# FairMon: A Tool for Monitoring and Visualizing Algorithmic Fairness

Jan Baumeister[1], Bernd Finkbeiner[1,2], Vladimir Krsmanovic[1], Frederik Scheerer(✉)[1], Julian Siber[1], and Tobias Wagenpfeil[1]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{jan.baumeister,finkbeiner,vladimir.krsmanovic,
frederik.scheerer,julian.siber,tobias.wagenpfeil}@cispa.de
[2] Technical University of Munich, Germany

**Abstract.** Runtime monitoring has recently been proposed as a rigorous method for analyzing algorithmic fairness of autonomous decision systems used in critical scenarios such as credit lending, job application, and the criminal justice system. Prior work has shown that runtime monitoring, in principle, can be an effective technique for establishing the kind of human oversight required by legislation such as the EU Artificial Intelligence Act. In practice, the available monitoring tools have not been developed with this application in mind and display several critical shortcomings in these scenarios. In this paper, we present FairMon, a runtime monitoring tool tailored to fairness analysis of high-stakes decision systems. FairMon uses RTLola as a flexible specification language for monitors, which we have extended with conditional probability operators that allow for concise descriptions of algorithmic fairness properties. The tool also features a real-time visualization of intermediary values, enabling human insight into the dynamics of the monitored system.

**Keywords:** Stream-Based Monitoring · Algorithmic Fairness

## 1 Introduction

Algorithmic fairness requires that systems act without bias against protected social groups [15]. Several inquiries have shown that critical autonomous decision systems often fail to adhere to this requirement. A prominent example is the COMPAS tool for recidivism risk prediction, which has displayed highly diverging true and false positive rates between Black and White Americans when predicting a high risk of recidivism [2]. Hence, the system violated the fairness requirement known as *equalized odds* [23], which requires that the true and false positive rates for an outcome $Y$ are within some bound $\epsilon$ between all groups:

$$\left| \mathbb{P}(\hat{Y} = 1 \mid G = 1, Y = 1) - \mathbb{P}(\hat{Y} = 1 \mid G = 0, Y = 1) \right| \leq \epsilon \ .$$

In this true positive part of the property, $\hat{Y}$ denotes the system's prediction, corresponding to a high-risk assessment in the COMPAS setting, and $Y$ is the

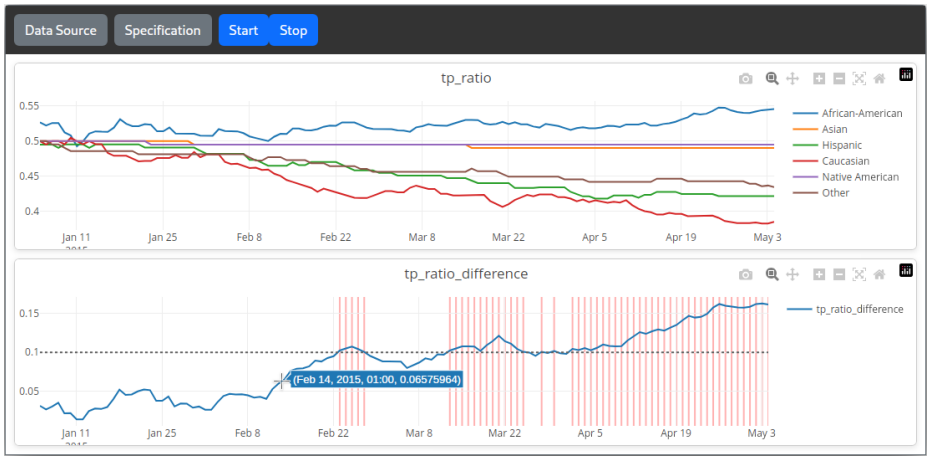



**Fig. 1:** Screenshot of FairMon monitoring the COMPAS dataset.

true outcome, indicating whether an individual actually reoffended. $G$ encodes membership in a protected group.

Runtime monitoring is a well-established technique in the formal methods community for analyzing executions of diverse high-stakes systems such as unmanned aircraft [8,28,32] and distributed networks [20]. Several recent studies have also highlighted its general applicability for monitoring algorithmic fairness [24,26]. In a recent case study [11], we have shown that stream-based monitoring with the specification language RTLola [9] can monitor algorithmic fairness requirements in decision and prediction systems over time. Building on this exploratory case study, we have developed a dedicated tool for this purpose: FairMon. It addresses shortcomings of existing stream-based monitoring frameworks, including RTLola, with respect to syntactical succinctness and the visualization during execution.

