Formally Verified Algorithmic Fairness using Information-Flow Tools (Extended Abstract)

Samuel Teuber[1], Bernhard Beckert[1]

[1]Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Abstract

This work presents results on the use of Information-Flow tools for the formal verification of algorithmic fairness properties. The problem of enforcing secure information-flow was originally studied in the context of information security: If secret information may "flow" through an algorithm in such a way that it can influence the program's output, we consider that to be insecure information-flow as attackers could potentially observe (parts of) the secret. Due to its wide-spread use, there exist numerous tools for analyzing secure information-flow properties. Recent work showed that there exists a strong correspondence between secure information-flow and algorithmic fairness: If protected group attributes are treated as secret program inputs, then secure information-flow means that these "secret" attributes cannot influence the result of a program. We demonstrate that off-the-shelf tools for information-flow can be used to formally analyze algorithmic fairness properties including established notions such as (conditional) demographic parity as well as a new quantitative notion named *fairness spread*.

Keywords

Program Analysis, Formal Methods, Demographic Parity, Quantitative Analysis, Qualitative Analysis

1. Introduction

Information-Flow is a well-studied concept originally introduced for software security analysis (see, e.g., [1, 2, 3, 4, 5, 6, 7, 8]). Consider a program P that checks whether a user entered the right password. P's inputs are the correct password (*high* security status) as well as the user entered password (*low* security status). A "bad" program that, for example, returns the full secret password to the user would be said to contain insecure information-flow. In general, we say that P contains insecure information-flow iff there exist two high security status values h, h' and a low security value l such that P returns different outputs for the inputs (h, l) and (h', l). We build upon previous work which observed similarities between the ideas in algorithmic fairness and information-flow analysis [9] and show that there is a strong correspondence between definitions in qualitative as well as quantitative information-flow and algorithmic fairness: By treating protected attributes such as race, gender, or age as "secret" inputs, we can analyze their influence on the program's outcome. We demonstrate that many off-the-shelf qualitative and quantitative information-flow tools can equally be used to verify algorithmic fairness properties such as demographic parity or a newly introduced metric which we call *Fairness Spread*.

EWAF'23: European Workshop on Algorithmic Fairness, June 07–09, 2023, Winterthur, Switzerland

teuber@kit.edu (S. Teuber); beckert@kit.edu (B. Beckert)
https://formal.kit.edu/~teuber (S. Teuber); https://formal.kastel.kit.edu/~beckert (B. Beckert)
0000-0001-7945-9110 (S. Teuber); 0000-0002-9672-3291 (B. Beckert)

2. Information-Flow and Demographic Parity

Qualitative Information-Flow. We demonstrate that information-flow provides an interesting framework for the analysis of disparate treatment. To this end, we consider decision-making programs P which process a protected group attribute $G \in \mathcal{G}$ and an independently distributed unprotected variable $U \in \mathcal{U}$ which can be comprised of a single value or a tuple of multiple values. The program has an outcome $d \in \mathcal{D}$. By assigning G to have high security status and U to have low security status, we can use existing tools to check whether a given program satisfies unconditional noninterference:

Definition 1 (Unconditional Noninterference (see e.g. [7]))**.** *A program P satisfies* unconditional noninterference *iff for all $g_1, g_2 \in \mathcal{G}$ and for all $u \in \mathcal{U}$ it holds that $P(g_1, u) = P(g_2, u)$.*

It can be shown that if P satisfies unconditional noninterference with respect to G and U it also satisfies the fairness notion of demographic parity:

Lemma 2 (Unconditional Noninterference implies Demographic Parity)**.** *Let P be a program and let G, U be distributed arbitrarily, but independently. If P satisfies unconditional noninterference, then the outcome of P satisfies demographic parity, i.e., for all $d \in \mathcal{D}, g_1, g_2 \in \mathcal{G}$ it holds that* $\Pr[P(G,U) = d \mid G = g_1] = \Pr[P(G,U) = d \mid G = g_2]$

Note that the inverse is not the case which makes unconditional noninterference a strictly stronger property. More refined flavors of information-flow which allow some level of information leakage lead to other kinds of, more restricted, fairness guarantees. Information-flow properties can be verified using off-the-shelf tools such as the deductive program verification tool KeY [7] or the static analysis tool Joana [10, 11].

