StatWhy: Formal Verification Tool for Statistical Hypothesis Testing Programs *

Yusuke Kawamoto[1][0000-0002-2151-9560], Kentaro Kobayashi[1,2], and Kohei Suenaga[3][0000-0002-7466-8789] **

[1] AIST, Tokyo, Japan
[2] University of Tsukuba, Ibaraki, Japan
[3] Kyoto University, Kyoto, Japan

Abstract. Statistical methods have been widely misused and misinterpreted in various scientific fields, raising significant concerns about the integrity of scientific research. To mitigate this problem, we propose a tool-assisted method for formally specifying and automatically verifying the correctness of statistical programs. In this method, programmers are required to annotate the source code of the statistical programs with the requirements for these methods. Through this annotation, they are reminded to check the requirements for statistical methods, including those that cannot be formally verified, such as the distribution of the unknown true population. Our software tool StatWhy automatically checks whether programmers have properly specified the requirements for the statistical methods, thereby identifying any missing requirements that need to be addressed. This tool is implemented using the Why3 platform to verify the correctness of OCaml programs that conduct statistical hypothesis testing. We demonstrate how StatWhy can be used to avoid common errors in various statistical hypothesis testing programs.

Keywords: Formal verification · Hypothesis testing · Program verification · Why3 platform.

1 Introduction

Statistical techniques have been essential for acquiring scientific knowledge from data in various academic fields. In particular, an increasing number of researchers have used *statistical hypothesis testing* [2, 11] to derive scientific conclusions from datasets. However, these statistical methods have been widely misused and misinterpreted, raising significant concerns about the integrity of scientific research [7, 24]. For example, the notion of *statistical significance*, assessed by calculating *p-values*, has been widely misused and misinterpreted [32].

For this reason, various guidelines for statistical analyses [25, 31] have been proposed to improve the quality and reproducibility of scientific research. However, owing to the absence of a formal language to describe procedures, we need

* The artifact of the paper is available at `https://github.com/fm4stats/statwhy` and `https://zenodo.org/records/13991312`.

** The authors are listed in alphabetical order.

to manually refer to these guidelines, written in natural language. As a result, the correctness of statistical analyses has not been checked automatically.

To mitigate these problems, we propose a new method for the formal specification and automatic verification of statistical program correctness. Specifically, programmers are required to annotate their source code with the requirements for the statistical methods and the interpretations of the analysis results. Then, our tool StatWhy automatically checks whether these requirements and interpretations are correctly annotated. For example, StatWhy can verify whether a p-value is correctly calculated in a program, thus preventing p-value hacking, i.e., a technique to manipulate statistical analyses to obtain a lower p-value.

The goal of StatWhy stems from the nature of statistics: many requirements for statistical methods cannot be proven mathematically because they are usually properties of an unknown true population that analysts seek to estimate from sampled data. For example, many statistical hypothesis testing methods require a population to follow a normal distribution. Since analysts cannot prove this requirement mathematically, they are responsible for judging whether the population appears to follow a normal distribution, possibly using their background knowledge about the population. For this reason, StatWhy asks analysts to explicitly write down the requirements for statistical methods—typically, the assumptions that they make about the population distributions— as an annotation in their source code. Then, the analysts are reminded to check these requirements empirically and approximately using their background knowledge.

To design StatWhy, we use the framework of belief Hoare logic (BHL) [17,19] and provide constructs to make writing statistical programs easier, as well as libraries for the specification of various hypothesis testing methods. For the implementation of this tool, we rely on the Why3 platform [8] to handle practical programming languages and to automatically discharge verification conditions using external SMT solvers.

Although the current implementation of StatWhy focuses on statistical hypothesis testing, the approach is not limited to a specific branch of statistics. Rather, it can be applied to any situation where the usage of statistical methods in programs needs to be checked. In future versions of the tool, we will include additional statistical methods beyond hypothesis testing.

Contributions. Our main contributions are summarized as follows:

1. We propose a new tool-assisted method for formally specifying and automatically verifying statistical programs (i.e., programs that perform hypothesis testing and calculate statistics under certain assumptions about an unknown population). This method requires programmers to annotate their source code with the requirements and the interpretations of statistical analyses, which makes a statistical procedure verifiable and explainable.
2. We implemented a software tool StatWhy based on our verification method. Given a program as input, StatWhy automatically verifies whether a programmer has correctly annotated it with the requirements for statistical hypothesis testing and the interpretation of the test results. StatWhy is available with a range of examples and comprehensive documentation [16].

3. We demonstrate how StatWhy can be used to avoid common errors in various popular statistical hypothesis testing methods.

To the best of our knowledge, StatWhy appears to be the first tool to automatically verify the requirements for the appropriate use of hypothesis tests. This work represents the first step in building a framework for specifying and verifying the integrity of scientific conclusions based on statistical analyses.

Related Work. *Logic for Statistics.* Several studies on modal logic have been proposed to express statistical properties [13, 18]. The work on statistical epistemic logic (StatEL) [13–15] is the first attempt to construct a modal logic to describe statistical properties of hypothesis testing. They introduce a belief modality weaker than S5 and interpret it in a Kripke model where an accessibility relation is defined using a statistical distance between possible worlds. However, these logics cannot reason about the procedures of statistical methods.

