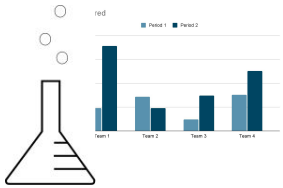
How to Evaluate Blame for Gradual Types

(Part 2)

Lukas Lazarek,
Ben Greenman, Matthias Felleisen,
Christos Dimoulas

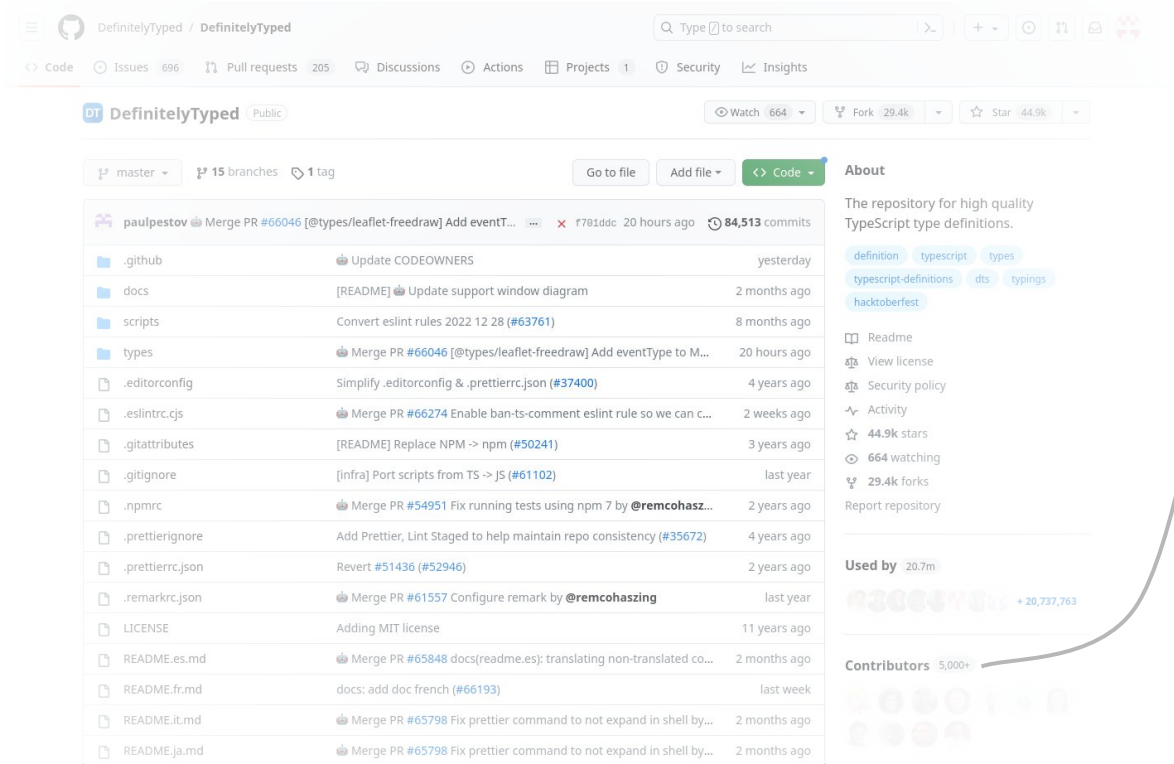
Research Question

How does Gradual Typing help with debugging mistaken type annotations?



