Work-in-Progress: Automatically Generated Response-Time Proofs as Evidence of Timeliness

Marco Maida, Sergey Bozhko, and Björn B. Brandenburg
Max Planck Institute for Software Systems (MPI-SWS)

I. Introduction

The purpose of a *response-time analysis* (RTA) is to obtain *safe* bounds on the worst-case response times of all critical tasks in a real-time system. To this end, the system is described with a mathematical model (typically, comprising a workload model, a resource model, and a scheduling policy), which is then analyzed to derive response-time bounds. This procedure requires **(i)** a *theory* that rigorously justifies that the RTA correctly characterizes the worst-case scenario, and **(ii)** an RTA *tool* that executes the concrete calculations.

Both are equally critical for the correctness of the computed bounds. An error in (i), such as an invalid over-generalization or a missed corner case, leads to a flawed theory. An error in (ii), which can be any significant bug in the tool, leads to a flawed implementation. Given the number of documented analysis mistakes in the real-time literature (*e.g.*, [5, 7, 9, 11]), and the reality that complex tools are rarely bug-free, both the research community and industry have shown a growing interest in the application of computer-assisted formal verification (*e.g.*, [10, 12]), while safety standards also started advising its use (*e.g.*, ISO 26262, DO-178C). Most relevant in the context of RTA, Cerqueira et al. introduced Prosa [4], the to-date largest *mechanized* (*i.e.*, machine-checked) framework for real-time schedulability analysis, based on the *Coq* proof assistant [1].

Applying the currently available proof assistants and verification tools to mechanize RTAs and formally verify RTA tools is neither an easy nor a cheap task. Once a theory — or the behavior of a program — is encoded in a proof assistant's specification language, it usually needs to be augmented with additional information (*e.g.*, step-by-step proofs, program invariants) before the verification procedure can succeed. This process requires human intervention and a significant amount of time. Moreover, developing and maintaining a verified tool requires advanced programming skills and specialized expertise. Lowering the cost and knowledge barriers blocking access to the benefits of formal verification remains, therefore, a major challenge for the adoption of formal methods in industry.

This project seeks to design and implement an RTA tool that can assert the correctness of its results with high confidence, without any need for the tool itself to be verified. Our RTA tool, called POET (*Prosa Obsigned Evidence of Timeliness*), works in conjunction with Bozhko and Brandenburg's abstract RTA theory [3]. When a problem instance (*i.e.*, a concrete task set, scheduling policy, and preemption model) is given as input, POET produces, along with the RTA results, a set of certificates of correctness that are then automatically machine-checked by the Coq proof assistant [1]. Each certificate contains a proof of correctness of the computed response-time bound of one task. The process is completely automated and does not require any expertise with formal verification on behalf of the user.

Given that POET directly produces formal proofs of correctness in Coq, it can be regarded as the first *foundational* [2] RTA tool. The critical advantage of foundational tools is that the *trusted computing base* (TCB) is reduced to the proof checker and its dependencies: the tool itself does *not* need to be trusted. Therefore, the source code can be updated, modified, ported, and optimized like any common non-critical software. In fact, we developed POET in Python, a convenient but notoriously complex-to-verify language.

Foremost, POET makes it possible for users unfamiliar with Coq to benefit from the power of formal verification and, in particular, from the Prosa open-source libraries [3, 4].

In this paper:

- We report on the ongoing development of POET, the first foundational and automated response-time analysis tool. The certificates produced by POET are short, readable, and fully commented Coq files that can be machine-checked in (usually) minutes.
- We describe the mechanisms implemented in POET to ensure the validity of the generated certificates.
- We report on a preliminary evaluation of the effectiveness and practicality of our approach on synthetic task sets.

II. Background

Coq is an interactive theorem prover with which mathematical theories can be formalized and machine-checked. It provides two languages: *Gallina*, a formal specification language to write mathematical definitions and functional programs, and *Ltac*, an untyped macro language used to steer the proof engine.

While proof checking does not require human intervention, Coq is not a fully automatic theorem prover: once a theorem is specified, it is necessary to provide a sequence of *tactic* applications (each of which can be seen as a single step of the proof) that, starting from the stated hypotheses, allows the proof engine to reach the claimed conclusion. Given that Coq allows the user to specify custom tactics via the Ltac language, it is possible to introduce domain-specific automation. This feature is heavily used in the implementation of POET.