*Dedicated Operators.* Encoding algorithmic fairness with RTLola as described in [11] requires elaborate usage of parameterized streams and aggregation. Not only is this an error-prone process even for experts, but it also results in a high threshold for adoption by practitioners. To make the specification of fairness properties more intuitive, we have added probability operators to RTLola. These directly encode (conditional) probabilities of events and, in this way, allow a direct encoding of properties such as equalized odds. Assuming the boolean variables are inputs, specifying this requirement with the original approach [11] required a minimum of five output streams. With the dedicated operators, all that is needed is the following specification, which mirrors the mathematical formulation of the equalized odds property:

```
1 | trigger abs(y_hat.prob(given: g∧¬y) - y_hat.prob(given: ¬g∧¬y)) > ϵ
```

This condition is evaluated at runtime by the monitor, and a warning is issued as soon as the trigger condition evaluates to true. Accordingly, the trigger condition is defined as the negation of the equalized odds property above.

*Visualization.* In addition, FairMon comes with a graphical user interface, depicted in Figure 1, that makes the monitoring process more accessible. The GUI enables the configuration of data sources, the editing of fairness specifications, and the visualization of the monitoring process through line plots of the important values. This allows the system's state and fairness to be tracked over time, with violations easily detectable in a visual manner.

*Integration.* FairMon is designed to integrate easily into existing decision-making systems through an integration framework. Its interface allows users to choose and configure input sources, supporting both domain-specific and general-purpose data sources. To demonstrate this flexible interface, we connect FairMon to the EU's DSA Transparency Database API [17], which provides a stream of moderation decisions of large online platforms and offers an example for a large-scale decision-making system.

## 2 Probabilistic RTLola

RTLola [9] is a stream-based specification language for runtime monitoring. A specification consists of stream definitions, each stream representing an infinite sequence of values. *Input streams* capture values reflecting the current state of the monitored system, while *output streams* compute new values by filtering or aggregating other streams. Violations of the specification are detected through *triggers*, which are special, boolean-valued, output streams: whenever a trigger evaluates to true, a violation of the specification has occurred.

*Example 1.* Consider the following specification computing the probability of a boolean event over a timed window:

```
input a : Bool
output sum_a @10s := a.aggregate(over: 60s, using: Σ)
output count_a @10s := a.aggregate(over: 60s, using: count)
output prob_a @10s := sum_a / count_a
trigger prob_a > 0.1
```

The specification declares a boolean-valued input stream `a` and computes two sliding window aggregations over the last 60 seconds. The first, `sum_a`, counts how often `a` has been true in that window, while the second, `count_a`, counts all events of `a`. Both produce a new value every 10 seconds. The ratio `prob_a` yields the empirical probability that `a` is true during the last minute and the trigger raises an alarm if this probability is higher than 0.1.

Such probability computations are common in fairness specification, as shown in our recent paper [11]. In both that work and the example above, probabilities

Table 1: Probability operators in RTLola.

| Syntax | Meaning |
|---|---|
| `e.prob(given: c)` | $\mathbb{P}(e \mid c)$ over the entire history |
| `e.prob(given: c, over: d)` | $\mathbb{P}(e \mid c)$ within a window of duration $d$ |
| `e.prob(given: c, prior: p, confidence: k)` | as above, smoothed toward prior $p$ with weight $k$ |
| `e.prob(given: c, prior: p, confidence: k, over: d)` | as above, windowed and smoothed |
| `s.prob(of: λ)` | fraction of instances of $s$ satisfying $\lambda$ |
| `s.prob(of: λ, given: μ)` | $\mathbb{P}(\lambda \mid \mu)$ across instances of $s$ |
| `s.prob(of: λ, given: μ, prior: p, confidence: k)` | as above, smoothed |

were represented as floating-point numbers. This choice, however, limits type checking and makes specifications less precise and harder to interpret. To address these issues, we introduced a dedicated value type `Prob`, representing a probability in the interval $[0, 1]$. Further, the new probability type allows runtime checks to ensure that probability values remain within the valid range of $[0, 1]$. It ensures that errors are detected as soon as they occur, rather than letting invalid values propagate through computations. Values outside $[0, 1]$ could silently distort fairness measures, leading to incorrect conclusions.

Building on the new `Prob` type, we extended RTLola to include probability-specific operations and aggregations. These extensions enable the expression of (conditional) probabilities through dedicated operators, allowing for concise formulations of fairness specifications.

*Example 2.* As an illustration, consider the definition of demographic parity [15]:

$$\big|\, \mathbb{P}(A = 1 \mid G = 1) - \mathbb{P}(A = 1 \mid G = 0) \,\big| \leq \epsilon \ .$$

With the new operators, this condition translates almost directly into RTLola:

```
1 | trigger abs(a.prob(given: g) - a.prob(given: ¬g)) > ϵ
```

In the above example, the shorthand method `prob` performs a probability aggregation. RTLola supports both unconditional and conditional probability aggregations, and allows specifying a prior [30,11] for when little data is available. The aggregations can be applied over sliding windows (such as in Example 1 using the shorthand `a.prob(over: 60s)`), over the full history (as in Example 2), or across instances of parameterized streams. A summary of all supported operators is given in Table 1.