Mistaken Types: A Story of Gradual Typing in Practice

Gradual Typing in Practice



Type interfaces

```
sort:
  any[]
  (any any → boolean)
  → any[]
```

Type Interface Mistakes Happen Often

The Academic Perspective on Type Mistakes

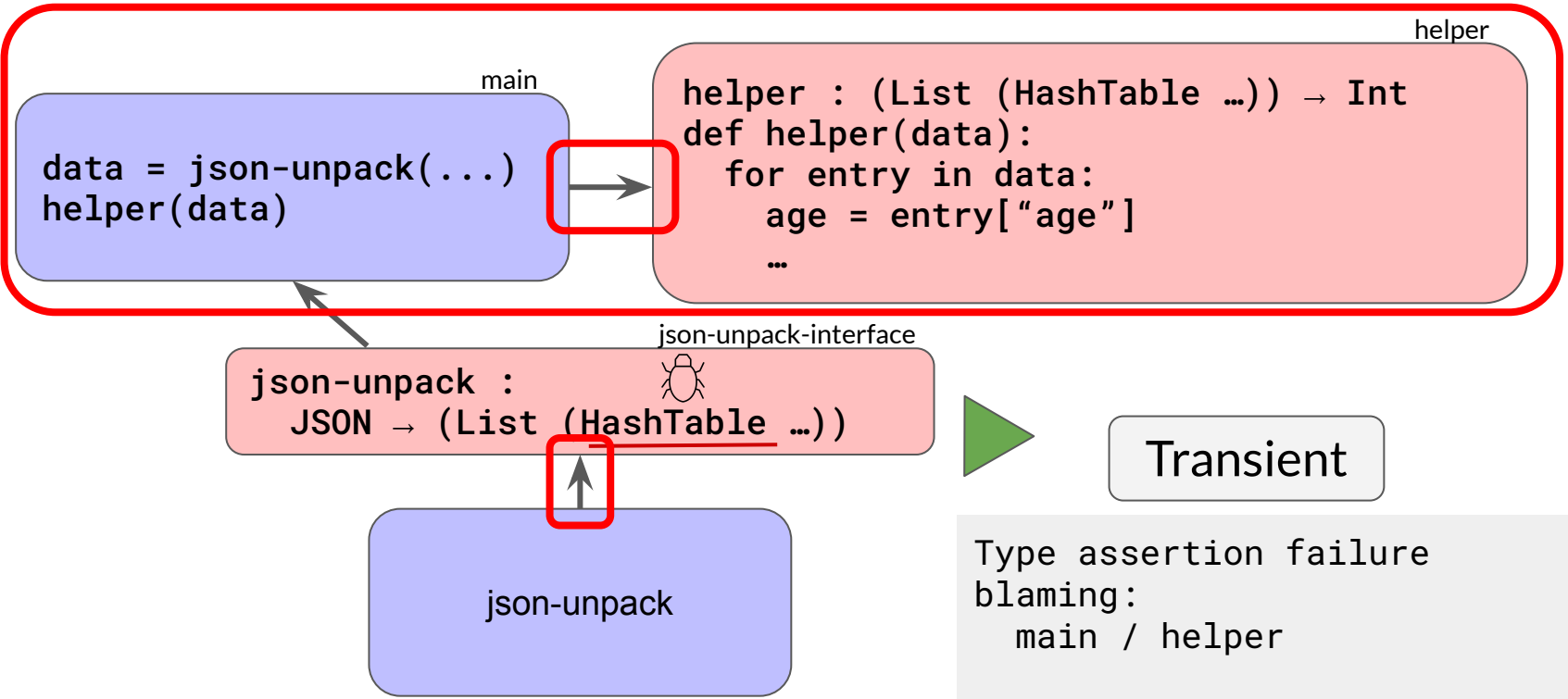
e.g. Reticulated Python  e.g. Typed Racket

Erasure	Transient	Natural
No dynamic checks for type annotations	Type assertions in typed code	Higher order contracts
Stacktraces	Blames many	Blames one

Research Question (refined)

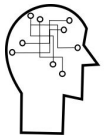
Given the same statics, which semantics for Gradual Typing provides better **error information** for systematically locating **type interface mistakes**?

Comparing Each Semantics, by Example



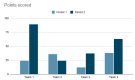
Comparing Each Semantics, Systematically

Hypothesis



Automated procedure(s)

Experiment testing procedures on real programs



Data providing evidence to support or refute the hypothesis

Does Bla

LUKAS LAZ
ALEXIS KIN
SAMANVIT
ROBERT BR
CHRISTOS

Contract syste
of blame is to n
system discove
In reality, ho
of bugs. The re
use contracts t
source of the p
This paper a
we introduce a
to translate bla
by how progra
shift blame to a
construct a fra
causes blame t
"closer" to the f
Our methodo
it to analyze Ra
of contracts an
sufficient to nar
that trigger sta
interfere with p
when searchin
design.

CCS Concepts
→ Empirical

Additional Key

ACM Referen
Lukas Lazarek,
Blame Shifting.
10.1145/337113

Authors' addres
King, Northwest
samanvithasunda
edu; Christos Dim

This work is lice

© 2020 Copyrigh
2475-1421/2020/
https://doi.org/10

65

How to Evaluate Blame for Gradual Types

LUKAS LAZAREK, PLT @ Northeastern University, USA
BEN GREENMAN, PLT @ Northeastern University, USA
MATTHIAS FELLEISEN, PLT @ Northeastern University, USA
CHRISTOS DIMOULAS, PLT @ Northeastern University, USA

Programming language theoreticians develop blame assignment systems and prove blame theorems for gradually typed programming languages. Practical implementations of gradual typing almost completely ignore the idea of blame assignment. This contrast raises the question whether blame provides any value to the working programmer and poses the challenge of how to evaluate the effectiveness of blame assignment strategies. This paper contributes (1) the first evaluation method for blame assignment strategies and (2) the results from applying it to three different semantics for gradual typing. These results cast doubt on the theoretical effectiveness of blame in gradual typing. In most scenarios, strategies with imprecise blame assignment are as helpful to a rationally acting programmer as strategies with provably correct blame.

CCS Concepts: • **Software and its engineering → Empirical software validation**; • **Theory of computation → Program specifications**.

Additional Key Words and Phrases: gradual typing, blame

ACM Reference Format:
Lukas Lazarek, Ben Greenman, Matthias Felleisen, and Christos Dimoulas. 2021. How to Evaluate Blame for Gradual Types. *Proc. ACM Program. Lang.* 5, ICFP, Article 68 (August 2021), 29 pages. https://doi.org/10.1145/3473573

68

1 DOES BLAME MATTER

Theoreticians of gradual typing have focused on blame theorems from the very beginning [Matthews and Findler 2009; Tobin-Hochstadt and Felleisen 2006]. "Well-typed [components][1] can't be blamed" turned the theorem into a slogan [Wadler and Findler 2009]. Academic systems (Reticulated Python [Vitousek et al. 2014, 2019, 2017] and Typed Racket [Tobin-Hochstadt and Felleisen 2006, 2008, 2010; Tobin-Hochstadt et al. 2017]) come with sophisticated checking and blame assignment strategies (sec. 2). Their academic creators embrace the idea that blame can help practicing programmers find impedance mismatches, that is, disagreements between the type ascriptions of a software component and its behavior.

