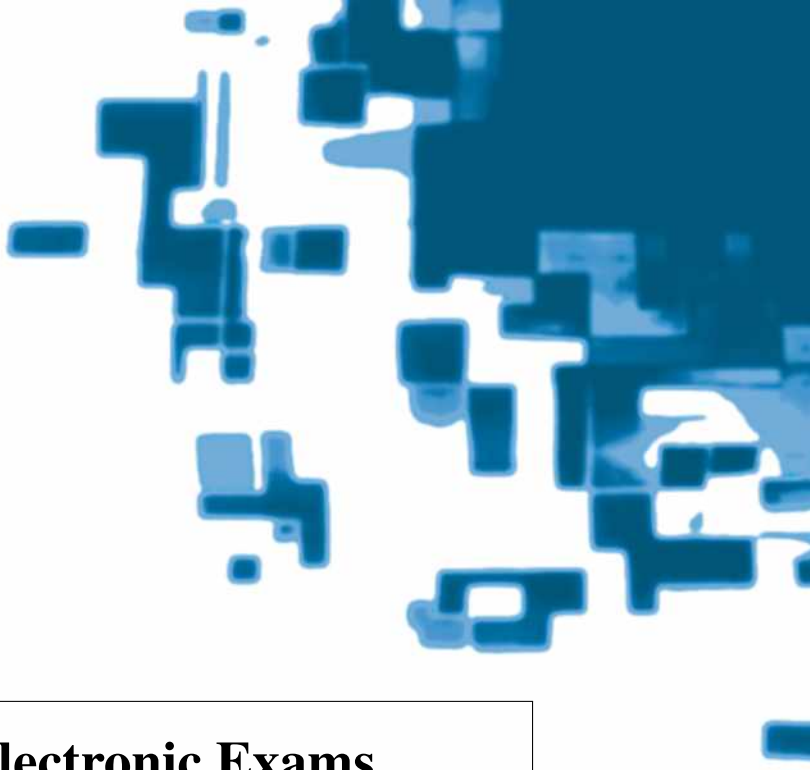

Monitoring Electronic Exams

Ali Kassem, Yliès Falcone, Pascal Lafourcade

Verimag Research Report n^o TR-2015-4

July 20, 2015

Reports are downloadable at the following address
http://www-verimag.imag.fr

Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UJF
Centre Équation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Monitoring Electronic Exams

Ali Kassem, Yliès Falcone, Pascal Lafourcade

Univ. Grenoble Alpes, VERIMAG, Grenoble, France, Ali.Kassem@imag.fr
Univ. Grenoble Alpes, Inria, LIG, Grenoble, France, Ylies.Falcone@imag.fr
Université d'Auvergne, LIMOS, France, Pascal.Lafourcade@udamail.fr

July 20, 2015

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, applied 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 contraventions and discrepancies between the specification and the implementation.

Keywords: Electronic exams, Monitoring, Runtime Verification, Quantified Event Automata (QEA), Log Analysis, Monitoring at runtime with QEA (MarQ)

Reviewers:

How to cite this report:

```
@techreport {TR-2015-4,
  title = {Monitoring Electronic Exams},
  author = {Ali Kassem, Yliès Falcone, Pascal Lafourcade},
  institution = {{Verimag, LIG} Research Report},
  number = {TR-2015-4},
  year = {2015}
}
```

Contents

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[1] and edX[2]) 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 [17]. Since several french medicine exams are multiple-choice tests, the French government plans to realize all medicine exams electronically by 2016.[3] Other institutions, such as ETS[4], CISCO[5], and Microsoft[6], have for long already adopted their own platforms to run, generally in qualified centers, electronic tests required to obtain their program certificates.

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[7]. 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 [9]. 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 [20].

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

[1] www.coursera.org [2] www.edx.org [3] The project is called *Épreuves Classantes Nationales informatisées*, see www.side-sante.org [4] www.etsglobal.org [5] www.cisco.com
[6] www.microsoft.com/learning/en-us/default.aspx [7] www.proctoru.com

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) [3, 18], 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. Then, we formalize as QEAs eight fundamental properties, which ensures that: 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 to detect the cause of the potential failures and the party responsible for them. 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 such as global variables and pre-initialization to QEAs. Second, we implement the monitors using MarQ[8] [19], 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 briefly introduce Quantified Event Automata. We define the events and a protocol for e-exams in Sec. 3. In Sec. 4, we specify the properties and propose the corresponding monitors. Then, in Sec. 5, we analyze two actual pharmacy e-exams at UJF. We discuss related work in Sec. 6. Finally, we conclude and present the avenues for future work in Sec. 7.

2 Quantified Event Automata (QEAs).

In this section, we present QEAs at an abstract level using intuitive terminology and refer to [3] 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 [3] by:

1. allowing variable declaration and initialization before reading the trace; and
2. introducing the notion of global variable shared among all event automaton instances.

[8] www.github.com/selig/qea

Note, we use global variables in our case study presented in Sec. 5.2 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 initial state of a QEA has an arrow pointing to it. 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.

3 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. 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.

In the following, 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 [10].

3.1 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 phases, where usually each phase ends before the next one begins. We consider the following four main 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 by the examiners;
4. *Notification*, when the marks are notified to the candidates.

3.2 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. Note, we can omit candidate identity i to capture anonymous marking.
- 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.

4 Properties of E-exams

In this section, we define eight properties expressed as QEAs [3, 18]. The properties state 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. Then, for each property we propose an alternative which additionally reports, in case of failure, all the entities that violates the requirements.

Each property models a different 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 to deliver a detailed report about the satisfied and unsatisfied properties.

4.1 Properties for Error Detection

First, for each property we propose a QEA which fails when its requirements are violated, and succeeds otherwise.

4.1.1 Candidate Registration

The first property is *Candidate Registration*, which states that only registered candidates submit answers to the exam. An exam run satisfies *Candidate Registration* if, for every candidate i, event $submit(i)$ is preceded by event $register(i)$. *Candidate Registration* does not reveal a weakness in the exam system when it is not satisfied (as long as the answers submitted from unregistered candidates are not accepted by the authority). However, it allows to detect the case where 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 Figure 1 with alphabet* $\Sigma_{\mathrm{CR}} = \{register(i), submit(i)\}$.

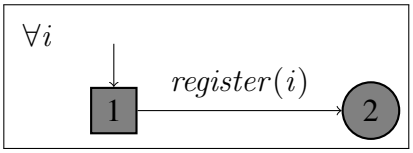

Figure 1: QEA for Candidate Registration

The input alphabet Σ_{CR} for *Candidate Registration* contains only events $register(i)$ and $submit(i)$, so any other events in the trace are ignored for this property. The QEA for *Candidate Registration* has two accepting states, and one quantified variable i. As the initial state is an accepting state, then 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. Henceforth, given a candidate i any trace starting with event $register(i)$ 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, 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 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 automaton instantiated with the appropriate value 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.