Quantitative Information-Flow. Quantitative Information-Flow (see e.g. [4]) transforms the qualitative security analysis of software into a metric for a software's security. In this context, one important notion is the idea of *Conditional Vulnerability*:

Definition 3 (Conditional Vulnerability)**.** *For a program P and random independent variables G, U, we define the* Conditional Vulnerability *V of G w.r.t. $P(G,U)$ and U as follows:*

$$V(G \mid P(G,U), U) = \sum_{u \in \mathcal{U}} \sum_{d \in \mathcal{D}} \max_{g \in \mathcal{G}} \left(\Pr[G = g \mid P(G,U) = d, U = u] \cdot \Pr[P(G,U) = d, U = u]\right).$$

For binary classifiers (i.e., for $|\mathcal{D}| = 2$), our work shows that Conditional Vulnerability can equally be used in the fairness context to compute a metric which we call *Fairness Spread.* For programs P with a binary outcome and G, U as before, the metric of Fairness Spread quantifies the weighted disparity in treatment between the most advantaged and most disadvantaged groups for each possible assignment of unprotected attributes, i.e.:

$$\sum_{u \in \mathcal{U}} \Pr[U = u] \left(\max_{g_1, g_2 \in \mathcal{G}} \left(\Pr[P(G,U) = 1 \mid G = g_1, U = u] - \Pr[P(G,U) = 1 \mid G = g_2, U = u]\right)\right).$$

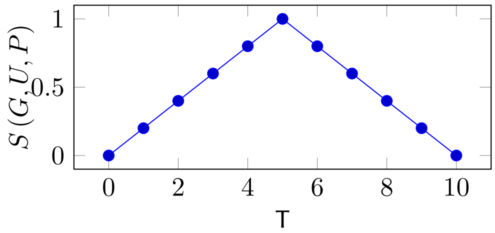
This metric satisfies three desirable requirements for a metric: 1. The metric returns its minimum (optimal) value if the program satisfies (un)conditional demographic parity. 2. A less fair program has a higher value of the metric. 3. It can be measured using off-the-shelf tools (e.g. [12]).

```
bool hire(int gender, int race, int college_rank,
          int experience) {
    int score = college_rank+experience;
    if (gender==0 && race == 0) {
        score -= 360;
    }
    score*=experience*college_rank;
    if (gender != 0) {
        score -= 360*college_rank;
    }
    score /= college_rank;
    if (score < 0) {
        if (score == -358) {
            score = (1 + 1) * 1;
        } else if (score > - 358) {
            score += 360;
        } else {
            score += 360*experience;
        }
    }
    return score > T;
}
```

(a) A complicated score-based hiring algorithm.

```
bool credit(int gender, int amount){
    if(gender==0) {
        return (amount <= T);
    } else {
        return (amount > (10-T));
    }
}
```

(b) An unfair credit algorithm.



(c) Fairness Spread for different values of T in Figure 1b

Figure 1: Examples of algorithms that can be examined through information-flow analyses

3. Exemplary Applications

As an exemplary application for information-flow analysis, consider the credit allocation algorithm in Figure 1b which contains a threshold parameter T: Assuming a uniform, independent distribution of gender and amount this program satisfies demographic parity for the parameter T set to $\texttt{T} = 5$, as then both groups are equally likely to obtain a credit. However, the program could still be considered unfair as one group only gets small credits while the other group only gets large credits. Indeed, for $\texttt{T} \in [1, 9]$ this program does not satisfy unconditional noninterference, and we can furthermore compute its Fairness Spread for differing values of T using counterSharp [12] (Figure 1c). This allows us to compare the fairness of different variants of the algorithm. For $\texttt{T} = 0$ and $\texttt{T} = 10$, the program *does* satisfy unconditional noninterference and, thus, demographic parity as well. This can be verified using KeY [7].

One might argue that the program in Figure 1b is an overly simplified version of realistic decision making procedures raising the question to which degree this analysis technique is scalable to larger decision making programs. To this end, it is worth to mention that the qualitative information-flow tool Joana [11, 10] is scalable to systems with up to 100k lines of Java code and the quantitative counterSharp tool has been tested on systems with up to 1000 lines of C code [12]. This is possible due to the use of the approximate model counter ApproxMC [13, 14] which allows the efficient counting of inputs satisfying given constraints even for very large input domains.

As a concrete example, consider the program in Figure 1a: While the previous example showed an obvious case of disparate treatment, this example is much more subtle: Initially, the program does make use of the variables gender and race. However, the computation of the final score is mostly unintelligible, which leaves the influence of the use of `gender` and `race` unclear. Figure 1a admittedly is a fictional example, however, unintelligible and possibly undocumented code is hardly an uncommon phenomenon in practice. For example, it may be the case that an engineer constructed the algorithm who has since left the company without documenting the reasoning behind the computation or the choice of the magic constant `360`. We can learn multiple lessons from this example: First, analyzing whether or not a certain variable has influence on the outcome of a computation is not a trivial problem. In fact, assuming a value range of 1 to 5 for the inputs `college_rank` and `experience`, the function `hire` satisfies unconditional noninterference and could thus be considered a fair algorithm under the definition of demographic parity. While a human will have a hard time seeing this directly from the program code, interactive theorem proving tools can once again be used to formally prove this property in KeY. Secondly, while tools for the information-flow-based analysis of machine-learning systems are currently not available, we can allow for the use of values learned by machine learning systems, e.g, the score threshold `T` which is a constant within the program `hire` (line 20). This allows, for example, to prove a program fair for all possible values of a machine learned threshold `T` and leaves room for a machine learning algorithm to learn the optimal threshold value. Finally, our approach allows the analysis of both single group attributes as well as mixtures of group attributes, e.g., the disparate treatment of individuals with `gender==0 && race == 0`.