Once a Coq source file (*.v*) has been written, it can be compiled into a lower-level representation (generating a *.vo* file) and finally machine-checked by the standalone checker *coqchk*. These two tools (the Coq compiler and the Coq checker) and their dependencies represent the entire TCB of POET.

Prosa, the proof framework underlying POET's certificates, is based on Coq and its popular extension *ssreflect* [13]. Starting from classic real-time systems concepts, such as *job*, *task*, *processor*, and *arrival curve*, the contributors of Prosa mechanized several classical results (*e.g.*, the optimality of the *earliest-deadline-first* scheduling policy) as well as new ones, the most relevant for this paper being *abstract RTA* (aRTA) [3].

aRTA formalizes the well-known concept of the busy-window principle to derive a generic RTA that is applicable to different types of workload, scheduling policy, and preemption model. This abstract core has been instantiated for two scheduling policies (*earliest-deadline-first, fixed-priority*) and for four preemption models (*preemptive, non-preemptive, limited-preemptive, floating non-preemptive*) in every possible combination, yielding eight different fully verified RTAs.

Given that all proofs are mechanized, aRTA is undoubtedly correct. However, a mechanized RTA theory, much like its traditional pen-and-paper counterpart, only describes *how* to obtain the response-time bounds, but it is not, per se, a program that can yield numerical results given a concrete task set. The theory of aRTA is, in fact, developed by treating the scheduler, the tasks and the claimed response-time bounds as variables on which a number of assumptions are made.

The key idea at the base of POET is that, by providing instantiations for all variables (*i.e.*, assigning a concrete value to each), and proving that they satisfy all assumptions, the theorems of aRTA can be put to use to formally verify precomputed response-time bounds.

III. POET: Design and Workflow

POET is a tool that generates *certificates* (*i.e.*, Coq files containing formal proofs of correctness for a given response-time bound) for some of the concrete RTAs supported in aRTA. At the time of writing, POET is already fully functional for *fully-preemptive fixed-priority* (FP-FP) and *non-preemptive fixed priority* (NP-FP) scheduling and *arbitrary-deadline periodic tasks*.

In the following, we discuss the challenges and design decisions involved in automating the generation of RTA certificates, and the resulting workflow.

Usability of the tool. For the idea behind POET to be successful, the tool must remain accessible to a general audience without any expertise in formal verification. In particular, users must not be expected to be proficient in authoring Coq proofs. To generate proven RTA results, POET indeed only requires a specification file containing the task set information, the scheduling policy, and the preemption model. An example is presented in Fig. 1.

Transparency and trustworthiness. Since the process of calculating the response-time bounds, generating formal proofs of their correctness, and then machine-checking the proofs is entirely automated, it is necessary to avoid silent failures and make it possible for a human to scrutinize the certificates. Moreover, there are ways in which correctly machine-checked

```
scheduling policy: FP    # fixed-priority
preemption model:  FP    # fully-preemptive
task set:

- id: 1
  worst-case execution time: 20
  period: 100
  deadline: 100
  priority: 2

- id: 2
  worst-case execution time: 10
  period: 120
  deadline: 100
  priority: 1
```

Fig. 1. An example input file (in YAML format) describing a task set with two tasks scheduled under the fully-preemptive fixed-priority (FP-FP) policy.

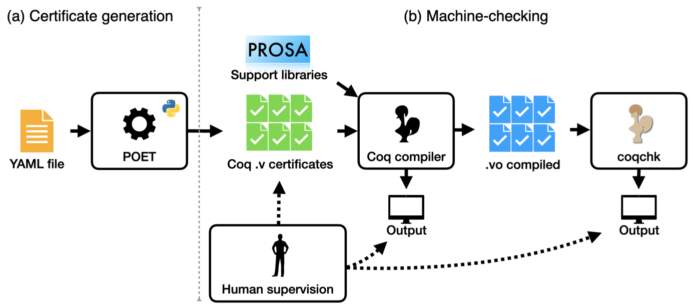

Fig. 2. The POET workflow.

proofs might still not justify the intended conclusions. This issue is further discussed in Section IV.