4.1.2 Candidate Eligibility

Property *Candidate Eligibility* states that no answer is accepted from an unregistered candidate. *Candidate Eligibility* is modeled by a QEA similar to that of *Candidate Registration* except that event $submit(i, q, a)$ has to be replaced by event $accept(i, q, a)$ in the related alphabet.

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

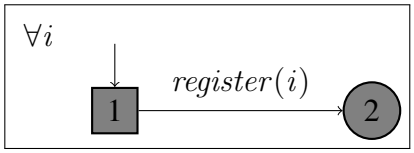

Figure 2: QEA for Candidate Eligibility

Candidate Eligibility captures the case where an answer is accepted from an unregistered candidate, *e.g.,* when an unregistered candidate participates in the exam. 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* as an answer is accepted from the candidate i_2 before he registers.

4.1.3 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 submits at least one answer to that question.

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

Answer Authentication allows to detect:

- the case where an answer is added by the authority without being submitted by a candidate;
- the case where the more than one answer is accepted from the same candidate for the same question; and
- the case no answer is accepted from a candidate to a question that he submitted at least one answer for it.

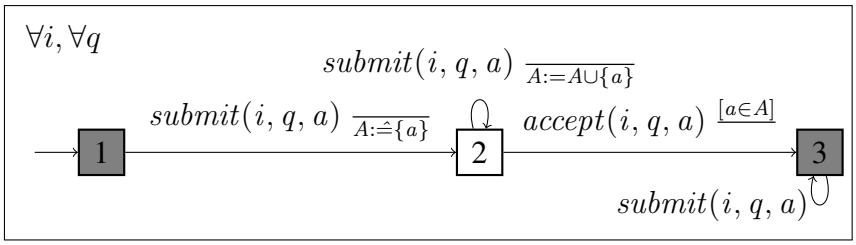

Figure 3: QEA for Answer Authentication

The QEA of *Answer Authentication* fails if an unsubmitted answer is accepted. A candidate can submit more than one answer to the same question, however 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 an answer from him to that question is accepted. 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 an accepted trace 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.

Decomposing *Answer Authentication*. Note, *Answer Authentication* can be further split into three different properties which respectively ensure that only submitted answers are accepted (*Answer Submission Authentication*), for every question an answer is accepted from a candidate that submits at least one answer (*Acceptance Assurance*), and only one answer is accepted from the same candidate for the same question (*Answer Singularity*) [9]. Note, the three properties are expressed over the same alphabet Σ_{AA}.

Definition 4. (Answer Submission Authentication). *Answer Submission Authentication is defined by the QEA depicted in Figure 3 with alphabet* $\Sigma_{\text{AA}} = \{submit(i, q, a), accept(i, q, a)\}$.

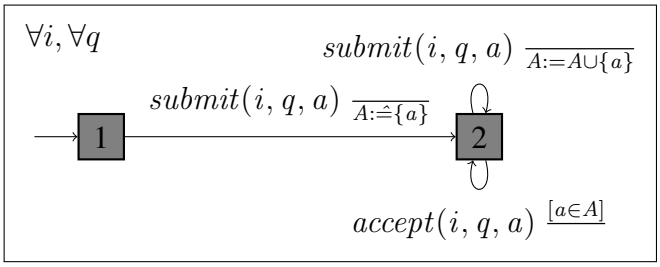

Figure 4: QEA for Answer Submission Authentication

The QEA of *Answer Submission Authentication* fails when an unsubmitted answer is accepted from a certain candidate regardless of whether that candidate submitted a different answer or not. Traces $accept(i_1, q_1, a_1)$ and $submit(i_1, q_1, a_1).accept(i_1, q_1, a_2)$ are both unaccepted traces by the QEA of Figure 4. In the former trace, an answer for question q_1 is accepted from candidate i_1, which did not submit any answer to question q_1. In the latter latter, candidate i_1 submitted an answer a_1 while a different answer a_2 is accepted from him.

Note, *Answer Submission Authentication* allows the case where no answer is accepted from a

[9] In our case study in Section 5.2, this property is violated on an e-exam run. Decomposing the property allows to locate the source of the error more precisely.

candidate, and the case where multiple answers are accepted from the same candidate as long as they are all submitted by that candidate.

Definition 5. (Acceptance Assurance). *Property* Acceptance Assurance *is defined by the QEA depicted in Figure 5 with alphabet* $\Sigma_{\text{AA}} = \{submit(i, q, a), accept(i, q, a)\}$.

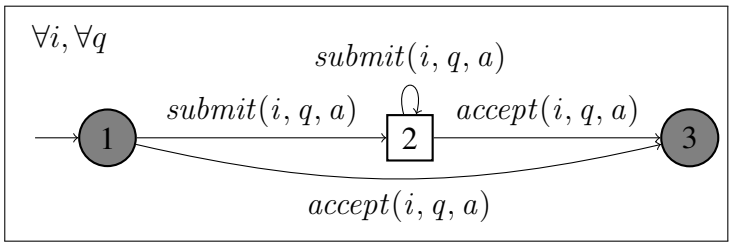

Figure 5: QEA for Acceptance Assurance.

The QEA of *Acceptance Assurance* fails when no answer is accepted from a candidate that submitted at least one answer. Note, the QEA succeed even if the accepted answer is not one of the submitted answers. Note, the latter case leads to the failure of *Answer Submission Authentication*.

Definition 6. (Answer Singularity). *Property* Answer Singularity *is defined by the QEA depicted in Figure 5 with alphabet* $\Sigma_{\text{AA}} = \{submit(i, q, a), accept(i, q, a)\}$.

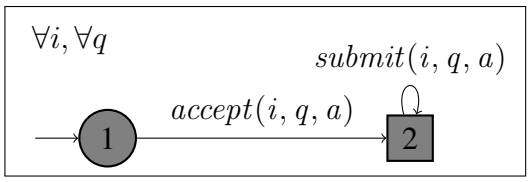

Figure 6: QEA for Answer Singularity

The QEA of *Answer Singularity* fails if more than one answer is accepted from the same candidate to the same question, even if all the accepted answers are submitted ones. Traces $accept(i_1, q_1, a_1)$ and $submit(i_1, q_1, a_1).accept(i_1, q_1, a_1)$ are an accepted traces. While trace $submit(i_1, q_1, a_1).accept(i_1, q_1, a_1).submit(i_1, q_1, a_2).accept(i_1, q_1, a_2)$ is not accepted as two answers are accepted from the same candidate to the same question.

4.1.4 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 7. (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 Figure 7 with alphabet $\Sigma_{\text{QO}} = \{get(i, q), accept(i, q)\}$.*

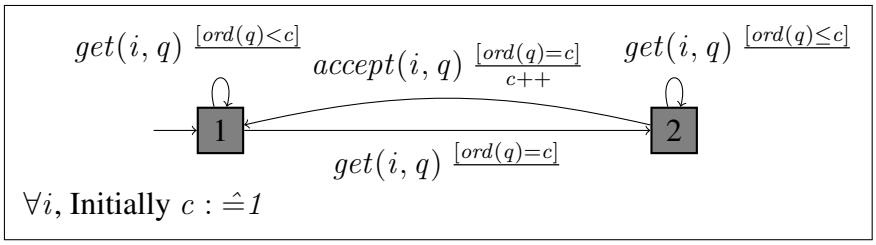

Figure 7: QEA for Question Ordering

The QEA of *Question Ordering* fails if, an answer is accepted from a candidate for a higher order question or if he gets a higher order question, before his answer to the current question is accepted. Note, variable c carries the order of the current question. Once an answer is accepted for the current question (transition from state (2) to state (1)), c is incremented and the candidate can then get the next question. Note, *Question Ordering* 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.