Belief Hoare Logic. Belief Hoare logic (BHL) [17, 19] is a program logic with an epistemic modality for statistical belief. Using this logic, we can derive the correctness of a hypothesis testing program (Sect. 2). Our verification tool, StatWhy, uses BHL as its theoretical foundation to produce a proof tree for the correctness of hypothesis testing programs within the tool. To develop StatWhy, we implemented several constructs specific to BHL using the WhyML language—the intermediate language used within Why3 framework. This allows verification conditions generated by StatWhy to be discharged by off-the-shelf SMT solvers.

Program Verification Tools. Various tools used for specifying the preconditions and postconditions of each function and statically verifying their correctness; for example, Frama-C [6] for C programs; Dafny [22] for imperative programs that compile to Boogie [21]; ESC/Java [9] and KeY [1] for Java programs. To the best of our knowledge, no attempt has been made to apply these tools for verifying the correct usage of statistical methods in programs.

The Verification Frameworks Used in Our Tool. StatWhy is implemented as an extension of Cameleer [29], a verifier for OCaml programs built on top of the Why3 framework [8]. Cameleer works as a translator from OCaml to the simple functional programming language WhyML. The translated WhyML code is then verified by Why3. StatWhy extends Cameleer to verify the correctness of hypothesis testing programs written in OCaml by incorporating the constructs and inference rules of BHL.

Verification of Statistical Algorithms. From a broader perspective, a number of studies have investigated the numerical accuracy of statistical algorithms [23], the formal verification of randomized algorithms [20, 26], and the PAC verification [10, 28] for approximately checking the correctness of statistical algorithms. Furthermore, formal methods have been used to verify a certain guarantee of the correctness of statistical machine learning models [30]. However, no prior work has provided a formal method tool for specifying or verifying the correct usage of statistical methods rather than the correctness of statistical algorithms.

Plan of the paper. The rest of the paper is organized as follows. In Sect. 2, we review basic notions in hypothesis testing and belief Hoare logic (BHL). In Sects. 3 and 4, we present StatWhy's design and implementation, respectively.

In Sect. 5, we show examples of StatWhy being applied to common errors in hypothesis testing. In Sect. 6, we present our final remarks.

2 Background

Statistical hypothesis testing [2, 11] is a method for inferring information about an unknown population x from a dataset y that has been sampled from the population x. In a hypothesis test, the *null hypothesis* φ_0 is a claim that we wish to test, while the *alternative hypothesis* φ_1 is a claim that we will accept if the null hypothesis is rejected. The goal of a hypothesis test is to determine whether we have sufficient evidence to reject the null hypothesis φ_0 in favor of the alternative hypothesis φ_1.

Example 1 (t-test for a population mean). For a population x following a normal distribution with an unknown mean, the *t-test for the population mean* is a hypothesis test to check whether the unknown mean $\mathsf{mean}(x)$ differs from a certain value μ_0 specified in the null hypothesis.

In the t-test, we want to show the alternative hypothesis $\varphi_1 \stackrel{\text{def}}{=} (\mathsf{mean}(x) \neq \mu_0)$ by rejecting the null hypothesis $\varphi_0 \stackrel{\text{def}}{=} (\mathsf{mean}(x) = \mu_0)$. First, we calculate the *t-test statistic* from a dataset y: $t(y) := \frac{\mathsf{mean}(y)-\mu_0}{s/\sqrt{n}}$ where n is the sample size of y and s is a sampled standard deviation, i.e., $s \stackrel{\text{def}}{=} \sqrt{\frac{\sum_{i=1}^{n}(y_i-\mathsf{mean}(y))^2}{n-1}}$. This statistic is compared to Student's t-distribution with $n-1$ degrees of freedom (i.e., the distribution of the t-statistic $t(y)$ when y is normally distributed). Specifically, we calculate the *p-value*: $\Pr_{d\sim N(\mu,\sigma^2)^n}[\,|t(d)| > |t(y)|\,]$ under the null hypothesis φ_0. For a smaller p-value, the dataset y is far from what we expect under the null hypothesis that $\mathsf{mean}(x) = \mu_0$. Hence, in the t-test, if the p-value is smaller than a certain threshold (e.g., 0.05), we reject the null hypothesis φ_0 and accept the alternative hypothesis φ_1, i.e., $\mathsf{mean}(x) \neq \mu_0$.

We remark that this t-test requires that the population x should follow a normal distribution. If this requirement is not satisfied, the use of the t-test is inappropriate and its result may be incorrect. Therefore, analysts need to check this requirement in some way. Since they cannot mathematically prove this requirement on the unknown population x, they use their background knowledge about the population x and check approximately whether the dataset y at hand (rather than the unknown population x itself) follows a normal distribution.