Industrial implementors of gradual typing systems have almost completely ignored blame assignment. Systems such as Flow, Hack, or TypeScript[2] exploit types for IDE actions and for finding

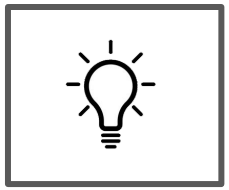
[1]The original authors got this word wrong. A program has many components; blame helps identify a faulty one.
[2]See https://flow.org, https://hacklang.org, and https://www.typescriptlang.org, respectively.

Authors' addresses: Lukas Lazarek, PLT @ Northwestern University, Evanston, Illinois, USA, lukas.lazarek@eecs.northwestern.edu; Ben Greenman, PLT @ Northeastern University, Boston, Massachusetts, USA, benjaminlgreenman@gmail.com; Matthias Felleisen, PLT @ Northeastern University, Boston, Massachusetts, USA, matthias@ccs.neu.edu; Christos Dimoulas, PLT @ Northwestern University, Evanston, Illinois, USA, chrdimo@northwestern.edu.

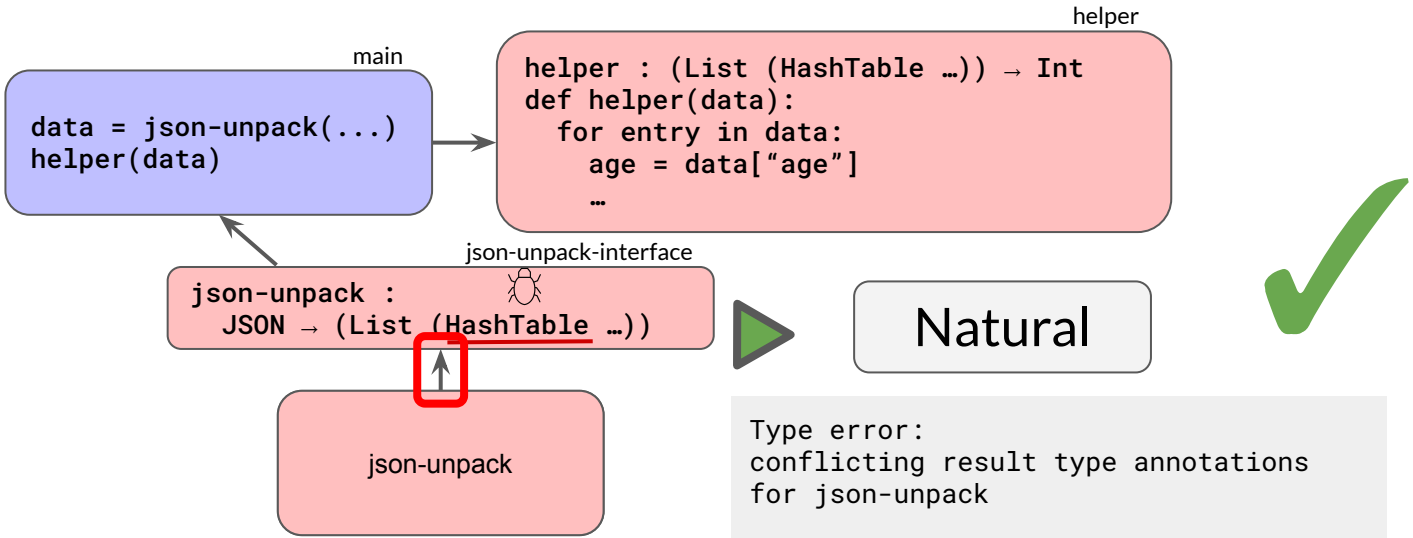
2475-1421/2021/8-ART68
https://doi.org/10.1145/3473573

Proc. ACM Program. Lang., Vol. 5, No. ICFP, Article 68. Publication date: August 2021.

