% of all participants detected Case2. But when Case2 is presented to the participants as a potential case in Task 2, the positive tendency increased to 77%.

TABLE VII: The results of labeling discriminatory cases

Manually Detected Case	Percentage of participants
Case1: against= "age", source= "income", through = Data Flow	54%
Case2: against = "age", source= "healthy", through= Data Flow	77%
Case3: against= "healthy", source="income", through= Data Flow	69%
Case4: against= "healthy", source= "healthy", through= Data Flow	62%

Concerning hypothesis H1: First, the weak performance in detecting discrimination manually in Task 1 (54%), indicates differences in the individuals' abilities in keeping a beady eye on the protected characteristic and their proxies while analyzing models. For example, although "healthy" was highlighted as a proxy for "age" in the provided proxy information to the participants, only a minority of 23% were able to detect Case2 manually in Task 1. Second, the variant in the participants' performance in Task 1 and Task 2 indicate differences in the individuals' abilities in tracking the data flow in the model. For instance, in Task 1, only 23%, 54%, 31% of all participants detected Case2, Case3 and Case4, respectively. But, in Task 2, the positive tendency increased to 77%, 69% and 62% respectively. These findings support our hypothesis H1.

Concerning hypothesis H2: we used the statistical measure, called Fleiss' kappa [38], to assess the inter-rater reliability of the participants' agreement with the cases of discrimination that can happen. The values of kappa κ range between <0 (poor agreement) and 1 (perfect agreement) indicating how much multiple judges agree with their decisions. Hence, the results are unbiased with our expected answers as each answer given by a participant is taken into account equally. Our results shows a low inter-reliability agreement between the participants. For Task 1, $\kappa = 0,077$ which indicates a slight agreement. For Task 2, $\kappa = -0,047$ which indicates a poor agreement. This result supports our hypothesis H2.

Since analysts overlook true-positive discriminatory cases when they analyze UML models manually (H1) and since the inter-reliability agreement between analysts on the discriminatory cases that can occur is low (H2), we conclude that analysts require support for fairness analysis, such as our MBFair methodology.

Threats to validity. A threat to e*xternal validity* is that our experimental material was based on one particular case study. Hence, the obtained results may not generalize to other case studies. A comprehensive study with a greater variety of case studies is left to future work.

A threat to *internal validity* is experimenter expectation. In Task 2 of our experiment, participants might have assessed the cases of discrimination without having understood the specification of the model. We point out that the participants were presented with detailed insights as provided by our training lecture. However, we did not examine the qualifications of the participants after the lecture. We believe that a more comprehensive training would have improved the results.

XI. Related Work

The works in the software fairness field can be classified into three broad categories: First, *understanding fairness:* works that aim at providing (un)formalized fairness definitions and help to understand how discrimination can happen in our systems [7, 39, 40]. Second, *mitigating discrimination:* works that aim at preventing discrimination. Approaches in this direction focus mainly on tackling discrimination in AI-based software [41, 42, 43]. Third, *discrimination detection:* works that aim at test/verify whether a system is fair with respect to certain fairness measure [5, 15, 44]. Despite the availability of many approaches that aim at detecting discrimination, we have not found an existing approach described in the literature that would use UML models for fairness analysis. In the following, we discuss the closest approaches to our work.

Model-based analysis. The need for integrating fairness in the design phase of software systems is first has been highlighted by the work in [13, 14]. However, the work in [13, 14] are not supported by an approach that realizes the idea of analyzing fairness based on UML models.

Temporal logic and model checking have been widely used in the literature for analyzing UML models against difficult system requirements [45]. The work in [46] is the most relevant work to this paper. The authors propose a UML-based approach that uses temporal logic and model checking to reason about cryptographic related. Other closest UML-based analysis checks to the MBFair methodology are: the *no down flow-check* in [27] and the *purpose-check* in [47]. According to [27], the no down flow-check verifies whether a system model produces the same outputs for any two traces of events that differ only in their secret data. However, different from our work, the no down flow-check does not consider proxies for secret data through the analysis. According to [47], the purpose-check can be used to identify whether a system processes users' data only for authorized purposes. With the purpose-check, one can detect unauthorized direct discrimination, but not indirect discrimination that may result from a data flow of protected characteristics or their proxies.

In [48], the authors propose a BPMN-based framework that supports the design of business processes considering data protection-related requirements, including fairness requirements. The goal of the work in [48] is to detect conflict between fairness, security, and data-minimization requirements.