These constructs allow fairness specifications to be expressed far more concisely. As a result, even non-monitoring experts can formulate fairness specifications naturally. By aligning the written specification as close as possible to the natural formulation of these fairness notions, we bridge the crucial gap between

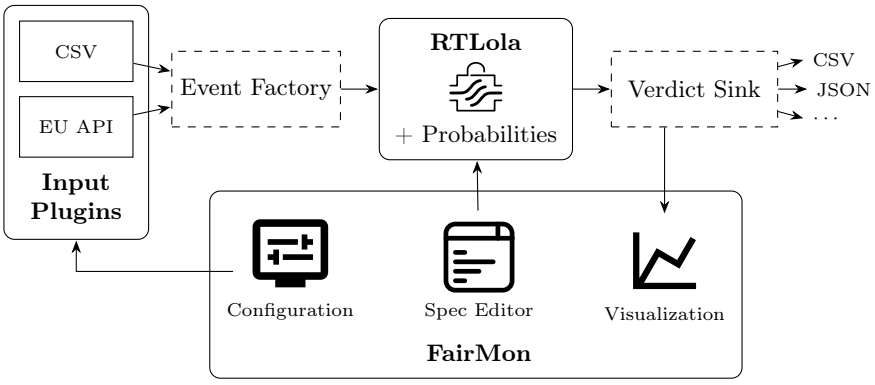



**Fig. 2:** Overview of the FairMon tool.

*what the users want to express* and *how it is expressed*. Closing this gap is essential for applying fairness toolkits to recommender and decision systems, as well as for monitoring machine learning systems [13,27].

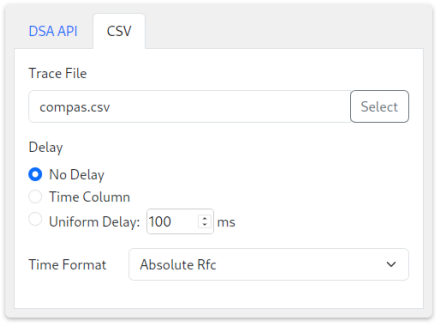
## 3 FairMon

In addition to concise and understandable specifications, the results produced by the monitor should be easy to interpret and updated continuously at runtime. For this purpose, we present FairMon[3], a graphical interface for fairness monitoring with RTLola [9,5]. It provides an intuitive configuration interface for the monitor and enables visualization of its results at runtime.

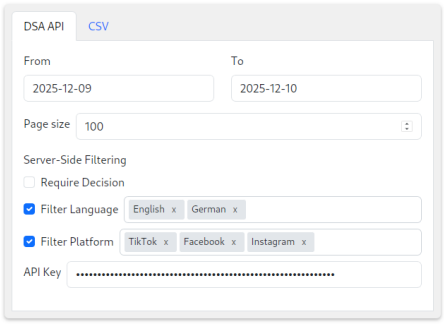
We show an overview of the FairMon tool in Figure 2. FairMon consists of three components: a specification editor, a configuration interface, and a visualization panel. To facilitate seamless integration into existing systems, our approach builds on the plugin-based integration framework for RTLola introduced in [8]. This framework separates between input and output plugins: input plugins extract input streams from the monitored system through a unified *Event Factory* interface, while output plugins integrate the monitor's output streams into external components using a *Verdict Sink*. The specification editor allows users to author and modify the RTLola fairness specification, which is passed directly to the monitor. Our tool provides a graphical output plugin that visualizes the monitor's internal state and provides live plots of relevant streams, as visible in Figure 1. Additionally, the graphical interface allows users to configure and select the input plugins for acquiring the data. Currently, we supply two predefined input plugins, a domain-specific one tailored to the DSA API, and a general-purpose plugin that accepts arbitrary data over CSV.

To improve the performance of the tool when querying large APIs or datasets, we implement a filtering mechanism to reduce the amount of data loaded. As

[3] Our tool is publicly available at `https://github.com/reactive-systems/rtlola-fairmon`.




**(a)** Configuration of the CSV plugin.




**(b)** Configuration of the DSA API plugin.

**Fig. 3:** Screenshots of the configuration options for different input plugins.

shown in Figure 3b, the required filters are integrated into our interface, enabling users to configure them directly in the frontend. As an example, Instagram emits roughly 5.5 million records per day to the DSA Transparency Database, while often only a fraction of the reported data is of interest. For example, a specification we present in Section 4 requires only a subset of 380,000 records. Querying the full records is unnecessary and wastes significant time and computing resources due to hitting API limits. To combat this type of problem, we expose configuration options for server-side filtering, significantly reducing overall query time. For the Instagram specification, this mechanism reduces the loading time by 93%.

For visualizing the stream values over time, we use Plotly, providing users with interactive features such as zooming and panning of the plots. The configuration of the plots, including which streams are displayed and how they are rendered, is controlled through annotations embedded directly in the RTLola specification. This annotation mechanism was originally introduced in [10] and allows for tightly binding the configuration to the specification and preventing the configuration and specification from diverging.