4.1.5 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 8. (Exam Availability). *Let t_0 and t_f be the starting and ending time instants of the exam, respectively. Property* Exam Availability *is defined by the QEA depicted in Figure 8 with alphabet $\Sigma_{\text{EA}} = \{get(i, t), change(i, t), submit(i, t), accept(i, t)\}$.*

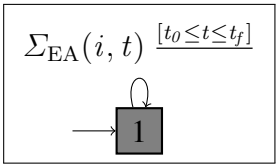

Figure 8: QEA for Exam availability

The QEA of *Exam Availability* checks that all the events in Σ_{EA} are emitted between t_0 and t_f. Note, any other event can be added to Σ_{EA} if required.

4.1.6 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 this possibility, we define *Exam Availability with Flexibility*. Property *Exam Availability with Flexibility* 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 9. (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 Figure 9 with alphabet $\Sigma_{\text{EA}} = \{begin(i,t), end(i), accept(i,t)\}$.*

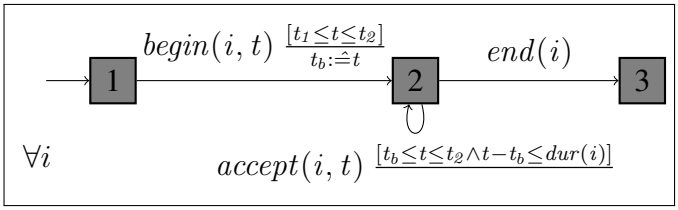

Figure 9: QEA for Exam availability with flexibility

Exam Availability with Flexibility also requires that, for each candidate i, there is only one event $begin(i,t)$ per exam, and also one event $end(i)$. A candidate can begin his exam at any time t_b such that $t_1 \leq t_b \leq t_2$. Note, an event $end(i)$ can be emitted after time t_2, however no answer can be accepted after t_2. Note also that, no answer can be accepted after the ending time t_2 even if the exam duration 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. However, traces $accept$ $(i_1, 5).begin\ (i_1, 20)$ and $begin(i_1, 0).accept(i_1, 91)$ 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, other events (*e.g.,* get and $submit$) can be considered. In such a case, the QEA in Figure 9 has to be edited by looping over state (2) with any added event.

4.1.7 Marking Correctness

Marking Correctness states that all answers are marked correctly.

Definition 10. (Marking Correctness). *Property* Marking Correctness *is defined by the QEA depicted in Figure 10 with alphabet* $\Sigma_{\mathrm{MC}} = \{corrAns(q, a), marked\}(q, a, b)\}$.

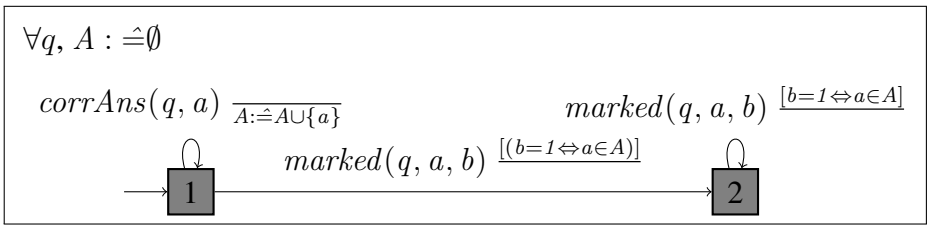

Figure 10: QEA for Marking correctness

In the QEA of *Marking Correctness*, the correct answers for the considered question are collected in a set A (self loop over state (1)). In state (1), once an answer to the considered question is marked correctly, a transition to state (2) is fired, otherwise if an answer is marked in a wrong way a transition to an implicit failure state occurs. In state (2), 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.

For example, trace $corrAns(q_1, a_c).marked(q_1, a_1, 1)$ where the wrong answer $a_1 \neq a_c$ is marked as correct, and trace $corrAns(q_1, a_{c1}).marked(q_1, a_{c1}, 1).corrAns(q_1, a_{c2})$ where the answer a_{c2} is declared as a correct answer to the question q_1 after an answer is already marked for that question, are not accepted by the QEA of *Marking Correctness*.

4.1.8 Mark Integrity

Property *Mark Integrity* states that all the 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 participating in the exam gets the correct mark corresponding to his answers.

Definition 11. (Mark Integrity). *Property* Mark Integrity *is defined by the QEA depicted in Figure 11 with alphabet* $\Sigma_{\mathrm{MI}} = \{accept(i, q, a), marked(q, a, b), assign(i, m)\}$.

The QEA of *Mark Integrity* collects, for each candidate, the answers that he submitted in set A. 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)).

4.2 Properties for Error Reporting

In the previous formalization, a property fails when its requirement is violated. However, it does not provide the entities that violate the requirement. In the following, we propose for

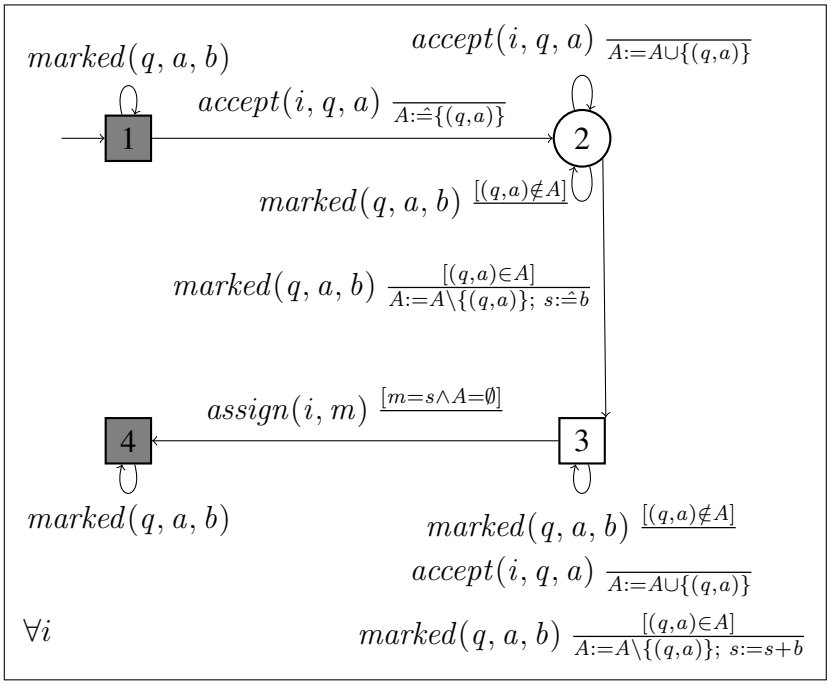

Figure 11: QEA for Mark integrity

each property an alternative that reports at the end all entities that violate the requirement of the property. In general, an alternative property ensures the same requirement(s) of the original regular property and has same input alphabet, but additionally reports some data at the end. Whenever the requirement formalized by the following QEAs differ from their counterparts in the previous subsection, we mention it.

