Formal Methods in Computer-Aided Design 2025

R2U2 Playground: Visualization of a Real-time, Temporal Logic Runtime Monitor

Alexis Aurandt
Iowa State University
Ames, IA, USA
aurandt@iastate.edu

Kristin Yvonne Rozier
Iowa State University
Ames, IA, USA
kyrozier@iastate.edu

Phillip H. Jones
Iowa State University
Ames, IA, USA
phjones@iastate.edu

Artifact Reviewed
10.5281/zenodo.16787010

***Abstract*—The Realizable, Responsive, Unobtrusive Unit (R2U2) is a real-time, temporal logic-based runtime monitoring engine that has been successfully deployed on-board a wide range of cyber-physical systems, from aircraft to spacecraft to robots, checking in real time whether these systems uphold specified system requirements. However, the efficacy of deploying runtime monitors is highly sensitive to their correct configuration. Moreover, there are many barriers to adopting runtime monitors, including a high learning curve and the challenge of eliciting formal specifications that accurately capture system requirements. Therefore, we present the R2U2 Playground, an interactive web-based playground that provides visualization of R2U2. The R2U2 Playground provides stepwise execution coupled with reactive timeline plotting and visualization of its internal abstract syntax tree architecture. To this extent, the R2U2 Playground provides insight into how R2U2 evaluates specifications, allowing for easier specification understanding and debugging.**

I. Introduction

The Realizable, Responsive, Unobtrusive Unit (R2U2) runtime monitoring engine is known for its real-time guarantees and resource-aware architecture [1]–[5]. Hence, R2U2 has been successfully deployed on-board various cyber-physical systems such as NASA's Robonaut2 [1], NASA's Lunar Gateway Vehicle System Manager [6]–[9], NASA's Swift UAS [2], [10], [11], NASA's DragonEye UAS [12], Iowa State's Nova Somnium sounding rocket [13], Iowa State's CySat-I CubeSat [14], and more [15]–[19]. R2U2 successfully analyzed whether these systems upheld formal system requirements during deployment. However, R2U2's efficacy is highly sensitive to its correct configuration and integration into the target system.

Furthermore, there are many barriers to adopting runtime monitors, such as the learning barrier, skepticism, lack of experience, etc. [20]–[22]. Additionally, once runtime monitors are adopted, there is the challenge of eliciting specifications that accurately capture system requirements [23]–[25]. To decrease the learning curve of the R2U2 runtime monitoring engine, we present the R2U2 Playground.[1][2] The R2U2 Playground is an interactive web-based playground that provides a visualization of the R2U2 runtime monitoring engine. Within the playground, an instantiation of R2U2 is executed given user-defined specifications and an input trace. The R2U2 Playground eases the elicitation and testing of these user-defined specifications by providing step-forward and step-backward debugging capabilities and timeline visualization of output verdicts produced by R2U2. It also provides a visualization for how R2U2 reasons about a (set of) specification(s) as an abstract syntax tree, and we enable configuring of various optimizations of the abstract syntax tree (e.g., common subexpression elimination [1], [5] and rewrite rules [26]) directly in the playground. The playground interface also includes easy import and export of the various input/output files, visualization images, and the compiled specification binary. Overall, the R2U2 Playground allows users to easily interact with R2U2, learn how R2U2 reasons about specification(s), and, as a result, also validate the correctness of their specification(s) and R2U2's configuration.

II. Current Visualizations for Runtime Monitors

Within the runtime verification community, system requirements are specified/formalized utilizing two main approaches: (1) stream-based specification languages and (2) temporal logics. The utilization of a stream-based specification language to specify system requirements to generate runtime monitors is seen in frameworks such as RTLola [27], [28], Striver [29], TeSSLa [30], and CoPilot [31], [32]. These stream-based specification languages are arguably more expressive than temporal logics, but they have a high learning curve as they are each tool-specific languages. To combat the learning curve, both RTLola [33] and TeSSLa [30] have interactive playgrounds for exploring their respective tools and to ease the understanding of specifications. On the other hand, other tools look at monitoring temporal logics directly, e.g., R2U2 [5], [34], Aerial [35], Reelay [36], MonPoly [37], [38], VeriMon [39], VeriMon+ [40], Hydra [41], and Vydra [42], but these monitors currently lack any explainability or visualization interface/tool. Recently, the Explanator2 [43] and WhyMon [44] monitors were developed to bridge this gap in explainability through outputting proof trees and providing optional GUI visualization of these proof trees for Metric Temporal Logic (MTL) and Metric First-Order Temporal Logic (MFOTL) specifications, respectively. Of these temporal logic monitors, R2U2 is the only one that can run on-board and provide