The visualization tool provides several mechanisms to aid the interpretability of the data and results, all of which can be seen in Figure 1. Triggers are rendered as red vertical bars, offering an immediate visual cue for when conditions are violated, with their associated thresholds represented as horizontal dashed lines. For parameterized streams, all instances can be displayed within a single plot, while different streams can also be combined in a single plot or displayed across separate plots. Crucially, all visualization properties are derived directly from the specification itself, requiring no external configuration. Streams are assigned to plots via an annotated plot identifier, where streams sharing the same identifier are grouped together into a single panel. Consider Section B for example specifications.

## 4 Examples

We illustrate our tool using two examples: COMPAS and the DSA Transparency API. The former serves as a well-known example from the literature and allows us to make a direct comparison with specifications from our earlier work. The DSA Transparency database application uses the newly introduced API from the EU to monitor online moderation data. All specifications and further details of the experiments can be found in the Appendix.

*COMPAS.* As a first example, we revisit the equalized odds specification introduced in our previous work [11]. We use the real COMPAS dataset, imported into FairMon via a CSV input plugin as shown in Figure 3a. The resulting visualization in Figure 1 displays the true positive rates across groups in the upper plot, and the maximal difference between the groups in the lower plot. The horizontal dotted line marks the threshold, while the vertical red bars highlight violations of the specification.

The specification must extract multiple independent trials from the inputs, and includes temporal logic to determine whether recidivism occurs within two years of the initial screening. As a result, the original specification was relatively large, spanning 30 lines. With the new probability operators, we can express the same behavior far more concisely: the true-positive specification is reduced by 57% (to 13 lines), and the number of output streams is reduced from 7 to 3.

*DSA Transparency API.* With the introduction of the Digital Services Act (DSA) [18] and the AI Act [19] by the European Commission, platforms classified as Very Large Online Platforms are legally obligated to report on practices that impact their users. One of the obligations under the DSA is that these platforms must provide *statements of reason* following all moderation decisions, which have to contain information about the mechanism used for detection and removal of harmful content, as well as legal reasoning for it.

We extended our tool with an input plugin that interfaces with the DSA database to monitor and visualize relevant fairness properties over these moderation decisions. In our experiments, we focused on three specifications: The first examines the ratio of automated decisions across content languages to assess whether groups speaking certain languages face disproportional automated moderation. The remaining two specifications, adopted from existing work [36], analyze the delay between content creation and moderation, and highlight the differences between the ratios of automated detection and decision across the different platforms, respectively. We examine the moderation decision made by the Google Play Store during December 2025, with the data being collected from the DSA Statements of Reason database using the API functionality of our tool. The specification can be inspected in Appendix B.

As a main insight, we discovered that some languages are significantly more likely to be manually moderated, potentially indicating a gap due to insufficient numbers of fluent human moderators and an overreliance on automated moderation. Figure 4 shows FairMon's output for a specification that compares the daily

automation ratios of moderation decisions between different languages, i.e., how likely reported content in the given language is moderated automatically. The figure shows this for Italian, English, and German. Over the whole month, the likelihood of Italian and English content to be moderated automatically is consistently over 90%, while it remains highly volatile for German, and frequently drops below 80%. A curious insight for German language moderation is that it approaches full automation on weekends (i.e., December 7, 14, ...).

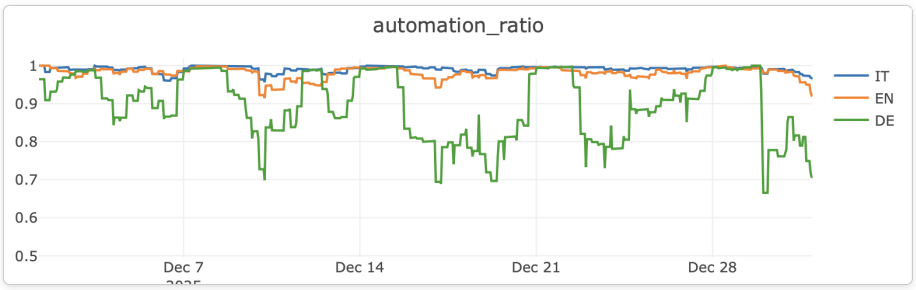



**Fig. 4:** Screenshot of FairMon monitoring the ratio of automated moderation in the Google Play Store for Italian (IT), English (EN), and German (DE).

Overall, the analysis with FairMon suggests that the manner and hence also the quality of content moderation may be influenced by content language and the weekday of the decision. This type of disparity has been a focus of existing legal actions made by the regulatory bodies regarding the obligations under the DSA, namely the EU Commission investigation against Meta for their actions during the 2024 Romanian Elections [16]. The use of tools such as FairMon would enable real-time monitoring of platform-level behavior, and enable proactive enforcement and regulatory actions instead of only after-the-fact investigation.