4.2.1 Candidate Registration with Auditing

Candidate Registration with Auditing fails if an unregistered candidate submitted an answer and at the same time collects all such candidates in a set F.

Definition 12. (Candidate Registration with Auditing). *Property* Candidate Registration with Auditing *is defined by the QEA depicted in Figure 12 with alphabet* $\Sigma_{\text{CR}} = \{register(i), submit(i)\}$.

The QEA of *Candidate Registration with Auditing* has three free variables I, F, and i, and no quantified variable. Instead of being instantiated for each candidate i, the QEA of *Candidate Registration with Auditing* collects all the registered candidates in set I, so that any occurrence of event $submit(i)$ 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, *i.e.,* those that violate the requirement. For example, trace $register(i_1).submit(i_2, q, a_2).submit(i_1, q, a_1)$ $.register(i_2)$ is not accepted by *Candidate Registration with Auditing*, and results in the set $F = \{(i_2, q, a_2)\}$.

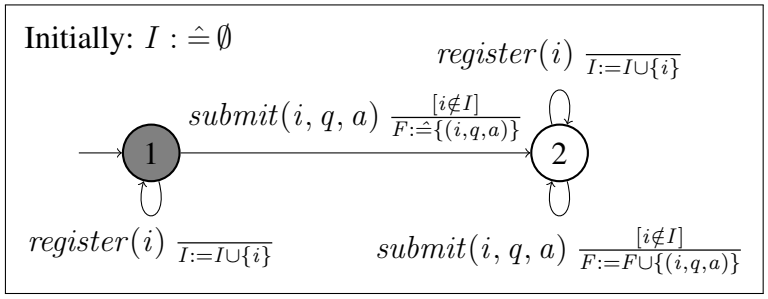

Figure 12: QEA for Candidate Registration with Auditing

4.2.2 Candidate Eligibility with Auditing

Candidate Eligibility with Auditing fails when an answer is accepted from an unregistered candidate, and reports all such candidates in a set F.

Definition 13. (Candidate Eligibility with Auditing). *Property* Candidate Eligibility with Auditing *is defined by the QEA depicted in Figure 13 with alphabet* $\Sigma_{\text{CE}} = \{register(i), accept(i, q, a)\}$.

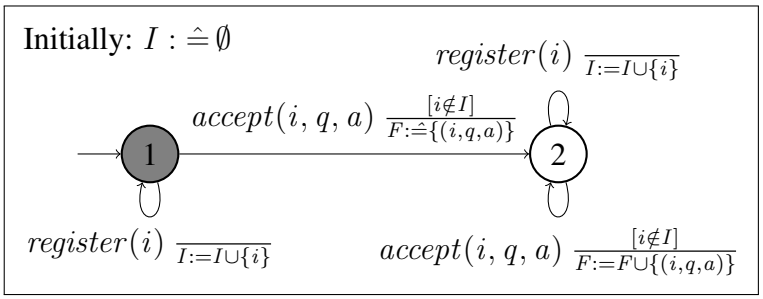

Figure 13: QEA for Candidate Eligibility with Auditing

The QEA of *Candidate Eligibility with Auditing* is similar to the QEA of *Candidate Registration with Auditing*, except that the event $submit(i, q, a)$ is replaced by the event $accept(i, q, a)$.

4.2.3 Answer Authentication with Auditing

Answer Authentication with Auditing fails when an unsubmitted answer is accepted, no answer to a certain question is accepted from a candidate that submitted an answer for that question, or more than one answer is accepted from the same candidate to the same question.

Definition 14. (Answer Authentication with Auditing). *Property* Answer Authentication *is defined by the QEA depicted in Figure 14 with alphabet* $\Sigma_{\text{AA}} = \{submit(i, q, a), accept(i, q, a)\}$.

The QEA of *Answer Authentication with Auditing* collects in a set F_1 all the unsubmitted answers that are accepted together with the corresponding candidates and questions. It also collects in a set F_2 the further accepted answers to the same question from the same candidate. Note, when the first answer is accepted the free variable c is set to 1. The sets F_1 and F_2 are

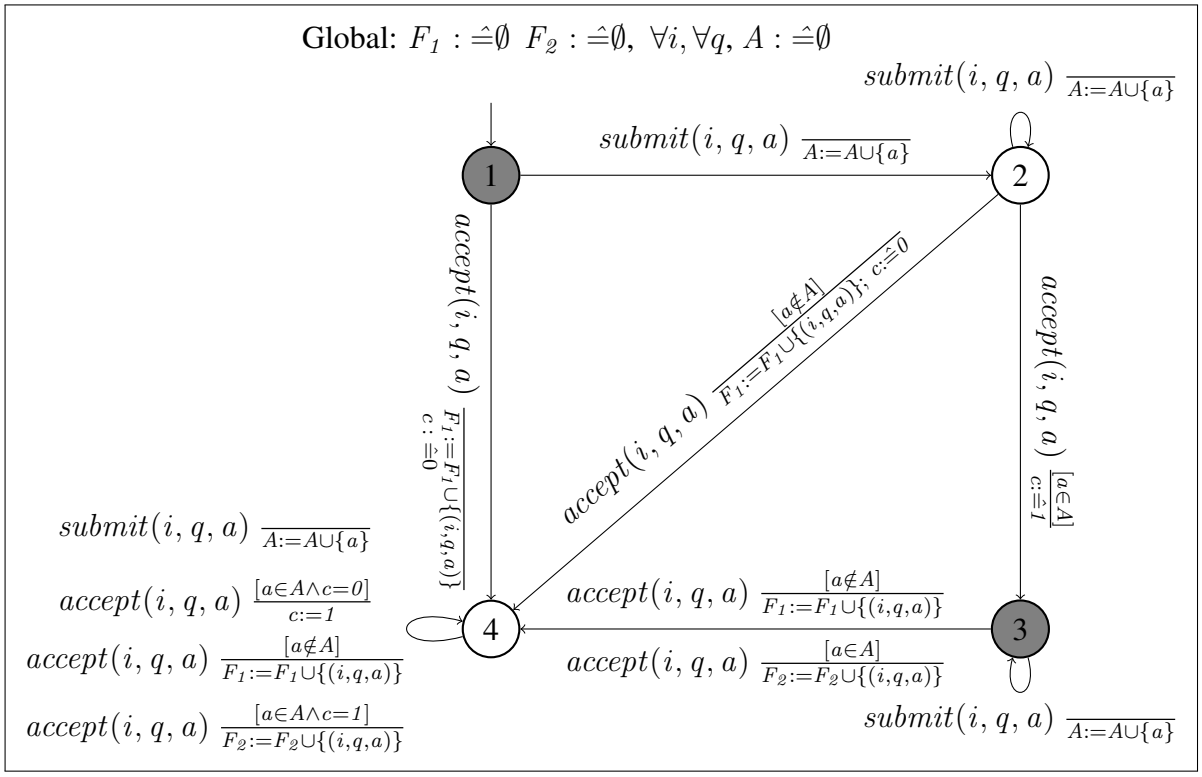

Figure 14: QEA for Answer Authentication with Auditing

globally defined, and thus they are sheared between all the instances of the QEA which result from the instantiation of the QEA for different i and q.