Supported by NSF:CPS Award 2038903.
[1] https://r2u2.temporallogic.org/playground/
[2] https://zenodo.org/records/16787011

https://doi.org/10.34727/2025/isbn.978-3-85448-084-6_18

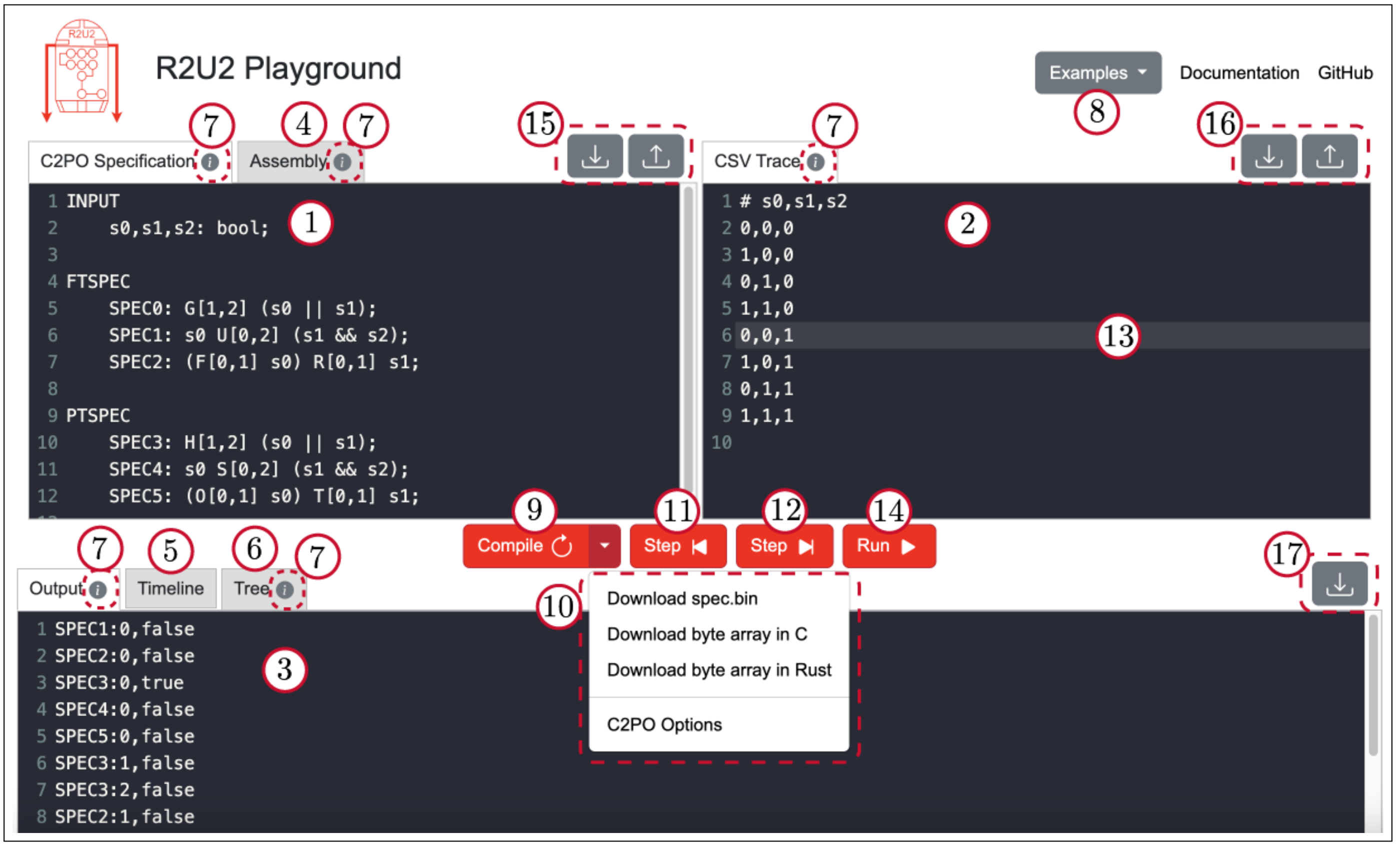


Fig. 1: R2U2 Playground: **(1)** C2PO specification; **(2)** CSV input trace; **(3)** R2U2 text output; **(4)** C2PO assembly output tab; **(5)** R2U2 timeline visualization tab; **(6)** R2U2 internal tree architecture visualization tab; **(7)** Info pop-up buttons; **(8)** Examples dropdown; **(9)** Compile C2PO specification button; **(10)** Compile dropdown menu; **(11)** Step-backward button; **(12)** Step-forward button; **(13)** Trace row highlighting based on current step; **(14)** Run button; **(15)** Download/Upload C2PO specification buttons; **(16)** Download/Upload CSV trace buttons; **(17)** Download output text, timeline image, or tree image button.

verdicts in real-time for both future- and past-time properties due to its unique architecture and specification language [6]; therefore, we focus on providing a visualization for R2U2, while also taking inspiration from the TeSSLa and RTLola playgrounds.

III. Playground Functionality

A. *Mission-time Linear Temporal Logic (MLTL) and past-time MLTL (ptMLTL) [2], [34], [45], [46]*

R2U2 reasons over specifications in MLTL/ptMLTL, a variant of LTL/ptLTL over finite traces with bounded, closed, and discrete intervals on temporal operators. MLTL and ptMLTL contain the standard boolean connectives (e.g., NOT, AND, OR, XOR, IMPLIES, EQUIVALENT) along with temporal operators that reason over future-time and past-time, respectively. Tables I and II provide a visualization of example formulas containing these temporal operators.