Discrimination detection approaches. Most of the proposed discrimination detection approaches in the literature can be used as fairness testing components during the *testing phase* of the software development life cycle, which is only after the software is implemented. Most of the proposed approaches in this direction are *black-box* approaches. Approaches in this direction apply different methods to study the changes in the overall software's output based on historical observations.

For example, FairTest [15] and Themis [5] are two methods that uncover discrimination in a software system by studying the correlations between its outputs and its inputs. Different from the FairTest [15], which requires a data set of possible inputs to be given, Themis [5] automatically generates input test cases based on a schema describing valid software inputs.

The authors of [49, 50] proposed a method that extracts classification rules from a dataset of historical decision records. The classification rules are then directly mined to search for discrimination with respect to protected characteristics and their proxies. In [51, 52], the authors propose an algorithm for detecting discrimination by analyzing a causal network that captures the causal structure among the data stored in a historical decision records. In [53], the authors deploy privacy attack strategies to detect discrimination under hard assumptions about a given data set of historical decision records, such as that the data set is preprocessed to hide illegal discrimination from being detected. In [54], a tool called AdFisher is proposed. AdFisher aims to detect violations of individual fairness with respect to the ads that web users receive when visiting a web page. AdFisher monitors the changes in the ads based on web users' behaviors and privacy preferences specified in their web browsers.

The main drawback of black-box approaches is their inability to provide witnesses that allow explaining the source of detected discrimination. To address this challenge, in [10], the authors proposed a *white-box* approach that analyzes decision-making software and returns witnesses, that describe how discrimination happened. FairSquare [11, 12] is another white-box approach that concerned with detecting discrimination *at run-time*. FairSquare verifies probabilistic assertions in the source code of decision-making software during its execution.

Different from the MBFair methodology, all the discussed approaches above require the software to be implemented. Hence, none of them can be used as a verification component during the design of software models.

XII. Conclusion and Future Work

We proposed MBFair, a model-based verification methodology, that applies model checking to UML models for detecting violations of individual fairness. We validated the applicability of MBFair using three case studies featuring a bank services system, a delivery system and a loan system. The result shows a promising outlook on the applicability of our MBFair in real-world settings as it detected all the true-positive cases of discrimination in the case studies as expected.

Using a user experiment, we reported on the necessity for an automated approach that analyses UML models to detect discrimination. The result shows that analysts overlook true-positive discriminatory cases when they analyze UML models manually. It also shows that the inter-reliability agreement between analysts on the discriminatory cases that can occur is low. These results indicate that analysts require support for fairness analysis, such as the MBFair methodology.

In future, we plan to extend MBFair to allow reasoning about *group fairness*. A software system preserves group fairness if it produces equally distributed outputs for each protected group [7]. Extending MBFair to permit group fairness analysis requires applying probabilistic model checking [55], a technique for verifying quantitative properties of systems.

References

[1] L. Carmichael, S. Stalla-Bourdillon, and S. Staab, “Data mining and automated discrimination: a mixed legal/technical perspective,” *IEEE Intelligent Systems*, vol. 31, no. 6, pp. 51–55, 2016.

[2] S. Staab, S. Stalla-Bourdillon, and L. Carmichael, “Observing and recommending from a social web with biases,” *arXiv preprint arXiv:1604.07180*, 2016, uRL https://arxiv.org/abs/1604.07180 (accessed: 15/02/2021).

[3] T. Zarsky, “The trouble with algorithmic decisions: An analytic road map to examine efficiency and fairness in automated and opaque decision making,” *Science, Technology, & Human Values*, vol. 41, no. 1, pp. 118–132, 2016.

[4] “Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data.” Official Journal of the European Union, 2016.

[5] S. Galhotra, Y. Brun, and A. Meliou, “Fairness testing: Testing Software for Discrimination,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 498–510.

[6] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel, “Fairness Through Awareness,” in *Proceedings of the 3rd innovations in theoretical computer science conference*. ACM, 2012, pp. 214–226.

[7] S. Verma and J. Rubin, “Fairness definitions explained,” in *2018 IEEE/ACM International Workshop on Software Fairness (FairWare)*. IEEE, 2018, pp. 1–7.

[8] Q. Ramadan, A. S. Ahmadian, J. Jürjens, S. Staab, and D. Strüber, “Explaining algorithmic decisions with respect to fairness,” *Software Engineering and Software Management 2019*, 2019.