4.2.4 Questions Order with Auditing

Questions Ordering with Auditing has the same requirement as *Question Ordering*.

Definition 15. (Questions Ordering with Auditing). *Let $q_1, \ldots, q_n$ be n questions such that the order $ord(q_k)$ of q_k is k. Property* Questions Ordering with Auditing *is defined by the QEA depicted in Figure 15 with alphabet $\Sigma_{\text{QO}} = \{get(i, q), accept(i, q)\}$.*

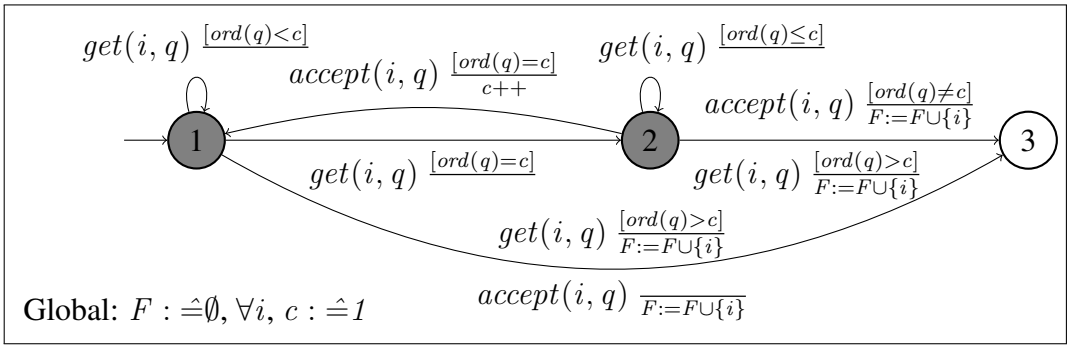

Figure 15: QEA for Questions Order with Auditing

The QEA of *Questions Ordering with Auditing* adds to a set F every candidate that:

1. gets a higher order question before answering his current question. This is guaranteed by the the two transitions from (1) to (3) and from (2) to (3) labeled by $get(i,q)$ $\frac{[ord(q)>c]}{F:=F\cup\{i\}}$; or
2. answers a question before getting it or answering an old question again. This is guaranteed by the transition from (1) to (3) labeled by $accept(i,q)$ $\frac{}{F:=F\cup\{i\}}$ and the transition from (2) to (3) labeled by $accept(i,q)$ $\frac{[ord(q)\neq c]}{F:=F\cup\{i\}}$.

Note, the set F is a global set.

4.2.5 Exam Availability with Auditing

The QEA of *Exam Availability with Auditing* checks that all the events in Σ_{EA} are emitted between t_0 and t_f. It also collects all the candidates that violet the property in a set F.

Definition 16. (Exam Availability with Auditing). *Let t_0 and t_f be the starting and finishing time of the exam respectively. Property* Exam Availability with Auditing *is defined by the QEA depicted in Figure 16 with alphabet $\Sigma_{\text{EA}} = \{get(i,t), submit(i,t), accept(i,t)\}$.*

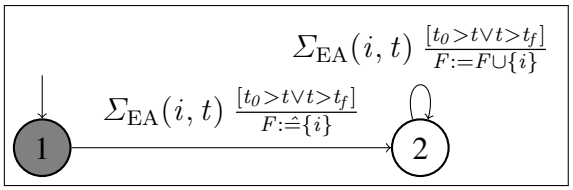

Figure 16: QEA for Exam availability with Auditing

4.2.6 Exam Availability with Flexibility Auditing

Questions Ordering with Auditing models the same requirement as *Question Ordering*, but additionally collects all candidates that violet the property in a global set F.

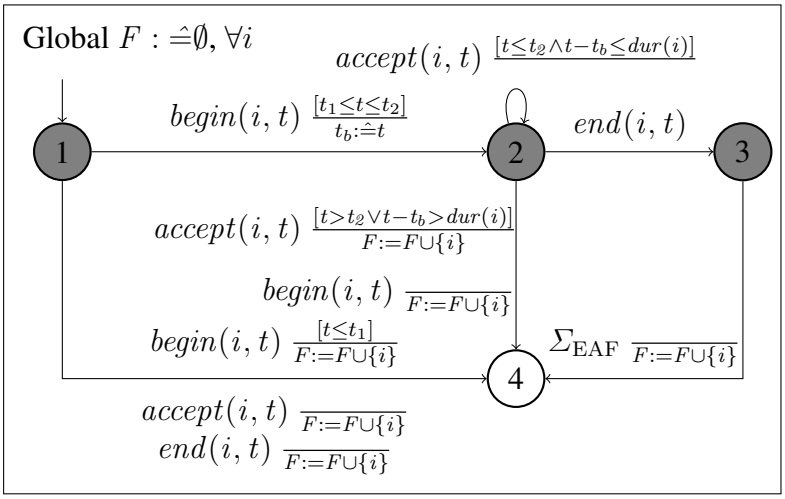

Figure 17: QEA for Exam availability with Flexibility Auditing

Definition 17. (Exam Availability With Flexibility Auditing). *Let t_1 and t_2 respectively be the starting and the finishing of the allowed period, and let $dur(i)$ be the exam duration of candidate i. Property* Exam Availability with Flexibility Auditing *is defined by the QEA depicted in Figure 17 with alphabet $\Sigma_{\text{EAF}} = \{begin(i,t),\ end(i),\ accept(i,t)\}$.*

The QEA of *Exam Availability with Flexibility Auditing* collects in a set F every candidate i that:

1. begins the exam before the starting instance t_1, or begins the exam twice; and
2. answers a question before the starting instance t_1, after the end of their duration, after he finish the exam (event $end(i)$) or after the ending instance t_2; and
3. make any action after he finish the exam (event $end(i)$); and
4. emits an event $end(i)$ before the starting instance t_1.

4.2.7 Marking Correctness with Auditing

Marking Correctness with Auditing collects all the answers that are marked in a wrong way in a global set F.

Note, the fact that, for a question q, no event $corrAns(q,a)$ can be emitted after the marking of the first answer is marked is relaxed. Simply, an answer that is not declared as an correct answer yet is considered a wrong answer.