B. *Configuration Compiler for Property Organization (C2PO) [5] Specification and R2U2 Configuration Output*

C2PO provides a specification language to make writing MLTL/ptMLTL specifications more natural and transparent. Notably, C2PO supports expressions over complex data types such as sets and structs. The language is also structured to include separate sections for defining structs, inputs, variables, macros, and MLTL/ptMLTL formulas. Within the R2U2 Playground, users can directly compose their C2PO specification (label 1 in Fig. 1), as well as upload/download C2PO specification files directly (label 15 in Fig. 1).

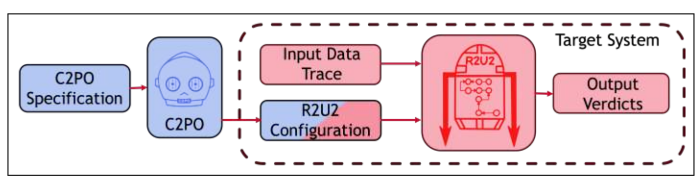


Fig. 2: C2PO and R2U2 Workflow: Blue shaded boxes indicate the process required to configure R2U2, and the red shaded boxes indicate the pieces required to execute R2U2.

C2PO encodes a C2PO specification into an R2U2 configuration by decomposing the specified expression(s) into subexpression(s) represented as an Abstract Syntax Tree (AST) and traverses the AST to produce assembly-style instructions for R2U2 to execute at runtime (Fig. 2). (Refer to Fig. 3 and 4 for

TABLE I: Pictorial Representation of Mission-time Linear Temporal Logic (MLTL). Timeline depicts a satisfying trace where $i = 0$.

Formula	Operator	Timeline
$\mathcal{G}_{[2,5]}\psi$	GLOBALLY$_{[2,5]}$	0 1 2(ψ) 3(ψ) 4(ψ) 5(ψ) 6
$\mathcal{F}_{[0,6]}\psi$	FUTURE$_{[0,6]}$	0 1 2 3 4 5 6(ψ)
$\psi\ \mathcal{U}_{[1,4]}\ \xi$	UNTIL$_{[1,4]}$	0 1(ψ) 2(ψ) 3(ξ) 4 5 6
$\psi\ \mathcal{R}_{[3,6]}\ \xi$	RELEASE$_{[3,6]}$	0 1 2 3(ξ) 4(ξ) 5(ξ) 6(ψ,ξ)

TABLE II: Pictorial Representation of past-time Mission-time Linear Temporal Logic (ptMLTL). Timeline depicts a satisfying trace where $i = 6$.

Formula	Operator	Timeline
$\mathcal{H}_{[2,5]}\psi$	HISTORICALLY$_{[2,5]}$	0 1(ψ) 2(ψ) 3(ψ) 4(ψ) 5 6
$\mathcal{O}_{[0,6]}\psi$	ONCE$_{[0,6]}$	0(ψ) 1 2 3 4 5 6
$\psi\ \mathcal{S}_{[1,4]}\ \xi$	SINCE$_{[1,4]}$	0 1 2 3(ξ) 4(ψ) 5(ψ) 6
$\psi\ \mathcal{T}_{[3,6]}\ \xi$	TRIGGER$_{[3,6]}$	0(ψ,ξ) 1(ξ) 2(ξ) 3(ξ) 4 5 6

the AST and assembly-style instructions required to encode SPEC0 from the C2PO specification in Fig. 1.) The R2U2 configuration (which is composed of these assembly-style instructions) is represented in binary format. The command line interfaces for both R2U2's C and Rust realizations expect the R2U2 configuration as a binary file, while if running R2U2 on-board a system, it may be more applicable to have the binary stored as an array; therefore, a user can easily download the R2U2 configuration as a binary file (i.e., spec.bin) or as a C/Rust file containing the configuration encoded as a byte array (label 10 in Fig. 1).

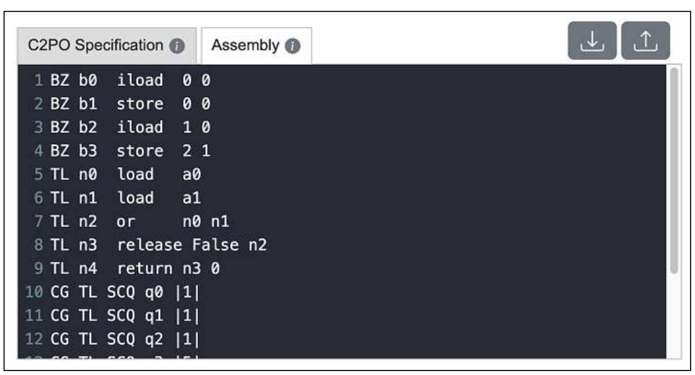


Fig. 3: Textual Representation of Assembly Output for SPEC0: G[1,2] (s0 || s1)

Once the C2PO file has been compiled in the playground by either pressing the Compile button (label 9 in Fig. 1) or by executing R2U2 (labels 11, 12, and 14 in Fig. 1), the textual representation of these instructions is displayed in the

