Application of ACTLChecker to the ICT Approval Protocol at IGFEJ as part of OptiGov

José João Alves dos Santos Ferreira

March 30, 2026

1 Introduction

The ACTLChecker tool was developed as part of my master's thesis [Fer25] to enable efficient offline monitoring of distributed protocol executions. Initially, it was applied theoretically to the two-phase commit protocol and concretely to OpenChord, a distributed hash table protocol, achieving highly satisfactory results.

Subsequently, an opportunity arose to adapt the tool, which had been designed with minimal future adaptation effort in mind, to support the verification of any protocol that could be logged in execution traces. This use case was precisely the verification and possible optimization of public administration processes.

In this context, we applied the tool to verify properties of a protocol from the Ministry of Justice of Portugal. This protocol requires that justice bodies, such as courts, submit requests for prior review to the *Instituto de Gestão Financeira e Equipamentos da Justiça* (IGFEJ) for approval of projects involving the acquisition or adaptation of information and communication technology (ICT) equipment or software. The process is represented as a BPMN diagram in Section 2.

To accomplish this, the tool was adapted (described in detail in Section 3), and new properties were defined using the formal logical framework introduced in my thesis [Fer25], Action Temporal Logic (ACTL). This framework is particularly suited for operations that involve time intervals and often run concurrently. Additionally, finite execution traces were generated to closely approximate the expected behavior of these processes throughout their dynamic execution.

The theoretical and formal work of my master's research, as well as the application of the tool to process optimization, was partially supported by the OptiGov project (ref. n. 2024.07385.IACDC, DOI: 10.54499/2024.07385.IACDC), fully funded by the *Plano de Recuperação e Resiliência* (PRR) under the investment *Ciência Mais Digital* (measure RE-C05-i08.m04). This project is framed within the financing agreement signed between the *Estrutura de Missão Recuperar Portugal* (EMRP) and *Fundação para a Ciência e a Tecnologia* (FCT) as an intermediary beneficiary. This work was also partially supported by Portuguese national funds through FCT under projects UID/50021/2025 and UID/PRR/50021/2025.

2 ICT Approval Protocol

The process for the acquisition or adaptation of ICT equipment within the justice system constitutes a formal collaboration between four key participants: the initiating *Judicial Body* (JB), the oversight entity (IGFEJ), the *Ministerial Representative* (MR), and the *Administrative Modernization Agency* (AMA). As depicted in the BPMN diagram in Figure 1, the workflow is orchestrated primarily by the IGFEJ, which acts as the central hub for receiving, analyzing, and resolving requests initiated by the courts.

