Monitoring Electronic Exams

Ali Kassem[1], Yliès Falcone[2], and Pascal Lafourcade[3]

[1] Univ. Grenoble Alpes, VERIMAG, Grenoble, France Ali.Kassem@imag.fr
[2] Univ. Grenoble Alpes, Inria, LIG, Grenoble, France Ylies.Falcone@imag.fr
[3] Université Clermont Auvergne, LIMOS, France Pascal.Lafourcade@udamail.fr

Abstract. Universities and other educational organizations are adopting computer-based assessment tools (herein called *e-exams*) to reach larger and ubiquitous audiences. While this makes examination tests more accessible, it exposes them to unprecedented threats not only from candidates but also from authorities, which organize exams and deliver marks. Thus, e-exams must be checked to detect potential irregularities. In this paper, we propose several monitors, expressed as Quantified Event Automata (QEA), to monitor the main properties of e-exams. Then, we implement the monitors using MarQ, a recent Java tool designed to support QEAs. Finally, we apply our monitors to logged data from real e-exams conducted by *Université Joseph Fourier* at pharmacy faculty, as a part of *Epreuves Classantes Nationales informatisées*, a pioneering project which aims to realize all french medicine exams electronically by 2016. Our monitors found discrepancies between the specification and the implementation.

1 Introduction

Electronic exams, also known as *e-exams*, are computer-based systems employed to assess the skills, or the knowledge of candidates. Running e-exams promises to be easier than running traditional pencil-and-paper exams, and cheaper on the long term. E-exams are deployed easily, and they are flexible in where and when exams can be set; their test sessions are open to a very large public of candidates and, if the implementation allows automatic marking, their results are immediately available.

We do not want to argue about the actual benefits of e-exams in promoting and supporting education, but as a matter of facts, their use has considerably raised (and will likely continue to raise). Nowadays, several universities, such as MIT, Stanford, and Berkeley, just to cite a few, have began to offer university courses remotely using the Massive Open Online Course platforms (*e.g.,* Coursera[4] and edX[5]) which offer e-exams. Even in a less ambitious and more traditional setting, universities start adopting e-exams to replace traditional exams, especially in the case of multiple-choice questions and short open answers. For example, pharmacy exams at *Université Joseph Fourier* (UJF) have been organized electronically using tablet computers since 2014 [1]. Since several french medicine exams are multiple-choice tests, the French government plans to realize all medicine exams electronically by 2016.[6] Other institutions, such as ETS[7], CISCO[8], and Microsoft[9], have for long already adopted their own platforms to run, generally in qualified centers, electronic tests required to obtain their program certificates.

[4] www.coursera.org [5] www.edx.org [6] The project is called *Épreuves Classantes Nationales informatisées*, see www.side-sante.org [7] www.etsglobal.org
[8] www.cisco.com [9] www.microsoft.com/learning/en-us/default.aspx

This migration towards information technology is changing considerably the proceeding of exams, but the approach in coping with their security still focuses only on preventing candidates from cheating with invigilated tests. Wherever it is not possible to have human invigilators, a software running on the student computer is used, *e.g.,* ProctorU[10]. However, such measures are insufficient, as the trustworthiness and the reliability of exams are today threatened not only by candidates. Indeed, threats and errors may come from the use of information technology, as well as, from bribed examiners and dishonest exam authorities which are willing to tamper with exams as recent scandals have shown. For example, in the Atlanta scandal, school authorities colluded in changing student marks to improve their institution's rankings and get more public funds [2]. The BBC revealed another scandal where ETS was shown to be vulnerable to a fraud perpetrated by official invigilators in collusion with the candidates who were there to get their visas: the invigilators dictated the correct answers during the test [3].

To address these problems, e-exams must be checked for the presence/absence of irregularities and provide evidence about the fairness and the correctness of their grading procedures. Assumptions on the honesty of authorities are not justifiable anymore. Verification should be welcomed by authorities since verifying e-exams provides transparency and then public trust. E-exams offer the possibility to have extensive data logs, which can provide grounds for the verification and checking process, however, the requirements to be satisfied by e-exams have to be clearly defined and formalized before.

Contributions. To the best of our knowledge, this paper proposes the first formalization of e-exams properties, using Quantified Event Automata (QEAs) [4, 5], and their off-line runtime verification on actual logs. Our contributions are as follows. First, we define an event-based model of e-exams that is suitable for monitoring purposes. Moreover, we formalize eight fundamental properties as QEAs: no unregistered candidate try to participate in the exam by submitting an answer; answers are accepted only from registered candidates; all accepted answers are submitted by candidates, and for each question at most one answer is accepted per candidate; all candidates answer the questions in the required order; answers are accepted only during the examination time; another variant of the latter that offers flexibility in the beginning and the duration of the exam; all answers are marked correctly; and the correct mark is assigned to each candidate. Our formalization also allows us for some properties to detect the cause of the potential failures and the party responsible for them. Note, formalizing the above properties entailed to add features to QEAs. Then, we implement the monitors using MarQ[11] [6], a Java tool designed to support QEA specification language. Finally, we perform off-line monitoring, based on the available data logs, for an e-exam organized by UJF; and reveal both students that violate the requirements, and discrepancies between the specification and the implementation.

Outline. In Sec. 2, we define the events and a protocol for e-exams. We specify the properties and propose the corresponding monitors in Sec. 3. Then, in Sec. 4, we analyze two actual e-exams organized by UJF. We discuss related work in Sec. 5. Finally, we conclude in Sec. 6. An extended version of this paper is available as [7].

[10] `www.proctoru.com` [11] `www.github.com/selig/qea`

2 An Event-based Model of E-exams

We define an *e-exam execution* (or *e-exam run*) by a finite sequence of events, called *trace*. Such event-based modelling of e-exam runs is appropriate for monitoring actual events of the system. In this section, we specify the parties and the phases involved in e-exams. Then, we define the events related to an e-exam run. Note, the e-exam model introduced in this section refines the one proposed in [8].