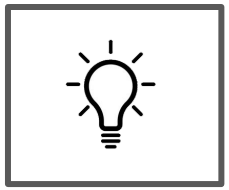
A Hypothesis, by Example



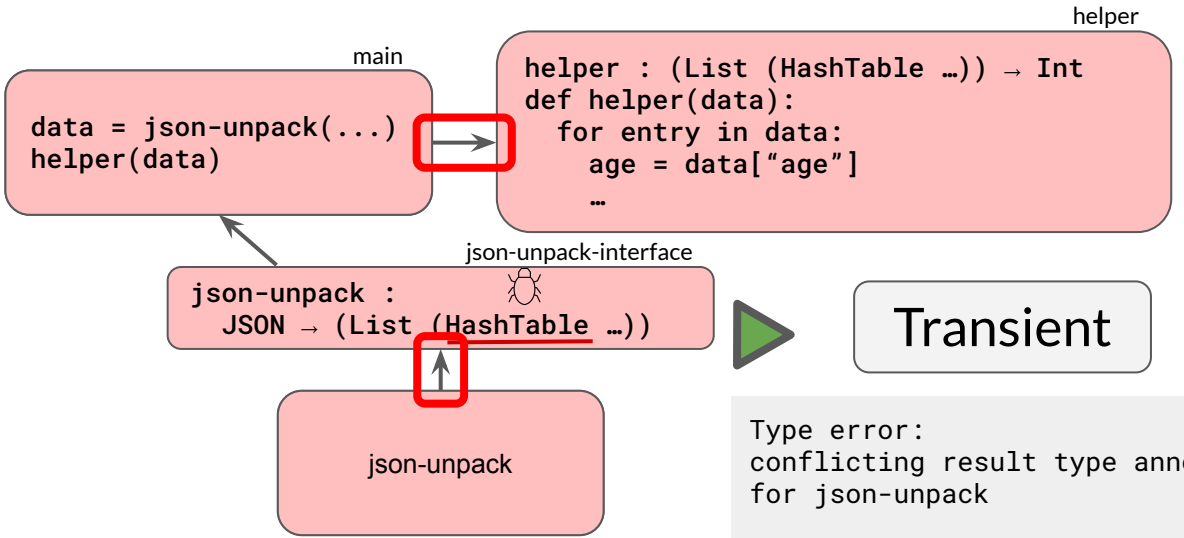
Hypothesis Using the type system, error information can be translated into the location of type interface mistakes



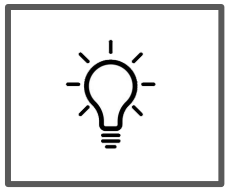
A Hypothesis, by Example



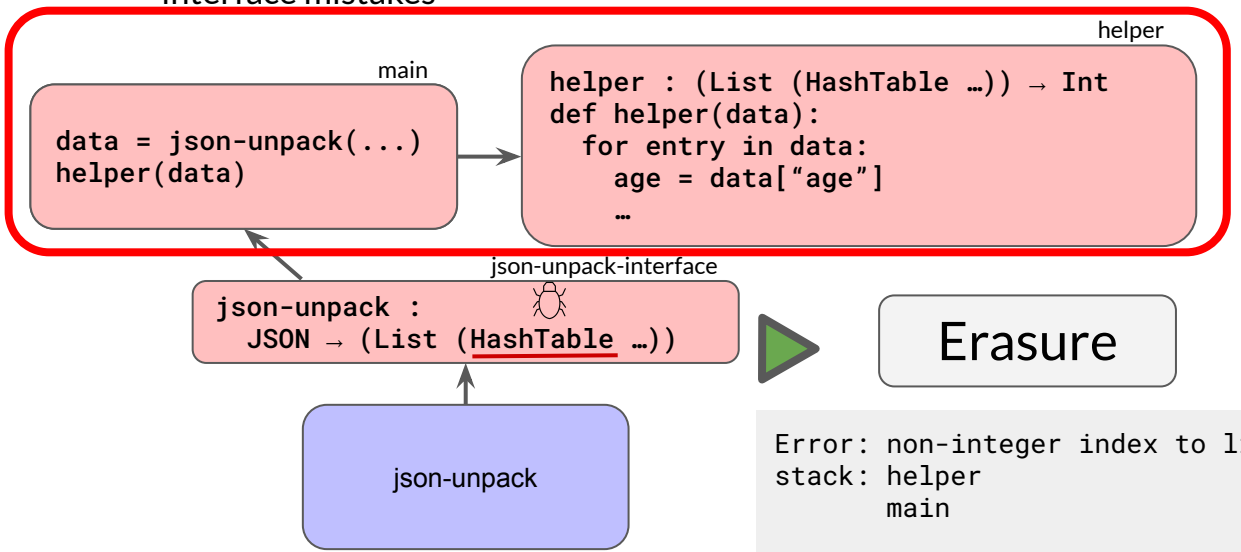
Hypothesis Using the type system, error information can be translated into the location of type interface mistakes



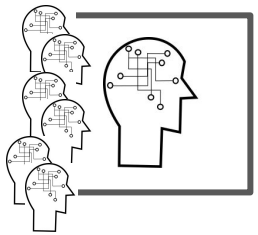
A Hypothesis, by Example



Hypothesis Using the type system, error information can be translated into the location of type interface mistakes



A Procedure Reifying Our Hypothesis



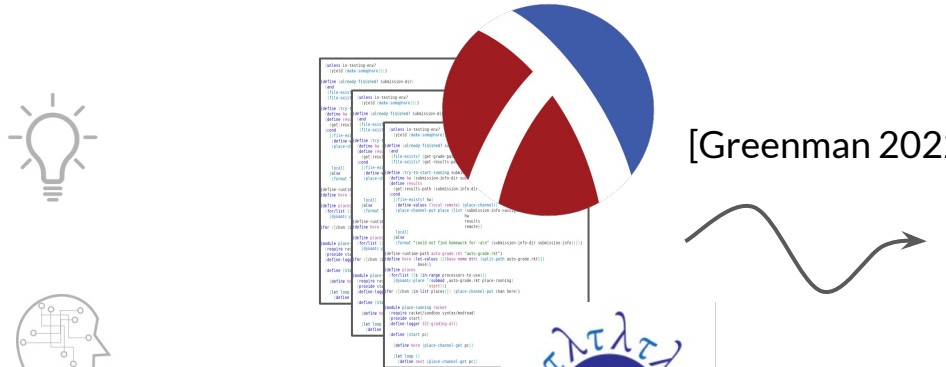
1. Run the program with Natural semantics to get blame
2. Identify the (untyped) blamed component
3. Try to type that component* (may fail)
4. Type-check the program
   - 4.1 If it type-checks: go to 1
   - 4.2 Otherwise: stop (success)