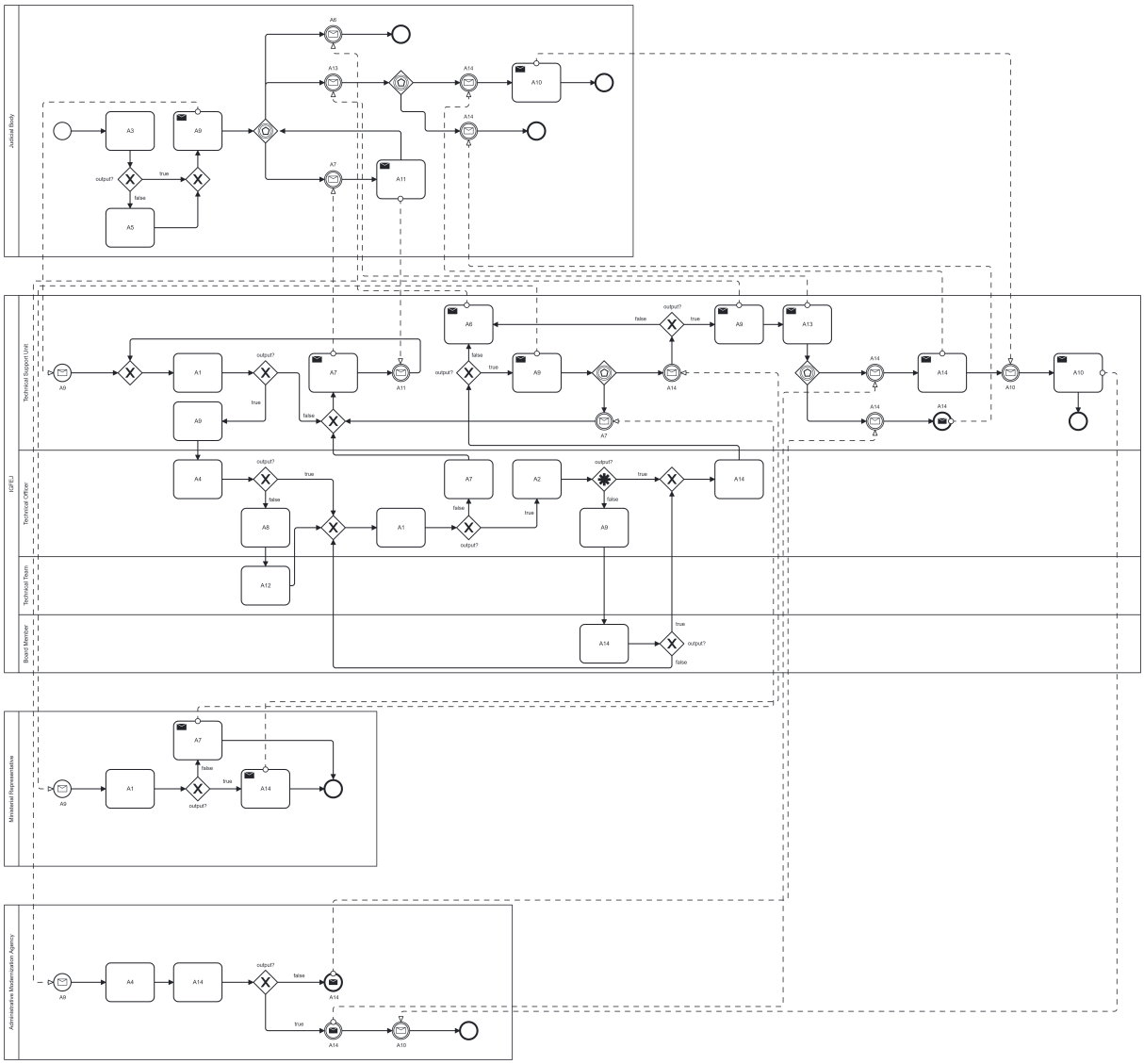
The IGFEJ, as far as an instance of this process is concerned, is constituted by the *Technical Support Unit* (TSU), the *Technical Officer* (TO), the *Technical Team* (TT), and the *Board Member* (BM).

The diagram in Figure 1 is the result of optimizing the workflow of a previous diagram that is not publicly available, with the purpose of becoming cleaner and more homogeneous. Therefore, to preserve the confidentiality of this protocol, the activities in the figure have been anonymized in a generic way.

The workflow begins in the *Judicial Body* pool, with the initiative of a JB, with activity A3, optionally followed by A5, before an opinion request is sent to the TSU as A9. We then enter into the IGFEJ pool, more specifically into the *Technical Support Unit* lane, where a TSU conducts A1; if information is missing, the process loops through A7 until the JB provides the required data in A11. Once complete, the TSU selects the A9 path. The assessment then moves through the internal hierarchy of IGFEJ. Inside the *Technical Officer* lane, a TO performs A4, may consult a TT if the process moves through A8, and, if needed, seeks the opinion of a BM. Notice that both the *Technical Team* and *Board Member* lanes are optional in an instance of a ICT approval request. This internal cycle produces a consolidated view. In parallel or subsequently, the TSU is to request external opinions from a MR and the AMA,

which return binding or advisory feedback. Therefore, the *Ministerial Representative* and *Administrative Modernization Agency* lanes are initiated through A9 - an opinion request. Finally, the TSU consolidates all inputs and informs the JB. A negative outcome ends the process; a positive one allows contracting. JB sends a contract to the TSU through activity A10, which is forwarded to the AMA, closing the procedure.

Figure 1: The optimized, clean and anonymized BPMN of the ICT Approval Protocol at IGFEJ.