2.1 Overview of an E-exam Protocol

An exam involves at least two roles: the *candidate* and the *exam authority*. An exam authority can have several sub-roles: the *registrar* registers candidates; the *question committee* prepares the questions; the *invigilator* supervises the exam, collects the answers, and dispatches them for marking; the *examiner* corrects the answers and marks them; the *notification committee* delivers the marking. Generally, exams run in four phases: 1) *Registration*, when the exam is set up and candidates enrol; 2) *Examination*, when candidates answer the questions, submit them to the authority, and have them officially accepted; 3) *Marking*, when the answers are marked; 4) *Notification*, when the grades are notified to the candidates. Usually, each phase ends before the next one begins.

2.2 Events Involved in an E-exam

Events flag important steps in the execution of the exam. We consider *parametric events* of the form $e(p_1, \ldots, p_n)$, where e is the event name, and $p_1, \ldots, p_n$ is the list of symbolic parameters that take some data values at runtime. We define the following events that are assumed to be recorded during the exam or built from data logs.

- Event $register(i)$ is emitted when candidate i registers to the exam.
- Event $get(i, q)$ is emitted when candidate i gets question q.
- Event $change(i, q, a)$ is emitted when candidate i changes on his computer the answer field of question q to a.
- Event $submit(i, q, a)$ is emitted when candidate i submits answer a to question q.
- Event $accept(i, q, a)$ is emitted when the exam authority receives and accepts answer a to question q from candidate i.
- Event $corrAns(q, a)$ is emitted when the authority specifies a as a correct answer to question q. Note that more than one answer can be correct for a given question.
- Event $marked(i, q, a, b)$ is emitted when the answer a from candidate i to question q is scored with b. In our properties we assume that the score b ranges over $\{0, 1\}$ (1 for correct answer and 0 for wrong answer), however other scores can be considered.
- Event $assign(i, m)$ is emitted when mark m is assigned to candidate i. We assume that the mark of a candidate is the sum of all the scores assigned to his answers. However, more complex functions can be considered (*e.g.,* weighted scores).
- Event $begin(i)$ is emitted when candidate i begins the examination phase.
- Event $end(i)$ is emitted when candidate i ends the examination phase. The candidate terminates the exam himself, *e.g.,* after answering all questions before the end of the exam duration.

In general, all events are time stamped, however we parameterize them with time only when it is relevant for the considered property. Moreover, we may omit some parameters from the events when they are not relevant to the property. For instance, we may use $submit(i)$ when candidate i submits an answer regardless of his answer. We also use $marked(q, a, b)$ instead of $marked(i, q, a, b)$ to capture anonymous marking.

3 Properties of E-exams

In this section, we define eight properties that aim at ensuring e-exams correctness. They mainly ensure that only registered candidates can take the exam, all accepted answers are submitted by the candidates, all answers are accepted during the exam duration, and all marks are correctly computed and assigned to the corresponding candidates. Note that in case of failure, two of the properties report all the individuals that violate the requirement of the property. This notion of reporting can be applied to all other properties (see [7]).

Each property represents a different e-exam requirement and can be monitored independently. An *exam run* may satisfy one property and fail on another one, which narrows the possible source of potential failures and allows us to deliver a detailed report about the satisfied and unsatisfied properties.

Quantified Event Automata (QEAs). We express properties as QEAs [4,5]. We present QEAs at an abstract level using intuitive terminology and refer to [4] for a formal presentation. A QEA consists of a list of quantified variables together with an *event automaton*. An event automaton is a finite-state machine with transitions labeled by parametric events, where parameters are instantiated with data-values at runtime. Transitions may also include guards and assignments to variables. Note, not all variables need to be quantified. Unquantified variables are left free, and they can be manipulated through assignments and updated during the processing of the trace. Moreover, new free variables can be introduced while processing the trace. We extend the initial definition of QEAs in [4] by i) allowing variable declaration and initialization before reading the trace, and ii) introducing the notion of global variable shared among all event automaton instances. Note, we use global variables in our case study presented in Sec. 4.2 and in the extended version of this paper. Global variables are mainly needed in QEAs to keep track and report data at the end of monitoring. Such QEAs may also require some manipulation of the quantified variables which is not currently supported by MarQ. Thus, we could not implement them and hence omitted them from the paper. The shaded states are final (accepting) states, while white states are failure states. Square states are closed to failure, *i.e.,* if no transition can be taken, then there is a transition to an implicit failure state. Circular states are closed to self (skip) states, *i.e.,* if no transition can be taken, then there is an implicit self-looping transition. We use the notation $\frac{[guard]}{assignment}$ to write guards and assignments on transitions: $:\hat{=}$ for variable declaration then assignment, $:=$ for assignment, and $=$ for equality test. A QEA formally defines a language (*i.e.*, a set of traces) over instantiated parametric events.

Correct Exam run. An exam run satisfies a property if the resulting trace is accepted by the corresponding QEA. A *correct exam run* satisfies all the properties. We assume

that an input trace contains events related to a single exam run. To reason about traces with events from more than one exam run, the events have to be parameterized with an exam run identifier, which has to be added to the list of quantified variables.

Candidate Registration. The first property is *Candidate Registration*, which states that only already registered candidates can submit answers to the exam. An exam run satisfies *Candidate Registration* if, for every candidate i, event $submit(i)$ is preceded by event $register(i)$. A violation of *Candidate Registration* does not reveal a weakness in the exam system (as long as the answers submitted from unregistered candidates are not accepted by the authority). However, it allows us to detect if a candidate tries to fake the system, which is helpful to be aware of *spoofing* attacks.