## 5 Related Work

There exists a broad range of literature on the theory of algorithmic fairness; we refer to [33,4] for a comprehensive overview. Runtime monitoring of algorithmic fairness has been studied in recent work [1,24,25,26,11]. Beyond monitoring, there are other approaches for formally specifying and obtaining fairness properties, particularly for testing [31,35] and learning [37]. While some visualization tools for fairness already exist [14,34], they do not offer the flexibility of stream-based specification languages for reasoning over time and about composite properties, i.e., they analyze conditional probabilities in a given dataset corresponding to a fixed snapshot of a system's history, wile FairMon analyzes and visualizes how the fairness behavior of a system changes during deployment. Various methods exist for visualizing the execution of runtime monitors, whether for offline anal-

ysis [21,3,29] or online scenarios [6,7], but the tools do not support conditional probabilities necessary for the specification of fairness properties.

## 6 Conclusion

FairMon extends RTLola with a probability type and dedicated operators, allowing fairness requirements to be stated concisely and naturally. This reduces the entry barrier for users without monitoring expertise and broadens access to fairness analysis. Paired with an interactive graphical interface that visualizes the runtime behavior, the tool supports both comprehension and practical use. Experiments with the COMPAS case and data from the DSA Transparency database further show that the approach performs well in real-world scenarios.

*Future Work.* The framework is designed to be easily extensible. Future work includes supporting a broader range of decision systems and expressing additional fairness requirements, particularly those relevant under EU regulations. We also aim to automate the extraction of API filters from specifications, reducing the need for manual configuration and further simplifying the fairness monitoring process. Since fairness monitoring often relies on privacy-sensitive demographic and behavioral data, another promising direction is the integration of RTLola's privacy mechanisms [22] into our tool to enable fairness analysis with formal privacy guarantees for monitored individuals.

**Acknowledgments.** This work was partially supported by the German Research Foundation (DFG) as part of TRR 248 (No. 389792660) and PreCePT (No. 521273327), and by the European Research Council (ERC) Grant HYPER (No. 101055412). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. J. Baumeister, F. Scheerer and J. Siber carried out this work as members of the Saarbrücken Graduate School of Computer Science.

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Albarghouthi, A., Vinitsky, S.: Fairness-aware programming. In: danah boyd, Morgenstern, J.H. (eds.) Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT* 2019, Atlanta, GA, USA, January 29-31, 2019. pp. 211–219. ACM (2019). https://doi.org/10.1145/3287560.3287588
2. Angwin, J., Larson, J., Mattu, S., Kirchner, L.: Machine bias. there's software used across the country to predict future criminals. and it's biased against blacks. ProPublica (2016), https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

3. Aurandt, A., Rozier, K.Y., Jones, P.H.: R2u2 playground: Visualization of a real-time, temporal logic runtime monitor. In: CONFERENCE ON FORMAL METHODS IN COMPUTER-AIDED DESIGN–FMCAD 2025. p. 126
4. Barocas, S., Hardt, M., Narayanan, A.: Fairness and Machine Learning: Limitations and Opportunities. MIT Press (2023)
5. Baumeister, J., Correnson, A., Finkbeiner, B., Scheerer, F.: An intermediate program representation for optimizing stream-based languages. In: Piskac, R., Rakamaric, Z. (eds.) Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part III. Lecture Notes in Computer Science, vol. 15933, pp. 393–407. Springer (2025). https://doi.org/10.1007/978-3-031-98682-6_20
6. Baumeister, J., Finkbeiner, B., Gumhold, S., Schledjewski, M.: Real-time visualization of stream-based monitoring data. In: Dang, T., Stolz, V. (eds.) Runtime Verification - 22nd International Conference, RV 2022, Tbilisi, Georgia, September 28-30, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13498, pp. 325–335. Springer (2022). https://doi.org/10.1007/978-3-031-17196-3_21
7. Baumeister, J., Finkbeiner, B., Kautenburger, J., Rubeck, C.: Rtlolamo3vis-a mobile and modular visualization framework for online monitoring. In: International Conference on Runtime Verification. pp. 192–202. Springer (2024)
8. Baumeister, J., Finkbeiner, B., Kohn, F., Löhr, F., Manfredi, G., Schirmer, S., Torens, C.: Monitoring unmanned aircraft: Specification, integration, and lessons-learned. In: Gurfinkel, A., Ganesh, V. (eds.) Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14682, pp. 207–218. Springer (2024). https://doi.org/10.1007/978-3-031-65630-9_10
9. Baumeister, J., Finkbeiner, B., Kohn, F., Scheerer, F.: A tutorial on stream-based monitoring. In: Platzer, A., Rozier, K.Y., Pradella, M., Rossi, M. (eds.) Formal Methods - 26th International Symposium, FM 2024, Milan, Italy, September 9-13, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14934, pp. 624–648. Springer (2024). https://doi.org/10.1007/978-3-031-71177-0_33
10. Baumeister, J., Finkbeiner, B., Scheerer, F.: Active monitoring with rtlola: A specification-guided scheduling approach. In: Könighofer, B., Torfah, H. (eds.) Runtime Verification - 25th International Conference, RV 2025, Graz, Austria, September 15-19, 2025, Proceedings. Lecture Notes in Computer Science, vol. 16087, pp. 181–201. Springer (2025). https://doi.org/10.1007/978-3-032-05435-7_11
11. Baumeister, J., Finkbeiner, B., Scheerer, F., Siber, J., Wagenpfeil, T.: Stream-based monitoring of algorithmic fairness. In: Gurfinkel, A., Heule, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part I. Lecture Notes in Computer Science, vol. 15696, pp. 60–81. Springer (2025). https://doi.org/10.1007/978-3-031-90643-5_4
12. Baumeister, J., Finkbeiner, B., Scheerer, F., Siber, J., Wagenpfeil, T.: Stream-based monitoring of algorithmic fairness. In: Gurfinkel, A., Heule, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, Canada, May 3–8, 2025. pp. 60–81. Lecture Notes in Computer Science, Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-90643-5_4