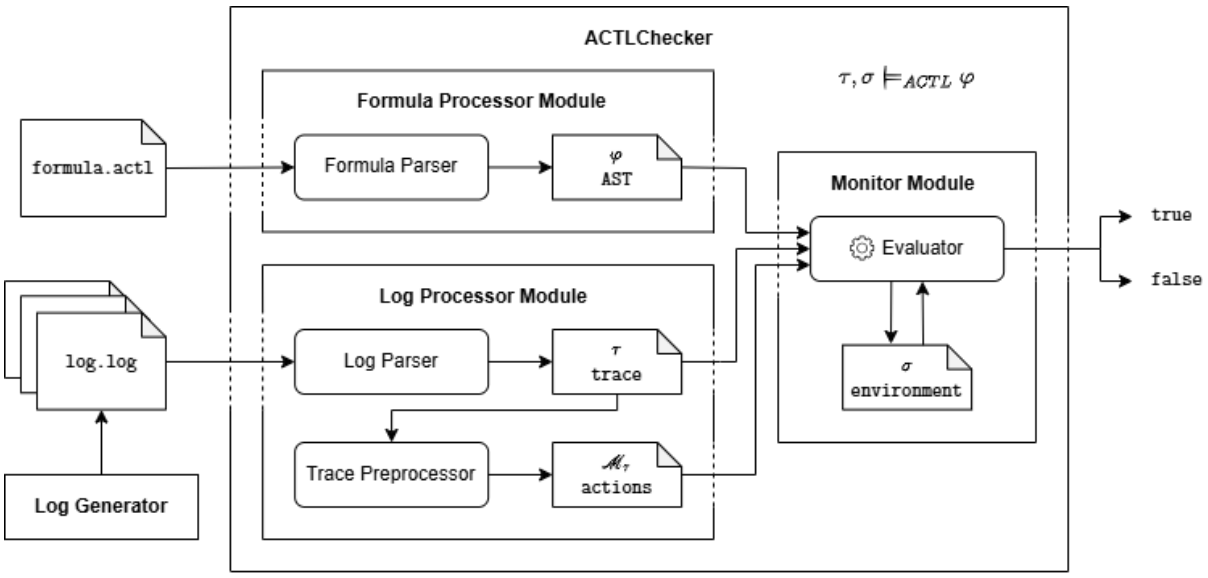
3 ACTLChecker

ACTLChecker is the tool we developed for the task of validation of protocol implementations using an offline testing pipeline. This section presents the design and structure of that checker, as well as the steps as how to use it properly. An overview of ACTLChecker is provided in Figure 2, which illustrates the general architecture of the system.

To apply the tool, users first specify formulas expressing the correctness properties of the protocol under analysis using the formal specification language introduced in Section 3.1. This language corresponds directly to the syntax of the temporal logic formalism underneath [Fer25], expressed in a machine-interpretable format to be parsed by the *Formula Parser* component inside the *Formula Processor* module, thus producing an *abstract syntax tree* φ, to be used for the evaluation of the property.

Subsequently, one must gather logs of the implementation of the protocol, whether through an instrumentation and execution of the protocol, as has been done previously for this same tool [Fer25], or as a log generation technique that closely follows the expected behavior of such protocol, if and when implemented, as done for this case study, and described in Section 3.2. A *Log Generator* is employed to generate a wide variety of execution scenarios, enabling the exploration of different system behaviors and the capture of representative traces. The *Log Processor* module then takes the log as input, parsing it using the *Log Parser* component to produce a trace τ. Using this trace, the *Trace Preprocessor* component

Figure 2: The general architecture of ACTLChecker.



generates an auxiliary mapping $\mathcal{M}_\tau$ which reduces complexity during trace evaluation [Fer25].

The tool, then, using the *Monitor* module, analyzes the execution traces against the formal specifications, determining whether the recorded behaviors satisfy the specified properties. It takes as input the *AST* φ derived from the property formula, the trace τ derived from the execution log, and the auxiliary mapping $\mathcal{M}_\tau$. Using its *Evaluator* component, the monitor checks whether τ satisfies φ, recursively traversing the nested formula structure. Its efficient trace validation algorithm is described in great detail [Fer25]. During evaluation, an environment σ is updated at each recursive call with the current variable bindings.

ACTLChecker is implemented in Python and is publicly accessible [1]. Each of the processor modules can be invoked individually, but running the monitor module is as straightforward as shown in Listing 1:

```
$ python checker.py -f formula.actl -l log.log [-d] [-n]
```

Listing 1: Command to run in order to use ACTLChecker to monitor a log using a formula.

where both the formula and the log can be specified as file paths or strings. The `-d` flag enables debugging, while the `-n` flag limits the number of log lines to be considered. And the outcome, with the debugging option disabled, should appear like Listing 2 in just a matter of seconds:

```
Evaluation: true / false
```

Listing 2: Example outputs of ACTLChecker evaluation: `true` and `false`.