Definition 1 (Candidate Registration). *Property* Candidate Registration *is defined by the QEA depicted in Fig. 1 with alphabet* $\Sigma_{\mathrm{CR}} = \{register(i), submit(i)\}$.

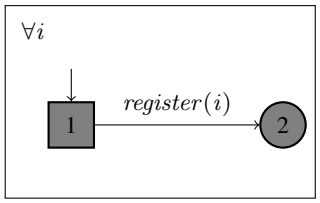

Fig. 1: QEA for *Candidate Registration*

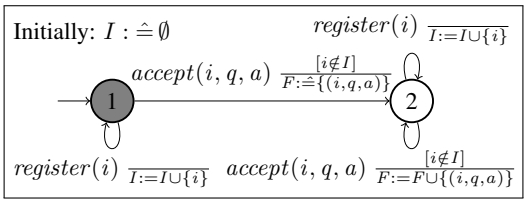

Fig. 2: QEA for *Candidate Eligibility*

The input alphabet Σ_{CR} for *Candidate Registration* contains only the events $register(i)$ and $submit(i)$, so any other event in the trace is ignored. The QEA for *Candidate Registration* has two accepting states, and one quantified variable i. Note, the empty trace is accepted by the QEA. State (1) is a square state, so an event $submit(i)$ that is not preceded by event $register(i)$ leads to a failure. An event $register(i)$ in state (1) leads to state (2) which is a skipping (circular) state, so after event $register(i)$ any sequence of events is accepted. The quantification $\forall i$ means that the property must hold for all values that i takes in the trace, *i.e.,* the values obtained when matching the symbolic events in the specification with concrete events in the trace. For instance, let us consider the following trace: $register(i_1).submit(i_2).submit(i_1).register(i_2)$. To decide whether it is accepted or not, the trace is sliced based on the values that can match i, resulting in the two slices: $i \mapsto i_1$: $register(i_1).submit(i_1)$, and $i \mapsto i_2$: $submit(i_2).register(i_2)$. Then, each slice is checked against the event automata instantiated with the appropriate values for i. The slice associated to i_1 is accepted as it reaches the final state (2), while the slice associated to i_2 does not reach a final state since event $submit(i_2)$ leads from state (1) to an implicit failure state. Therefore, the whole trace is not accepted by the QEA. Note, we omit parameters q and a from event $submit(i, q, a)$ since only the fact that a candidate i submits an answer is significant for the property, regardless of the question he is answering, and the answer he submitted.

Candidate Eligibility. Property *Candidate Eligibility* states that no answer is accepted from an unregistered candidate. *Candidate Eligibility* can be modeled by a QEA similar to that of *Candidate Registration* depicted in Fig. 1, except that event $submit(i, q, a)$ has to be replaced by $accept(i, q, a)$ in the related alphabet. However, we formalize *Candidate Eligibility* in a way that, in addition to checking the main requirement, it reports all the candidates that violate the requirement, *i.e.,* those that are unregistered but some answers are accepted from them. Note, *Candidate Registration* can also modeled similarly by replacing $accept(i, q, a)$ with $submit(i, q, a)$.

Definition 2 (Candidate Eligibility). *Property* Candidate Eligibility *is defined by the QEA depicted in Fig. 2 with alphabet* $\Sigma_{\mathrm{CE}} = \{register(i), accept(i, q, a)\}$.

The QEA of *Candidate Eligibility* has three free variables I, F, and i, and no quantified variables. Instead of being instantiated for each candidate i, the QEA of *Candidate Eligibility* collects all the registered candidates in set I, so that any occurrence of event $accept(i, q, a)$ at state (1) with $i \notin I$ fires a transition to the failure state (2). Such a transition results in the failure of the property since all transitions from state (2) are self-looping transitions. Set F is used to collect all the unregistered candidates that submitted an answer. Note, variable I is pre-declared and initialized to $\emptyset$. Trace $register(i_1).accept(i_2, q_0, a_2).accept(i_1, q_0, a_1).register(i_2)$ is not accepted by *Candidate Eligibility*, and results in $F = \{(i_2, q_0, a_2)\}$. Note, reporting the candidates that violates the requirements requires to monitor until the end of the trace.

Answer Authentication. Property *Answer Authentication* states that all accepted answers are submitted by candidates. Moreover, for every question, exactly one answer is accepted from each candidate that submitted at least one answer to that question.

Definition 3 (Answer Authentication). *Property* Answer Authentication *is defined by the QEA depicted in Fig. 3 with alphabet* $\Sigma_{\mathrm{AA}} = \{submit(i, q, a), accept(i, q, a)\}$.

The QEA of *Answer Authentication* fails if an unsubmitted answer is accepted. A candidate can submit more than one answer to the same question, but exactly one answer has to be accepted. Note, any answer among the submitted answers can be accepted. However, the QEA can be updated to allow only the acceptance of the last submitted answers by replacing set A with a variable, which acts as a placeholder for the last submitted answer. If no answer is accepted after at least one answer has been submitted, the QEA ends in the failure state (2), while acceptance of an answer leads to the accepting state (3). A candidate can submit after having accepted an answer from him to that question. However, if more than one answer is accepted, an implicit transition from state (3) to a failure state is fired. Trace $submit(i_1, q_0, a_1).submit(i_1, q_0, a_2).accept(i_1, q_0, a_2)$ – where candidate i_1 submits two answers a_1 and a_2 to question q_0, then only a_2 is accepted – is accepted by *Answer Authentication*. While the traces $accept(i_1, q, a)$, where an unsubmitted answer is accepted from i_1, and $submit(i_1, q, a_1).\ submit(i_1, q, a_2).\ accept(i_1, q, a_1).accept(i_1, q, a_2)$, where two answers to the same question are accepted from same candidate, are not accepted.