13. Beattie, L., Taber, D., Cramer, H.: Challenges in translating research to practice for evaluating fairness and bias in recommendation systems. In: Golbeck, J., Harper, F.M., Murdock, V., Ekstrand, M.D., Shapira, B., Basilico, J., Lundgaard, K.T., Oldridge, E. (eds.) RecSys '22: Sixteenth ACM Conference on Recommender Systems, Seattle, WA, USA, September 18 - 23, 2022. pp. 528–530. ACM (2022). https://doi.org/10.1145/3523227.3547403
14. Cabrera, Á.A., Epperson, W., Hohman, F., Kahng, M., Morgenstern, J., Chau, D.H.: FAIRVIS: visual analytics for discovering intersectional bias in machine learning. In: Chang, R., Keim, D.A., Maciejewski, R. (eds.) 14th IEEE Conference on Visual Analytics Science and Technology, IEEE VAST 2019, Vancouver, BC, Canada, October 20-25, 2019. pp. 46–56. IEEE (2019). https://doi.org/10.1109/VAST47406.2019.8986948
15. Dwork, C., Hardt, M., Pitassi, T., Reingold, O., Zemel, R.S.: Fairness through awareness. In: Goldwasser, S. (ed.) Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. pp. 214–226. ACM (2012). https://doi.org/10.1145/2090236.2090255
16. European Commission: Commission preliminarily finds TikTok and Meta in breach of their transparency obligations under the Digital Services Act. Press Release, IP/25/2503 (Oct 2025), https://ec.europa.eu/commission/presscorner/detail/en/ip_25_2503, directorate-General for Communications Networks, Content and Technology
17. European Commission-DG CONNECT: Digital services act transparency database (2023). https://doi.org/10.2906/134353607485211
18. European Parliament and Council of the European Union: Regulation (EU) 2022/2065 on a single market for digital services (Digital Services Act). Official Journal of the European Union, L 277, 1–102 (2022), https://eur-lex.europa.eu/eli/reg/2022/2065/oj
19. European Parliament and Council of the European Union: Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). Official Journal of the European Union, L 2024/1689 (2024), https://eur-lex.europa.eu/eli/reg/2024/1689/oj
20. Faymonville, P., Finkbeiner, B., Schirmer, S., Torfah, H.: A stream-based specification language for network monitoring. In: Falcone, Y., Sánchez, C. (eds.) Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10012, pp. 152–168. Springer (2016). https://doi.org/10.1007/978-3-319-46982-9_10
21. Finkbeiner, B., Kohn, F., Schledjewski, M.: Leveraging static analysis: An IDE for rtlola. In: André, É., Sun, J. (eds.) Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14216, pp. 251–262. Springer (2023). https://doi.org/10.1007/978-3-031-45332-8_13
22. Finkbeiner, B., Scheerer, F.: Differentially private runtime monitoring. In: Darulova, E., Lin, A.W., Rümmer, P. (eds.) Computer Aided Verification - 38th International Conference, CAV 2026, Lisbon, Portugal, July 26-29, 2026, Proceedings, Part I. Lecture Notes in Computer Science, vol. 16682, pp. 449–472. Springer (2026). https://doi.org/10.1007/978-3-032-32519-8_23
23. Hardt, M., Price, E., Srebro, N.: Equality of opportunity in supervised learning. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona,