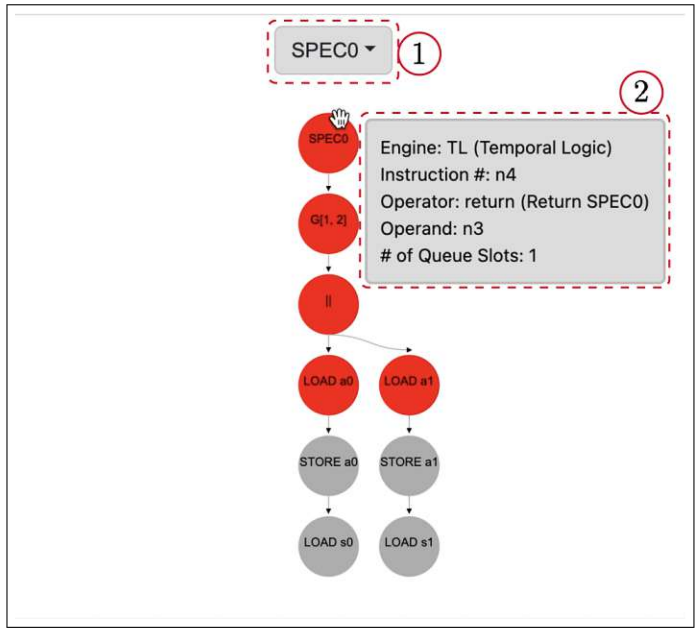


Fig. 4: Tree Visualization of SPEC0: G[1,2] (s0 || s1): **(1)** Specification selection dropdown menu; **(2)** More information dialog box displayed upon mouse over of node.

Assembly tab (label 4 in Fig. 1). Fig. 3 displays the Assembly tab if only SPEC0 was defined in the C2PO specification given in Fig. 1. This textual representation may be difficult to parse; therefore, a visual representation of the AST is available in the Tree tab (label 6 in Fig. 1). Fig. 4 displays the Tree tab with SPEC0 selected, where SPEC0 was defined in the C2PO specification given in Fig. 1. Within the Tree tab, a dropdown menu (label 1 in Fig. 4) allows a user to select any specification that has been defined, or they can select to view all specifications at once (as shown in Fig. 6). The tree is also color-coded such that red nodes represent MLTL/ptMLTL operator nodes, while grey nodes represent Booleanizer [5] nodes. (The Booleanizer is an optional module that computes atomic propositions from non-Boolean expressions utilizing arithmetic, bitwise, set aggregation, and relational operators.) A user can also zoom in and out of the AST view, move the entire tree around, move individual nodes around, and mouse over nodes to find out more information (label 2 in Fig. 4). Optionally, a user may also download the image to save locally (label 17 in Fig. 1).

C2PO also allows users to enable or disable certain features (i.e., the Booleanizer, auxiliary data, various rewriting rules [26], Common Subexpression Elimination (CSE) [1], [5], and SAT checking) to customize the AST for a given target system. A user can directly enable/disable these features by clicking on the C2PO Options button from the Compile dropdown (label 10 in Fig. 1) to bring up the C2PO options pop-up as shown in Fig. 5. Fig. 6 illustrates how the AST may change based on the features enabled. Fig. 6a displays the AST for all specifications defined in the C2PO specification from Fig. 1 based on the default C2PO options displayed in Fig. 5, and Fig. 6b displays

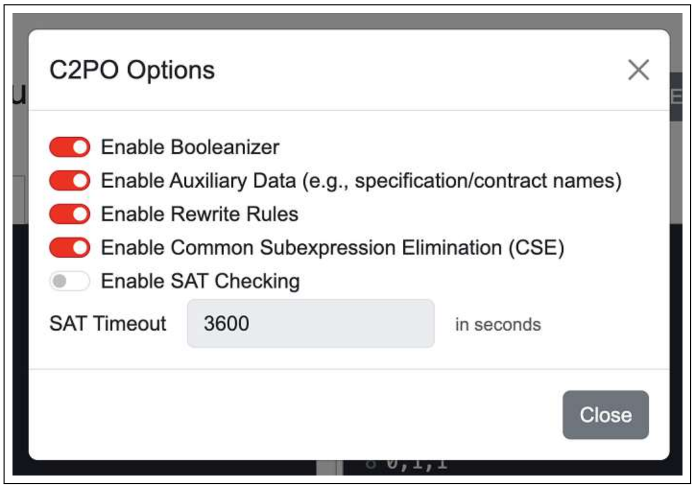
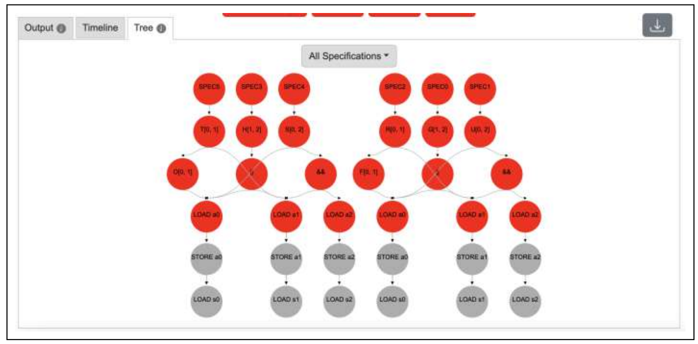


Fig. 5: C2PO Options Pop-Up Window: **(a)** Enable Booleanizer [5] module; **(b)** Enable Auxiliary Data such as specification names (e.g., "SPEC0") to be stored and provided with output verdicts; **(c)** Enable Rewrite Rules given in [26]; **(d)** Enable CSE allows to share common nodes between subexpressions [1], [5]; **(e)** Enable SAT Checking of the MLTL/ptMLTL specifications powered by Z3 [47].