ACTLChecker is particularly suited for testing executions of complex, data-intensive systems characterized by interdependent operations and multiple correctness properties, such as distributed protocols. Its modules are all independent of the implementation of the protocol under testing, requiring only that the formulas and the generated logs be in the formats described ahead, respectively in Section 3.1 and Section 3.2.

3.1 Specification Language of ACTL

The machine-readable specification language that was developed for this ACTLChecker tool follows the syntax of Action Temporal Logic (ACTL), a formalism that was created, alongside the tool, in the context of the master's thesis [Fer25].

The ACTL syntax was developed with care to support efficient automated monitoring and verification. For more rigorous and formal details regarding the syntax and semantics of this same logic fragment, especially the concept of actions, please refer to the source [Fer25].

While this type of formulas are concise and readable for humans, they are not directly practical for machine interpretation. To address this, we have designed a formal specification language that can be parsed by our monitoring tool using only alphanumeric characters readily available on standard keyboards. The language uses a prefix notation with nested parentheses, which preserves readability while enabling

[1] https://github.com/jjasferreira/actl-checker

machine traversal and evaluation. Once familiar with both representations, the translation between mathematical formulas and the specification language is straightforward.

As an example, consider below, the *A1-A7-A14 property*, a property that must hold for the ICT Approval Protocol, both in the mathematical format and in the specification language one in Listing 3.

(PROPERTY) **A1-A7-A14:** if an entity $ent1$ performs an A1 with output $false$ on the proposal identified by id, then, eventually, that same entity must request something in A7 from the entity $ent2$ it directly responds to hierarchically, in accordance to A14.

$$\forall\, \mathbf{a1}(ent1, id) \overset{i}{\rightsquigarrow} (false)\,.\, \exists\, \mathbf{a7}(ent1, id) \overset{j}{\rightsquigarrow} (ent2)\,.$$
$$\exists\, \mathbf{a14}(ent1, id) \overset{k}{\rightsquigarrow} (-, ent2)\,.\, Before(i, j) \wedge Before(j, k)$$

```
(forall a1 i (ent1 id) ('false)
    (exists a7 j (ent1 id) (ent2)
        (exists a14 k (ent1 id) (- ent2)
            (and
                (before i j)
                (before j k)
            )
        )
    )
)
```

Listing 3: *A1-A7-A14 property* in the ACTL specification language.

3.2 ICT Approval Log Generation

Before reasoning about traces and ACTL formulas, it is necessary to obtain execution logs by generating implementations of the ICT Approval protocol.

During the log generation process, we first refined the provided BPMN model and standardized the terminology, obtaining the version shown in Figure 1. Based on this model, we developed a log generator designed to be as realistic as possible. Each BPMN task is translated into a pair of *begin* and *end* events in the log, while each gateway is modeled as a probabilistic branch, with a 66% probability of following the *true* path and a 33% probability of following the *false* path.

The generator assigns random process-level identifiers to the involved entities and supports the generation of logs containing multiple concurrent process instances, controlled by a parameter specifying the number of parallel executions. The command used to run the log generator, which is available in the repository [2], is shown in Listing 4:

```
$ python generate_log.py --num-requests 20 --log log.log
```

Listing 4: Command to run in order to generate a log with 20 proposals, to be interpreted by ACTLChecker.

An excerpt of a log in the format read by the ACTLChecker tool is shown in Listing 5. The names of the events have been hidden in accordance to the anonymized diagram terminology, i.e. using the same naming convention (A1, A2, A3, etc.). Each log entry follows the following format:

`<timestamp>, <operation>, <operation ID>, <fields*>`

where:

- `<timestamp>` is the time the event occurred;
- `<operation>` is the type of operation (e.g., `A1`, `ReplyA12`). Reply messages are prefixed with `Reply` to distinguish them from initiating operations;
- `<operation ID>` is a unique identifier for the operation;
- `<fields*>` are fields relevant to the operation.