Natural
Transient
Erasure

Natural-exceptions
Transient-exceptions

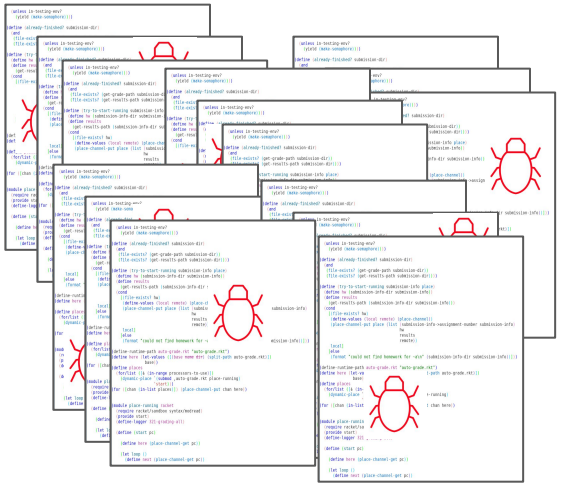
~~"Modes"~~ blame

Creating an Experiment to Test Our Hypothesis

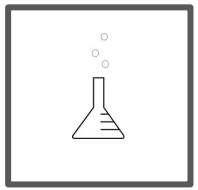


[Greenman 2022]

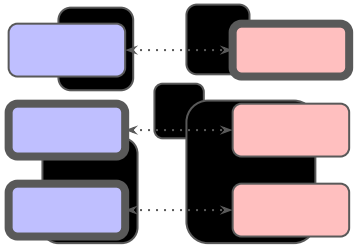
[Greenman et al. 2019, Greenman 2023]



Mutation
[Lipton 1971, DeMillo et al. 1978]

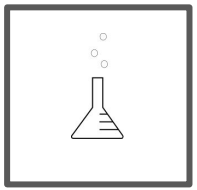


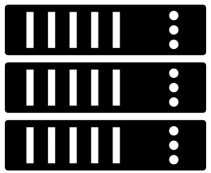
configuration



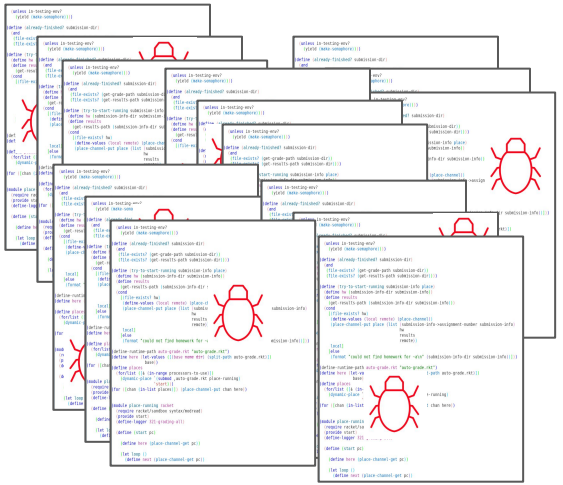
… for types

Creating an Experiment to Test Our Hypothesis



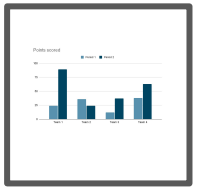


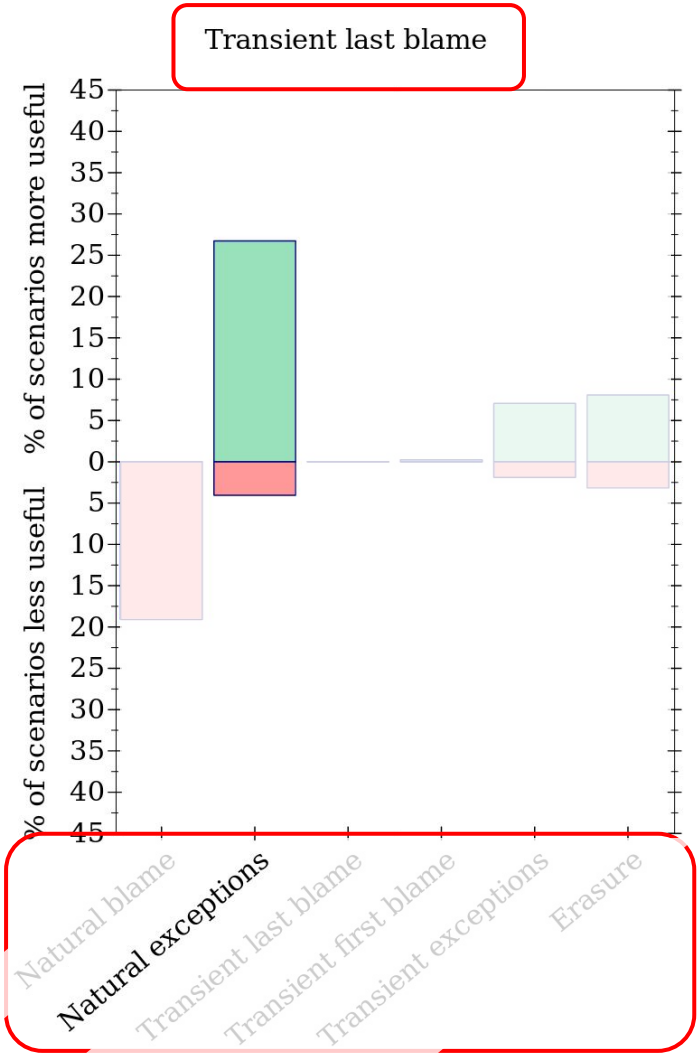
[... details omitted ...]



