The PL-Detective Revisited

Christoph Reichenbach
Dept. of Computer Science
Lund University
Lund, Sweden
christoph.reichenbach@cs.lth.se

Abstract

The semantics of programming languages comprise many concepts that are alternatives to each other, such as by-reference and by-value parameter passing. To help teach these concepts, Diwan et al. introduced the programming language Mystery, with fixed syntax but configurable semantics, and described how this language enables new approaches to teaching programming languages concepts. In this paper, we reproduce the studies by Diwan et al. in a Swedish setting, describe extensions to the original system, and introduce a new technique for evaluating the utility of student experiments. We largely confirm the earlier findings and show how our evaluation technique helps us in our understanding of student experiments.

CCS Concepts: • **Applied computing** → *E-learning*; • **Social and professional topics** → **Computing education**; • **Software and its engineering** → **Language features**.

Keywords: PL-Detective, Education, Programming Language Concepts

ACM Reference Format:
Christoph Reichenbach. 2020. The PL-Detective Revisited. In *Proceedings of the 2020 ACM SIGPLAN SPLASH-E Symposium (SPLASH-E '20), November 20, 2020, Virtual, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3426431.3428655

1 Introduction

The PL-Detective [Diwan et al. 2004] is a tool for encouraging active experimentation when teaching classes on the *Concepts of Programming Languages*. Instructors can integrate it into their exercises and online exams to allow students to discover the *semantics* of a programming language whose *syntax* they already know.

For example, the students may have to determine whether the language checks types statically or dynamically. In this example, the instructor would configure the PL-Detective for either static or dynamic type checking. Then, the students must come up with an *experiment*, a program that behaves differently depending on whether types are checked at runtime or before runtime, and encode this experiment in the PL-Detective's Mystery language. When the students submit their experiment to the PL-Detective (whose configuration they cannot see), they either learn that (a) the program had an error, or (b) the program terminated, possibly with some output. The students can run multiple experiments to collect *evidence*, which they can then use to justify their answer to the question.

In this way, the PL-Detective can facilitate an experiential learning process [Kolb and Kolb 2006]: when the instructor asks students to determine the semantics of Mystery, the students must first review the concepts discussed in class to identify how different semantics materialise in practice (*Abstract Conceptualisation*), then translate these differences into an experiment (*Active Experimentation*) before collecting evidence (*Concrete Experience*). Ideally, the students will then reflect on their insights (*Reflective Observation*) before deciding on whether they have completed the task or should continue the learning cycle to create a new experiment.

We have re-implemented, extended, and adopted the PL-Detective in a Swedish classroom setting and here re-examine its utility 15 years after Diwan et al.'s original study.

To understand the utility of the PL-Detective as a learning tool, we focus on the following research questions, the first two of which we re-use directly from Diwan et al. [2004]:

- **RQ1**: Did the students *effectively* use the PL-Detective as an information source?
- **RQ2**: Did the students *efficiently* use the PL-Detective as an information source?
- **RQ3**: Does combining multiple small exercises into larger exercises encourage student experimentation?
- **RQ4**: Does the students' ability to experiment with the PL-Detective improve over the course of the semester?
- **RQ5**: Do the mistakes that the students make with the PL-Detective evolve over the course of the semester?
- **RQ6**: Does success with the PL-Detective affect student grades?

Our contributions are as follows:

- We partially reproduce the 2005 PL-Detective study by Diwan et al. in a Swedish classroom setting.

SPLASH-E '20, November 20, 2020, Virtual, USA

ACM ISBN 978-1-4503-8180-2/20/11.
https://doi.org/10.1145/3426431.3428655

- We study a number of new exercises, including several combinations of exercises suggested in the original study.
- We describe an extended Open Source re-implementation of the PL-Detective framework.
- We introduce a new evaluation technique for PL-Detective style experimentation, SAGE (Section 4.1).

2 Language Overview

We have re-implemented the Mystery language with minimal syntactic changes. As an example, consider the following program in our dialect, Mystery2020:

```
BEGIN
  PRINT 1 + 2 == 2
END
```

Our semantics represent truth values as integers, where `1` represents "True" and `0` represents "False". The above program thus prints either `0` or `2`, depending on whether the language evaluates `1 + 2` or `2 == 2` first. This in turn depends on the operator precedence that the instructor has configured for the language. Depending on this configuration, the above may also trigger an error, e.g. if both addition and equality have the same precedence but are non-associative.

The original Mystery language provides one form of literal values (integers), three binary operators (less-than, addition, and boolean `AND`), statements for printing, conditional execution and `WHILE` loops, as well as local and global variables together with suitable assignment statements, arrays and associated operations, user-defined and possibly nested procedures that can return values, and user-defined types.

The language uses up to five type constructors (depending on its configuration): integers, subrange types (e.g., **[**1 **TO** 42**]**, the type of all integers i where $1 \leq i \leq 42$), array types, procedure types (for variables that store closures), and user-defined types, which may act as fresh types or type aliases.

2.1 Language Extensions

Our Mystery2020 language adds two modest extensions to the original Mystery: a binary operator for checking equality (==), and the option to omit type annotations.

We included the equality operator to allow us to demonstrate different forms of equality checking for arrays. Specifically, the instructor can configure the operator to perform structural equality checking of arrays (comparing the arrays element-wise), or by-reference equality checking (testing whether the array variables point to the same memory).

The option to omit type annotations allows instructors to illustrate the syntactic difference between languages with and without explicit type information. We envision this feature for new forms of exercises that use the PL-Detective as a *gradually typed language*.