Scalability. Numerically-large computations are infeasible with the standard unary-based representation of numbers employed by Coq and ssreflect. However, POET's certificates need to support large numbers due to the numerical magnitude of real-world task sets (periods, costs, and deadlines are often expressed in nanoseconds or processor cycles). We discuss how to obtain nonetheless efficient calculations in Section V.

Workflow. The aforementioned design issues lead to the workflow illustrated in Fig. 2. Conceptually, the entire procedure comprises two parts, namely **(a)** the generation of certificates starting from an input file provided by the user, and **(b)** the compilation and machine-checking done by Coq.

The input file (recall Fig. 1), contains the necessary information about the task set, the scheduling policy, and the preemption model. Given an input file, POET produces one certificate (*.v* file) per task by instantiating a template specific to the given scheduling policy and preemption model. During this phase, Coq itself is not involved in any way. Once the certificates are in place, POET triggers first the Coq compiler, which will produce compiled files (*.vo*) containing low-level *proof terms*, and finally the Coq checker (coqchk), which will verify them. Note that parts (a) and (b) are independent and performed by different tools (POET and Coq). In particular, the second part — compilation and verification of the certificates

— may be performed repeatedly and on different machines.

The user does not need to act at any step, but we expect some degree of supervision, as discussed next.

IV. Trustworthiness of the procedure

Since POET itself does not need to be trusted, in this section we focus on how machine-checked, auto-generated certificates could still be subject to fallacies leading to wrong conclusions. Only the Coq toolchain is assumed to work correctly.

A. Incomplete proofs

Since POET is not assumed to be correct, measures need to be taken to ensure that no incomplete proofs can be silently generated and machine-checked. Coq and POET are working independently of one another; hence POET might conceivably generate an incomplete (or, in the extreme case, even completely empty) certificate file that Coq would then successfully machine-check. Furthermore, Coq gives the possibility to *admit* theorems, *i.e.*, to accept them as valid without giving any proof, therefore treating them as axioms.

Though these edge cases could be easily programmatically detected by POET itself, doing so would implicitly turn the tool into a trusted component. For the same reason, POET cannot be in charge of reporting the results of the verification attempt to the user. After the certificates have been generated, POET must not intervene in any way. Therefore, any action that needs to be executed after the creation of the certificates *is handled entirely in the Coq environment*. This includes printing, which is done directly in Coq, and checking that no theorem has been *admitted* (using coqchk).

To completely eliminate the need to trust POET, the certificates are designed to be human-readable. Supervision is required to **(1)** check that the input file and the generated certificates match in terms of task set, scheduling policy and preemption model and **(2)** observe the output of the Coq compiler and checker to assess whether they succeeded.

Finally, it is possible for an experienced user to scrutinize the complete list of proof steps (*i.e.*, tactic applications) that lead to the response-time bound. Though it is generally not necessary to closely inspect the certificates, striving for readability increases their quality and renders them suitable as transparent evidence of temporal correctness since they can be dissected and studied up to their fundamental definitions (as provided by Prosa).

B. Contradicting hypotheses

A second, more subtle type of error is related to the possible existence of contradicting hypotheses inside the certificates. In this scenario, conclusions reached in a sound way may still be incorrect (note that, in general, it is not possible to detect contradicting hypotheses automatically). This potential pitfall has been described in-depth by Cerqueira et al. [4]. It is hence necessary to show that it is possible to instantiate each of the variables such that all hypotheses are respected.

POET's certificates generalize over only one variable, namely, the *arrival sequence*, whose purpose is to yield, for each given instant, a sequence of new jobs (each of which has a specific cost) that have been released by their respective tasks. This is done because, like most RTAs, aRTA considers every possible combination of job arrivals and job costs that respect the workload constraints (*e.g.*, that jobs arrive periodically).