Definition 18. (Marking Correctness with Auditing). *Property* Marking Correctness with Auditing *is defined by the QEA depicted in Figure 18 with alphabet $\Sigma_{\text{MC}} = \{corrAns(q,a),\ marked\}(q,a,b)\}$.*

4.2.8 Mark Integrity with Auditing

Property *Mark Integrity with Auditing* states that all the accepted answers are marked, and that exactly one mark is assigned to each candidate, the one attributed to his answers.

Note, a candidate that does not participate in the exam, *i.e.,* no answer is accepted from him, is assigned no mark. So, if a mark is assigned to a candidate that do not participate in the exam, the property fail (transition from (1) to (5)). While, a candidate that participate in the exam but none of his answers is a correct answer is assigned mark 0.

Set F_1 collects all the candidates that their first assigned mark is wrong. While, set F_2 collects all the further marks assigned for the candidates regardless if they are correct or not. Note if a candidate assigned a mark on transition from (3) to (4), his identity and mark are stored in i_c and m_c respectively. So that, if any further answer is accepted from him, then his identity is added to F_1 since this means that the mark i_c assigned to him is wrong.

Definition 19. (Mark Integrity with Auditing). *Property* Mark Integrity with Auditing *is defined by the QEA depicted in Figure 19 with alphabet* $\Sigma_{\text{MI}} = \{accept(i, q, a),\ marked(q, a, b),\ assign(i, m)\}$.

5 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[10] 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[11] [19] (Monitoring At Runtime with QEA) to model the QEAs

[10] www.theia.fr [11] https://github.com/selig/qea

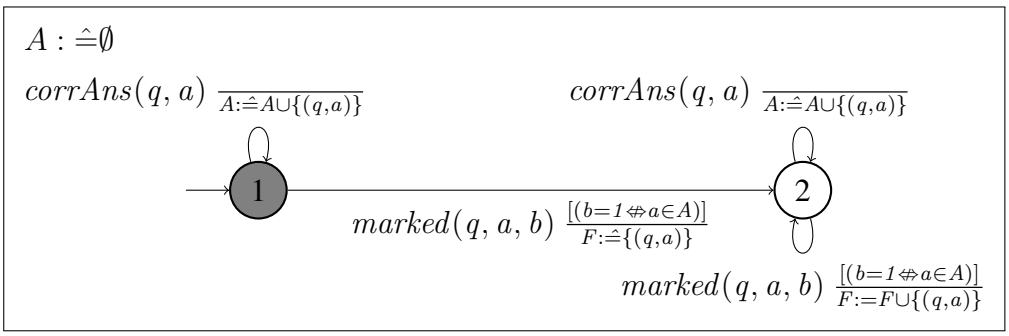

Figure 18: QEA for Marking correctness with Auditing

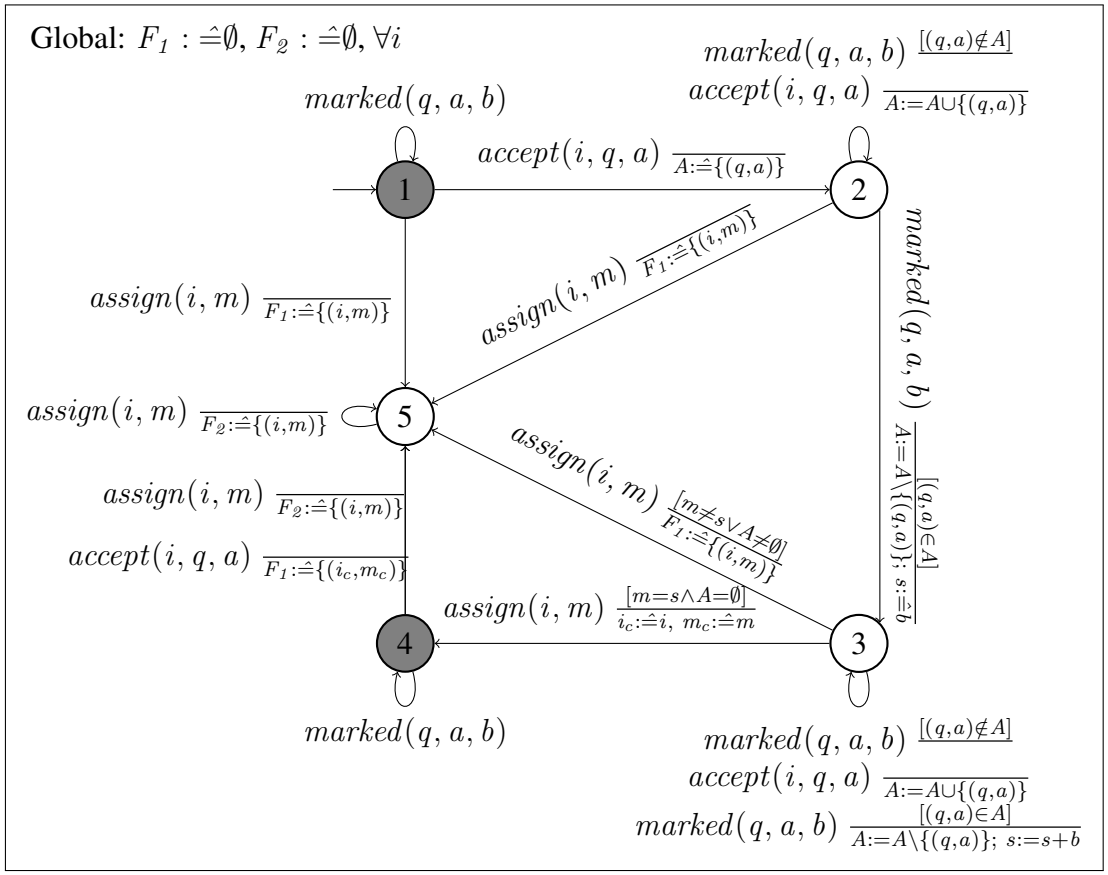

Figure 19: QEA for Mark integrity with Auditing

and perform the verification. We provide a description for this system that we call *UJF e-exam*[12], then we present the results of our analysis.

5.1 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 was provided with 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, and, only the exam and authentication servers can be accessed from the tablets. 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 pedagogi-

[12] 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.

cal 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 examiner 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.

5.2 Analysis

We analyzed two exams held in 2015: Exam 1 involves 233 candidates and contains 42 questions for a duration of 1 hour and 35 minutes. Exam 2 involves 90 candidates, contains 36 questions for a duration of 5 hours and 20 minutes. 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. For each property, the number of violations obtained from running the auditing variant of each QEA is also mentioned whenever it was possible to run them with the available data. (✓) means satisfied, (×) means not satisfied, and $[1]$ indicates the number of violations. Only four of the eight general properties presented in Sec. 4 were compatible with UJF E-exam. Note, some additional properties specific for the *UJF e-exam* are also considered.

Candidate Registration: On both e-exams, property *Candidate Registration* held, that is, no unregistered candidate submitted an answer during the two analyzed exams. Hence, there was no need to verify *Candidate Registration with Auditing*. However, we implemented *Candidate Registration with Auditing* to double check the result.