Answer Authentication can be further split into three different properties which allow us to precisely know whether, for a certain question, an unsubmitted answer is accepted, no answer is accepted from a candidate that submitted an answer, or more

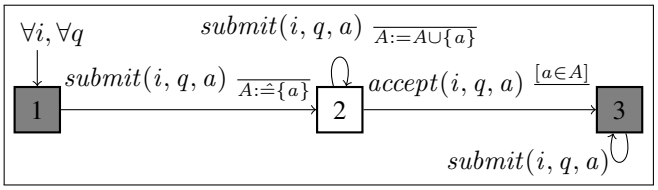

Fig. 3: QEA for *Answer Authentication*

than one answer is accepted from the same candidate. For instance, updating the QEA depicted in Fig. 3 by getting rid of state (3), converting state (2) into an accepting state, and adding a self loop transition on state (2) labeled by $accept(i, q, a)$ $\overline{[a\in A]}$ results in a QEA that fails only when an unsubmitted answer is accepted (see [7] for more details).

Question Ordering. The previous properties formalize the main requirements that are usually needed concerning answer submission and acceptance. However, additional requirements might be needed. For example, candidates may be required to answer questions in a certain order: a candidate should not get a question before validating his answer to the previous question. This is ensured by *Question Ordering*.

Definition 4 (Question Ordering). *Let $q_1, \ldots, q_n$ be n questions such that the order $ord(q_k)$ of q_k is k. Property* Question Ordering *is defined by the QEA depicted in Fig. 4 with alphabet $\Sigma_{\mathrm{QO}} = \{get(i, q), accept(i, q)\}$.*

The QEA of *Question Ordering* fails if a candidate gets (or an answer is accepted from him for) a higher order question before his answer to the current question is accepted. Note, *Question Ordering* also allows only one accepted answer per question. Otherwise, there is no meaning for the order as the candidate can re-submit answers latter when he gets all the questions.

Exam availability. An e-exam must allow candidates to take the exam only during the examination phase. *Exam Availability* states that questions are obtained, and answers are submitted and accepted only during the examination time.

Definition 5 (Exam Availability). *Let t_0 be the starting instant, and t_f be the ending instant of the exam. Property* Exam Availability *is defined by the QEA depicted in Fig. 5 with alphabet $\Sigma_{\mathrm{EA}} = \{get(i, t), change(i, t), submit(i, t), accept(i, t)\}$.*

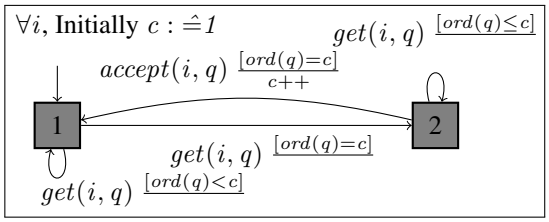

Fig. 4: QEA for *Question Ordering*

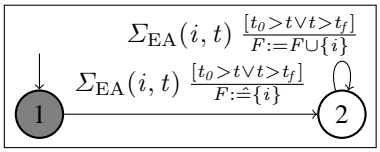

Fig. 5: QEA for *Exam Availability*

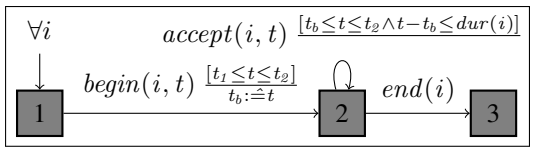

Fig. 6: QEA for *Exam Availability with Flexibility*

The QEA of *Exam Availability* checks that all the events in Σ_{EA} are emitted between t_0 and t_f. It also collects all the candidates that violates the requirements in a set F. Note, any other event can be added to Σ_{EA} if required.

Exam availability with flexibility. Some exams offer flexibility to the candidates, so that a candidate is free to choose the beginning time within a certain specified period. To capture that, we define *Exam Availability with Flexibility* which states that no answer can be accepted from a candidate before he begins the exam, after he terminates the exam, after the end of his exam duration, or after the end of the specified period. The beginning time of the exam may differ from one candidate to another, but in any case it has to be within a certain specified period. The exam duration may also differ between candidates. For example, an extended duration may be offered to certain candidates with disabilities.

Definition 6 (Exam Availability With Flexibility). *Let t_1 and t_2 respectively be the starting and the ending time instants of the allowed period, and let $dur(i)$ be the exam duration for candidate i. Property* Exam Availability with Flexibility *is defined by the QEA depicted in Fig. 6 with alphabet $\Sigma_{\mathrm{EA}} = \{begin(i,t), end(i), accept(i,t)\}$.*

Exam Availability with Flexibility also requires that, for each candidate i, there is only one event $begin(i,t)$ per exam. Hence, it fails if event $begin(i)$ is emitted more than once. A candidate can begin his exam at any time t_b such that $t_1 \leq t_b \leq t_2$. Note, no answer can be accepted from a candidate after then ending time t_2 of the period, if the duration of the candidate is not finished yet. Assume that $t_1 = 0$, $t_2 = 1,000$, $dur(i_1) = 90$, and $dur(i_2) = 60$. Then, trace $begin(i_1, 0).accept(i_1, 24).begin(i_2, 26).accept(i_2, 62).accept(i_1, 90)$ is accepted. While, trace $accept(i_1, 5).begin\ (i_1, 20)$ and trace $begin(i_1, 0).accept(i_1, 91)$ are not accepted since in the first one an answer is accepted from candidate i_1 before he begins the exam, and in the second one an answer is accepted after the exam duration expires.

Event $submit$ is not included in Σ_{EA}, thus an answer submission outside the exam time is not considered as an irregularity if the answer is not accepted by the authority. However, again other events (*e.g.,* get and $submit$) can be considered. In such a case, the QEA in Fig. 6 has to be edited by looping over state (2) with any added event.