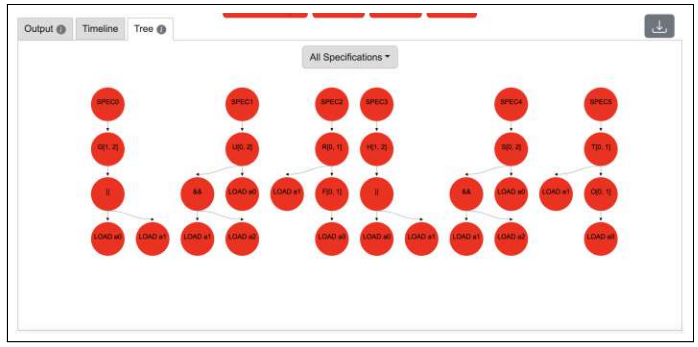
the resulting AST when the Booleanizer and CSE are disabled. The specifications given in the C2PO specification in Fig. 1 only reason over Boolean inputs; therefore, the Booleanizer is not required for this C2PO specification and can be disabled to save resources as shown by the elimination of the grey nodes in Fig. 6b. On the other hand, if we disable CSE, we no longer share common nodes between subexpressions; therefore, we also observe a larger amount of red nodes in Fig. 6b.

C. R2U2 Execution

There are three options for executing the R2U2 runtime monitoring engine: step-forward (label 11 in Fig. 1), step-backward (label 12 in Fig. 1), and run (label 14 in Fig. 1). To define the inputs into R2U2, a CSV formatted input signal trace can be composed directly within the playground (label 2 in Fig. 1), as well as uploaded/downloaded (label 16 in Fig. 1). Within the CSV trace, each line typically represents the input signals at a discrete timestamp, but to support longer traces as requested by users at NASA (without having to manually specify the input of each time-step), we added the ability to encode multiple time-steps in a single line where the symbol $@T$ at the beginning of a line encodes the input signal as occurring at time T and since the last time-step included in the input trace. This format directly works with R2U2's aggregated storing of results (refer to [1] and [34] for more details on R2U2's aggregated storage). The $@T$ symbol is optional, and if not included, the line simply represents the next timestamp of execution. Fig. 7 provides an example of employing the $@T$ symbol. Utilizing the step-forward and step-backward buttons, the CSV trace is iterated through step-by-step, and the current step in the CSV trace is highlighted



(a) Booleanizer and Common Subexpression Elimination (CSE) are enabled



(b) Booleanizer and CSE are disabled

Fig. 6: Tree Visualization of All Specifications (i.e., SPEC0, SPEC1, SPEC2, SPEC3, SPEC4, and SPEC5 from Fig. 1)

```
# p,q,i,j,k
0,1,5,2,3
0,1,5,2,3
0,1,5,2,3
0,0,4,2,3
0,0,4,2,3
1,1,2,6,10
```

(a) Typical trace where each line represents a sequential discrete timestamp starting at timestamp 0

```
# p,q,i,j,k
@2 0,1,5,2,3
@4 0,0,4,2,3
@5 1,1,2,6,10
```

(b) Trace encoding multiple time-steps utilizing the $@T$ symbol

Fig. 7: Different Equivalent CSV Input Trace Representations: Note that "$\# \ p, q, i, j, k$" defines the variable-to-column mapping scheme for the trace.

(label 13 in Fig. 1). Additionally, the entire CSV trace can be evaluated at once using the run button.

The textual representation of R2U2's output verdicts is given in the Output tab (label 3 in Fig. 1), but a timeline visualization is provided in the Timeline tab (label 5 in Fig. 1). Fig. 8 displays the Timeline tab with SPEC0 selected, but a dropdown menu (label 1 in Fig. 8) allows a user to visualize

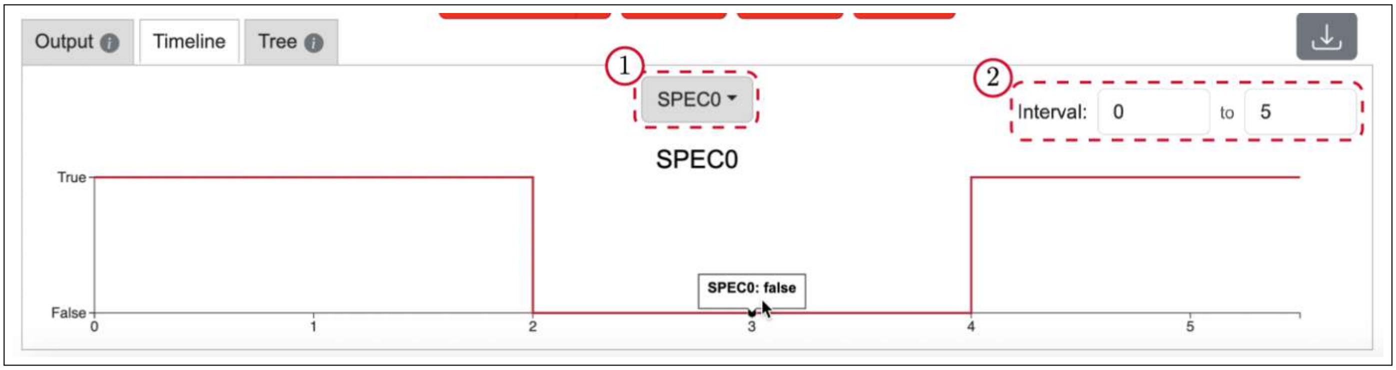


Fig. 8: Timeline Visualization of SPEC0: G[1,2] (s0 || s1): **(1)** Specification selection dropdown menu; **(2)** Interval zooming.

any specification that has been defined. When performing stepwise execution, both the textual representation and the timeline visualization will react accordingly, allowing the user to visualize how R2U2 produces output verdicts over time. Optionally, the user can also zoom in to a certain interval of interest (label 2 in Fig. 8) and download the timeline image to save locally (label 17 in Fig. 1).