[9] C. Dwork and C. Ilvento, “Fairness Under Composition,” *arXiv preprint arXiv:1806.06122*, 2018, uRL https://arxiv.org/abs/1806.06122 (accessed: 15/02/2021).

[10] A. Datta, M. Fredrikson, G. Ko, P. Mardziel, and S. Sen, “Proxy non-discrimination in data-driven systems,” *arXiv preprint arXiv:1707.08120*, 2017, uRL https://arxiv.org/abs/1707.08120 (accessed: 15/02/2021).

[11] A. Albarghouthi, L. D’Antoni, S. Drews, and A. V. Nori, “FairSquare: Probabilistic Verification of Program Fairness,” *Proceedings of the ACM on Programming Languages*, no. Object-Oriented Programming, Systems, Languages & Applications, 2017.

[12] A. Albarghouthi and S. Vinitsky, “Fairness-aware Programming,” in *Proceedings of the Conference on Fairness, Accountability, and Transparency*. ACM, 2019, pp. 211–219.

[13] Y. Brun and A. Meliou, “Software Fairness,” in *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 754–759.

[14] Q. Ramadan, A. S. Ahmadian, D. Strüber, J. Jürjens, and S. Staab, “Model-based discrimination analysis: a position paper,” in *2018 IEEE/ACM International Workshop on Software Fairness (FairWare)*. IEEE, 2018, pp. 22–28.

[15] F. Tramèr, V. Atlidakis, R. Geambasu, D. J. Hsu, J.-P. Hubaux, M. Humbert, A. Juels, and H. Lin, “Discovering Unwarranted Associations in Data-Driven Applications with the Fairtest Testing Toolkit,” *CoRR, abs/1510.02377*, 2015.

[16] “The German General Act on Equal Treatment,” 2006, uRL http://www.gesetze-im-internet.de/englisch_agg/ (accessed: 15/02/2021).

[17] D. E. Denning, “A Lattice Model of Secure Information Flow,” *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[18] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge university press, 2004.

[19] “SPIN.” Bell Labs in the Unix group of the Computing Sciences Research Center, 1991, uRL http://spinroot.com/spin/whatisspin.html (Version 6.4.9, release date: 2018, accessed: 15/02/2021).

[20] G. J. Holzmann, “The model checker spin,” *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[21] K. Y. Rozier, “Linear temporal logic symbolic model checking,” *Computer Science Review*, vol. 5, no. 2, pp. 163–203, 2011.

[22] G. J. Holzmann, “The model checker spin,” *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[23] “OMG® Unified Modeling Language® (OMG UML®) Version 2.5.1,” no. formal/2017-12-05. Object Management Group (OMG), December 2017, uRL https://www.omg.org/spec/UML/2.5.1/PDF (accessed: 15/02/2021).

[24] “Chapter 1: life expectancy and healthy life expectancy,” Public Health England. GOV.UK, 2017, uRL http://shorturl.at/bzBEW (accessed: 15/02/2021).

[25] “Distribution of median and mean income and tax by age range and gender.” Office for National Statistics, 2020.

[26] O. Lenhart, “The effects of income on health: new evidence from the earned income tax credit,” vol. 17, no. 2. Springer, 2019, pp. 377–410.

[27] J. Jürjens, *Secure Systems Development with UML*. Springer Science & Business Media, 2005.

[28] “Hugo/RT.” Departement of computer science, Augsburg University, 2015, uRL https://www.informatik.uni-augsburg.de/en/chairs/swt/sse/hugort/ (Version 0.8a, accessed: 15/02/2021).

[29] D. Ingold and S. Soper, “Amazon Doesn’t Consider the Race of Its Customers. Should It?” bloomberg.com, Ed., April 2016, uRL https://www.bloomberg.com/graphics/2016-amazon-same-day/ (accessed: 15/02/2021).

[30] M. Salnitri, M. Alizadeh, D. Giovanella, N. Zannone, and P. Giorgini, “From security-by-design to the identification of security-critical deviations in process executions,” in *International Conference on Advanced Information Systems Engineering*. Springer, 2018, pp. 218–234.

[31] “BPI 2012,” 2012, uRL https://www.win.tue.nl/bpi/doku.php?id=2012:challenge (accessed: 15/02/2021).

[32] Q. Ramadan, M. Salnitriy, D. Strüber, J. Jürjens, and P. Giorgini, “From Secure Business Process Modeling to Design-level Security Verification,” in *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2017, pp. 123–133.