Belief Hoare logic (*BHL*) [17, 19] is a program logic equipped with epistemic modal operators for the statistical beliefs acquired via hypothesis testing. The epistemic modal logic used in BHL is defined by:

$$\varphi ::= \eta(u_1, \ldots, u_n) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{K}\varphi \mid \mathbf{K}^{\leq\epsilon}_{y,A}\,\varphi$$

for a predicate symbol η, terms $u_1, \ldots, u_n$, a dataset y, a hypothesis test A, and a p-value ϵ. The knowledge modality $\mathbf{K}$ is defined in the S5 modal logic system with axioms T, 4, and 5. Intuitively, $\mathbf{K}\varphi$ represents that we know φ. The epistemic possibility $\mathbf{P}$ is defined as usual by $\mathbf{P}\varphi \stackrel{\text{def}}{=} \neg\mathbf{K}\neg\varphi$. $\mathbf{K}^{\leq\epsilon}_{y,A}\,\varphi$ represents that by a hypothesis test A on a dataset y, we believe φ with a p-value $\alpha \leq \epsilon$.

The semantics of this epistemic logic is based on a Kripke model [17,19]. The satisfaction of an epistemic formula φ in a possible world w is denoted by $w \models \varphi$ and is defined straightforwardly in a Kripke model where each possible world is equipped with a test history that is updated by performing hypothesis tests.

In the framework of BHL, we express a procedure for hypothesis testing as a program C using a programming language. Then, we use epistemic modal logic to describe the requirements for the hypothesis tests as a precondition formula, e.g., $\psi_{\mathsf{pre}} \stackrel{\text{def}}{=} y \leftsquigarrow N(\mu_0, \sigma^2) \wedge \mathbf{P}\varphi_1 \wedge \kappa_\emptyset$, where the atomic formula $y \leftsquigarrow N(\mu, \sigma^2)$ represents that a dataset y is sampled from the population that follows a normal distribution $N(\mu, \sigma^2)$ with an unknown mean μ and an unknown variance σ^2. The modal formula $\mathbf{P}\varphi$ represents that, before conducting the hypothesis test, we have the *prior belief* that the alternative hypothesis φ *may be* true. The formula $\kappa_\emptyset$ represents that no hypothesis test has been conducted previously.

The statistical belief we acquire from the hypothesis test is specified as a *postcondition* formula, e.g., $\varphi_{\mathsf{post}} \stackrel{\text{def}}{=} \mathbf{K}^{\leq 0.05}_{y,A}\varphi_1$, representing that by a hypothesis test A on the dataset y, we believe φ with a p-value $\alpha \leq 0.05$. Since the result of the hypothesis test may be wrong, we use the statistical belief modality $\mathbf{K}^{\leq 0.05}_{y,A}$ instead of the knowledge modality $\mathbf{K}$.

Finally, we combine all the above and describe the whole statistical inference as a *judgment* $\Gamma \vdash \{\psi_{\mathsf{pre}}\}\ C\ \{\varphi_{\mathsf{post}}\}$, representing that whenever the precondition ψ_{pre} is satisfied, the execution of the program C results in the satisfaction of the postcondition φ_{post}. By deriving this judgment using derivation rules in BHL, we conclude that the program C for the statistical inference results in the statistical belief φ_{post} whenever the requirement ψ_{pre} is satisfied.

BHL has been applied only to pen-and-paper analyses of a few simple examples of statistical hypothesis testing in previous work [17,19] and has not yet been used to build a computer-aided verification tool.

3 The Design of StatWhy

3.1 Running Examples

```
let p = exec_ttest_1samp t_n 1.0 d Two
(*@ requires  sampled d t_n
    ensures  (World (!st) interp) |= StatB p fmlA  *)
```

Fig. 1: An OCaml program that calls a t-test command for a mean of a population.

Simple Example. We present the main idea of our formal specification and automated verification method using the program in Fig. 1. This program shows an OCaml source code that executes a command `exec_ttest_1samp` for the one-sample t-test (Example 1 in Sect. 2) with an alternative hypothesis `fmlA` (e.g., representing `mean(t_n) != 1.0`).

To specify the requirements and the interpretation of this t-test command, a programmer writes the precondition in the `requires` clause and the postcondition in the `ensures` clause using the specification language Gospel [5].

```
val exec_ttest_1samp (p:population) (mu:real) (y:dataset (list real)) (alt:alternative):real
 ...
 requires {
  match p with
  | (NormalD _ _) -> sampled y p /\ ... /\
    match alt with
    | Two -> (World !st interp) |= Possible ((mean p) <' (const_term mu)) /\
             (World !st interp) |= Possible ((mean p) >' (const_term mu)) ...
 }
 ensures {
  pvalue result /\
  let h = match alt with
    | Two -> (mean p) $!= (const_term mu) ...
  end in !st = Cons ("ttest_1samp", h, Eq result) !(old st)
 }
```

Fig. 2: The specification of `exec_ttest_1samp`

In this code, `sampled` is a predicate defined in WhyML, and the precondition `sampled d t_n` expresses that the dataset `d` has been sampled from a population that has a normal distribution type `t_n` with an unknown mean and an unknown variance. In the WhyML implementation of BHL, a dataset is implemented as a record with a field storing the distribution type of the population.