3 PL-Detective Re-Implementation

We have re-implemented Diwan et al.'s PL-Detective in the JastAdd reference attribute grammar system [Ekman and Hedin 2007]. Unlike the original PL-Detective, our system is an interpreter rather than a compiler. At 2970 lines of Java code and 1291 of JastAdd code (which includes Java code), the system has a similar size as the original PL-Detective but allows program analysis with reference attributes. We have found the plug-in modules for different semantic options to be of comparable size to those of the original PL-Detective, as reported by Diwan et al. [2004].

The PL-Detective automatically limits the number of instructions, the size of arrays, and the number of calls (to prevent infinite recursion). We have made it publicly available[1]. We use the PL-Detective together with a web interface that is tied to a custom course management system (Section 3.2) that we expect to publish soon.

3.1 Configurable Syntax and Semantics

Our PL-Detective re-implementation includes four binary operators (addition, equality, less-than, and boolean AND). On the *syntactic* level, Instructors can freely configure the precedence levels and associativities (left, right, non-associative) of each operator. Our system reports ambiguity due to non-associativity as an error.

The remaining *semantic* configuration options are:

- **Parameter Evaluation Order** for procedure calls, either *left-to-right* or *right-to-left*.
- **Operand Evaluation Order** for binary operators, analogously either *left-to-right* or *right-to-left*.
- **Short Circuit Evaluation** for the AND operator, which can be *enabled* or *disabled*.
- **Type Equality for Named Types**, which can be *structural* (as in C) or *nominal* (as in Go).
- **Type Checking**, which can be set to *dynamic checking*, *static checking*, or *disabled*, which uses deferred error reporting analogously to JavaScript.
- **Default Types**, used when the program omits a type declaration. These can be *disabled* (meaning that type declarations are mandatory), *integer*, or the special *Any* type that is the supertype of all other types.
- **Array Assignment and By-Value Passing Semantics**, which can be set to *copy* arrays (as in Go) or to *share* arrays by reference (as in C).
- **Parameter Passing**, which supports passing by *Value*, *Result*, *Value-Result*, *Reference*, *Name*, and *Need* (as in Haskell).
- **Locals Storage Binding**, which can be set to bind local variables to either *static* or *stack-dynamic* storage.
- **Scoping**, which can be *static* or *dynamic*.

[1] https://github.com/lu-cs-sde/mystery2020

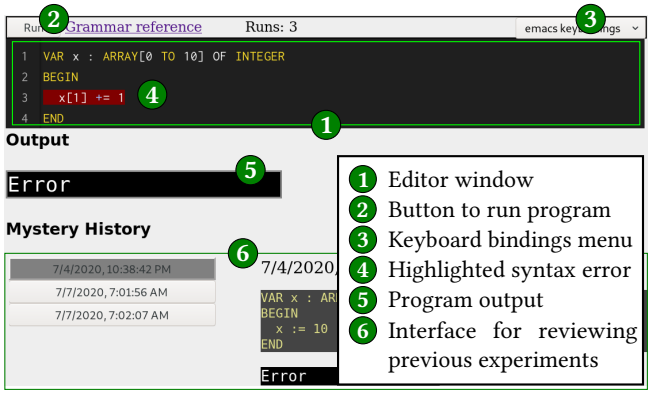

Figure 1. Annotated screenshot of the PL-Detective user interface[2]. The program output can be a number, an empty box, or the message `Error` (as in the example). The interface for reviewing previous experiments does not track experiments with syntax errors.

- **Environment Binding** for closures, allowing *deep binding* (i.e., the environment of the closure is determined by the static scope as in Scala or Haskell) and *shallow binding* (i.e., the environment is determined by the call site, as in LISP 1.5).
- **Array Equality**, which can be *structural* (as in Go) or *by reference* (as in Java).
- **Procedure Argument Subtyping**, which determines how the argument types of two procedure types can vary in order for them to be considered subtypes of each other. The possible settings are *invariant*, *covariant*, *contravariant* (as in many object-oriented languages), and *bivariant*.
- **Procedure Return Subtyping**, which is analogous to procedure argument subtyping but refers to the variance of the return type.
- **Literal Number Type**, which allows setting the type for integer literals; e.g., 5 can either have the type Integer or the type **[**5 **TO** 5**]**.

Our implementation is thus not a strict extension of the original PL-Detective, as we omitted three configuration options from Diwan et al. [2004] and only partially implemented their *Type Equality* feature. In exchange, we provide eight new configuration categories (not including precedence/associativity) and extend on several of the existing configuration categories. We expect that adding the missing features would be straightforward.

3.2 User Interface

Students interact with the PL-Detective through a web-based user interface (Figure 1) as part of a custom course management system. This UI provides syntax highlighting and keyboard commands in the style of of popular editors.

When students interact with the PL-Detective through our UI, we automatically record all experiments that do not have syntactic errors in a database. The PL-Detective reports any output printed by the students' programs plus at most one error message and line number. We currently follow the original PL-Detective setup to hide line numbers and detailed error information and only report the message "Error" on an error, with the exception of syntactic and lexical errors, for which we provide students with the error line number.

When it comes to the language syntax, students thus get a level of comfort close to that of modern IDEs, while we avoid holding their hands for questions of semantics.

3.3 Managing PL-Detective Exercises

As in the original PL-Detective, our system tracks the number of executed prints and the number of runs (experiments that did not have a syntax error) per exercise. Unlike the original system, ours can randomly select different configurations for each student group.

4 Evaluation

We used our system for the first time as part of a new course on programming language concepts at Lund University. We assigned students eleven exercises with limited runs or prints and four without such limits (Table 2).