[33] A. Rodríguez, E. Fernández-Medina, J. Trujillo, and M. Piattini, “Secure business process model specification through a UML 2.0 activity diagram profile,” *Decision Support Systems*, vol. 51, no. 3, pp. 446–465, 2011.

[34] H. Hofmann, “Statlog (german credit data) data set,” Hamburg University. UCI Machine Learning Repository, 1994, uRL https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data) (accessed: 15/02/2021).

[35] M. Ochoa, J. Jürjens, and D. Warzecha, “A sound decision procedure for the compositionality of secrecy,” in *International Symposium on Engineering Secure Software and Systems*. Springer, 2012, pp. 97–105.

[36] I. Salman, A. T. Misirli, and N. Juristo, “Are students representatives of professionals in software engineering experiments?” in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 1. IEEE, 2015, pp. 666–676.

[37] G. Charness, U. Gneezy, and M. A. Kuhn, “Experimental methods: Between-subject and within-subject design,” *Journal of Economic Behavior & Organization*, vol. 81, no. 1, pp. 1–8, 2012.

[38] J. L. Fleiss, “Measuring nominal scale agreement among many raters.” *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.

[39] E. Ntoutsi, P. Fafalios, U. Gadiraju, V. Iosifidis, W. Nejdl, M.-E. Vidal, S. Ruggieri, F. Turini, S. Papadopoulos, E. Krasanakis *et al.*, “Bias in data-driven artificial intelligence systems—an introductory survey,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 10, no. 3, p. e1356, 2020.

[40] B. Hutchinson and M. Mitchell, “50 years of test (un) fairness: Lessons for machine learning,” in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2019, pp. 49–58.

[41] F. Calmon, D. Wei, B. Vinzamuri, K. N. Ramamurthy, and K. R. Varshney, “Optimized pre-processing for discrimination prevention,” in *Advances in Neural Information Processing Systems*, 2017, pp. 3992–4001.

[42] M. B. Zafar, I. Valera, M. Gomez Rodriguez, and K. P. Gummadi, “Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment,” in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 1171–1180.

[43] F. Kamiran, S. Mansha, A. Karim, and X. Zhang, “Exploiting reject option in classification for social discrimination control,” *Information Sciences*, vol. 425, pp. 18–33, 2018.

[44] J. A. Adebayo *et al.*, “Fairml: Toolbox for diagnosing bias in predictive modeling,” Ph.D. dissertation, Massachusetts Institute of Technology, 2016.

[45] M. d. M. Gallardo, P. Merino, and E. Pimentel, “Debugging UML Designs with Model Checking,” *Journal of Object Technology*, vol. 1, no. 2, pp. 101–117, 2002.

[46] J. Jürjens and P. Shabalin, “Tools for Secure Systems Development with UML,” *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 5-6, pp. 527–544, 2007.

[47] A. S. Ahmadian, S. Peldszus, Q. Ramadan, and J. Jürjens, “Model-Based Privacy and Security Analysis with CARiSMA,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 989–993.

[48] Q. Ramadan, D. Strüber, M. Salnitri, J. Jürjens, V. Riediger, and S. Staab, “A semi-automated bpmn-based framework for detecting conflicts between security, data-minimization, and fairness requirements,” *Software and Systems Modeling*, pp. 1–37, 2020.

[49] S. Ruggieri, D. Pedreschi, and F. Turini, “Data Mining for Discrimination Discovery,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 4, no. 2, p. 9, 2010.

[50] D. Pedreschi, S. Ruggieri, and F. Turini, “Integrating Induction and Deduction for Finding Evidence of Discrimination,” in *Proceedings of the 12th International Conference on Artificial Intelligence and Law*. ACM, 2009, pp. 157–166.

[51] L. Zhang, Y. Wu, and X. Wu, “On Discrimination Discovery Using Causal Networks,” in *International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation*. Springer, 2016, pp. 83–93.

[52] L. Zhang and X. Wu, *International Journal of Data Science and Analytics*, vol. 4, no. 1, 2017.

[53] S. Ruggieri, S. Hajian, F. Kamiran, and X. Zhang, “Anti-discrimination Analysis Using Privacy Attack Strategies,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2014, pp. 694–710.

[54] A. Datta, M. C. Tschantz, and A. Datta, “Automated Experiments on Ad Privacy Settings,” *Proceedings on privacy enhancing technologies*, vol. 2015, no. 1, pp. 92–112, 2015.