Spain. pp. 3315–3323 (2016), https://proceedings.neurips.cc/paper/2016/hash/9d2682367c3935defcb1f9e247a97c0d-Abstract.html
24. Henzinger, T.A., Karimi, M., Kueffner, K., Mallik, K.: Monitoring algorithmic fairness. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13965, pp. 358–382. Springer (2023). https://doi.org/10.1007/978-3-031-37703-7_17
25. Henzinger, T.A., Karimi, M., Kueffner, K., Mallik, K.: Runtime monitoring of dynamic fairness properties. In: Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency, FAccT 2023, Chicago, IL, USA, June 12-15, 2023. pp. 604–614. ACM (2023). https://doi.org/10.1145/3593013.3594028
26. Henzinger, T.A., Kueffner, K., Mallik, K.: Monitoring algorithmic fairness under partial observations. In: Katsaros, P., Nenzi, L. (eds.) Runtime Verification - 23rd International Conference, RV 2023, Thessaloniki, Greece, October 3-6, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14245, pp. 291–311. Springer (2023). https://doi.org/10.1007/978-3-031-44267-4_15
27. Holstein, K., Vaughan, J.W., III, H.D., Dudík, M., Wallach, H.M.: Improving fairness in machine learning systems: What do industry practitioners need? In: Brewster, S.A., Fitzpatrick, G., Cox, A.L., Kostakos, V. (eds.) Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019. p. 600. ACM (2019). https://doi.org/10.1145/3290605.3300830
28. Johannsen, C., Jones, P.H., Kempa, B., Rozier, K.Y., Zhang, P.: R2U2 version 3.0: Re-imagining a toolchain for specification, resource estimation, and optimized observer generation for runtime verification in hardware and software. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13966, pp. 483–497. Springer (2023). https://doi.org/10.1007/978-3-031-37709-9_23
29. Kallwies, H., Leucker, M., Schmitz, M., Schulz, A., Thoma, D., Weiss, A.: Tessla - an ecosystem for runtime verification. In: Dang, T., Stolz, V. (eds.) Runtime Verification - 22nd International Conference, RV 2022, Tbilisi, Georgia, September 28-30, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13498, pp. 314–324. Springer (2022). https://doi.org/10.1007/978-3-031-17196-3_20
30. Mitchell, T.M.: Machine learning, International Edition. McGraw-Hill Series in Computer Science, McGraw-Hill (1997), https://www.worldcat.org/oclc/61321007
31. Morales, S., Clarisó, R., Cabot, J.: Langbite: A platform for testing bias in large language models. arXiv preprint arXiv:2404.18558 (2024)
32. Perez, I., Goodloe, A.E., Dedden, F.: Runtime verification in real-time with the copilot language: A tutorial. In: Platzer, A., Rozier, K.Y., Pradella, M., Rossi, M. (eds.) Formal Methods - 26th International Symposium, FM 2024, Milan, Italy, September 9-13, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14934, pp. 469–491. Springer (2024). https://doi.org/10.1007/978-3-031-71177-0_27
33. Pessach, D., Shmueli, E.: Algorithmic Fairness, pp. 867–886. Springer International Publishing, Cham (2023). https://doi.org/10.1007/978-3-031-24628-9_37
34. Saleiro, P., Kuester, B., Stevens, A., Anisfeld, A., Hinkson, L., London, J., Ghani, R.: Aequitas: A bias and fairness audit toolkit. CoRR **abs/1811.05577** (2018), http://arxiv.org/abs/1811.05577

35. She, Y., Biswas, S., Kästner, C., Kang, E.: Fairsense: Long-term fairness analysis of ml-enabled systems. In: 47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025. pp. 782–794. IEEE (2025). https://doi.org/10.1109/ICSE55347.2025.00159, https://doi.org/10.1109/ICSE55347.2025.00159
36. Trujillo, A., Fagni, T., Cresci, S.: The dsa transparency database: Auditing self-reported moderation actions by social media. Proceedings of the ACM on Human-Computer Interaction **9**(2), 1–28 (2025)
37. Zhang, H., Chu, X., Asudeh, A., Navathe, S.B.: Omnifair: A declarative system for model-agnostic group fairness in machine learning. In: Proceedings of the 2021 international conference on management of data. pp. 2076–2088 (2021)

# A COMPAS Specification

In the paper, we refer to the following (true-positive) equalized-odds specification for COMPAS with the operators presented in this paper. The COMPAS specification without special operators can be found in the corresponding paper [12]. For the line counts, we disregard empty lines, comments, and visualization annotations.

## A.1 Without Probability Operators

```
input event : String
input id : Int64
input group : String
input score : Int64

// Defendant Information
output days_per(i)
  spawn with id
  eval @Global(1d) with days_per(i).last(or: 0) + 1
  close when days_per(i) = 730
output has_re(i)
    spawn with id
    eval when id == i with event == "RECIDIVISM"
    close @Global(1d) when days_per(i).hold(or: 0) = 730
output tp_event(i, g, s)
  spawn with (id, group, score)
  eval @Global(1d)
    when days_per(i).hold(or: 0) = 730 ∧ has_re(i).aggregate(over:
        730d, using: ∃) with s > 6
  close @Global(1d) when days_per(i).hold(or: 0) = 730

// TP Ratio
output abs_re(g) : UInt64
    spawn with group
    eval @Global(1d) with abs_re(g).last(or: 100) +
    tp_event.aggregate(over_instances: All(ii, ig, is => ig = g),
        using: count)
output abs_hr_re(g) : UInt64
  spawn with group
  eval @Global(1d) with abs_hr_re(g).last(or: 50) +
    tp_event.aggregate(over_instances: All(ii, ig, is => ig = g),
        using: sum)

#[plot]
output tp_ratio(g)
    spawn with group
    eval when abs_re(g) != 0
        with cast<UInt64, Float64>(abs_hr_re(g)) / cast<UInt64,
            Float64>(abs_re(g))
```

```
36
37 #[plot]
38 #[threshold="0.1"]
39 output tp_rate_diff @1d tp_ratio.aggregate(over_instances: all,
       using: max).defaults(to: 0.0) -
       tp_ratio.aggregate(over_instances: all, using: min).defaults(to:
       0.0) > 0.1
40
41 #[plot="tp_rate_diff"]
42 trigger tp_rate_diff > 0.1
```