[2] https://github.com/AlessandroGianola/optigov-actl-check

```
2026-09-29 05:33:17, A8, 4-45cf2376-82cd, TO-1-2, ID-476
2026-09-29 08:39:20, ReplyA8, 4-45cf2376-82cd, TT-1-5
2026-09-29 18:54:22, ReplyA12, 5-a5fc8364-69a4, TO-1-7
2026-09-29 20:12:27, A12, 4-982fb0c6-caf5, TT-1-5, ID-476
2026-09-30 04:42:29, A1, 5-7b54525a-1804, TO-1-7, ID-842
2026-09-30 15:46:39, ReplyA9, 15-25b4c915-d21f, MR-5
2026-09-30 19:07:43, ReplyA12, 4-982fb0c6-caf5, TO-1-2
2026-09-30 20:18:56, A1, 4-c18aad11-9e0c, ID-476
2026-10-01 04:20:00, ReplyA9, 18-ae5df583-6b68, BM-1-3
2026-10-01 15:23:09, ReplyA1, 4-c18aad11-9e0c, True
2026-10-01 16:50:36, A2, 4-4acb2ac0-882e, TO-1-2, ID-476
2026-10-01 19:16:43, A14, 18-8cada550-9164, BM-1-3, ID-192
2026-10-02 00:37:50, ReplyA1, 5-7b54525a-1804, True
```

Listing 5: Extract of a log got from the ICT Approval Protocol log generator.

4 Evaluation

To ensure a comprehensive and thorough verification of the system's operations, we specified properties with a high, near-atomic level of granularity and in a cascaded manner, such that if they are all satisfied, as is the case, we can ensure completeness. By validating these individual properties, we can confirm that the ICT Approval protocol functions as an integrated whole, in full alignment with the BPMN diagram. This approach was chosen over the use of generalized properties which, while applicable across multiple entities, would sacrifice the unique functional nuances of each component. Table 1 presents the full set of specified properties, whose names have been, once again, redacted to maintain confidentiality.

Table 1: All the 95 properties covering every expected behavior of every action

List of Properties			
jb_jb_uniqueness	tsu_a7_previously	to_a4_eventually	bm_bm_uniqueness
jb_a3_exclusivity	tsu_a7_eventually	to_a8_exclusivity	bm_a14_exclusivity
jb_a3_uniqueness	tsu_a9_exclusivity	to_a8_previously	bm_a14_previously
jb_a3_eventually	tsu_a9_uniqueness	to_a8_eventually	bm_a14_eventually
jb_a5_uniqueness	tsu_a9_previously	to_a1_previously	mr_mr_uniqueness
jb_a5_previously	tsu_a9_eventually	to_a1_eventually	mr_a1_previously
jb_a5_eventually	tsu_a6_exclusivity	to_a7_exclusivity	mr_a1_eventually
jb_a9_exclusivity	tsu_a6_uniqueness	to_a7_previously	mr_a7_exclusivity
jb_a9_uniqueness	tsu_a6_previously	to_a7_eventually	mr_a7_previously
jb_a9_previously	tsu_a13_exclusivity	to_a2_exclusivity	mr_a7_eventually
jb_a9_eventually	tsu_a13_uniqueness	to_a2_previously	mr_a14_exclusivity
jb_a11_exclusivity	tsu_a13_previously	to_a2_eventually	mr_a14_uniqueness
jb_a11_previously	tsu_a13_eventually	to_a9_exclusivity	mr_a14_previously
jb_a11_eventually	tsu_a14_exclusivity	to_a9_previously	mr_a14_eventually
jb_a10_exclusivity	tsu_a14_uniqueness	to_a9_eventually	ama_ama_uniqueness
jb_a10_uniqueness	tsu_a14_previously	to_a14_exclusivity	ama_a4_exclusivity
jb_a10_previously	tsu_a14_eventually	to_a14_previously	ama_a4_uniqueness
jb_a10_eventually	tsu_a10_exclusivity	to_a14_eventually	ama_a4_previously
igfej_igfej_uniqueness	tsu_a10_uniqueness	tt_tt_exclusivity	ama_a4_eventually
tsu_tsu_exclusivity	tsu_a10_previously	tt_tt_uniqueness	ama_a14_exclusivity
tsu_tsu_uniqueness	to_to_exclusivity	tt_a12_exclusivity	ama_a14_uniqueness
tsu_a1_previously	to_to_uniqueness	tt_a12_previously	ama_a14_previously
tsu_a1_eventually	to_a4_exclusivity	tt_a12_eventually	ama_a14_eventually
tsu_a7_exclusivity	to_a4_previously	bm_bm_exclusivity	