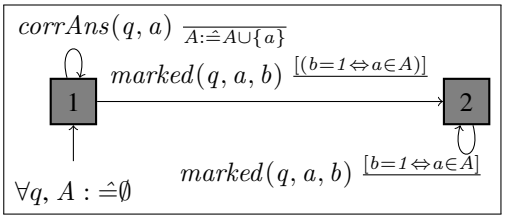

Fig. 7: QEA for *Marking Correctness*

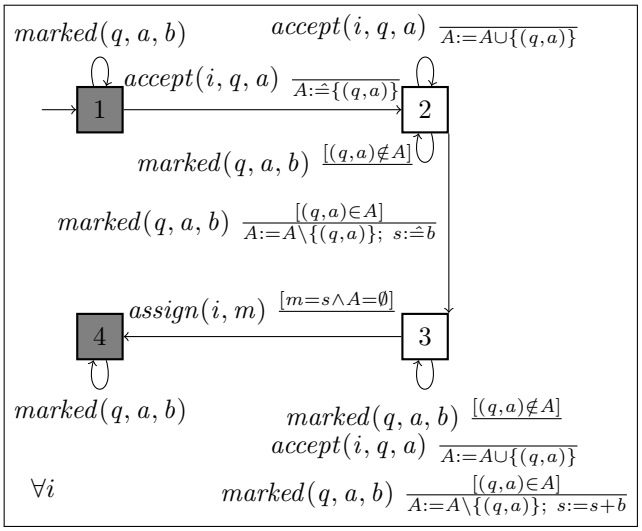

Fig. 8: QEA for *Mark Integrity*

Marking correctness. The last two properties state that each candidate should get the correct mark, the one computed correctly from his answers. *Marking Correctness* states that all answers are marked correctly. In the QEA of *Marking Correctness*, the correct answers for the considered question are collected in a set A (self loop over state (1)).

Definition 7 (Marking Correctness). *Property* Marking Correctness *is defined by the QEA depicted in Fig. 7 with alphabet* $\Sigma_{\mathrm{MC}} = \{corrAns(q, a), marked\}(q, a, b)\}$.

In state (1), once an answer to the considered question is marked correctly, a transition to state (3) is fired, otherwise if an answer is marked in a wrong way a transition to an implicit failure state occurs. In state (3), the property fails either if an answer is marked in a wrong way, or if an event $corrAns(q, a)$ is encountered as this means that certain answers are marked before all the correct answers are set.

Mark integrity. Property *Mark Integrity* states that all accepted answers are marked, and that exactly one mark is assigned to each candidate, the one attributed to his answers. *Mark Integrity* together with *Marking Correctness*, guarantee that each candidate participated in the exam gets the correct mark corresponding to his answers. The QEA of *Mark Integrity* collects, for each candidate, the submitted answers in a set A.

Definition 8 (Mark Integrity). *Property* Mark Integrity *is defined by the QEA depicted in Fig. 8 with alphabet* $\Sigma_{\mathrm{MI}} = \{accept(i, q, a), marked(q, a, b), assign(i, m)\}$.

For each accepted answer, the QEA accumulates the corresponding score b in the sum s. If the accepted answers are not marked, the property fails (failure state (2)). If the candidate is not assigned a mark or assigned a wrong mark the property fails (failure state (3)). Once the the correct mark is assigned to the candidate, if another mark is assigned or any other answer is accepted from him, the property fails (square state (4)).

4 Case Study: UJF E-exam

In June 2014, the pharmacy faculty at UJF organized a first e-exam, as a part of *Epreuves Classantes Nationales informatisées* project which aims to realize all medicine exams electronically by 2016. The project is lead by UJF and the e-exam software is developed by the company THEIA[12] specialized in e-formation platforms. This software is currently used by 39 french universities. Since then, 1,047 e-exams have been organized and 147,686 students have used the e-exam software.

We validate our framework by verifying two real e-exams passed with this system. All the logs received from the e-exam organizer are anonymized; nevertheless we were not authorized to disclose them. We use MarQ[13] [6] (Monitoring At Runtime with QEA) to model the QEAs and perform the verification. We provide a description for this system that we call *UJF e-exam*[14], then we present the results of our analysis.

4.1 Exam Description

Registration. The candidates have to register two weeks before the examination time. Each candidate receives a username/password to authenticate at the examination.
Examination. The exam takes place in a supervised room. Each student handled a previously-calibrated tablet to pass the exam. The internet access is controlled: only IP addresses within an certain range are allowed to access the exam server. A candidate starts by logging in using his username/password. Then, he chooses one of the available exams by entering the exam code, which is provided at the examination time by the invigilator supervising the room. Once the correct code is entered, the exam starts and the first question is displayed. The pedagogical exam conditions mention that the candidates have to answer the questions in a fixed order and cannot get to the next question before answering the current one. A candidate can change the answer as many times as he wants before validating, but once he validates, then he cannot go back and change any of the previously validated answers. Note, all candidates have to answer the same questions in the same order. A question might be a one-choice question, multiple-choice question, open short-text question, or script-concordance question.
Marking. After the end of the examination phase, the grading process starts. For each question, all the answers provided by the candidates are collected. Then, each answer is evaluated anonymously by an examiner to 0 if it is wrong, $0 < s < 1$ if it is partially correct, or 1 if it is correct. An example of a partially-correct answer is when a candidate provides only one of the two correct answers for a multiple-choice question. The professor specifies the correct answer(s) and the scores to attribute to correct and partially-correct answers, as well, as the potential penalty. After evaluating all the provided answers for all questions, the total mark for each candidate is calculated as the summation of all the scores attributed to his answers.
Notification. The marks are notified to the candidates. A candidate can consult his submission, obtain the correct answer and his score for each question.