We based our exercises and limits on the exercises from Diwan et al. [2004] (reproduced in Table 1), with the following changes:

- We conducted our study at Lund University, with mainly Swedish students, and in a 7-week intensive course, as opposed to a 13/14-week course with mainly US-American students as at the University of Colorado at Boulder.
- We provided an improved UI (Section 3.2).
- We cut the exercise on deep/shallow binding (sc-depth).
- Following recommendations by Diwan et al. [2004], we combined four exercises with only two configuration options with two new form of semantic variability (ev-op-order, ar-eq), forming three larger exercises with four possible configurations each.
- We added two new exercises (sy-prec and sy-assoc) related to precedence and associativity.
- We configured our system to randomly select one possible configuration for each group, rather than hard-wiring the same configuration for all groups.

We used our four exercises with unlimited resources partly to familiarise the students with the syntax and UI. We do not consider these exercises further in the following, except where expressly mentioned.

37 two-person groups participated in our exercises in total, submitting a total of 383 answers (each corresponding to a unique ⟨group, exercise⟩ pair), plus 143 more answers for

[2]Edited for space: empty space removed.

Table 1. Reproduction of the list of PL-Detective experiments as reported by Diwan et al. [2004] (Table I), with exercise **Code**s added for cross-referencing with Table 2. All exercises had a limit on either the number of prints allowed or the number of runs (column **Limit**), not counting experiments with syntax errors. We reproduce the table in the order in which the students encountered them (assignment number / week in column **Assignment**).

Code	Description	Limit	Assignment
sb-local	What is the storage binding of local variables?	3 prints	3
ty-eq	When are two types equal?	8 runs	3
ty-con	Do type declarations create a new type?	2 runs	3
sc-lookup	Does Mystery use static of dynamic scoping?	3 runs	4
ar-assign	What are the semantics of array assignments?	6 prints	5
ev-p-order	What is the evaluation order in a procedure call?	4 prints	5
ev-short	Does Mystery use short-circuit evaluation?	3 prints	6
ty-p-variance	When is one procedure type a subtype of another?	6 runs	6
pp-m	What is the parameter passing mechanism (A)?	10 prints	7
pp-m	What is the parameter passing mechanism (B)?	10 prints	7
(sc-depth)	Does Mystery use deep or shallow binding?	4 prints	8

Table 2. Summary of the PL-Detective exercises in our course. Exercise codes in boldface are new exercises. Column **Opt** presents the number of options that students had to distinguish between. For sy-assoc, sb-local, and pp-m, we limited the actual number of options that students could encounter (actual number in parentheses).

Code	Description	Limit	Assignm.	Opt
	WARMUP: Booleans, > and ==	—	1	—
sy-prec	What is the relative operator precedence of >, +, ==?	5 prints	1	6
sy-assoc	What is the associativity of >?	3 runs	1	3 (2)
	WARMUP: Write and use a subprogram	—	2	—
sb-local	What is the storage binding of local variables?	3 prints	2	2 (1)
ty-eq + ty-con	When are two (fresh) types equal?	4 runs	2	4
ev-short + **ev-op-order**	How does Mystery2020's AND evaluate its arguments?	3 prints	2	4
ev-p-order	What is the evaluation order in a procedure call?	4 prints	2	2
	WARMUP: Write a nested subprogram	—	3	—
	WARMUP: Use a closure as a value	—	3	—
sc-lookup	Does Mystery2020 use static of dynamic scoping?	3 runs	3	2
pp-m (a)	What is the parameter passing mechanism (A)?	10 prints	3	6 (3)
pp-m (b)	What is the parameter passing mechanism (B)?	10 prints	3	6 (3)
ar-assign + **ar-eq**	What are the semantics of array assignments and equality?	8 prints	4	4
ty-p-variance	When is one procedure type a subtype of another?	6 runs	4	9

exercises with unlimited resources. After manual inspection, we excluded three of the answers to exercises with limited resources from our study (Section 4.2.1).

All groups completed at least two exercises with limited resources, and all but four groups completed all eleven. The groups submitted between six and 155 experiments (mean = 40) in total across all of their answers.

4.1 Evaluation with SAGE

To evaluate student experiments, we developed a new automatic technique that groups them by whether they are (a) relevant to the given exercise and (b) expose new insights over previous experiments in the same exercise.

Our technique re-runs each syntactically well-formed student experiment with all possible configurations for the given exercise and clusters the output by behaviour that is indistinguishable to the students (i.e., produces the same output, which may be no more than the message `Error`).

For example, in an exercise in which students explore whether the == operator associates (1) to the left, (2) not at all, or (3) to the right, the following program prints 1 for configurations (1) and (3):

```
BEGIN
  PRINT 1 == 1 == 1
END
```

while for configuration (2) it produces an error. We thus find this program **relevant**: it yields *evidence* that can help the students solve this particular exercise. (Note that the program would *not* be relevant e.g. to an exercise on scoping.)

If the above example printed the number 1, there are still two configuration alternatives that the students must explore, namely (1) left-associativity and (3) right-associativity. Submitting a further **relevant** program that distinguishes between the alternatives {(1), (3)} and {(2)} would not help the students progress (they have already eliminated (2)), so we introduce a new category of student experiments: **insightful** experiments are experiments that produce *new* evidence that allows students to shrink the set of configuration alternatives that explain the set of observations so far.

In this paper, we use this classification technique to understand whether student submissions are effective at grouping and eliminating configuration alternatives. Our technique, SAGE[3], classifies each experiment as one of the following:

- **insightful**: discovered new, relevant evidence
- **relevant**: re-discovered relevant evidence that was not insightful
- **trivial**: ran without error, produced no insight
- **erroneous**: ran with error, produced no insight
- **copy**: was identical (modulo whitespace changes and variable renaming) to a previous submission

Moreover, if the set of student experiments in a homework submission narrows down the configuration alternatives to just one option, we consider the evidence **conclusive**.

Using SAGE, we found that the students submitted a total of 1678 syntactically well-formed programs, of which 89 (5%) were copies, 428 (26%) were erroneous, 273 (16%) were trivial, 354 (21%) were relevant and 534 (32%) insightful.