To prove the absence of contradictions beyond any doubt, POET generates, in addition to the certificate of correctness of the general response-time bound, a *second* certificate for each task free of any variables or hypotheses (*i.e.*, a concrete arrival sequence with fixed job costs) and for which the response-time bound is proven once more to hold. Instantiating any valid arrival sequence suffices to show that the hypotheses are contradiction-free; we chose the concrete arrival sequence that, at each instant t, maximizes the number of arrivals in the interval $[0, t]$ while respecting all workload constraints.

V. Scalability of the certification procedure

POET's certificates depend on Prosa, and therefore implicitly on ssreflect. We found that, without adopting any further solution, certification time grew prohibitively with increasing costs, periods, and deadlines. The root cause turned out to be that ssreflect employs a *unary* representation of numbers. This has clear advantages when writing proofs, as it simplifies inductive reasoning and case analyses. However, even a moderately large unary number (like one billion) takes considerable time to instantiate and can easily trigger a stack overflow in the Coq compiler. Roughly, on our test machine (described in Section VI), the certification process stays somewhat feasible (*i.e.*, takes hours) despite the unary representation as long as costs, periods, and deadlines remain in the order of 10^6.

For example, consider the task set in Fig. 1, which uses milliseconds as its unit of measure. If we instead express all parameters in microseconds (*i.e.*, multiply worst-case execution times, periods, and deadlines by 10^3), we do not, in principle, change the complexity of the RTA problem. However, when using ssreflect's number representation, this has an enormous impact on performance, pushing the certification time from less than three seconds to around 15 minutes. A unary representation hence renders it impossible to support nanosecond resolution (and would therefore severely limit the applicability of POET).

Instead, POET's certificates employ a *binary representation* of numbers. However, this is easier said than done: since the aRTA library [3] expects unary numbers, trying to apply its definitions and functions to binary-encoded inputs would result in type-checking errors. Therefore, the necessary aRTA calculations were re-implemented to support binary numbers and connected to the existing aRTA proofs using CoqEAL [6], a framework for changes in data representation. The support code necessary to speed up the certifications of FP-FP task sets is roughly 600 LOCs of definitions, proofs, and tactics, and a similar amount is required for NP-FP. Note that CoqEAL is not part of the TCB since, as a Coq library, it is itself subject to full verification by coqchk when a certificate is machine-checked.

The switch to binary representation dramatically impacts runtime and memory needs, rendering nearly instantaneous previously infeasible operations. Consider once again the task set in Fig. 1 (expressed in milliseconds). With the binary

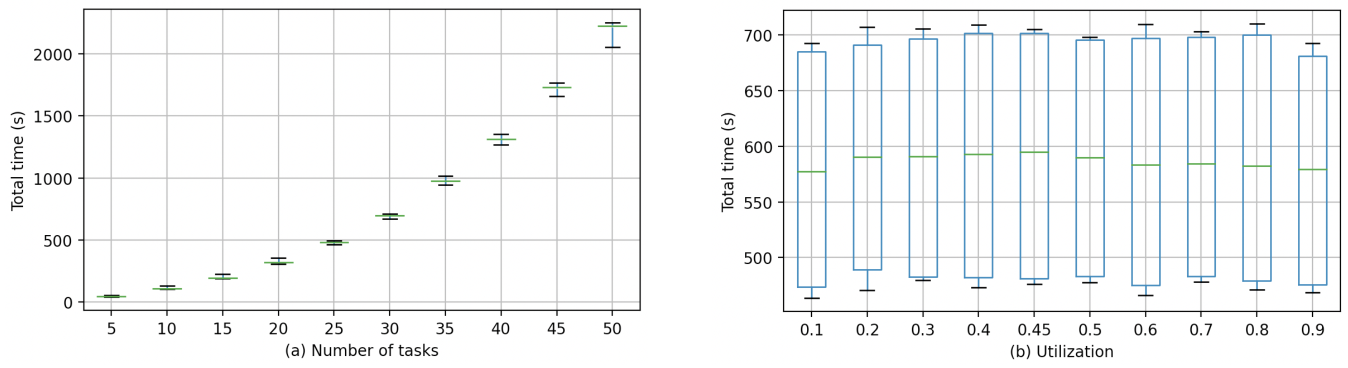

Fig. 3. Total time taken to generate and machine-check the certificates for an entire task set, w.r.t. (**a**) number of tasks in the task set and (**b**) total utilization. Each box shows the *interquartile range* (IQR); the median is indicated by a horizontal line. Boxes are omitted in (a) for clarity. Whiskers extend to the minimum and maximum measured values. Inset (b) shows workloads with 25 and 30 tasks.