[12] `www.theia.fr` [13] https://github.com/selig/qea [14] We have also designed an event-based behavioral model of the e-exam phases that is not reported in this paper for space reasons. The description was obtained and validated through discussions with the engineers at THEIA.

4.2 Analysis

We analyzed two exams: Exam 1 involves 233 candidates and contains 42 questions for a duration of 1h35. Exam 2 involves 90 candidates, contains 36 questions for a duration of 5h20. The resulting traces for these exams are respectively of size 1.85 MB and 215 KB and contain 40,875 and 4,641 events. The result of our analysis together with the time required for MarQ to analyze the whole trace on a standard PC (AMD A10-5745M–Quad-Core 2.1 GHz, 8 GB RAM), are summed up in Table 1. (✓) means satisfied, (×) means not satisfied, and $[1]$ indicates the number of violations. Only four of the eight general properties presented in Sec. 3 were compatible with UJF E-exam. We considered five additional and specific properties for the *UJF e-exam*.

Property *Candidate Registration* was satisfied, that is, no unregistered candidate submits an answer. *Candidate Eligibility* is also satisfied. We note that, in MarQ tool the *Candidate Eligibility* monitor stops monitoring as soon as a transition to state (2) is made since there is no path to success from state (2). Thus, only the first candidate that violates the property is reported. In order to report all such candidates, we had to add an artificial transition from state (2) to an accepting state that could never be taken. Then, monitoring after reaching state (2) remains possible. Moreover, the current implementation of MarQ does not support sets of tuples. Consequently, we could only collect the identities i in a set F instead of the tuples (i, q, a).

Answer Authentication was violated only in Exam 1. We reported the violation to the e-exam's developers. The violation actually revealed a discrepancy between the initial specification and the current features of the e-exam software: a candidate can submit the *same answer* several times and this answer remains accepted. Consequently, an event $accept$ can appear twice but only with the same answer. To confirm that the failure of *Answer Authentication* is only due to the acceptance of a same answer twice, we updated property *Answer Authentication* and its QEA presented in Fig. 3 by storing the accepted answer in a variable a_v, and adding a self loop transition on state (3) labeled by $accept(i, q, a)$ $\frac{[a=a_v]}{}$. We refer to this new weaker property as *Answer Authentication* *, which differs from *Answer Authentication* by allowing the acceptance of the same answer again; but it still forbids the acceptance of a different answer. We found out that *Answer Authentication* * is satisfied, which confirms the claim about the possibility of accepting the same answer twice. After diagnosing the source of failure, we defined property *Answer Authentication Reporting* presented in Fig. 9, which fails if more than one answer (identical or not) is accepted from the same candidate to the same question. At the same time, it collects all such candidates in a set F. *Answer Authentication Reporting* is defined by the QEA depicted in Fig. 9 with the input alphabet $\Sigma_{\mathrm{AAR}} = \{accept(i, q, a)\}$. The analysis of *Answer Authentication Reporting* shows that, for Exam 1, there is only one candidate such that more than one answer are accepted from him to the same question. The multiple answers that are accepted for the same question are supposed to be equal since *Answer Authentication* * is satisfied. Note that MarQ currently does not support global variables, so for *Answer Authentication Reporting*, a set is required for each question. Note for Exam 1, *Answer Authentication* required less monitoring time than *Answer Authentication* * and *Answer Authentication Reporting* as the monitor for *Answer Authentication* stops monitoring as soon as it finds a violation.

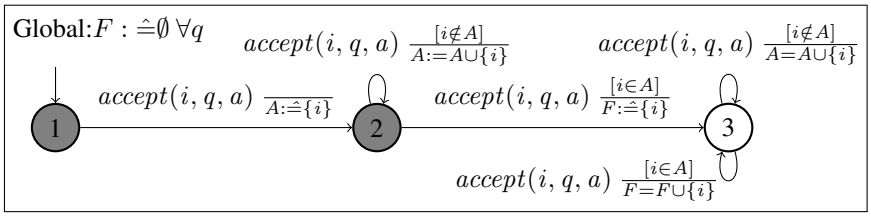

Fig. 9: QEA for *Answer Authentication Reporting*

Furthermore, *UJF exam* has a requirement stating that after acceptance the writing field is "blocked" and the candidate cannot change it anymore. Actually, in *UJF exam* when a candidate writes a potential answer in the writing field the server stores it directly, and once the candidate validates the question the last stored answer is accepted. As *Answer Authentication* shows, several answers can still be accepted after the first acceptance, then the ability of changing the answer in the writing field could result in an acceptance of a different answer. For this purpose, we defined property *Answer Editing* that states that a candidate cannot change the answer after acceptance. *Answer Editing* is defined by the QEA depicted in Fig. 10 with the input alphabet $\Sigma_{\mathrm{AE}} = \{change(i,q), accept(i,q,a)\}$.

Note, we allowed the acceptance of the same answer to avoid the bug found by *Answer Authentication*. Our analysis showed that *Answer Editing* was violated in Exam 2: at least one student was able to change the content of the writing field after having his answer accepted.

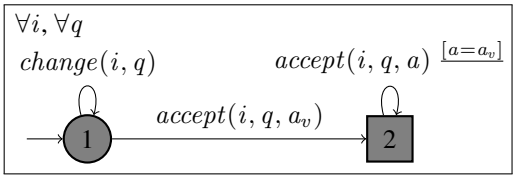

Fig. 10: QEA for *Answer Editing*

Concerning *Question Ordering* the developers did not log anything related to the event $get(i,q)$. However, we defined *Question Ordering* * which fails if a candidate changes the writing field of a future question before an answer for the current question is accepted. *Question Ordering* * is defined by the QEA depicted in Fig. 11 with the input alphabet $\Sigma_{\mathrm{QO'}} = \{change(i,q), accept(i,q)\}$. The idea is that if a candidate changes the answer field of a question, he must have received the question previously. Moreover, we allow submitting the same answer twice, and also changing the previous accepted answers to avoid the two bugs previously found. Note, *UJF exam* requires the candidate to validate the question even if he left it blank, thus we also allow acceptance for the current question before changing its field (self loop above state (2)). The analysis showed that *Question Ordering* * was violated in both exams.