The postcondition `(World (!st) interp) |= StatB p fmlA` represents the interpretation of the result of the hypothesis test. Specifically, we obtain a *statistical belief*—denoted by the logical formula `StatB p fmlA`—that an alternative hypothesis `fmlA` holds with a p-value `p`, in the possible world `(World (!st) interp)` equipped with the record `st` of all hypothesis tests executed so far. This postcondition encodes the satisfaction of the epistemic formula $w \models \mathbf{K}^{\leq \mathtt{p}}_{y,A}\, \mathtt{fmlA}$ in the world w in the Kripke model addressed in Sect. 2.

By applying StatWhy to this source code, the program verification fails because other requirements are missing in the precondition. However, since Why3 finds the failure to discharge the verification conditions corresponding to such requirements, the tool user can easily find out the missing requirements.

We remark that the specification of `exec_ttest_1samp` is defined in StatWhy using WhyML so that it (1) checks the requirements for the hypothesis test in the precondition, (2) asserts the conclusion implied by the hypothesis test in the postcondition, and (3) updates the test history `st` with the test result (Fig. 2).

In the `requires`-clause, the two-tailed (`TWO`) t-test requires the prior belief that both tails $\mathsf{mean}(p) < \mu$ and $\mathsf{mean}(p) > \mu$ are possible. In the `ensures`-clause, the test result consisting of the test name, the hypothesis `h`, and the p-value `result` is added to the test history `st`. This specification of the two-tailed t-test encodes an instance of the following inference rule in BHL [17, 19]:

$$\frac{\Gamma \models \psi \rightarrow (\varpi \wedge \mathbf{P}\varphi_{\mathsf{L}} \wedge \mathbf{P}\varphi_{\mathsf{U}})}{\Gamma \vdash \{\psi \wedge \kappa_{\emptyset}\}\ \alpha := f_A(y)\ \{\psi \wedge \kappa_{y,A} \wedge \mathbf{K}^{\alpha}_{y,A}(\varphi_{\mathsf{L}} \vee \varphi_{\mathsf{U}})\}} \quad \text{(Two-HT)}$$

where the precondition ψ includes the assumption ϖ on the population distribution and the prior beliefs on the two tails $\mathbf{P}\varphi_{\mathsf{L}}$ and $\mathbf{P}\varphi_{\mathsf{U}}$; the postcondition updates the empty test history $\kappa_{\emptyset}$ to the history $\kappa_{y,A}$ recording the test result.

```
let ex_hack trial1 trial2 =
  let p1 = exec_ttest_1samp ppl_new 1.0 trial1 Two in
  let p2 = exec_ttest_1samp ppl_new 1.0 trial2 Two in
  let p = min p1 p2 in (* This is INCORRECT *)
  (p1, p2, p)
(*@ (p1, p2, p) = ex_hack trial1 trial2
  requires
    is_empty (!st) /\ sampled trial1 ppl_new /\ sampled trial2 ppl_new /\
    (World (!st) interp) |= Possible h_new_l /\ (World (!st) interp) |= Possible h_new_u
  ensures
    (Leq p = compose_pvs h_new !st
      && (World !st interp) |= StatB (Leq p) h_new) *)
```

Fig. 3: An OCaml program that performs the p-value hacking.

***P*-Value Hacking.** Fig. 3 is another example presenting how StatWhy detects an error in the code conducting the *p-value hacking* (a.k.a. *data dredging*), a technique to manipulate statistical analyses to obtain a lower p-value.

In this program, we execute a t-test `exec_ttest_1samp` on a dataset `trial1` and another on another dataset `trial2`. Given the p-values `p1` and `p2` for these two t-tests, we should report `p1` + `p2` as the p-value of these experiments. However, this program reports only the experiment showing the lower p-value (i.e., `min p1 p2`) by ignoring the other showing higher p-value. By ensuring that all hypothesis tests are described in the program, StatWhy can automatically check whether the p-values are correctly calculated, thus preventing p-value hacking.

We remark that in the precondition, the atomic expression `is_empty (!st)` with the dereference operator '`!`' represents that the test history `st` is empty; i.e., no hypothesis test has performed before. The expression `sampled trial1 ppl_new` represents that the dataset `trial1` is sampled from a population `ppl_new`. For specific predicates such as `is_empty` and `sampled`, we can use abbreviated expressions where "`(World !st interp) |= `" is omitted for the sake of simplicity. In the postcondition, `compose_pvs h_new !st` obtains the correct p-value from the test history `st`, which is found to be different from the p-value `p` incorrectly calculated in this program.

3.2 More Details on Specifications

The latest version of StatWhy can automatically verify the correctness of specifications written in the WhyML language [8]. It can also verify programs written in the subset of OCaml supported by Cameleer [29], a verifier for OCaml programs. Specifically, Cameleer covers the core OCaml language except for several features, including object-oriented programming, generalized algebraic data types (GADTs), and polymorphic variants.

StatWhy requires minimal modifications to the source code. Programmers need to insert annotations into the OCaml program to describe the requirements and interpretations for hypothesis testing. More specifically, a requirement (respectively, interpretation) for a hypothesis testing command is expressed as a logical formula written in the Gospel language [5], representing a precondition (respectively, postcondition) for the command.

```
use ocamlstdlib.Stdlib
let function p =
  [@vc:white_box]
  (begin
     requires { sampled d t_n }
     returns { p -> (World (!st) interp) |= (StatB p fmlA) }
     exec_ttest_1samp t_n 1.0 d Two
   end)
```

Fig. 4: A WhyML program calling a t-test command for a mean of a population.