[55] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, “Automated verification techniques for probabilistic systems,” in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2011, pp. 53–113.

XIII. Appendix

In what follows we explain how we identify proxies of protected characteristics based on historical data. We also describe the algorithm of our LTL generator and the algorithm of our fairness reporter.

(A) Generating Proxies from a database

In the MBFair methodology, when expert knowledge of proxy information is not available, a database that contains personal data of a true distribution can be a source for identifying such proxy information. To find whether a data attribute in a UML model is a proxy of a protected characteristic, the data attributes in the class diagram have to be mapped to the database schema. The mapping could be given by the user or determined (semi-)automatically. Proxies will be statistically derived by using a correlation metric and a threshold for being a proxy. Various correlation metrics have been used in the literature to identify proxies such as the conditional probability, information gain, and others [15]. The metric and the threshold can be provided by domain experts. In the following, we show through an example how we identify proxies of protected characteristics based on historical data.

TABLE VIII: Customer data.

id	age	income	goodCredit History	healthy	income*
1	Young	1500	False	True	False
2	Young	1800	False	True	False
3	MidAge	2800	False	True	True
4	MidAge	3800	False	True	True
5	MidAge	2500	False	True	True
6	MidAge	3500	False	True	True
7	MidAge	2700	False	True	True
8	Old	2300	True	False	False
9	Old	2200	True	False	False
10	Old	1800	True	False	False

TABLE IX: Results of uncertainty measurement.

Conditional Entropy	Uncertainty
Entropy(age | goodCreditHistory)	0.60
Entropy(age | income*)	0.48
Entropy(age | healthy)	0.60
Entropy(healthy | goodCreditHistory)	0.0
Entropy(healthy | income*)	0.60
Entropy(healthy | age)	0.0

Table VIII contains data based on real statistics [24, 25, 26]. The statistics show a high correlation between age, income and health status. The attributes namely “age”, “income”, “goodCreditHistory”, and “healthy” are identical to those in Fig. 1. Before looking for correlations, each data attribute in the database with a large range of possible values has to be transformed into a Boolean data attribute based on its usage context in the UML model. For example, based on the “(income >= 2500)” condition in Fig. 1(b), the model will deal with any value of the “income” as either a value that can evaluate the “(income >= 2500)” condition to a true or a false. Therefore, a new column called “income*” is added to Table VIII. The “income*” shows “True” if the corresponding cell in the “income” column satisfies the “(income >= 2500)” condition and “False” otherwise.

To identify proxies, we use the “conditionalEntropy” and a threshold equals to “0.6”. The conditional entropy measures the *uncertainty* in a data attribute given another data attribute. The output of the conditional entropy is a value between “0” and “1”, where “0” means low uncertainty and “1” means high uncertainty. The results are summarized in Table IX. Each data attribute that reduces the uncertainty in a protected characteristic to the specified threshold or less is a proxy for that proctored characteristic. Concerning the results provided in Table IX, the following data attributes are considered as proxies: “goodCreditHistory” and “income*” are both proxies for “age” and “healthy”. We also identify that “age” and “healthy” as proxies for each other.

(B) LTL generator

For generating LTL formulas, our LTL generator considers every atomic guard condition g whose data attribute has the following properties:

- First, the data attribute is not derived.
- Second, the data attribute is defined as a protected characteristic in the *{protectedData}* tag or is a proxy of a protected characteristic, that does not belong to the explanatory knowledge of the sensitive decision s in the *{explanatoryKnowledge}* tag.

With the first property, we avoid generating LTL formulas with respect to masks of proxies. For example, in Fig. 1, “ownPoints” is a derived attribute that depends on “income” and “healthy”, which are proxies for “age”. In this case, “ownPoints” is a mask for “income” and “healthy”. Hence, knowing that a discrimination happened because of using “ownPoints”, will not uncover the original source of the discrimination.

With the second property, we avoid generating LTL formulas with respect to explanatory knowledge. In Fig. 1, “goodCreditHistory” is defined as a proxy for “age”. But nevertheless our LTL formulas generator ignores the “goodCreditHistory” when generating LTL formulas with respect to “hasFreeAccount”, because the “goodCreditHistory” is defined as explanatory for “hasFreeAccount” in Fig. 1.