We note that, the current implementation of MarQ tool does not support the pre-initialization of variables. However, this is not a problematic for *Candidate Registration with Auditing* since the set is created when the first element is added to the set, and guard $[i \notin I]$ succeeds if set I is not created yet. 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). We

reported this issue to MarQ team.

We also note that, in the 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.

Candidate Eligibility: Property *Candidate Eligibility* also held on both e-exams. Then, all the answers were accepted from registered candidates in both exams.

Note the same comments about *Candidate Registration with Auditing*, are also hold for *Candidate Eligibility with Auditing*.

Answer Authentication: *Answer Authentication* was violated only in Exam 1, while it is satisfied by the Exam 2. To locate the source of error we check properties *Answer Submission Authentication*, *Acceptance Assurance*, and *Answer Singularity*. As a result, *Answer Submission Authentication* and *Acceptance Assurance* held, but *Answer Singularity* was violated. This means that more than one answer was accepted from the same candidate to the same question.

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: in the implementation, 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 is only due to the acceptance of a same answer twice, we updated property *Answer Singularity* and its QEA presented in Figure 3 by storing the accepted answer in a variable a_v on the transition from (1) to (2), and adding a self loop transition on state (2) labeled by $accept(i, q, a) \xrightarrow{[a=a_v]}$. We refer to this new weaker property as ***Answer Singularity*** *, which differs from *Answer Singularity* by allowing the acceptance of the same answer again; but it still forbids the acceptance of a different answer. We found out that *Answer Singularity* * is satisfied, which confirms the claim about the possibility of accepting the same answer twice.

We could not implement the QEA of *Answer Authentication with Auditing* using MarQ since it does not support the manipulation of quantified variables. However, we defined property *Answer Singularity with Auditing* presented in Figure 20, 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 global set F. Note, in the QEA of *Answer Singularity with Auditing*, only q is quantified and every candidate i who answers q is added to a local set A. Consequently, if a second answer is encountered from the same candidate i ($[i \in A]$), then i is added to F. The analysis of *Answer Singularity with Auditing* 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. Consequently, for *Answer Singularity with Auditing*, a set is required for each question. Note also that, for Exam 1, *Answer Authentication* required less monitoring time than *Answer Authentication* * and *Answer Singularity with Auditing* as the monitor for *Answer Authentication* stops monitoring as

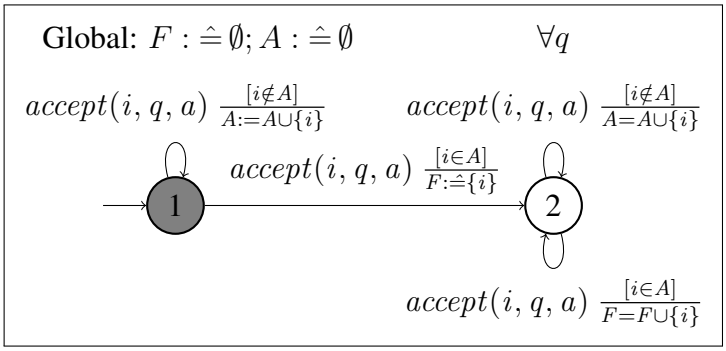

Figure 20: QEA for *Answer Singularity with Auditing*

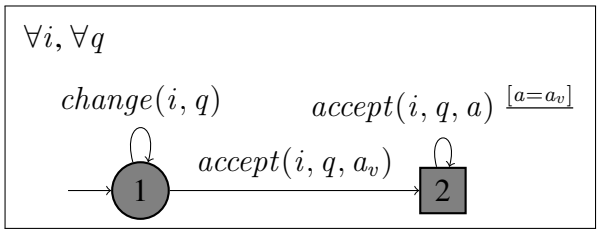

Figure 21: QEA for *Answer Integrity*

soon as it finds a violation.

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 Integrity* that states that a candidate cannot change the answer after acceptance. *Answer Integrity* is defined by the QEA depicted in Figure 21 with the input alphabet $\Sigma_{\text{AE}} = \{change(i, q), accept(i, q, a)\}$.

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

Questions Order: Regarding *Question Ordering*, the developers did not log anything related to the event $get(i, q)$. However, as we expect exam developers to add necessary logging, 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 Figure 22 with the input alphabet $\Sigma_{\text{QO}'} = \{change(i, q), accept(i, q)\}$. 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

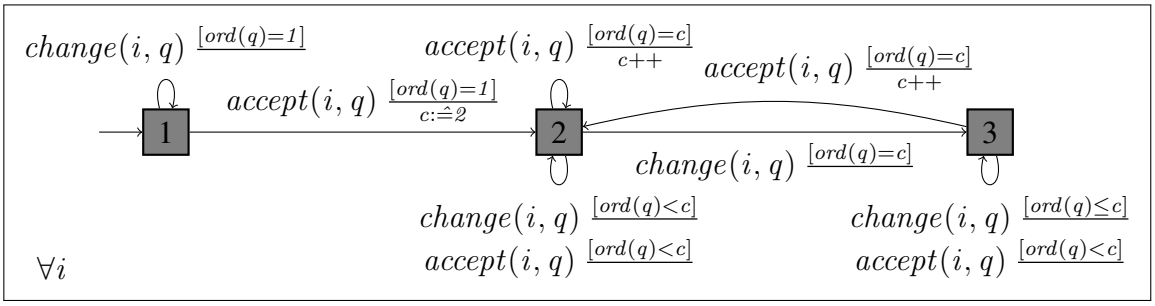

Figure 22: QEA for *Question Ordering* *

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.

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.

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 the violation of *Question Ordering* * is due to skipping some questions, we defined property *Acceptance Order* that, for each candidate, ignores the question he skipped (those that he did not validate) and checks whether all the accepted answers do not violates the required order order, *i.e.,* there should be no answer accepted for a question that is followed by an accepted answer for a lower order question. Note, *Acceptance Order* the allows a candidate to skip some questions, but he cannot violates the global order of the questions. It is defined by the QEA depicted in Figure 23 with the input alphabet $\Sigma_{\text{AO}} = \{accept(i, q, a)\}$.

Exam Availability: *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 also implement *Exam Availability with Auditing* which confirms the obtained result.

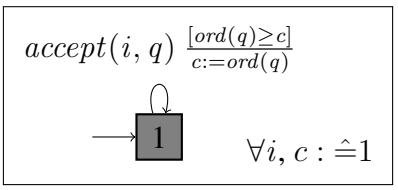

Figure 23: QEA for *Acceptance Order*

Note, we could not analyze *Exam Availability with Flexibility*, since the two analyzed exam have fixed starting and ending time instances. So, all the candidates have to take the exam during the specified instances, and thus there is no $begin$ and end events. Moreover, all the candidates have the same time duration.