representation in place, the total certification time is around five seconds on our testing machine (*i.e.*, slightly slower than before) but stays roughly the same irrespective of whether the task set is expressed in microseconds or nanoseconds. For task sets with small parameters (on the order of 10^2), the unary representation remains faster as the overhead of the translation to binary exceeds the cost of the calculation. However, in this edge case, the total certification time is quite low, and hence the difference between the two approaches is negligible.

VI. Preliminary Evaluation

To assess whether POET can produce certificates for task sets of realistic complexity (in terms of task count, utilization, and numerical magnitude of the data), we ran the certification process on 540 synthetic task sets generated using Emberson et al.'s unbiased task-set generator [8]. The number of tasks ranged from 5 to 50 in steps of 5. Utilization ranged from 0.1 to 0.9 in steps of 0.1. We expressed time in nanoseconds, which means that the average numerical magnitude of parameters was in the order of 10^9. All experiments were run on a 2.5 GHz Intel Xeon Platinum 8180 processor with 376 GB RAM.

Considering once again the workflow as depicted in Fig. 2, our experiments have shown that the processing time is largely dominated by part (b). In other words, the time taken by POET to perform the RTA and generate the certificates is negligible w.r.t. the time taken by Coq to check the certificates. Hence, the preliminary evaluation does not distinguish between the two and only shows the total time taken for both (a) and (b).

As can be seen in Fig. 3, certification time grows with the number of tasks (Fig. 3a) while it is not affected by the total utilization (Fig. 3b). The super-linear growth apparent in Fig. 3a is due to the fact that adding a task both increases the number of certificates that must be checked while also increasing the the complexity of the certificates of all prior tasks.

Although machine-checking times significantly differ from task to task (and, consequently, from task set to task set), in every test POET was able to certify the task set under analysis in at most 45 minutes. We interpret these results to indicate that the design of POET works and scales to realistic task sets.

VII. Conclusion

Our preliminary results show that foundational RTA tools that certify the bounds they produce with machine-checkable proofs of correctness, like POET, can be a viable alternative to verified RTA tools. Going forward, we plan to extend POET to support sporadic tasks. Moreover, we intend to explore RTAs for different preemption models and *earliest-deadline first* scheduling.

References

[1] "The Coq Proof Assistant," https://coq.inria.fr.
[2] A. W. Appel, "Foundational proof-carrying code," in *LICS*, 2001.
[3] S. Bozhko and B. B. Brandenburg, "Abstract Response-Time Analysis: A Formal Foundation for the Busy-Window Principle," in *ECRTS*, 2020.
[4] F. Cerqueira, F. Stutz, and B. B. Brandenburg, "PROSA: A case for readable mechanized schedulability analysis," in *ECRTS*, 2016.
[5] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley *et al.*, "Many suspensions, many problems: a review of self-suspending tasks in real-time systems," *Real-Time Systems*, vol. 55, no. 1, 2019.
[6] C. Cohen, M. Dénès, and A. Mörtberg, "Refinements for free!" in *CPP*, 2013.
[7] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, 2007.
[8] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *WATERS*, 2010.
[9] A. Gujarati, F. Cerqueira, and B. B. Brandenburg, "Schedulability analysis of the Linux push and pull scheduler with arbitrary processor affinities," in *ECRTS*, 2013.
[10] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish *et al.*, "seL4: Formal verification of an OS kernel," in *SOSP*, 2009.
[11] K. Lakshmanan, D. de Niz, and R. Rajkumar, "Coordinated task scheduling, allocation and synchronization on multiprocessors," in *RTSS*, 2009.
[12] X. Leroy, S. Blazy, D. Kästner, B. Schommer, M. Pister, and C. Ferdinand, "CompCert-a formally verified optimizing compiler," in *ERTS*, 2016.
[13] A. Mahboubi and E. Tassi, *Mathematical Components*. Zenodo, 2021.