Of the 380 student answers, 275 (72%) had conclusive evidence, and 254 (67%) passed the exercise, close to the 70% reported by Diwan et al. [2005]. Figure 2 breaks down these numbers by exercise.

4.2 Manual Investigation

We manually explored a subset of the 383 student submissions to confirm that the PL-Detective was working as intended and to better understand how effective SAGE was at indicating student progress,

We focused our efforts on exercises in which (a) the students had sufficient evidence to find the correct answer but failed to obtain 50% or more of the grade for the question as per the assessment of the teaching assistants (11 cases), (b) exercises in which the students had insufficient data to reach a conclusion but obtained 75% or more of the grade for the question as per TA assessment (35 cases), (c) a manual inspection of the TA's feedback otherwise directly contradicted SAGE (approximately 30 cases), and (d) all student answers in which our assessment had disagreed with TA assessment more than five times (exercises sb-local, pp-m (b), and ty-p-variance, for approximately 85 additional cases).

[3]Semantic Alternative Grouping and Elimination

Table 3. Obtaining complete and insightful information compared against completing the exercise successfully.

	Fully correct	**Not fully correct**
Conclusive	250 (65.8%)	25 (6.6%)
Insightful but inconclusive	3 (0.8%)	88 (23.2%)
No insightful experiments	1 (0.3%)	13 (3.4%)

4.2.1 Grading with Bugs.

During this process we discovered and fixed several bugs in our PL-Detective implementation, including (1) a misconfiguration in our resource limiter that allowed students to allocate arbitrarily large arrays, (2) several bugs that produced incorrect error messages (which sometimes led to the students getting an empty reply that neither counted as a run nor produced information), (3) a nondeterministic bug in precedence parsing due to concurrent evaluation, and three bugs in specific subsystems that would incorrectly cause or suppress errors in corner cases.

We carefully re-examined all 39 homework submissions in which at least one experiment behaved differently after our fixes. Most of the impact did not affect the outcome or was caught by the teaching assistants (when the incorrect PL-Detective output gave rise to an alternative explanation). However, we found three homework submissions in which the students had not provided a satisfactory answer at least partly due to confusion caused by one of the above bugs. We eliminated these three cases for the rest of this study.

During this process, we re-graded each student submission that we examined as "pass" or "fail". For the homework submissions that we had not classified by hand (approximately 55%), the TA's grade and feedback largely agreed with the SAGE assessment and we treated the submission as "pass" unless SAGE indicated otherwise or the TA's feedback clearly found the students' explanation to be insufficient.

4.3 Research Questions

We now use the results from SAGE together with our partitioning of student answers into "pass" or "fail" to investigate our research questions.

4.3.1 RQ1: Did the Students Effectively use the PL-Detective?

Table 3 summarises how often students were able to complete exercises successfully, compared to whether they had obtained conclusive evidence or at least one piece of insightful evidence. Among groups who obtained conclusive evidence, 91% were able to translate this evidence into a correct answer, whereas for groups that had at least *some* evidence, the number was 69%. This number is lower than the 77% reported by Diwan et al., and the difference persists even

if we split out exercises with only two options (where "some evidence" is identical to "conclusive evidence"). One possible explanation is the difference in maturity of the courses: we taught our course for the first time in this setting, whereas the earlier course was in at least its third iteration.

Figure 2 splits the four main categories (conclusive / inconclusive evidence vs. pass / fail) across our exercises. We observe considerable variation: students perform best in some of the earliest exercises and struggle most in exercise ty-p-variance. Two exercises (sy-assoc, ev-p-order) stand out: here, five or more groups found sufficient evidence but failed to capitalise on it. In most of these cases, the groups found the correct answer but did not give a sufficient explanation.

In four cases, students were able to complete the exercises with incomplete information (according to SAGE). In all four of those cases, the students submitted effective experiments that triggered an unrelated error that the students interpreted as evidence. In one of these four cases, the error was due to a bug in our implementation, while in the other three cases the students had included an unrelated bug in their code that was incidental to the point of the exercise.

Figure 3 breaks down the SAGE categories by individual experiments. For most exercises, the overwhelming majority of groups were able to submit an insightful experiment immediately as their first experiment, with the exceptions of sb-local, ty-eq+ty-con, and ty-p-variance, three of the four most overall failure-prone exercises (cf. Figure 2). This suggests that students may not have been prepared well enough for these exercises. In particular, we see that students submitted a large number of experiments to sb-local, which had only two options for students to explore experimentally but two other options that students could eliminate by reading the information available to them.

Exercises ty-p-variance and ty-eq+ty-con saw a large number of errors initially, as students struggled with type declarations in Mystery2020. Similarly, sb-local saw many type-related errors, as students encountered variables and type annotations for the first time. These three exercises were the exercises most closely connected to type checking, suggesting that student found types to be particularly challenging.

Perhaps surprisingly, students also started out with a high rate of insightful experiments to pp-m (b), the exercise with the third-lowest success rate. This effect is easier to understand when comparing it to pp-m (a): both exercises asked students to determine the correct parameter passing mode out of six options, but pp-m (a) randomly selected among three common options (by-value, by-reference, by-value-result) while pp-m (b) selected among the remaining, more challenging options. Since all but two groups started with (a) before starting on (b) and 25 of the 35 participating groups completed (a) before starting on (b), we expect that most groups were able to initially capitalise on their experiences from pp-m (a) to define an effective first set of queries.

4.3.2 RQ2: Did the Students Efficiently use the PL-Detective?

To encourage efficient experimentation, we penalised assignment task scores by up to 50% when groups exceeded resource limits.

Figure 4 shows the number of prints or runs (depending on the resource limit for the exercise) that the students used. For all exercises, most students stayed within the limit, though for ev-p-order, many used more prints than we expected.