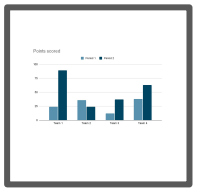
Mutation
[Lipton 1971, DeMillo et al. 1978]

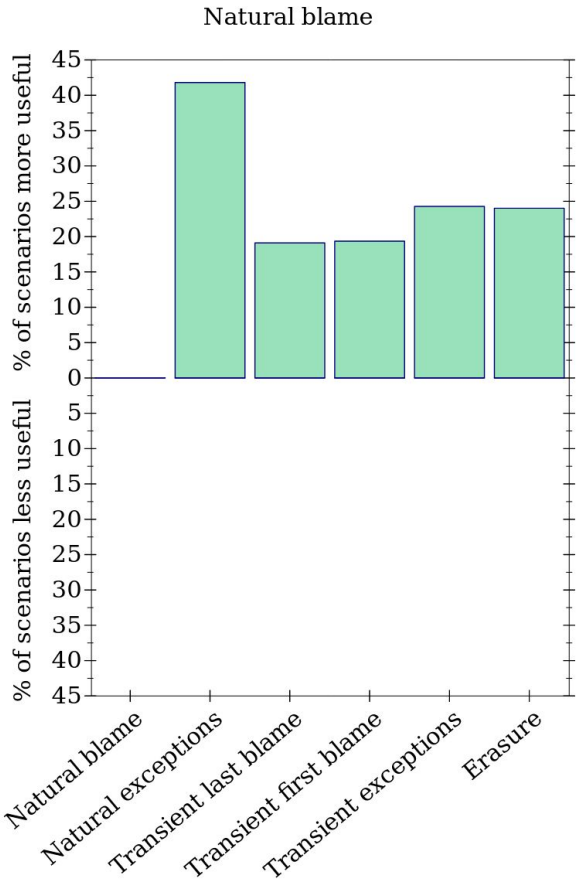
Results

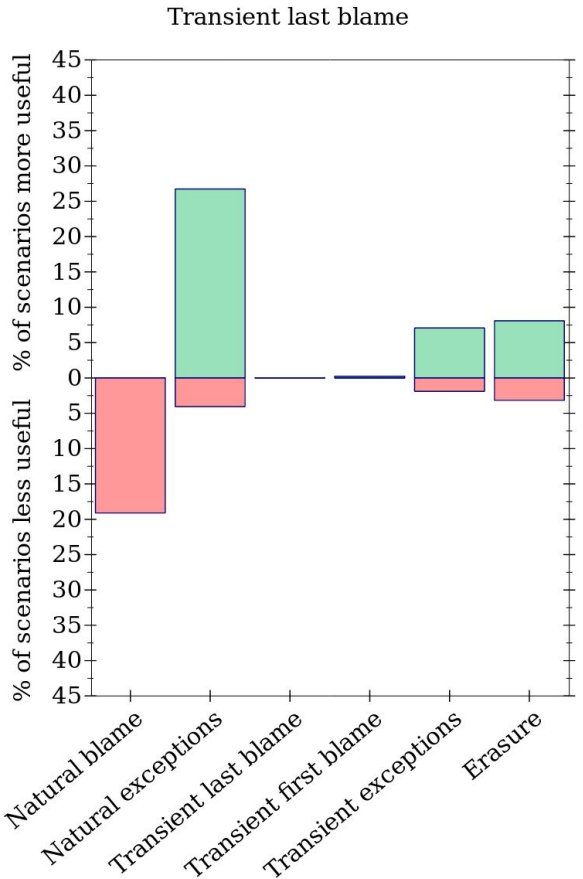



Transient last blame
% of scenarios more useful
% of scenarios less useful
45
40
35
30
25
20
15
10
5
0
5
10
15
20
25
30
35
40
45
Natural blame
Natural exceptions
Transient last blame
Transient first blame
Transient exceptions
Erasure