Alternatives to *Answer Editing* and *Question Ordering* can be defined to report all the candidates who violate the requirement (see [7]). However, it cannot be implemented using MarQ as it requires the ability either to manipulate the quantified variables or to build sets of pairs which are both currently not supported by MarQ. However, the tool still outputs the first candidate who violates the property.

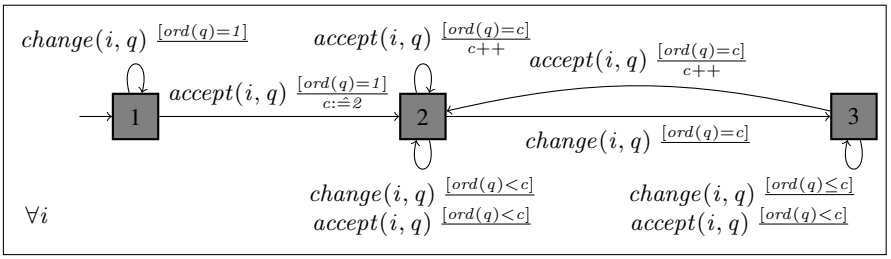

Fig. 11: QEA for *Question Ordering* *

Note, the manual check of *Question Ordering* * showed that some candidates were able to skip certain questions (after writing an answer) without validating them, and then validating the following questions.

As we found a violation for *Question Ordering* *, we defined *Acceptance Order* that checks, for each candidate, whether all the accepted answers are accepted in order, *i.e.,* there should be no answer accepted for a question that is followed by an accepted answer for a lower order question. *Acceptance Order* is defined by the QEA depicted in Fig. 12 with the input alphabet $\Sigma_{\mathrm{AO}} = \{accept(i, q, a)\}$.

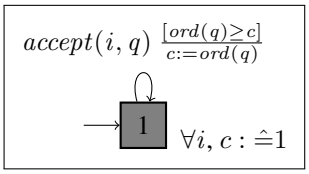

Fig. 12: QEA for *Acceptance Order*

Exam Availability is also violated in Exam 2. A candidate was able to change and submit an answer, which is accepted, after the end of the exam duration. We could not analyze *Exam Availability with Flexibility*, since it is not supported by the exam. We also did not consider *Marking Correctness*, and *Mark Integrity* properties since the developers did not log anything concerning the marking and the notification phase is done by each university and we were not able to get the logs related to this phase. This shows that universities only look for cheating candidates, and do not look for internal problems or insider attacks. We expect the developers of the e-exam software to include logging features for every phase. Note, we implemented all properties in MarQ and validated them on toy traces as we expect to obtain the actual traces of the marking phase in the near future.

5 Related Work and Discussion

To the best of our knowledge, this is the first paper to address the runtime verification of e-exams. However, a formal framework for checking verifiability properties of e-exams based on abstract tests has been proposed by Dreier *et al.* in [8]. Note, the proposed tests in [8] need to be instantiated for each exam depending on its specifications. The authors of [8] have validated their framework by i) modeling two exams in the applied π-calculus [9], and then ii) analyzing them using ProVerif [10]. More precisely, they proposed a set of individual and universal properties that allow to verify the correctness of e-exams. The individual properties allow the candidate to check himself whether he received the correct mark that corresponds to his answers. While the universal properties allow an outsider auditor to check whether only registered candidates participate in

Table 1: Results of the off-line monitoring of two e-exams.

	Exam 1		Exam 2	
Property	Result	Time (ms)	Result	Time (ms)
Candidate Registration	✓	538	✓	230
Candidate Eligibility	✓	517	✓	214
Answer Authentication	×	310	✓	275
Answer Authentication *	✓	742	✓	223
Answer Authentication Reporting	×[1]	654	✓	265
Answer Editing	✓	641	×	218
Question Ordering *	×	757	×	389
Acceptance Order	✓	697	✓	294
Exam Availability	✓	518	×[1]	237

the exam, all accepted answers are marked correctly, and all marks are assigned to the corresponding candidates. The universal properties that we proposed revisit the properties defined in [8]. However, as mentioned before, this paper is concerned with the monitoring of actual exam executions rather than operating on the abstract models of the exam specification. Furthermore, in general, formal verification techniques such as the one in [8] suffer from the so-called state explosion that may limit the size of systems that can be verified. Moreover, as formal methods operate on models, they introduce an additional question concerning the correctness of the abstraction. In contrast, as runtime verification operates only on the actual event traces, it is less dependent on the size of the system and, at the same time, does not require as much abstraction. Our properties can be monitored only by observing the events of trace from an exam run.

System verification is also addressed in some other related domains *e.g.,* in auctions [11], and voting [12, 13]. Back to e-exams, Dreier *et al.* also propose a formal framework, based on π-calculus, to analyze other security properties such as authentication and privacy [14]. These complementary approaches study security and not verification, however both aspects are important to develop such sensitive systems.

All the mentioned related approaches only allow symbolic abstract analysis of the protocols specifications, mainly looking for potential flaws in the used cryptographic primitives. What is more, these approaches support neither on-line nor off-line analysis of the actual logs obtained from system executions.