In ty-p-variance, we see that most students came close to exceeding the number of runs, suggesting that the number of runs available to the students was too low.

We found three extreme outliers, each by a different very persistent group that initially took a wrong path but ultimately converged on the correct experimentation strategy. Two of these outliers found a correct solution.

As we see in Figure 5, a substantial fraction of the successful groups submitted additional experiments even after they had already found conclusive evidence, especially in exercises with a limited number of prints but unlimited runs: even successful groups were not always effective at minimising resource usage. However, our evaluation scheme did not penalise groups who experimented more than necessary, as long as they remained below the resource limit.

We checked for correlation between resource usage and group success in any given homework exercise using Spearman's correlation coefficient. We found no significant correlation between the fraction of prints used and whether a group succeeded, but we did observe a significant negative correlation ($p = 0.001$, $r = -0.335$) between the fraction of submitted experiments (without syntactic errors) and whether the group succeeded, confirming the findings by Diwan et al. [2004]. Our hypothesis is that groups that submit fewer programs are more careful in trying to understand previous outcomes and designing subsequent experiments.

On average, we found that students used about 75% of the runs allocated to them, and 60% of the prints. Thus, the students were less efficient than the students in the original PL-Detective study (which reported the students using "a little over 50% of the limited attempts").

4.3.3 RQ3: Does Combining Multiple Small Exercises Encourage Experimentation?

Diwan et al. [2004] noted that in their setup there were "several exercises where students had to distinguish between only two possibilities and thus did not engage in experimentation" and suggested adding more possibilities e.g. by "combining two exercises with two possibilities each into a larger exercises with more possibilities." We experimented with this idea in three exercises, ty-eq + ty-con, ev-short + ev-op-order, and arr-assign + arr-eq, and found these to be exercises that provoked nontrivial amounts of experimentation, which supports Diwan et al.'s hypothesis.

Conversely, for the three exercises in which *our* students had only two options, we found that only for sc-lookup the

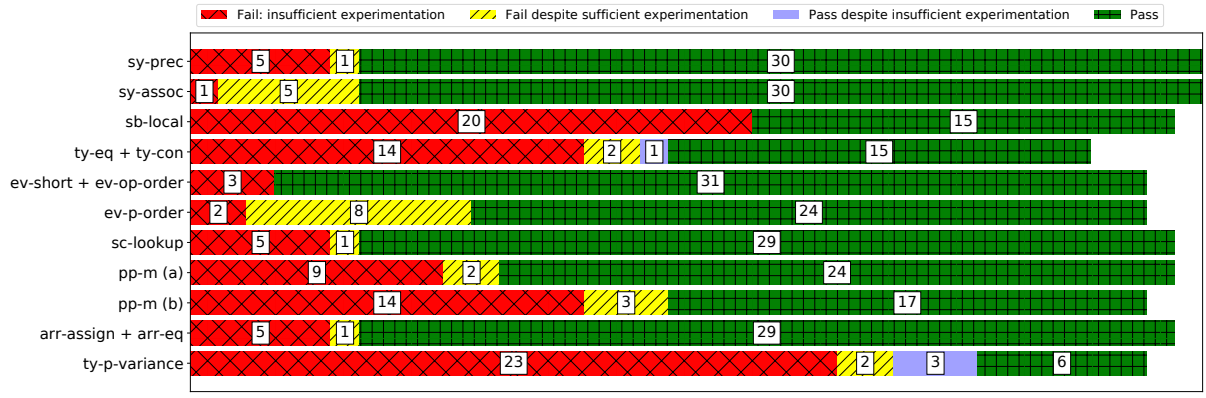

Figure 2. Aggregate student answers by failure, success, and completeness of evidence, broken down by exercise.

students largely finished experimenting after one or two experiments. For sb-local and ev-p-order, the students instead produced many trivial or erroneous experiments. This may be due to the latter two being limited by the number of prints, while sc-lookup was limited by the number of runs.

4.3.4 RQ4: Do Students Improve Their Usage of the PL-Detective?

We further examined whether students' ability to take advantage of the PL-Detective evolved over the course of the semester but could not find any obvious patterns to support this hypothesis. As Figure 3 shows, the final exercise had the lowest fraction of initial **Progress** experiments, while the first two exercises had among the highest. We could not observe any significant differences even after discriminating groups based on their overall ability to successfully complete PL Detective homework tasks.

Instead, when we checked for *perfect* student submissions that only contained insightful experiments, we observed a trend that suggested that the number of such submissions *decreased* over the semester, from 15, 25, and 4 for the first three exercises to 8, 8, and 0 for the last three exercises. This suggests that the students' ability to succeed in the exercises was dominated by other factors, such as the overall difficulty of the exercise or the students' stress levels.

4.3.5 RQ5: Do Student Mistakes with the PL-Detective Evolve?

Table 4 summarises the error kinds that students encountered during experiments that were erroneous (i.e., triggered errors that they could not use as evidence). The error types are largely connected to the types of exercises that the students worked with, though we do see indications that the number of 'Kind' errors (in which students declared a user-defined type and used is a variable, or vice versa) shrunk over the course of the semester, pointing to students learning about the shared scope between type names and variable names. Otherwise, we see no indication that students became more skilled at using Mystery2020, most likely due to the substantial differences between the tasks.

Table 4. Total number of experiments per exercise (excluding experiments with syntax errors), and errors among them: Type (**Ty**), Name (**Nm**), Name kind (**K**, using variable names as type names or vice versa), Resource limit (**Lt**, mainly array size limits), and **Other**, which includes out of bounds accesses (33), numeric literals too large for 32 bit numbers (17), and assignments to out-mode variables (7, all pp-m (b)).