In this paper, an atomic guard condition g whose data attribute satisfies the properties above is called a *used-condition*. To explain the generation of sets of LTL formulas with respect to a state machine sm annotated with the *«individualFairness»* stereotype, we assume the following:

- $s_i \in SensitiveDecisions$, where $SensitiveDecisions$ is a set of size n and it contains all the data attributes and call operations that are defined in the *{sensitiveDecisions}* tag of sm.
- $g_j \in usedConditions_{s_i}$, where $usedConditions_{s_i}$ is the set of the used conditions with respect to s_i and it is of the size k.
- $v_h \in V_{s_i}$, where V_{s_i} is the set of all values in the range space of a sensitive decision s_i and it is of the size l.

- $event\_queues?[call\_s_i]$ is a logical condition. It returns true if the call-operation event s_i belongs to the $event\_queues$ set, which includes the triggered events while executing the PROMELA model M that is corresponding to the UML model being analyzed. Otherwise, it returns false.

Given the above notations, Def. 1 shows how we define the set B, that represent the output of our LTL formulas generator with respect to a state machine sm annotated with the *«individual-Fairness»* stereotype.

Definition 1 (Sets of LTL formulas): Given sm is an *«individual-Fairness»*-annotated state machine in a UML model, the set B of all sets of the LTL formulas with respect to sm is defined as follow:

$$B=\Big\{ \bigcup_{s_i=1}^{s_n} \bigcup_{g_j=1}^{g_k} set_{s_i g_j} \Big\}, where:$$

(Rule 1) if s_i is a data attribute:

$$set_{s_i g_j}=\Big\{ \bigcup_{v_h=1}^{v_l} \Big(g_j \rightarrow \Diamond\Box(s_i{=}{=}v_h), \neg g_j \rightarrow \Diamond\Box(s_i{=}{=}v_h) \Big) \Big\}$$

(Rule 2) if s_i is a call operation:

$$set_{s_i g_j}= \Big\{ \Big(g_j \rightarrow \Diamond event\_queues?[call\_s_i], \neg g_j \rightarrow \Diamond event\_queues?[call\_s_i] \Big), \Big(g_j \rightarrow \neg\Diamond event\_queues?[call\_s_i], \neg g_j \rightarrow \neg\Diamond event\_queues?[call\_s_i] \Big) \Big\}$$

According to Def. 1, for each sensitive decision s_i with k used conditions, k sets have to be generated, each set with respect to a used condition g_j of s_i. When s_i is a *data attribute*, a $set_{s_i g_j}$ has to be defined according to Rule 1 of Def. 1. In this case, the $set_{s_i g_j}$ is a set of pairs. According to Rule 1, the number of the pairs in a $set_{s_i g_j}$ is l, where l is the number of the possible values that can be assigned to a decision data-attribute s_i. Following Rule 1, each pair belongs to a $set_{s_i g_j}$ consists of two formulas. The first formula asserts that there is some instantiation of (non-derived) data attributes in the model and some execution, such that if the g_j condition is $true$, later at some point ($\Diamond$) the value of the sensitive decision s_i equals v_h always ($\Box$), where v_h is a possible value that can be assigned to s_i.

The second formula of Rule 1 asserts that there is some instantiation of (non-derived) data attributes in the model and some execution, such that if the g_j condition is *false*, later at some point ($\Diamond$) the value of the sensitive decision s_i equals v_h always ($\Box$). For instance, recall the pairs p1 and p2 in Table IIa. By Rule 1, p1 and p2 represent a set of formulas with respect to the "DecisionsMaking" state machine in Fig. 1(b).

When s_i is a *call-operation* event, a $set_{s_i g_j}$ has to be defined according to Rule 2 of Def. 1. In this case, the $set_{s_i g_j}$ consists of two pairs only. This is because an operation is either can be invoked or never invoked. We are not using the always ($\Box$) symbol in the formulas of Rule 2 because a call-operation event can be triggered several times at different points in a single execution.

The first formula of Rule 2 in Def. 1 asserts that there is some instantiation of (non-derived) data attributes in the system and some execution, such that if the g_j condition is $true$, later at some point ($\Diamond$) the call-operation event s_i is added to the $event\_queues$ (i.e., s_i is triggered). The second formula asserts that there is some instantiation of (non-derived) data attributes in the system and some execution, such that if the g_j condition is *false*, later at some point ($\Diamond$) the call-operation event s_i is added to the $event\_queues$. The following *Example* explains how our generator generate LTL formulas with respect to a sensitive call-operation event.