To describe these annotations in Gospel, we introduce types for *terms*, *atomic formulas*, and *logical formulas* of belief Hoare logic (BHL) as follows.

```
type term = RealT real_term | PopulationT population | ...
type atomic_formula = Pred psymb (list term)
type formula = Atom atomic_formula | Not formula
             | Conj formula formula | Disj formula formula
             | Possible formula | Know formula | StatB pvalue formula | ...
```

where a term can express a real number and a population; an atomic formula consists of a predicate symbol (e.g., `is_normal`) and a list of terms; a formula is built using `Possible`, `Know`, and `StatB`, each corresponding to the modal epistemic operators $\mathbf{P}$, $\mathbf{K}$, and $\mathbf{K}^{\leq\epsilon}_{y,A}$, respectively, shown in Sect. 2.

In the WhyML grammar, an atomic expression is of the form `World (!st) interp |= formula` representing that the formula `formula` is satisfied in the possible world equipped with the test history `st` (i.e., the record of all hypothesis tests executed so far) and the interpretation `interp` of private-variables in the Kripke model for BHL [17, 19]. For the non-modal formulas using only specific predicates (e.g., `is_empty` or `sampled`), we allow an abbreviation that can omit "`World (!st) interp |=`" from an expression. We can also use function symbols (e.g., mean and `ppl`) to simplify expressions.

For clarity in hypothesis testing specifications, programmers can use abbreviation operators. Since hypothesis testing programs often involve repeated comparisons among multiple groups of data, StatWhy provides a set of folding operations to simplify the repetition of similar conditions in specifications. In particular, folding operators can be used to briefly describe the iteration of the hypothesis tests that compare all combinations of groups in multiple comparison.

3.3 Verification of Statistical Programs

To verify a given OCaml program, StatWhy first transforms it into a program written in the WhyML language [8]. This preprocessing is performed using our extension of Cameleer [29], a static verifier for OCaml. For example, the OCaml program in Fig. 1 is transformed into the WhyML program in Fig. 4.

Next, StatWhy proves the correctness of a WhyML program using the Why3 platform [8]. Specifically, the tool internally synthesizes a proof tree using the proof rules of Belief Hoare logic and derives the verification condition for the program. Then, it automatically discharges these conditions by using external SMT solvers, e.g., CVC5 [3] or Z3 [27]. If the verification succeeds, StatWhy outputs a proof tree that attests to the correctness of the program. If the verification fails or times out, the tool reports the failure. Even in that case, the tool users can identify any missing or incorrect requirements and interpretations for statistical methods so that they can re-specify the requirements and interpretations..

The verification process using StatWhy guarantees the following correctness. If StatWhy successfully verifies a program, for any function `f` defined as `let f d1 ... dn = e`, annotated with a precondition ψ_{pre} and a postcondition φ_{post}, the judgment $\{\psi_{\mathsf{pre}}\}\ [v_1/\mathtt{d1}, \ldots, v_n/\mathtt{dn}]\,\mathtt{e}\ \{\varphi_{\mathsf{post}}\}$ holds for any value v_i of type $\mathtt{d}i$, assuming the soundness of the Why3 framework. By the soundness of BHL, if the expression `e` is evaluated under the environment satisfying the precondition ψ_{pre}, then the resulting environment satisfies the postcondition φ_{post}.

We remark that StatWhy focuses on automatically verifying the *procedure* and the *annotations* in a statistical program by assuming the correctness of the implementation of each hypothesis testing method used in the procedure as a subroutine. In other words, our automated verification tool only checks that a program correctly uses hypothesis testing functions. Technically, StatWhy checks whether the preconditions and the postconditions of hypothesis testing functions are satisfied when the program calls these functions as subroutines. By using StatWhy, programmers are encouraged to pay attention to the requirements, the interpretations, and the choices of hypothesis testing methods.

We also remark that StatWhy verifies a statistical program only under the assumption that all requirements about an unknown true population are satisfied (e.g., a population follows a normal distribution). Thus, such an assumption needs to be checked externally; for instance, the analysts are responsible for judging whether the population appears to approximately follow the normal distribution[4], possibly using background knowledge about the population.

4 The Implementation of StatWhy

In this section, we explain and discuss the implementation of the StatWhy tool. More details on the tool is available in the user documentation [16].

4.1 The Architecture of StatWhy

Fig. 5 shows the architecture of StatWhy. To specify the requirements and interpretations of hypothesis tests as preconditions and postconditions, StatWhy has modules for real number arithmetic, basic statistics, and individual hypothesis testing commands (e.g., t-test) that cover most of the popular hypothesis testing methods [12]. To reason about the interpretation of hypothesis testing, the tool also has modules for BHL [17, 19], an epistemic logic with statistical belief explained in Sect. 2, and for the record of the hypothesis tests performed so far.

[4] There are hypothesis testing methods for checking the normality approximately. Such tests are applied to the *dataset* (instead of the actual *population*) and cannot prove the normality of the population mathematically.