Exercise	Total	Ty	Nm	K	Lt	Other
sy-prec	119	0	0	0	0	0
sy-assoc	50	0	0	0	0	0
sb-local	150	23	22	9	2	0
ty-eq+ty-con	115	12	12	8	0	1
ev-short + ev-op-order	126	5	8	0	4	4
ev-p-order	72	4	3	0	0	0
sc-lookup	55	1	1	0	0	0
pp-m (a)	251	7	37	4	2	25
pp-m (b)	306	9	17	1	0	7
arr-assign + arr-eq	173	9	18	0	8	24
ty-p-variance	261	115	20	0	5	1

4.3.6 RQ6: Do PL-Detective Exercises Affect Grades?

After the final exam for the course, we examined whether group performance in the homework exercises was correlated with group member performance in the exam. Following Wolfe [1990], we tried to avoid evaluating learning success by summary grades and instead directly examined seven exam questions that were closely connected to homework assignments, though we had to eliminate one from this consideration because all students were able to give a fully satisfactory answer to it in the exam.

When comparing across all students, we found no significant correlation. We then explored the hypothesis that one student in each group might have benefit more than the other. We separated groups into the 'weaker' and the

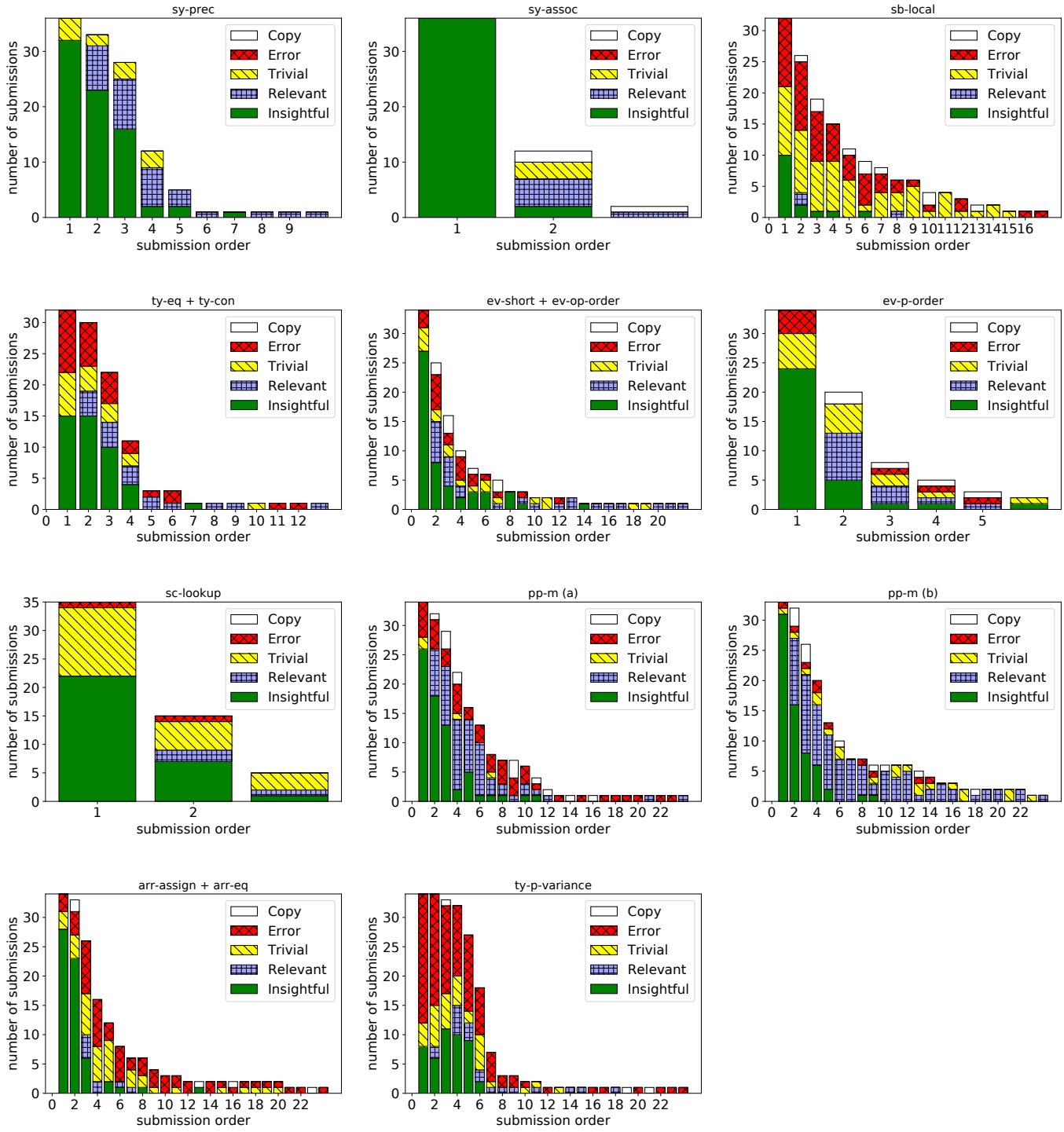

Figure 3. Effectiveness of student experiments in gaining insight in an exercise. The x axes represents the number of experiments, the y axes represents the number of experiments in the five SAGE categories. Despite several instance of **Copy** we observed no instances of the students resubmitting the exact preceding program with no changes. All tables are clipped at 24 experiments, though we saw one outlier each in pp-m (a) (83 experiments), pp-m (b) (116 experiments), and pp-m (b) (77 experiments), by three different groups. The first two of these outliers were successful in identifying the correct configuration, but had already obtained conclusive evidence after their first experiment.