Example. Consider the state machine in Fig. 3. In this state machine, if the "ownPoints" is greater than or equals to 70, the "predict()" call-operation event will be triggered, which then will call a black box operation that implements a free account decision-making algorithm. Assume that: (1) the "VerifyingFreeAccountApplication" state machine in our example model shown in Fig. 1(b) is replaced with the state machine in Fig. 3. (2) "predict()" is defined as a sensitive call-operation event in the *{sensitiveDecisions}* tag of the "DecisionsMaking" state machine in Fig. 1(b).

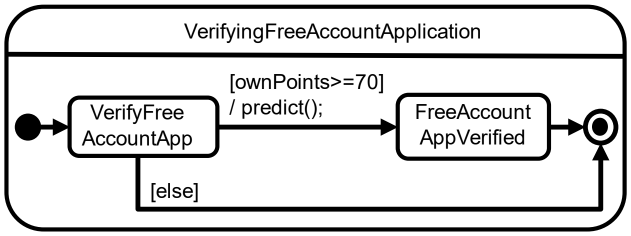


Fig. 3: Example: call-operation event.

Based on the assumptions above, by Rule 2 of Def. 1, the LTL formulas in Table X represents a set with respect to the "DecisionsMaking" state machine. There might be different decision algorithms in the system. With the LTL formulas in Table X we can check whether that the same algorithm is called in the given cases. Specifically, we can automatically check, using a model checker, whether the *predict* event will be triggered for every two free account applicants which have identical data except for the *income*. Further analysis is required to ensure the fairness of the called algorithm. In the following we describe the algorithm of our LTL generator.

The algorithm of LTL formulas generator. Algorithm 1 describes the generation of LTL formulas.

The LTL formulas generator takes as input a UML model m annotated with the UMLfair profile and a list proxy information $proxList$. The algorithm returns B (the set of all

TABLE X: A batch with respect to a call-operation event

Pair	LTL Formula
p1	$income{>=}2500 \rightarrow \lozenge event\_queues?[call\_predict]$
	$\neg income{>=}2500 \rightarrow \lozenge event\_queues?[call\_predict]$
p2	$income{>=}2500 \rightarrow \neg\lozenge event\_queues?[call\_predict]$
	$\neg income{>=}2500 \rightarrow \neg\lozenge event\_queues?[call\_predict]$

Algorithm 1: LTL formulas generator

```
1  generateBatchesOfFormulas ($m, proxList$);
   Inputs : a UMLfair-annotated model $m$ and a a list of
            proxy information $proxList$
   Output: a collection of sets of LTL formulas $B$
2  $sm \leftarrow getIndividualFairnessStateMachine(m)$;
3  $protectedSet \leftarrow \emptyset$;
4  $sensitiveSet \leftarrow \emptyset$;
5  $guardSet \leftarrow \emptyset$;
6  $protectedSet \leftarrow getProtected(m)$;
7  $sensitiveSet \leftarrow getSensitive(sm)$;
8  $guardSet \leftarrow getGaurds(m)$;
9  $B \leftarrow \emptyset$ ;
10 foreach $s \in sensitiveSet$ do
11     $explanatorySet \leftarrow \emptyset$;
12     $explanatorySet \leftarrow getExplanatory(sm, s)$;
13     $usedConditionsSet \leftarrow \emptyset$;
14     $usedConditionsSet \leftarrow getUsedConditions(m, s,$
         $proxList, protectedSet, explanatorySet, guardSet)$;

15     if $s$ is a data attribute then
16         $rangeSpaceSet \leftarrow \emptyset$;
17         $rangeSpaceSet \leftarrow getRangeSpace(s, m)$;
18         foreach $g \in usedConditionsSet$ do
19             $set \leftarrow \emptyset$;
20             foreach $v \in rangeSpaceSet$ do
21                 $set.add(\{g \rightarrow \lozenge\Box(s == v)\},$
                     $\{!g \rightarrow \lozenge\Box(s == v)\})$;
22             end
23             $B.add(set)$;
24         end
25     end
26     else
27         foreach $g \in usedConditionSet$ do
28             $set \leftarrow \emptyset$;
29             $set.add($
30             $\{g \rightarrow \lozenge event\_queues?[call\_s]\},$
                 $\{!g \rightarrow \lozenge event\_queues?[call\_s]\})$;
31             $set.add($
32             $\{g \rightarrow \lozenge !event\_queues?[call\_s]\},$
                 $\{!g \rightarrow \lozenge !event\_queues?[call\_s]\})$;
33             $B.add(set)$;
34         end
35     end
36 end
37 return $B$
```

sets of LTL formulas) with respect to sm, where sm is an *«individual-Fairness»*-annotated state machine in m.