### A.2 With Probability Operators

```
1  input event : String
2  input id : Int64
3  input group : String
4  input score : Int64
5
6  output user_info(user)
7    spawn with id when event == "SCORE"
8    eval when user == id with (score, group)
9    close @Local(2y)
10
11 #[plot]
12 output tp_rate(r)
13   spawn with group
14   eval when event == "RECIDIVISM" with prob(of: score(id).hold(or: 0)
         > 6, given: user_info(id).exists(over: 2y) && user_info(id).1
         == r)
15
16 #[plot]
17 #[threshold="0.1"]
18 output tp_rate_diff @Global(1d) := tp_rate.aggregate(over_instances:
       all, using: max).defaults(to: 0.0) -
       tp_rate.aggregate(over_instances: all, using: min).defaults(to:
       0.0)
19
20 #[plot="tp_rate_diff"]
21 trigger tp_rate_diff > 0.1
```

## B DSA Specifications

### B.1 Automation per Content Language

```
1 input event : String
2 input id : Int64
3 input group : String
```

```
input score : Int64

output user_info(user)
  spawn with id when event == "SCORE"
  eval when user == id with (score, group)
  close @Local(2y)

#[plot]
output tp_rate(r)
  spawn with group
  eval when event == "RECIDIVISM" with prob(of:
      user_info(id).0.hold(or: 0) > 6, given:
      user_info(id).1.hold(or: "") == r)

#[plot]
#[threshold="0.1"]
output tp_rate_diff @Global(1d) := tp_rate.aggregate(over_instances:
    all, using: max).defaults(to: 0.0) -
    tp_rate.aggregate(over_instances: all, using: min).defaults(to:
    0.0)

#[plot="tp_rate_diff"]
trigger tp_rate_diff > 0.1
```

### B.2 Decision Delay

```
import time
input id: Int64
input content_language: String
input automated_decision: String
input automated_detection: Bool
input platform_name: String
input application_date: String
input created_at: String

output application_date_time := time_from_rfc(application_date)
output created_at_time := time_from_rfc(created_at)

output delay_per(p)
    spawn with platform_name
    eval when platform_name == p
    with
        cast<UInt64,Float64>(created_at_time.time_diff(application_date_time).to_days())

#[plot]
output avg_delay_per(p)
    spawn with platform_name
    eval @1h with delay_per(p).aggregate(over: 1d, using:
        avg).defaults(to: 0.0)
```

### B.3 Automation Ratios

```
input id: Int64
input content_language: String
input automated_decision: String
input automated_detection: Bool
input platform_name: String

output automation (p, dec, det)
    spawn with (platform_name, automated_decision,
        automated_detection)
    eval with prob(of: automated_decision == dec, given:
        automated_detection == det)

output automation_plot(p, dec, det)
    spawn with (platform_name, automated_decision,
        automated_detection)
    eval @60s with automation(p, dec, det).hold(or: 0.0)

#[plot]
output tiktok(dec, det)
    spawn with (automated_decision, automated_detection)
    eval @60s with automation_plot("TikTok", dec, det).hold(or: 0.0)

#[plot]
output shopify(dec, det)
    spawn with (automated_decision, automated_detection)
    eval @60s with automation_plot("Shopify", dec, det).hold(or: 0.0)

#[plot]
output idealo(dec, det)
    spawn with (automated_decision, automated_detection)
    eval @60s with automation_plot("Idealo", dec, det).hold(or: 0.0)

#[plot]
output ebay(dec, det)
    spawn with (automated_decision, automated_detection)
    eval @60s with automation_plot("eBay", dec, det).hold(or: 0.0)
```

## C Query Delay Details

We retrieve the data for 26.11.2025 using a page size of 1000. If a quota limit is reached, the process pauses and resumes with one-second intervals until the quota becomes available again.

| Scenario | Pages | Full time (s) | Hits | Mean latency (s) |
|---|---|---|---|---|
| 1d_all | 5501 | 6662.741 | 5500000 | 0.928 |
| 1d_instagram | 379 | 484.664 | 378449 | 0.986 |