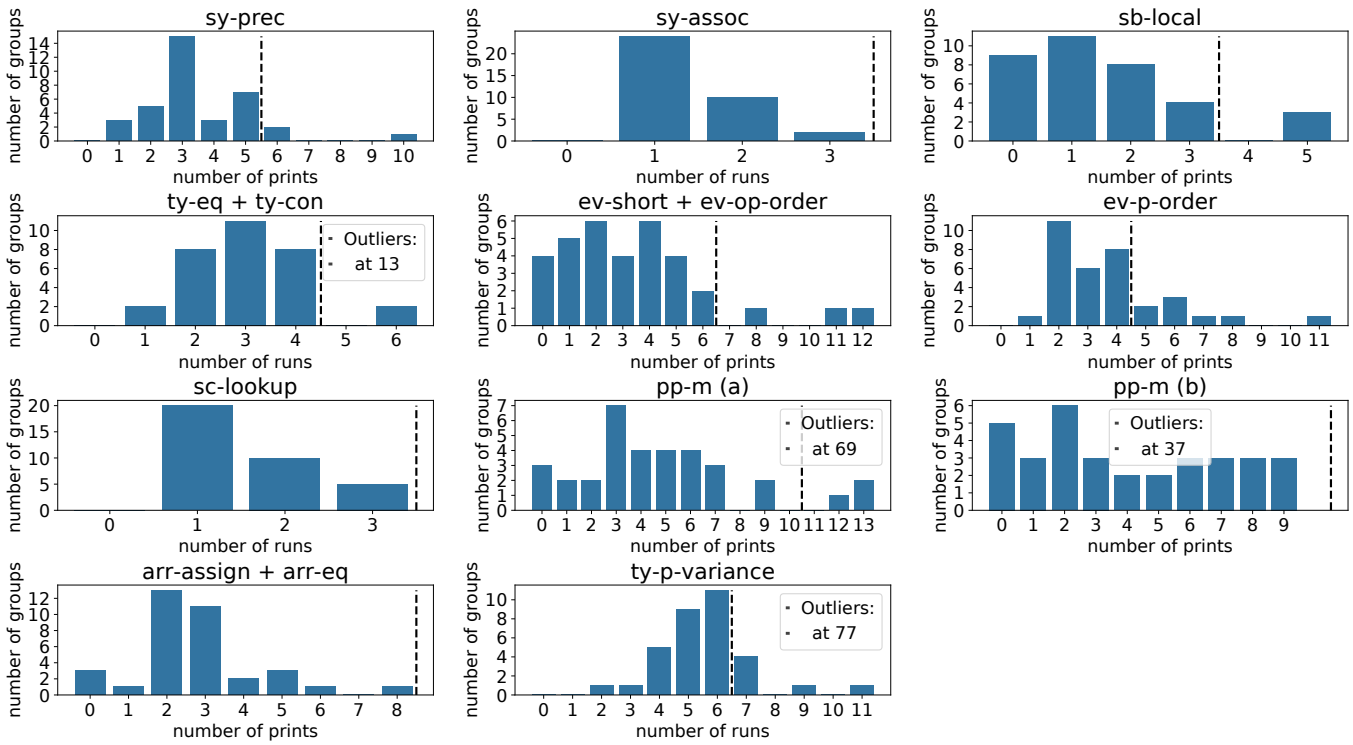

Figure 4. Resource usage by exercise (for prints or runs, depending on the exercise). The x axis represents the number of runs or prints used, the y axis the number of groups in that category. The dashed line is the soft resource limit for the exercise.

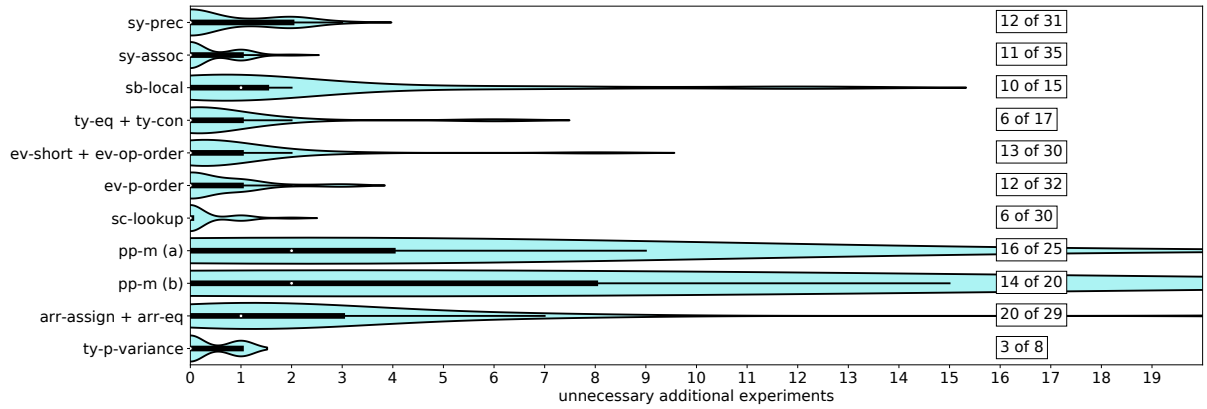

Figure 5. Number of experiments that the students ran after they had already gathered conclusive evidence.

'stronger' student, based on their exam grades. We found no evidence that the weaker students' performance was correlated to their group's performance in related homework questions but we did find evidence that the stronger students' performance in relevant exam questions was correlated to group performance in two of the six cases: for pp-a ($p = 0.02$ and $r = 0.42$) and for sb-local ($p = 0.02$ and $r = 0.45$).

One possible explanation for these correlations could be that in many groups, only one student contributed to these homework questions. Leonardi et al. [2009] notes that engineering students often prefer to work alone, meaning that the students may have split up the work instead of collaborating. However, we find the evidence for this case to be weak: the exam included three exam questions on different parameter passing modes (pp-a), while we only saw a significant correlation for one of them.