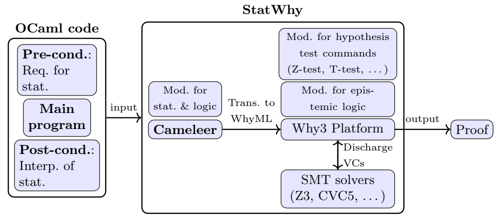

Fig. 5: The architecture of the StatWhy tool.

Since the goal of our verification tool is to ensure the correct usage of the hypothesis testing methods in programs, the StatWhy tool reasons about p-values in hypothesis testing, which requires basic reasoning about probabilities. Specifically, we extended Cameleer so that it can access the real number arithmetic formalized in Why3 standard library, and added basic statistics functions (e.g., mean) and their lemmas. To reason about p-values appearing in epistemic formulas in simplifying verification conditions, we added Why3 lemmas about the composition of p-values under multiple-tests and about the comparisons between p-values. With these lemmas in the StatWhytool, we avoid the need for SMT solvers to handle probability computations, while maintaining the soundness and correctness in the statistical context.

To accept OCaml programs as input, StatWhy internally calls our extension of Cameleer [29] to translate an OCaml program to a WhyML program. Then the tool verifies a WhyML program by using the Why3 platform.

4.2 Extension of Cameleer

The programs given to StatWhy share several peculiar features. For example, StatWhy specifications often involve repeated comparisons among multiple groups of data, which are expressed using folding operations. Furthermore, the programs often involve the array structure of records of hypothesis tests executed so far. We have found that SMT solvers often get stuck if we try to discharge the verification conditions that involve such folding operations.

To improve the performance of StatWhy, we implemented a custom proof strategy—a combination of proof tactics and transformations—that exploits these characteristics of hypothesis testing programs to accelerate the proof search. Our strategy first applies Why3's default proof strategies (e.g., `split_vc` for splitting conjunctive verification conditions into smaller ones and `compute_in_goal` for applying computations and simplifications to proof goals). These invocations of the proof strategies are interleaved with calls to SMT solvers, whose timeouts are set to small values. If these applications of the default proof strategies fail to discharge the VCs, then we apply aggressive transformation strategies that unfold the definitions of the functions and predicates defined in StatWhy.

These extensions are implemented as follows. We added WhyML modules for StatWhy to the standard library of Why3 at the installation of our extension of Cameleer. We also extended Cameleer so that `Uterm`, the module for OCaml untyped terms, and `Why3ocaml_driver`, the module for translation from OCaml to WhyML, can handle floating-point numbers. We also extended the Why3 plugins provided by Cameleer by adding `plugin/cameleerBHL.ml` and modifying `plugin/plugin_cameleer.ml` to handle the algebraic data types in StatWhy.

```
let cmp_with_existing_drugs d_new d_drug1 d_drug2 =
  let p_drug1 = exec_ttest_ind_eq ppl_new ppl_drug1 (d_new, d_drug1) Up in
  let p_drug2 = exec_ttest_ind_eq ppl_new ppl_drug2 (d_new, d_drug2) Up in
  p_drug1 +. p_drug2
 (*@ p = testing d_new d_drug1 d_drug2
    requires
      is_empty (!st) /\ non_paired d_new d_drug1 /\ non_paired d_new d_drug2 /\
      sampled d_new ppl_new /\ sampled d_drug1 ppl_drug1 /\ sampled d_drug2 ppl_drug2 /\
      (World (!st) interp) |= Possible h_new_drug1 /\
      (World (!st) interp) |= Not (Possible h_new_drug1_c) /\
      (World (!st) interp) |= Possible h_new_drug2 /\
      (World (!st) interp) |= Not (Possible h_new_drug2_c)
    ensures
      (Leq p) = compose_pvs (Disj h_new_drug1 h_new_drug2) !st &&
      (World !st interp) |= StatB (Leq p) (Disj h_new_drug1 h_new_drug2) *)
```

Fig. 6: An OCaml program that performs multiple comparison.

Table 1: The execution times (sec) for verifying hypothesis testing programs with practical numbers of disjunctive (OR) and conjunctive (AND) hypotheses.

#hypotheses	2	3	4	5	6	7	8	9	10
OR	8.77	8.89	8.84	8.94	9.01	9.01	9.04	9.16	9.23
AND	8.82	8.72	8.86	8.98	8.95	9.03	9.11	9.17	9.46

5 Case Studies on Common Errors in Hypothesis Testing

We present examples to demonstrate how StatWhy can be used to avoid common errors in a variety of popular hypothesis testing programs.

Multiple Comparison Problems. Analysts occasionally make mistakes in computing the p-value in comparing more than two groups, which is called a *multiple comparison problem* [4].

Let us consider an experiment comparing the efficacy of a new drug with that of two existing drugs `drug1` and `drug2`. We conduct two one-tailed t-tests: one comparing the new drug with `drug1` and another with `drug2`. Then the alternative hypotheses `h_new_drug1` and `h_new_drug2` for these tests represent that the new drug has a better efficacy than `drug1` and `drug2`, respectively. For the p-values p_1 and p_2 of these two tests, the p-value p for the combined test with the disjunctive hypothesis `h_new_drug1` $\vee$ `h_new_drug2` satisfies $p \leq p_1 + p_2$, which is known as *Bonferroni correction*. In contrast, the p-value p for the conjunctive hypothesis `h_new_drug1`$\wedge$`h_new_drug2` satisfies $p \leq \min(p_1, p_2)$.