4. Future Work

As a next step we plan to analyze larger scale systems to prove or disprove fairness properties of such systems. Our approach currently has two limitations: First, we assume the independence of the protected attribute G and U. To this end, causal models or Bayesian networks may provide a methodology to model important dependencies between G and U within the analyzed code. Moreover, we can only analyze “classical” software currently. Therefore, we plan to develop techniques for analyzing information-flow properties on machine learning models.

References

[1] B. W. Lampson, A note on the confinement problem, Commun. ACM 16 (1973) 613–615. doi:10.1145/362375.362389.

[2] D. E. R. Denning, Secure information flow in computer systems., Ph.D. thesis, Purdue University, 1975.

[3] Á. Darvas, R. Hähnle, D. Sands, A theorem proving approach to analysis of secure information flow, in: D. Hutter, M. Ullmann (Eds.), Security in Pervasive Computing, Second International Conference, SPC 2005, Boppard, Germany, April 6-8, 2005, Proceedings, volume 3450 of *LNCS*, Springer, Cham, 2005, pp. 193–209. doi:10.1007/978-3-540-32004-3_20.

[4] G. Smith, On the foundations of quantitative information flow, in: L. de Alfaro (Ed.), Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of

Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings, volume 5504 of *LNCS*, Springer, Cham, 2009, pp. 288–302. doi:10.1007/978-3-642-00596-1_21.

[5] B. Beckert, D. Bruns, V. Klebanov, C. Scheben, P. H. Schmitt, M. Ulbrich, Information flow in object-oriented software, in: G. Gupta, R. Peña (Eds.), Logic-Based Program Synthesis and Transformation, 23rd International Symposium, LOPSTR 2013, Madrid, Spain, September 18-19, 2013, Revised Selected Papers, volume 8901 of *LNCS*, Springer, Cham, 2013, pp. 19–37. doi:10.1007/978-3-319-14125-1_2.

[6] V. Klebanov, Precise quantitative information flow analysis - a symbolic approach, Theor. Comput. Sci. 538 (2014) 124–139. doi:10.1016/j.tcs.2014.04.022.

[7] W. Ahrendt, B. Beckert, R. Bubel, R. Hähnle, P. H. Schmitt, M. Ulbrich (Eds.), Deductive Software Verification - The KeY Book - From Theory to Practice, volume 10001 of *LNCS*, Springer, Cham, 2016. doi:10.1007/978-3-319-49812-6.

[8] F. Biondi, M. A. Enescu, A. Heuser, A. Legay, K. S. Meel, J. Quilbeuf, Scalable approximation of quantitative information flow in programs, in: I. Dillig, J. Palsberg (Eds.), Verification, Model Checking, and Abstract Interpretation - 19th International Conference, VMCAI 2018, Los Angeles, CA, USA, January 7-9, 2018, Proceedings, volume 10747 of *LNCS*, Springer, Cham, 2018, pp. 71–93. doi:10.1007/978-3-319-73721-8_4.

[9] B. Beckert, M. Kirsten, M. Schefczyk, Algorithmic Fairness and Secure Information Flow (Extended Abstract), in: C. Heitz, C. Hertweck, E. Viganò, M. Loi (Eds.), European Workshop on Algorithmic Fairness (EWAF '22), Lightning round track, 2022. URL: https://sites.google.com/view/ewaf22/accepted-papers.

[10] G. Snelting, D. Giffhorn, J. Graf, C. Hammer, M. Hecker, M. Mohr, D. Wasserrab, Checking probabilistic noninterference using joana, it - Information Technology 56 (2014) 280–287. doi:10.1515/itit-2014-1051.

[11] J. Graf, M. Hecker, M. Mohr, Using joana for information flow control in java programs - a practical guide, in: Proceedings of the 6th Working Conference on Programming Languages (ATPS'13), Lecture Notes in Informatics (LNI) 215, Springer Berlin / Heidelberg, 2013, pp. 123–138.

[12] S. Teuber, A. Weigl, Quantifying software reliability via model-counting, in: A. Abate, A. Marin (Eds.), Quantitative Evaluation of Systems - 18th International Conference, QEST 2021, Paris, France, August 23-27, 2021, Proceedings, volume 12846 of *LNCS*, Springer, Cham, 2021, pp. 59–79. doi:10.1007/978-3-030-85172-9_4.

[13] S. Chakraborty, K. S. Meel, M. Y. Vardi, Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls, in: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, 2016, pp. 3569–3576.

[14] M. Soos, K. S. Meel, BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting, in: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, 2019, pp. 1592–1599. doi:10.1609/aaai.v33i01.33011592.