5 Discussion

While our results largely confirm the main findings of the original 2004 study as well as the merit of same study's proposal for merging (or expanding on) smaller exercises, we also observe room for further improvement.

Improvements to the Approach. On average, each group failed on one third of the PL-Detective exercises. Considering that our SAGE technique allows us to partly automate the grading of the students' submissions (at the cost of skipping the stage in which they explain their reasoning), we could allow students to re-do these exercises, in the style of Mastery Learning [Bloom 1968]. We could further automate the judgement of student reasoning by asking students to write down what behaviour they expect for the alternative configurations, which is trivial to machine-check.

Another concern that we observed is that 45% of the time when groups had gathered conclusive evidence, they submit further experiments (Figure 5). This may indicate that the groups are not applying (or deferring) *reflective observation* [Kolb and Kolb 2006]. One option to encourage reflection is to ask students to explain what they learned from each individual experiment.

Alternatively, we could use SAGE to give direct feedback on whether the students are "on the right track" [Feldman et al. 2019]. To avoid over-reliance on SAGE, we could delay SAGE feedback, which could also encourage students to start their homework early [Leonardi et al. 2009].

Threats to Validity. Our study is subject to a number of threats to internal validity, and we have listed the threats that we are aware of as part of the preceding evaluation.

One goal of our study was to address threats to external validity in the original PL-Detective study. Our study differs by us using a new (though largely equivalent) implementation and interface, and integrating with a different course structure at a different level of maturity. Moreover, both the experimenter and the study subjects were different, with the latter having a substantially different educational and generational background. The differences to the original study that we observed in Sections 4.3.1 and 4.3.2 suggest that further studies may be necessary to better understand their causes.

6 Related Work

There is now a variety of techniques to support online student learning [Diwan et al. 2005; Feldman et al. 2019; Wrigstad and Castegren 2017] in the form of "blended learning" [Stacey and Gerbic 2008] that combines online activities with in-person meetings. Our work re-validates one of these techniques and sheds more light on the approach's strengths and weaknesses. Moreover, it is to the best of our knowledge the first reproduction of an evaluation for such a teaching tool.

The above-mentioned techniques provide instructors with new abilities, such as automatic (varied) repetition to support mastery learning [Bloom 1968; Wrigstad and Castegren 2017], exercise synthesis [Radošević et al. 2010] and MOOC support. Not all of these are immediately applicable to the PL-Detective, though we sketch some ideas in Section 5 for some future extensions to the PL-Detective approach.

7 Conclusions

We have reproduced Diwan et al.'s 2004 PL-Detective study with a re-implementation of the PL-Detective system. We added an improved user interface and several extensions, evolved the original exercises as recommended by the original authors, and used a novel technique to evaluate the student submissions in greater detail. Our analysis shows that students in our setting were somewhat less effective (69% vs 77%) and efficient (75%/60% vs. 50% of the available resources) at using PL-Detective than the students in the earlier study. At the same time, we confirmed a correlation reported in the earlier study that shows that student groups with fewer experiments tend to be more successful. Finally, we identified a more detailed analysis technique that sheds additional insight into how students use the PL-Detective.

Acknowledgments

This work was partially supported by Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. The author is grateful to Görel Hedin and the anonymous SPLASH-E reviewers for valuable feedback.

References

Benjamin S Bloom. 1968. Learning for Mastery. Instruction and Curriculum. Regional Education Laboratory for the Carolinas and Virginia, Topical Papers and Reprints, Number 1. *Evaluation comment* 1, 2 (1968), n2.

Amer Diwan, Michele H. Jackson, William M. Waite, and Jacob Dickerson. 2005. PL-Detective: Experiences and Results. *SIGCSE Bull.* 37, 1 (Feb. 2005), 221–225. https://doi.org/10.1145/1047124.1047423

Amer Diwan, William M. Waite, Michele H. Jackson, and Jacob Dickerson. 2004. PL-Detective: A System for Teaching Programming Language Concepts. *J. Educ. Resour. Comput.* 4, 4 (Dec. 2004), 1–es. https://doi.org/10.1145/1086339.1086340

Torbjörn Ekman and Görel Hedin. 2007. The JastAdd system—modular extensible compiler construction. *Science of Computer Programming* 69, 1-3 (2007), 14–26.

Molly Q Feldman, Yiting Wang, William E Byrd, François Guimbretière, and Erik Andersen. 2019. Towards answering "Am I on the right track?" automatically using program synthesis. In *Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E*. 13–24.

Alice Y Kolb and David A Kolb. 2006. Learning styles and learning spaces: A review of the multidisciplinary application of experiential learning theory in higher education. In *Learning styles and learning: A key to meeting the accountability demands in education*. Nova Science Publishers New York, 45–91.

Paul M Leonardi, Michele H Jackson, and Amer Diwan. 2009. The Enactment-Externalization Dialectic: Rationalization and the Persistence of Counterproductive Technology Design Practices in Student Engineering. *Academy of Management Journal* 52, 2 (2009), 400–420.

Danijel Radošević, Tihomir Orehovački, and Zlatko Stapić. 2010. Automatic on-line generation of student's exercises in teaching programming. In *Radošević, D., Orehovački, T., Stapić, Z:" Automatic On-line Generation of Students Exercises in Teaching Programming", Central European Conference on Information and Intelligent Systems, CECIIS.*

Elizabeth Stacey and Philippa Gerbic. 2008. Success factors for blended learning. In *Proceedings of the 25th ASCILITE Conference* (Melbourne, Australia). 964–968.

Joseph Wolfe. 1990. The evaluation of computer-based business games: Methodology, findings, and future needs. *Guide to business gaming and experiential learning* (1990), 279–300.

Tobias Wrigstad and Elias Castegren. 2017. Mastery Learning-Like Teaching with Achievements. In *SPLASH-E.*