On the other hand, off-line runtime verification of user-provided specifications over logs has been addressed in the context of several tools in the runtime verification community [15]: Breach for Signal Temporal Logic. RiTHM and StePr for (variants of) Linear Temporal Logic, LogFire for rule-based systems, and Java-MOP for various specification formalisms provided as plugins. MarQ [6] is a tool for monitoring Quantified Event Automata [4, 5]. Our choice of using QEA stems from two reasons. First, QEAs is one of the most expressive specification formalism to express monitors. The second reason stems from our interviews of the engineers who were collaborating with us and responsible for the development of the e-exam software at UJF. To validate our

formalization of the protocol and the desired properties for e-exams, we presented the existing alternative specification languages. QEAs turned out to be the specification language that was most accepted and understood by the engineers. Moreover, MarQ came top in the 1st international competition on Runtime Verification (2014)[15] , showing that MarQ is one of the most efficient existing monitoring tools for both off-line and on-line monitoring. Note, off-line runtime verification was successfully applied to other case studies, *e.g.,* for monitoring financial transactions with LARVA [16], and monitoring IT logs with MonPoly [17].

6 Conclusions and Future Work

We define an event-based model for e-exams, and formalize several essential properties as Quantified Event Automata, enriched with global variables and pre-initialization. Our model handles e-exams that offer flexible independent beginning time and/or different exam duration for the candidates. We validate the properties by analyzing real logs from e-exams at UJF. We perform off-line verification of certain exam runs using the MarQ tool. We find several discrepancies between the specification and the implementation. Analyzing logs of real e-exams requires only a few seconds on a regular computer. Due to the lack of logs about the marking and notification phases, we were not able to analyze all properties. The *UJF E-exam* case study clearly demonstrates that the developers do not think to log these two phases where there is less interaction with the candidates. However, we believe that monitoring the marking phase is essential since a successful attempt from a bribed examiner or a cheating student can be very effective.

Several avenues for future work are opened by this paper. First, we intend to analyze more existing e-exams: from other universities and the marking phase of the pharmacy exams at UJF. We encourage universities and educational institutions to incorporate logging features in their e-exam software. Moreover, we plan to perform on-line verification during live e-exams, and to study to what extent runtime enforcement (cf [18] for an overview) can be applied during a live e-exam run. Finally, we plan to study more expressive and quantitative properties that can detect possible collusion between students through similar answer patterns.

Acknowledgment. The authors would like to thank François Géronimi from THEIA, Daniel Pagonis from TIMC-IMAG, and Olivier Palombi from LJK for providing us with a description of e-exam software system, for sharing with us the logs of some real french e-exams, and for validating and discussing the properties presented in this paper. The authors also thank Giles Reger for providing us with help on using MarQ. The authors also would like to thank the "Digital trust" Chair from the University of Auvergne Foundation for the support provided to conduct this research.

References

1. Le Figaro: Etudiants: les examens sur tablettes numériques appellés à se multiplier. Press release (2015) Available at `goo.gl/ahxQJD`.

[15] `http://rv2014.imag.fr/monitoring-competition/results`

2. Copeland, L.: School cheating scandal shakes up atlanta. USA TODAY (2013) Available at http://goo.gl/wGr40s.
3. Watson, R.: Student visa system fraud exposed in BBC investigation. http://www.bbc.com/news/uk-26024375 (2014)
4. Barringer, H., Falcone, Y., Havelund, K., Reger, G., Rydeheard, D.E.: Quantified event automata: Towards expressive and efficient runtime monitors. In: FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings. Volume 7436 of Lecture Notes in Computer Science., Springer (2012) 68–84
5. Reger, G.: Automata Based Monitoring and Mining of Execution Traces. PhD thesis, University of Manchester (2014)
6. Reger, G., Cruz, H.C., Rydeheard, D.E.: MarQ: Monitoring at runtime with QEA. In: Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS, London, UK. (2015) 596–610
7. Kassem, A., Falcone, Y., Lafourcade, P.: Monitoring electronic exams. Technical Report TR-2015-4, Verimag, Laboratoire d'Informatique de Grenoble Research Report (2015)
8. Dreier, J., Giustolisi, R., Kassem, A., Lafourcade, P., Lenzini, G.: A framework for analyzing verifiability in traditional and electronic exams. In: 11th International Conference on Information Security Practice and Experience (ISPEC'15). (2015)
9. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL'01, New York, ACM (2001) 104–115
10. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: CSFW, Cape Breton, Canada, IEEE Computer Society (2001) 82–96
11. Dreier, J., Jonker, H., Lafourcade, P.: Defining verifiability in e-auction protocols. In: 8th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '13, Hangzhou, China. (2013) 547–552
12. Kremer, S., Ryan, M., Smyth, B.: Election verifiability in electronic voting protocols. In: ESORICS'10. Volume 6345 of LNCS., Springer (2010) 389–404
13. Backes, M., Hritcu, C., Maffei, M.: Automated verification of remote electronic voting protocols in the applied pi-calculus. In: CSF. (2008) 195–209
14. Dreier, J., Giustolisi, R., Kassem, A., Lafourcade, P., Lenzini, G., Ryan, P.Y.A.: Formal analysis of electronic exams. In: SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria. (2014) 101–112
15. Bartocci, E., Bonakdarpour, B., Falcone, Y.: First international competition on software for runtime verification. [19] 1–9
16. Colombo, C., Pace, G.J.: Fast-forward runtime monitoring - an industrial case study. In Qadeer, S., Tasiran, S., eds.: Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers. Volume 7687 of Lecture Notes in Computer Science., Springer (2012) 214–228
17. Basin, D.A., Caronni, G., Ereth, S., Harvan, M., Klaedtke, F., Mantel, H.: Scalable offline monitoring. [19] 31–47
18. Falcone, Y.: You should better enforce than verify. In Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G.J., Rosu, G., Sokolsky, O., Tillmann, N., eds.: Runtime Verification - First International Conference, RV 2010, Proceedings. Volume 6418 of Lecture Notes in Computer Science., Springer (2010) 89–105
19. Bonakdarpour, B., Smolka, S.A., eds.: Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. Volume 8734 of Lecture Notes in Computer Science., Springer (2014)