First, in line 2 of the algorithm, the state machine that is annotated with the *«individual-Fairness»* stereotype will be retrieved. In the lines from 3-8, the followings are declared and initialized:

1) the $protectedSet$. A set contains all the data attributes that are defined as protected characteristics in the *{protectedData}* tags of the UML model m.
2) the $sensitiveSet$. A set contains all the data attributes that are defined as sensitive decisions in the *{sensitiveDecisions}* of sm.
3) the $guardSet$. A set contains all atomic guards conditions in the model m.

In line 9 of Algorithm 1, an empty set B is declared. This set will be used to store all sets of LTL formulas that can be generated. In line from 10-14 of the algorithm, for each sensitive decision s, the followings will be performed:

1) in the lines from 11-12, the explanatory data with respect to s will be retrieved and stored in the *explanatorySet*.
2) in the lines from 13-14, all the used conditions in m with respect to s will be stored into the *usedConditionsSet*. This will be performed by calling the $getUsedConditions$ function. This function takes as input a UML model m, the sensitive decision s, the list of proxies $proxList$, the set of protected data $protectedSet$, the set of explanatory characteristics *explanatorySet* and the set of the atomic guard conditions $guardSet$ in the model. The function returns the used conditions in m with respect to the sensitive decision s.

In the line 15, for each sensitive decision s that is defined as a data attribute in m , the followings will be performed:

1) in the lines from 16-17, all the possible values that can be assigned to s will be retrieved from the UML model m and stored in the $rangeSpaceSet$.
2) in the lines from 18-24, for each used condition g with respect to the sensitive decisions s, a set of LTL formulas will be defined as follows: for each value belongs to the $rangeSpaceSet$, a pair of formulas will be defined and added to the set. Each pair should have the format of the pair in line 21 of Algorithm 1. After iterating overall values in $rangeSpaceSet$, the set will be added to the set of all sets of LTL formulas B.

In the line 27, for each sensitive decision s that is defined as a call-operation event in m, the followings will be performed:

1) in the lines from 28-34, for each used condition g with respect to the sensitive decisions s, a set of LTL formulas will be defined. Each set must consist of two formulas that conform to the format of the formulas in line 30 of the algorithm.
2) in the line 33, each generated set will be added to the set of all sets of LTL formulas B.

(C) The individual fairness reporter

Algorithm 2 explains the functionality of our individual fairness reporter. Algorithm 2 takes as inputs: (1) the set of generated sets of LTL formulas B with respect to a state machine sm annotated with *«individual-Fairness»* stereotype. (2) R a set contains the results of checking the formulas in B against the PROMELA model M of the targeted UML model m by using a model checking technique. Algorithm 2 returns $fairnessReport$ that shows the detected violations of individual fairness.

Algorithm 2: Individual fairness reporter

```
1  indFairness ($B, R$);
   Inputs : a set of sets of LTL formulas $B$ and a set
            contains the formulas' checking results $R$
   Output: $fairnessReport$ a report shows the result of
            the individual fairness analysis
2  create $fairnessReport$;
3  foreach $set \in B$ do
4      if ($\exists$ pair $\in$ set whose formulas are satisfied while
         the formulas of set $\setminus\{pair\}$ are unsatisfied) OR
         (all formulas $\in$ set are satisfied) then
5          do nothing;
6      end
7      else
8          create $report$;
9          $report$.add($getResutls(set, R)$);
10         $fairnessReport$.add("a violation for individual
             fairness in the model is detected"+ $report$);
11     end
12 end
13 if fairnessReport is empty then
14     $fairnessReport$.add("the analyzed state machine
         preserves individual fairness");
15 end
16 return $fairnessReport$;
```

First, in line 2 of Algorithm 2, an empty *fairnessReport* will be created. In line 3, for each set in B, if one of the following is true, we do nothing:

- if there is only one pair in the set whose formulas are satisfied while the formulas of other pairs in the set are all unsatisfied.
- if all the formulas in the set are satisfied.

Otherwise, a violation of individual fairness will be reported together with all verification results of the LTL formulas in that set. In line 13, if after iterating over all the sets in B the fairness report remains empty, the following sentence will be added to the fairness report: *"the analyzed state machine preserves individual fairness"*.