These properties are available in the repository [3] and have been formally verified against the protocol logs using the ACTLChecker tool. The specifications for a selection of these properties are detailed below:

(JUDICIAL BODY) **jb_a10_exclusivity:** if a judicial body *jb* sends something in A10, it must be to a technical support unit *tsu* and if a technical support unit receives something in A10, it must come from a judicial

[3] https://github.com/AlessandroGianola/optigov-actl-check

body.

$$\forall\, \mathbf{a10}(jb,-,-) \overset{i}{\rightsquigarrow} (tsu) \begin{cases} \exists\, \mathbf{jb}(jb)\ ^{j}\ .\ During(i,j) \Rightarrow \exists\, \mathbf{tsu}(tsu,-)\ ^{k}\ .\ During(i,k) \\ \exists\, \mathbf{tsu}(tsu,-)\ ^{j}\ .\ During(i,j) \Rightarrow \exists\, \mathbf{jb}(jb)\ ^{k}\ .\ During(i,k) \end{cases}$$

(IGFEJ) **igfej_uniqueness:** there can exist at most one IGFEJ; any two observed instances must refer to the same entity.

$$\forall\, \mathbf{igfej}(igfej_1)\ ^{i}\ .\ \forall\, \mathbf{igfej}(igfej_2)\ ^{j}\ .\ (i = j) \wedge (igfej_1 = igfej_2)$$

(TECHNICAL SUPPORT UNIT) **tsu_a9_previously:** if a technical support unit *tsu* requests something in A9 for proposal *id* involving entity *ent*, then a valid preceding action must have occurred: an A1 if requesting TO, or an A14 for MR and AMA.

$$\forall\, \mathbf{a9}(tsu,id) \overset{i}{\rightsquigarrow} (e) \begin{cases} \exists\, \mathbf{to}(e,-)\ ^{j}\ .\ During(i,j) \Rightarrow \exists\, \mathbf{a1}(tsu,id) \overset{k}{\rightsquigarrow} (true)\ .\ Before(k,i) \\ \exists\, \mathbf{mr}(e)\ ^{j}\ .\ During(i,j) \Rightarrow \exists\, \mathbf{a14}(to,id) \overset{k}{\rightsquigarrow} (true,tsu)\ . \\ \qquad \exists\, \mathbf{to}(to,-)\ ^{l}\ .\ During(k,l) \wedge Before(k,i) \\ \exists\, \mathbf{ama}(e)\ ^{j}\ .\ During(i,j) \Rightarrow \exists\, \mathbf{a14}(mr,id) \overset{k}{\rightsquigarrow} (true,tsu)\ . \\ \qquad \exists\, \mathbf{mr}(mr)\ ^{l}\ .\ During(k,l) \wedge Before(k,i) \end{cases}$$

(TECHNICAL OFFICER) **to_a1_eventually:** if a technical officer *to* performs an A1 with $false$ output for proposal *id*, then it requests something in A7 from a technical support unit, otherwise it performs A3.

$$\forall\, \mathbf{a1}(to,id) \overset{i}{\rightsquigarrow} (output) \begin{cases} \exists\, \mathbf{to}(to,-)\ ^{j}\ .\ During(i,j) \Rightarrow \\ \quad \begin{cases} (output = false) \Rightarrow \exists\, \mathbf{a7}(to,id) \overset{k}{\rightsquigarrow} (-)\ .\ Before(i,k) \\ (output = true) \Rightarrow \exists\, \mathbf{a3}(to,id) \overset{k}{\rightsquigarrow} (-)\ .\ Before(i,k) \end{cases} \end{cases}$$

(TECHNICAL TEAM) **tt_tt_exclusivity:** a technical team *tt* must belong to the IGFEJ.

$$\forall\, \mathbf{tt}(tt,igfej)\ ^{i}\ .\ \exists\, \mathbf{igfej}(igfej)\ ^{j}\ .\ During(i,j)$$