IV. Implementation

The R2U2 Playground is comprised of two main components: the frontend and the backend. The frontend consists of the visual user interface as shown in Fig. 1. The frontend is strictly written in HTML, JavaScript, and CSS with the aid of Bootstrap [48], CodeMirror [49], and D3 [50]. The frontend purely provides the visualization elements of the R2U2 Playground, while the execution of the R2U2 runtime monitoring engine occurs on the backend. The backend is a Rust-based server built with the warp framework [51] that directly executes the Rust realization of R2U2 [34]. (Note that we could have easily utilized the R2U2's C realization in the backend but chose the Rust realization due to its memory-safe qualities and ease-of-use.) The frontend sends applicable requests with the C2PO specification and CSV input trace to the backend, and the backend will then respond with the applicable response (e.g., R2U2 configuration and verdicts). The frontend then performs static analysis on the response to provide its various visualizations.

To promote extensibility, the R2U2 Playground is also available open-source under the Creative Commons Attribution 4.0 International license.[3]

V. Discussion

The R2U2 Playground allows users to easily run C2PO specifications against various CSV input traces without requiring local installation or interacting with command line interfaces. The playground is also easy to use with several optional examples available through the Example dropdown (label 8 in Fig. 1) and optional more-information pop-up windows (label 7 in Fig. 1). Consequently, the R2U2 Playground lowers the barrier to entry for utilizing the R2U2 runtime monitoring engine, allowing for easier adoption of R2U2. To this extent, we have witnessed the easier adoption of R2U2 through sharing the R2U2 Playground with various interested industrial entities (e.g., Collins Aerospace, Kansas State University, and NASA).

Through the playground, users can more easily debug their specifications through stepwise debugging capabilities and reactive timeline visualization. Additionally, users can enable or disable C2PO features and view how this changes the internal representation (i.e., the AST) of their specifications in R2U2. Previously, the Resource Estimation GUI from [5] provided limited capabilities to perform similar analysis, but the information was not intuitively displayed. Additionally, this previous work only calculated resource usage for running R2U2 and did not provide an interface to execute R2U2 or debug specifications. In the future, we plan to incorporate the resource estimation analysis of this previous work into the R2U2 Playground to visualize the resource requirements of the AST. Furthermore, if a user's end goal is to run R2U2 on-board a system outside the R2U2 Playground, they can easily export their compiled specifications to a range of popular formats R2U2 might expect (e.g., a binary file or a byte array in C or Rust). Overall, the R2U2 Playground can be utilized to both learn how R2U2 reasons about specifications and debug/test these specifications to make sure they are accurately capturing the system requirements, while also ensuring the correct configuration of R2U2.

Given the prevalence of MLTL/ptMLTL as a specification logic and the popularity of runtime monitoring as a light-weight formal verification technology, the impact of the R2U2 Playground also extends beyond R2U2, providing a tool for learning MLTL/ptMLTL and runtime monitoring. We utilized the R2U2 Playground as an educational tool for teaching these aspects of formal methods in the 2025 Summer School on Formal Techniques [52], and in the Applied Formal Methods course at Iowa State University [53]. Future expansions of the R2U2 Playground will include additional educational tooling in response to questions and feedback from students.

References

[1] B. Kempa, P. Zhang, P. H. Jones, J. Zambreno, and K. Y. Rozier, "Embedding Online Runtime Verification for Fault Disambiguation on

[3]https://zenodo.org/records/16787011

Robonaut2," in *Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, ser. Lecture Notes in Computer Science (LNCS). Vienna, Austria: Springer, September 2020, pp. 196–214.

[2] T. Reinbacher, K. Y. Rozier, and J. Schumann, "Temporal-logic based runtime observer pairs for system health management of real-time systems," in *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, ser. Lecture Notes in Computer Science (LNCS), vol. 8413. Springer-Verlag, April 2014, pp. 357–372.

[3] J. Schumann, P. Moosbrugger, and K. Y. Rozier, "Runtime Analysis with R2U2: A Tool Exhibition Report," in *Proceedings of the 16th International Conference on Runtime Verification (RV16)*. Madrid, Spain: Springer-Verlag, September 2016.

[4] K. Y. Rozier and J. Schumann, "R2U2: Tool Overview," in *Proceedings of International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CUBES)*, vol. 3. Seattle, WA, USA: Kalpa Publications, September 2017, pp. 138–156. [Online]. Available: https://easychair.org/publications/paper/Vncw

[5] C. Johannsen, P. Jones, B. Kempa, K. Y. Rozier, and P. Zhang, "R2U2 Version 3.0: Re-Imagining a Toolchain for Specification, Resource Estimation, and Optimized Observer Generation for Runtime Verification in Hardware and Software," in *International Conference on Computer Aided Verification*. Springer, 2023, pp. 483–497.