StatWhy can automatically check that the program in Fig. 6 correctly calculates the p-value of the disjunctive hypothesis `Disj h_new_drug1 h_new_drug2`.

Scalability of StatWhy We evaluated the scalability of the performance of the program verification using StatWhy to (i) the complex hypotheses and (ii) the larger number of compared groups. For the evaluation, we conducted experiments on a MacBook Pro with Apple M2 Max CPU and 96 GB memory using the external SMT solver CVC5 1.0.6.

Table 2: The execution times (sec) for various multiple comparison methods. #groups (respectively, #comparisons) represents the (practical) number of groups (respectively, combinations of groups) compared in the testing.

		#groups					
Test methods	Metric	2	3	4	5	6	7
Tukey's HSD test	Times (sec)	0.37	9.09	9.33	9.81	15.27	16.39
	#comparisons	1	3	6	10	15	21
Dunnett's test	Times (sec)	0.48	8.98	9.17	9.61	9.62	9.77
	#comparisons	1	2	3	4	5	6
Williams' test	Times (sec)	0.48	8.90	9.04	9.16	9.23	9.58
	#comparisons	1	2	3	4	5	6
Steel-Dwass' test	Times (sec)	0.44	9.05	9.43	9.76	15.10	16.24
	#comparisons	1	3	6	10	15	21
Steel's test	Times (sec)	0.49	8.79	8.92	9.11	9.43	9.74
	#comparisons	1	2	3	4	5	6

Table 1 shows the execution times for StatWhy to verify hypothesis testing programs for practical numbers of disjunctive/conjunctive hypotheses. These experiments took roughly the same amount of time for a larger number of hypotheses. Table 2 provides the execution times for the most common multiple comparison methods described in standard textbooks. The numbers of groups compared in the experiments are practical but challenging, as the number of comparisons grows rapidly with the number of groups. The verification of these programs is efficient, since our proof strategy (Sect. 4) accelerates the proof search for programs with folding operations and test histories.

6 Conclusion

We proposed a tool-assisted method for formally specifying and automatically verifying the correctness of statistical programs. In particular, we presented the StatWhy tool for automatically checking whether the programmers have properly specified the requirements and the interpretations of the statistical methods.

In future work, we will extend StatWhy to verify the procedures for power analyses and interval estimation and to deal with other types of statistical methods and other programming languages, e.g., a subset of Python. We also plan to work on the formal verification of the correctness of the implementation of each hypothesis testing function called from statistical software as a subroutine.

Acknowledgments. The authors are supported by JSPS KAKENHI Grant Number JP24K02924, Japan. Yusuke Kawamoto is supported by JST, PRESTO Grant Number JPMJPR2022, Japan. Kohei Suenaga is supported by JST CREST Grant Number JPMJCR2012, Japan.

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

References

1. Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M.: Deductive Software Verification - The KeY Book - From Theory to Practice, Lecture Notes in Computer Science, vol. 10001. Springer (2016). https://doi.org/10.1007/978-3-319-49812-6
2. Arbuthnot, J.: An argument for divine providence, taken from the constant regularity observ'd in the births of both sexes. Philosophical Transactions of the Royal Society of London **27**(328), 186–190 (1710). https://doi.org/10.1098/rstl.1710.0011
3. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: Proc. the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2022), Part I. Lecture Notes in Computer Science, vol. 13243, pp. 415–442. Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_24
4. Bretz, F., Hothorn, T., Westfall, P.: Multiple Comparisons Using R. Chapman and Hall/CRC (2010). https://doi.org/10.1201/9781420010909
5. Charguéraud, A., Filliâtre, J., Lourenço, C., Pereira, M.: GOSPEL - providing ocaml with a formal specification language. In: Proc. of the 24th International Symposium on Formal Methods (FM 2019). Lecture Notes in Computer Science, vol. 11800, pp. 484–501. Springer (2019). https://doi.org/10.1007/978-3-030-30942-8_29
6. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C - A software analysis perspective. In: Proc. 10th International Conference on Software Engineering and Formal Methods (SEFM 2012). Lecture Notes in Computer Science, vol. 7504, pp. 233–247. Springer (2012). https://doi.org/10.1007/978-3-642-33826-7_16
7. Fernandes-Taylor, S., Hyun, J.K., Reeder, R.N., Harris, A.H.: Common statistical and research design problems in manuscripts submitted to high-impact medical journals. BMC Research Notes **4**(1), 304 (2011). https://doi.org/10.1186/1756-0500-4-304
8. Filliâtre, J., Paskevich, A.: Why3 - where programs meet provers. In: Proc. of the 22nd European Symposium on Programming (ESOP 2013). Lecture Notes in Computer Science, vol. 7792, pp. 125–128. Springer (2013). https://doi.org/10.1007/978-3-642-37036-6_8
9. Flanagan, C., Leino, K.R.M., Lillibridge, M., Nelson, G., Saxe, J.B., Stata, R.: PLDI 2002: Extended static checking for Java. ACM SIGPLAN Notices **48**(4S), 22–33 (2013). https://doi.org/10.1145/2502508.2502520
10. Goldwasser, S., Rothblum, G.N., Shafer, J., Yehudayoff, A.: Interactive Proofs for Verifying Machine Learning. In: Lee, J.R. (ed.) Proc. of the 12th Innovations in Theoretical Computer Science Conference (ITCS 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 185, pp. 41:1–41:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.ITCS.2021.41
11. Hogg, R.V., McKean, J.W., Craig, A.T.: Introduction to mathematical statistics. Pearson Education India (2013)
12. Kanji, G.K.: 100 statistical tests. Sage (2006)
13. Kawamoto, Y.: Statistical epistemic logic. In: The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Pri-