We also did not consider *Marking Correctness*, and *Mark Integrity* properties since the developers did not log anything concerning 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 usually universities only try to detect cheating candidates, and do not look for internal problems or in-

Property	Exam 1		Exam 2	
	Result	Time (ms)	Result	Time (ms)
Candidate Registration (Definition 1)	✓	538	✓	230
Candidate Eligibility (Definition 2)	✓	517	✓	214
Answer Authentication (Definition 3)	×	310	✓	275
Answer Submission Authentication (Definition 4)	✓	497	✓	216
Acceptance Assurance (Definition 5)	✓	523	✓	227
Answer Singularity (Definition 6)	×(1)	587	✓	261
Answer Singularity * (Page 22)	✓	742	✓	223
Answer Integrity (Figure 21)	✓	641	×	218
Question Ordering * (Definition 7)	×	757	×	389
Acceptance Order (Figure 23)	✓	697	✓	294
Exam Availability (Definition 8)	✓	518	×(1)	237
Total		**6,327**		**2,784**

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

sider attacks. We expect the developers of the e-exam software to include logging features for every phase. Note however, we implemented *Exam Availability with Flexibility*, *Marking Correctness*, and *Mark Integrity* in MarQ and validated them on toy traces as we expect to obtain the actual traces of the marking phase in the near future. We also implemented *Marking Correctness with Auditing* in MarQ, however *Exam Availability with Flexibility Auditing* and *Mark Integrity with Auditing* are currently not supported by MarQ since they require global variables and manipulation of quantified variables.

All in all, the total analysis times for the first and second e-exam were 6,327 and 2,784 ms respectively; thus showing the usability of the approach.

6 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 [10]. Note, the proposed tests in [10] need to be instantiated for each exam depending on its specifications. The authors of [10] have validated their framework by i) modeling two exams in the applied π-calculus [1], and then ii) analyzing them using ProVerif [6]. 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 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 [10]. 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 [10] 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 [12], and voting [2, 15]. 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 [11]. These complementary approaches study security and not verification, however both aspects are important to develop such sensitive systems.

Küsters *et al.* [16] studied accountability ensuring that, when verifiability fails, one can identify the participant responsible for the failure. The authors also give symbolic and computational definitions of verifiability, which they recognize as a weaker variant of accountability. However, their framework needs to be instantiated for each application by identifying relevant verifiability goals. Guts *et al.* [14] defined auditability as the quality of a protocol that stores sufficient evidence to convince an honest judge that specific properties are satisfied. In case of failure, we are able to detect the responsible since our monitors are be to find sufficient evidences stored in the logs. This corresponds to a kind of accountability and auditability of such e-exams.

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 [4]: 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 [19] is a tool for monitoring Quantified Event Automata [3, 18]. 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, showing that MarQ is one of the most

efficient existing monitoring tools for both off-line and on-line monitoring. [13] Note, off-line runtime verification was successfully applied to other industrial case studies, *e.g.,* for monitoring financial transactions with LARVA [8], and monitoring IT logs with MonPoly [5].

7 Conclusions and Future Work

7.1 Conclusions

We define an event-based model of e-exams, and formalize several essential properties as Quantified Event Automata (QEA), 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 e-exams at UJF. We find several discrepancies between the specification and the implementation. We perform off-line verification of certain exam runs using the MarQ tool. 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.

7.2 Future Work

Several avenues for future work are open 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, using MarQtool, on-line verification with our monitors during live e-exams, and to study to what extent runtime enforcement (cf. [13] for an overview) can be applied during a live e-exam run. Finally, we plan to study more expressive and quantitative properties that might 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 from University of Manchester for answering our questions, and for providing us with help on using the

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

MarQ tool. This research was conducted with the support of the Digital trust Chair from the University of Auvergne Foundation.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL'01*, pages 104–115, New York, 2001. ACM. 6

[2] M. Backes, C. Hritcu, and M. Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *CSF*, pages 195–209, 2008. 6

[3] H. Barringer, Y. Falcone, K. Havelund, G. Reger, and D. E. Rydeheard. 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*, pages 68–84. Springer, 2012. 1, 2, 4, 6

[4] E. Bartocci, B. Bonakdarpour, and Y. Falcone. First international competition on software for runtime verification. In Bonakdarpour and Smolka [7], pages 1–9. 6

[5] D. A. Basin, G. Caronni, S. Ereth, M. Harvan, F. Klaedtke, and H. Mantel. Scalable offline monitoring. In Bonakdarpour and Smolka [7], pages 31–47. 6

[6] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSFW*, pages 82–96, Cape Breton, Canada, JUN 2001. IEEE Computer Society. 6

[7] B. Bonakdarpour and S. A. Smolka, editors. *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. 4, 5

[8] C. Colombo and G. J. Pace. Fast-forward runtime monitoring - an industrial case study. In S. Qadeer and S. Tasiran, editors, *Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers*, volume 7687 of *Lecture Notes in Computer Science*, pages 214–228. Springer, 2012. 6

[9] L. Copeland. School cheating scandal shakes up Atlanta. http://www.usatoday.com/story/news/nation/2013/04/13/atlanta-school-cheatring-race/2079327/, April 2013. 1

[10] J. Dreier, R. Giustolisi, A. Kassem, P. Lafourcade, and G. Lenzini. A framework for analyzing verifiability in traditional and electronic exams. In *11th International Conference on Information Security Practice and Experience (ISPEC'15)*, 2015. 3, 6

[11] J. Dreier, R. Giustolisi, A. Kassem, P. Lafourcade, G. Lenzini, and P. Y. A. Ryan. Formal analysis of electronic exams. In *SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria*, pages 101–112, 2014. 6

[12] J. Dreier, H. Jonker, and P. Lafourcade. Defining verifiability in e-auction protocols. In *8th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '13, Hangzhou, China*, pages 547–552, 2013. 6

[13] Y. Falcone. You should better enforce than verify. In H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. J. Pace, G. Rosu, O. Sokolsky, and N. Tillmann, editors, *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, volume 6418 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 2010. 7.2

[14] N. Guts, C. Fournet, and F. Zappa Nardelli. Reliable evidence: Auditability by typing. In *ESORICS'09*, volume 5789 of *LNCS*, pages 168–183. Springer, 2009. 6

[15] S. Kremer, M. Ryan, and B. Smyth. Election verifiability in electronic voting protocols. In *ESORICS'10*, volume 6345 of *LNCS*, pages 389–404. Springer, 2010. 6

[16] R. Küsters, T. Truderung, and A. Vogt. Accountability: definition and relationship to verifiability. In *CCS'10*, pages 526–535. ACM, 2010. 6

[17] Le Figaro. Etudiants: les examens sur tablettes numériques appellés à se multiplier. Press release, January 2015. Available at goo.gl/ahxQJD. 1

[18] G. Reger. Automata Based Monitoring and Mining of Execution Traces. PhD thesis, University of Manchester, 2014. 1, 4, 6

[19] G. Reger, H. C. Cruz, and D. E. Rydeheard. MarQ: Monitoring at runtime with QEA. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS, London, UK*, pages 596–610, 2015. 1, 5, 6

[20] R. Watson. Student visa system fraud exposed in BBC investigation. http://www.bbc.com/news/uk-26024375, feb 2014. 1