[6] J. B. Dabney, J. M. Badger, and P. Rajagopal, "Adding a verification view for an autonomous real-time system architecture," in *Proceedings of SciTech Forum*, ser. 2021-0566. AIAA, January 2021, p. Online.

[7] J. B. Dabney, "Using assume-guarantee contracts in autonomous spacecraft," Flight Software Workshop (FSW) Online: https://www.youtube.com/watch?v=zrtyiyNf674, February 2021.

[8] J. B. Dabney, P. Rajagopal, and J. M. Badger, "Using assume-guarantee contracts for developmental verification of autonomous spacecraft," Flight Software Workshop (FSW) Online: https://www.youtube.com/watch?v=HFnn6TzblPg, February 2022.

[9] J. B. Dabney, J. M. Badger, and P. Rajagopal, "Trustworthy autonomy for gateway vehicle system manager," in *2023 IEEE Space Computing Conference (SCC)*. IEEE, 2023, pp. 57–62.

[10] J. Geist, K. Y. Rozier, and J. Schumann, "Runtime observer pairs and bayesian network reasoners on-board FPGAs: flight-certifiable system health management for embedded systems," in *International Conference on Runtime Verification*. Springer, 2014, pp. 215–230.

[11] J. Schumann, K. Y. Rozier, T. Reinbacher, O. J. Mengshoel, T. Mbaya, and C. Ippolito, "Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems," in *International Journal of Prognostics and Health Management*, 2015.

[12] J. Schumann, P. Moosbrugger, and K. Y. Rozier, "R2U2: monitoring and diagnosis of security threats for unmanned aerial systems," in *Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22-25, 2015. Proceedings*. Springer, 2015, pp. 233–249.

[13] B. Hertz, Z. Luppen, and K. Y. Rozier, "Integrating runtime verification into a sounding rocket control system," in *NASA Formal Methods Symposium*. Springer, 2021, pp. 151–159.

[14] A. Aurandt, P. H. Jones, and K. Y. Rozier, "Runtime verification triggers real-time, autonomous fault recovery on the CySat-I," in *NASA Formal Methods Symposium*. Springer, 2022, pp. 816–825.

[15] M. Cauwels, A. Hammer, B. Hertz, P. Jones, and K. Y. Rozier, "Integrating Runtime Verification into an Automated UAS Traffic Management System," in *DETECT*. L'Aquila, Italy: Springer, September 2020.

[16] C. Johannsen, M. Anderson, W. Burken, E. Diersen, J. Edgren, C. Glick, S. Jou, A. Kumar, J. Levandowski, E. Moyer *et al.*, "Openuas version 1.0," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2021, pp. 1449–1458.

[17] Z. A. Luppen, D. Y. Lee, and K. Y. Rozier, "A case study in formal specification and runtime verification of a cubesat communications system," in *AIAA Scitech 2021 forum*, 2021.

[18] Z. Luppen, M. Jacks, N. Baughman, M. Stilic, R. Nasers, B. Hertz, J. Cutler, D.-Y. Lee, and K. Y. Rozier, "Elucidation and analysis of specification patterns in aerospace system telemetry," in *NASA Formal Methods Symposium*. Springer, 2022, pp. 527–537.

[19] J. Schumann, I. Roychoudhury, and C. Kulkarni, "Diagnostic reasoning using prognostic information for unmanned aerial systems," in *Annual Conference of the PHM Society*, vol. 7, no. 1, 2015.

[20] J. P. Bowen and M. G. Hinchey, "Ten commandments of formal methods," *Computer*, vol. 28, no. 4, pp. 56–63, 1995.

[21] ——, "Ten commandments of formal methods... ten years later," *Computer*, vol. 39, no. 1, pp. 40–48, 2006.

[22] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: Practice and experience," *ACM computing surveys (CSUR)*, vol. 41, no. 4, pp. 1–36, 2009.

[23] A. Goodloe, "Challenges in high-assurance runtime verification," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2016, pp. 446–460.

[24] K. Y. Rozier, "Specification: The biggest bottleneck in formal methods and autonomy," in *Verified Software. Theories, Tools, and Experiments: 8th International Conference, VSTTE 2016, Toronto, ON, Canada, July 17–18, 2016, Revised Selected Papers 8*. Springer, 2016, pp. 8–26.

[25] L. Teixeira, B. Miranda, H. Rebêlo, and M. d'Amorim, "Demystifying the challenges of formally specifying API properties for runtime verification," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2021, pp. 82–93.

[26] C. Johannsen, B. Kempa, P. H. Jones, K. Y. Rozier, and T. Wongpiromsarn, "Impossible made possible: Encoding intractable specifications via implied domain constraints," in *International Conference on Formal Methods for Industrial Critical Systems*. Springer, 2023, pp. 151–169.

[27] J. Baumeister, B. Finkbeiner, F. Kohn, and F. Scheerer, "A tutorial on stream-based monitoring," in *International Symposium on Formal Methods*. Springer, 2024, pp. 624–648.