vacy. LNCS, vol. 11760, pp. 344–362. Springer (2019). https://doi.org/10.1007/978-3-030-31175-9_20
14. Kawamoto, Y.: Towards logical specification of statistical machine learning. In: Proc. SEFM 2019. pp. 293–311 (2019). https://doi.org/10.1007/978-3-030-30446-1_16
15. Kawamoto, Y.: An epistemic approach to the formal specification of statistical machine learning. Software and Systems Modeling **20**(2), 293–310 (2020). https://doi.org/10.1007/s10270-020-00825-2
16. Kawamoto, Y., Kobayashi, K., Suenaga, K.: User Documentation for StatWhy v.1.2 (2025), available at https://github.com/fm4stats/statwhy
17. Kawamoto, Y., Sato, T., Suenaga, K.: Formalizing statistical beliefs in hypothesis testing using program logic. In: Proc. KR'21. pp. 411–421 (2021). https://doi.org/10.24963/kr.2021/39
18. Kawamoto, Y., Sato, T., Suenaga, K.: Formalizing statistical causality via modal logic. In: Proc. JELIA'23. Lecture Notes in Computer Science, vol. 14281, pp. 681–696. Springer (2023). https://doi.org/10.1007/978-3-031-43619-2_46
19. Kawamoto, Y., Sato, T., Suenaga, K.: Sound and relatively complete belief Hoare logic for statistical hypothesis testing programs. Artif. Intell. **326**, 104045 (2024). https://doi.org/10.1016/J.ARTINT.2023.104045
20. Kozen, D.: A probabilistic PDL. J. Comput. Syst. Sci. **30**(2), 162–178 (1985). https://doi.org/10.1016/0022-0000(85)90012-1
21. Leino, K.R.M.: This is Boogie 2. manuscript KRML **178**(131), 9 (2008)
22. Leino, K.R.M.: Dafny: An automatic program verifier for functional correctness. In: Proc. the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'16). Lecture Notes in Computer Science, vol. 6355, pp. 348–370. Springer (2010). https://doi.org/10.1007/978-3-642-17511-4_20
23. Lesage, J.P., Simon, S.D.: Numerical accuracy of statistical algorithms for microcomputers. Computational Statistics & Data Analysis **3**, 47–57 (1985). https://doi.org/https://doi.org/10.1016/0167-9473(85)90057-X
24. Makin, T.R., de Xivry, J.J.O.: Science forum: Ten common statistical mistakes to watch out for when writing or reviewing a manuscript. Elife **8**, e48175 (2019)
25. Moher, D., Hopewell, S., Schulz, K.F., Montori, V., Gøtzsche, P.C., Devereaux, P., Elbourne, D., Egger, M., Altman, D.G.: Consort 2010 explanation and elaboration: updated guidelines for reporting parallel group randomised trials. International journal of surgery **10**(1), 28–55 (2012)
26. Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Trans. Program. Lang. Syst. **18**(3), 325–353 (1996). https://doi.org/10.1145/229542.229547
27. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Proc. the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008). Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
28. Mutreja, S., Shafer, J.: Pac verification of statistical algorithms. In: Neu, G., Rosasco, L. (eds.) Proceedings of Thirty Sixth Conference on Learning Theory. Proceedings of Machine Learning Research, vol. 195, pp. 5021–5043. PMLR (2023)
29. Pereira, M., Ravara, A.: Cameleer: A deductive verification tool for OCaml. In: Proc. of the 33rd International Conference on Computer Aided Verification (CAV 2021), Part II. Lecture Notes in Computer Science, vol. 12760, pp. 677–689. Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_31
30. Seshia, S.A., Sadigh, D., Sastry, S.S.: Toward verified artificial intelligence. Commun. ACM **65**(7), 46–55 (2022). https://doi.org/10.1145/3503914

31. Von Elm, E., Altman, D.G., Egger, M., Pocock, S.J., Gøtzsche, P.C., Vandenbroucke, J.P.: The strengthening the reporting of observational studies in epidemiology (strobe) statement: guidelines for reporting observational studies. Bulletin of the World Health Organization **85**, 867–872 (2007)
32. Wasserstein, R.L., Lazar, N.A.: The ASA statement on p-values: Context, process, and purpose. The American Statistician **70**(2), 129–133 (2016). https://doi.org/10.1080/00031305.2016.1154108