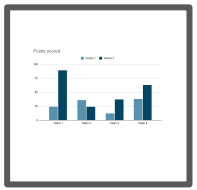
How Blame Stacks Up

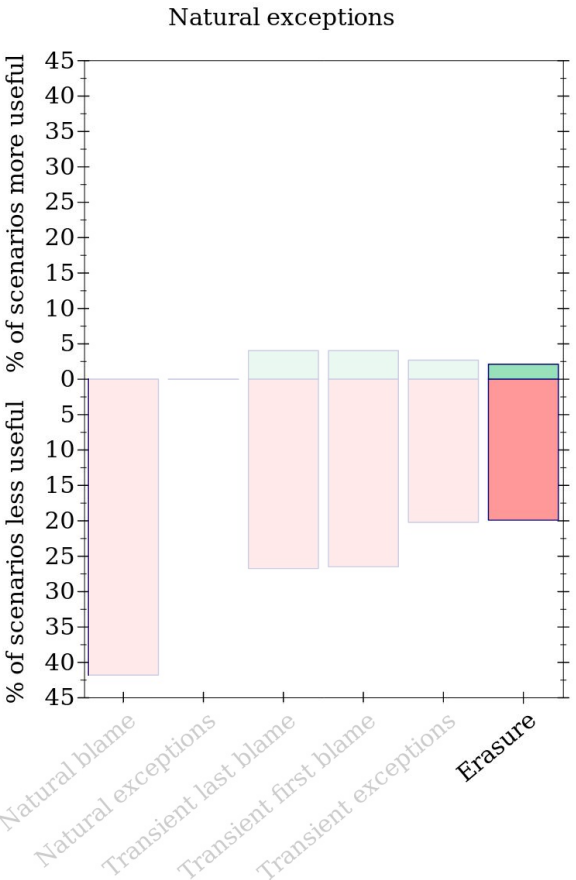


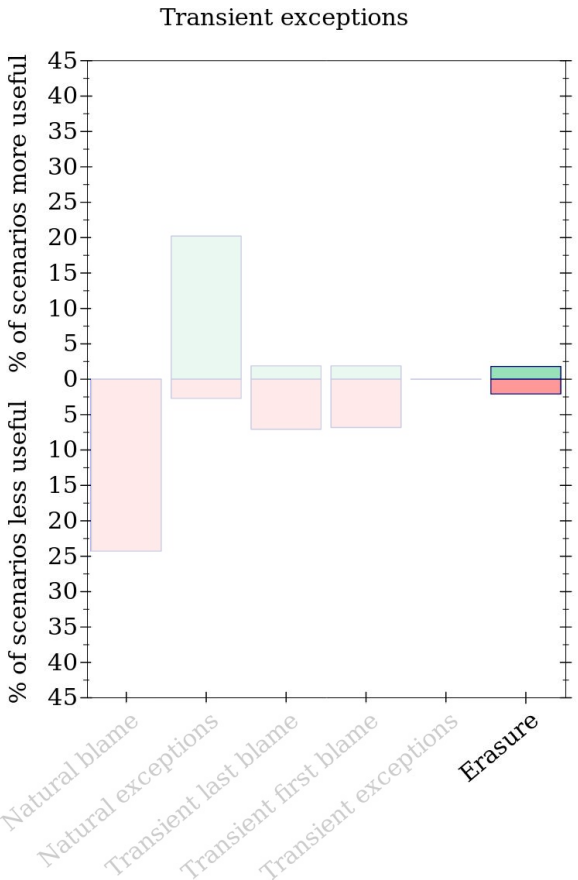




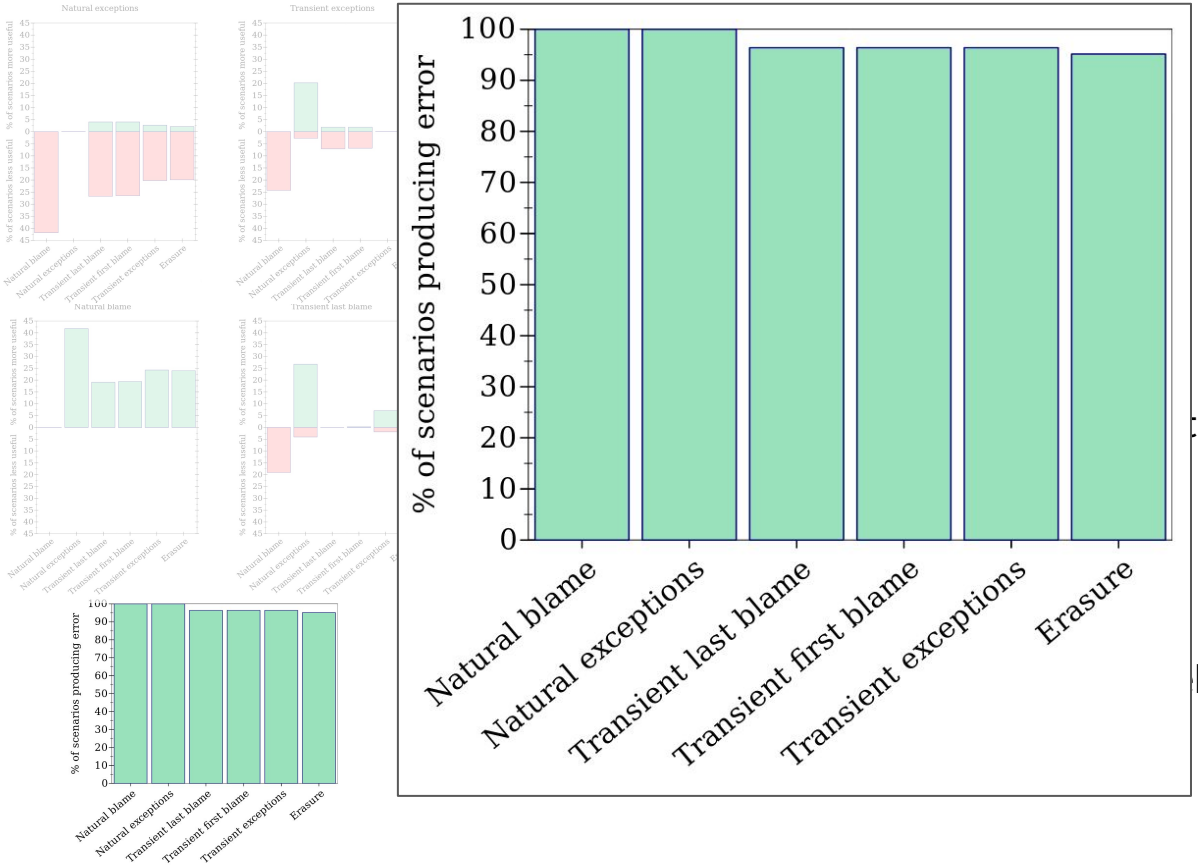
Checks *Without Blame* Don't







Takeaways

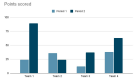


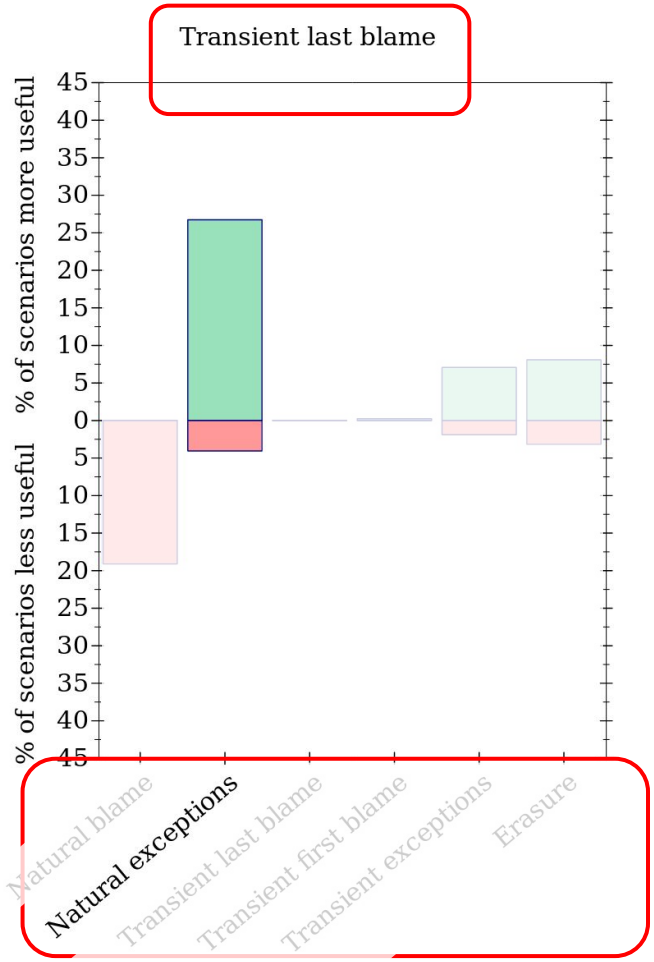
make sense?

the location of bugs.

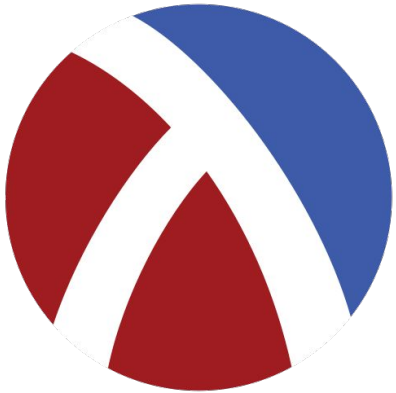
bugging mode be a useful

Results




Transient last blame
% of scenarios more useful
% of scenarios less useful