[28] P. Faymonville, B. Finkbeiner, M. Schledjewski, M. Schwenger, M. Stenger, L. Tentrup, and H. Torfah, "StreamLAB: stream-based monitoring of cyber-physical systems," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 421–431.

[29] F. Gorostiaga and C. Sánchez, "Striver: Stream runtime verification for real-time event-streams," in *International Conference on Runtime Verification*. Springer, 2018, pp. 282–298.

[30] H. Kallwies, M. Leucker, M. Schmitz, A. Schulz, D. Thoma, and A. Weiss, "TeSSLa–an ecosystem for runtime verification," in *International Conference on Runtime Verification*. Springer, 2022, pp. 314–324.

[31] I. Perez, A. E. Goodloe, and F. Dedden, "Runtime verification in real-time with the Copilot language: A tutorial," in *International Symposium on Formal Methods*. Springer, 2024, pp. 469–491.

[32] L. Pike, A. Goodloe, R. Morisset, and S. Niller, "Copilot: A hard real-time runtime monitor," in *International Conference on Runtime Verification*. Springer, 2010, pp. 345–359.

[33] B. Finkbeiner, F. Kohn, and M. Schledjewski, "Leveraging static analysis: An IDE for RTLola," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2023, pp. 251–262.

[34] A. Aurandt, P. H. Jones, and K. Y. Rozier, "Towards a Safe, Verified Runtime Monitor for Embedded Systems: R2U2 in Embedded Rust," in *NASA Formal Methods Symposium*. Springer, 2025, pp. 31–53.

[35] D. A. Basin, S. Krstic, and D. Traytel, "AERIAL: Almost event-rate independent algorithms for monitoring metric regular properties." *RV-CuBES*, vol. 3, pp. 29–36, 2017.

[36] D. Ulus, "Online monitoring of metric temporal logic using sequential networks," *arXiv preprint arXiv:1901.00175*, 2019.

[37] D. Basin, M. Harvan, F. Klaedtke, and E. Zălinescu, "MONPOLY: Monitoring usage-control policies," in *Runtime Verification: Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers 2*. Springer, 2012, pp. 360–364.

[38] D. A. Basin, F. Klaedtke, and E. Zalinescu, "The MonPoly monitoring tool." *RV-CuBES*, vol. 3, pp. 19–28, 2017.

[39] J. Schneider, D. Basin, S. Krstić, and D. Traytel, "A formally verified monitor for metric first-order temporal logic," in *Runtime Verification: 19th International Conference, RV 2019, Porto, Portugal, October 8–11, 2019, Proceedings 19*. Springer, 2019, pp. 310–328.

[40] D. Basin, T. Dardinier, L. Heimes, S. Krstić, M. Raszyk, J. Schneider, and D. Traytel, "A formally verified, optimized monitor for metric first-order dynamic logic," in *Automated Reasoning: 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part I 10*. Springer, 2020, pp. 432–453.

[41] M. Raszyk, D. Basin, S. Krstić, and D. Traytel, "Multi-head monitoring of metric temporal logic," in *Automated Technology for Verification and Analysis: 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28–31, 2019, Proceedings 17*. Springer, 2019, pp. 151–170.

[42] M. Raszyk, D. Basin, and D. Traytel, "Multi-head monitoring of metric dynamic logic," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2020, pp. 233–250.

[43] L. Lima, A. Herasimau, M. Raszyk, D. Traytel, and S. Yuan, "Explainable online monitoring of metric temporal logic," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2023, pp. 473–491.

[44] L. Lima, J. J. Huerta y Munive, and D. Traytel, "WhyMon: A runtime monitoring tool with explanations as verdicts," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2024, pp. 76–90.

[45] J. Elwing, L. Gamboa-Guzman, J. Sorkin, C. Travesset, Z. Wang, and K. Y. Rozier, "Mission-time LTL (MLTL) formula validation via regular expressions," in *International Conference on Integrated Formal Methods*. Springer, 2023, pp. 279–301.

[46] J. Li, M. Y. Vardi, and K. Y. Rozier, "Satisfiability checking for mission-time LTL," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 3–22.

[47] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340. [Online]. Available: https://doi.org/10.1007/978-3-540-78800-3_24

[48] Mark Otto and Jacob Thornton, "Bootstrap," 2011–2025. [Online]. Available: https://getbootstrap.com/docs/5.3/getting-started/introduction/

[49] Marijn Haverbeke, "Codemirror," 2007–2025. [Online]. Available: http://codemirror.net/

[50] Mike Bostock and Observable, Inc., "D3 (or D3.js)," 2011–2025. [Online]. Available: https://d3js.org/

[51] Sean McArthur, "warp," 2018–2024. [Online]. Available: https://crates.io/crates/warp

[52] SRI, "Summer School on Formal Techniques," Menlo Park, California, USA, , May 2025.

[53] K. Y. Rozier, "On teaching applied formal methods in aerospace engineering," in *Proceedings of the Formal Methods Teaching Workshop (FMTea) at the 3rd World Congress on Formal Methods*, ser. Lecture Notes in Computer Science (LNCS), vol. 11758. Porto, Portugal: Springer, October 2019, pp. 111–131. [Online]. Available: https://fmtea.github.io/