(BOARD MEMBER) **bm_bm_uniqueness:** all board members *bm* must have unique identifiers.

$$\forall\, \mathbf{bm}(bm_1,-)\ ^{i}\ .\ \forall\, \mathbf{bm}(bm_2,-)\ ^{j}\ .\ (i \neq j) \Rightarrow (bm_1 \neq bm_2)$$

(MINISTERIAL REPRESENTATIVE) **mr_a7_previously:** if a ministerial representative *mr* requests something in A7 for proposal *id*, then a preceding A1 must have had $false$ as output.

$$\forall\, \mathbf{a7}(mr,id) \overset{i}{\rightsquigarrow} (-)\ .\ \exists\, \mathbf{mr}(mr)\ ^{j}\ .\ During(i,j) \Rightarrow \exists\, \mathbf{a1}(mr,id) \overset{k}{\rightsquigarrow} (false)\ .\ Before(k,i)$$

(ADMINISTRATIVE MODERNIZATION AGENCY) **ama_a14_eventually:** if the Administrative Modernization Agency sends something in A14 for proposal *id*, then the technical support unit *tsu* must eventually follow by sending that same output.

$$\forall\, \mathbf{a14}(ama,id) \overset{i}{\rightsquigarrow} (output,tsu) \begin{cases} \exists\, \mathbf{ama}(ama)\ ^{j}\ .\ During(i,j) \Rightarrow \\ \qquad \exists\, \mathbf{a14}(tsu,id) \overset{k}{\rightsquigarrow} (output,-)\ .\ Before(i,k) \end{cases}$$

Table 2 displays the average execution times for verifying a selection of sample properties across 10 logs of increasing scale. The log sizes scale logarithmically relative to the number of concurrent ICT Approval proposals, allowing us to evaluate the tool's efficiency and performance under high-stress conditions with exceptionally large log files. The approximate number of log entries corresponding to the concurrent request totals are as follows: 100 lines for 1 proposal; 5.7×10^2 for 10 concurrent proposals; 5.6×10^3 for 100; 5.5×10^4 for 1,000; 5.5×10^5 for 10,000; and 5.4×10^6 for 100,000. In instances where a hyphen (—) appears in the table, the execution time exceeded the predefined threshold of 1,000 seconds. These

benchmarks were conducted on a system featuring an Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz and 12 GB of RAM.

Table 2: Performance metrics for a set of representative properties

	Time (s) / # concurrent executions					
Property	10^0	10^1	10^2	10^3	10^4	10^5
jb_a5_eventually	.00933	.01618	.11540	1.4852	45.673	-
jb_a10_exclusivity	.01867	.02681	.12438	1.3051	14.624	-
igfej_igfej_uniqueness	.01152	.02181	.12241	1.3729	15.337	-
tsu_a9_exclusivity	.03227	.04197	.14971	1.4209	15.869	-
tsu_a9_previously	.03451	.04394	.46651	34.225	-	-
tsu_a6_uniqueness	.00891	.01756	.11268	1.3236	14.431	-
to_a1_previously	.01883	.03092	.88511	103.02	-	-
to_a1_eventually	.01977	.02581	.27349	21.012	-	-
tt_tt_exclusivity	.00846	.01797	.10536	1.1839	14.716	-
tt_a12_previously	.00881	.01903	.12103	1.3424	32.801	-
bm_bm_uniqueness	.01136	.02169	.11115	1.1939	14.660	-
bm_a14_eventually	.01977	.03035	.31293	20.919	-	-
mr_a7_previously	.01201	.02377	.15235	3.9538	208.94	-
mr_a14_uniqueness	.01215	.02271	.18109	1.2248	15.863	-
ama_a4_exclusivity	.00881	.01762	.11488	1.2099	14.364	-
ama_a14_eventually	.01254	.02337	.18420	6.8567	708.15	-

ACTLChecker proves to be both versatile and highly efficient for case studies extending beyond distributed protocols. Any system that can be formalized according to our notation is verifiable, offering significant advantages for complex concurrent systems, intricate temporal properties, and workflows involving time-dependent data and actions. The ICT Approval protocol serves as a primary example of such a system.

References

[Fer25] José João Ferreira. Specifying and Testing Distributed Protocols with Action Temporal Logic. Master's thesis, University of Lisbon - Instituto Superior Técnico, November 2025.