45
40
35
30
25
20
15
10
5
0
5
10
15
20
25
30
35
40
45
Natural blame
Natural exceptions
Transient last blame
Transient first blame
Transient exceptions
Erasure

Creating an Experiment to Test Our Hypothesis

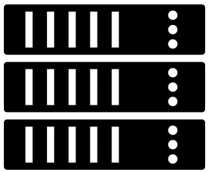


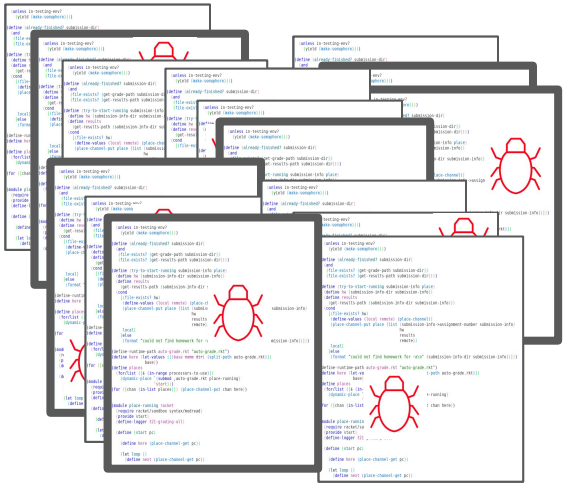
[Greenman 2022]

Erasure | Transient | Natural

Creating an Experiment to Test Our Hypothesis

30,000 scenarios





~300 mutants

2 million scenarios
(mutant ✕ configuration)

Erasure Detects Most of the Bugs