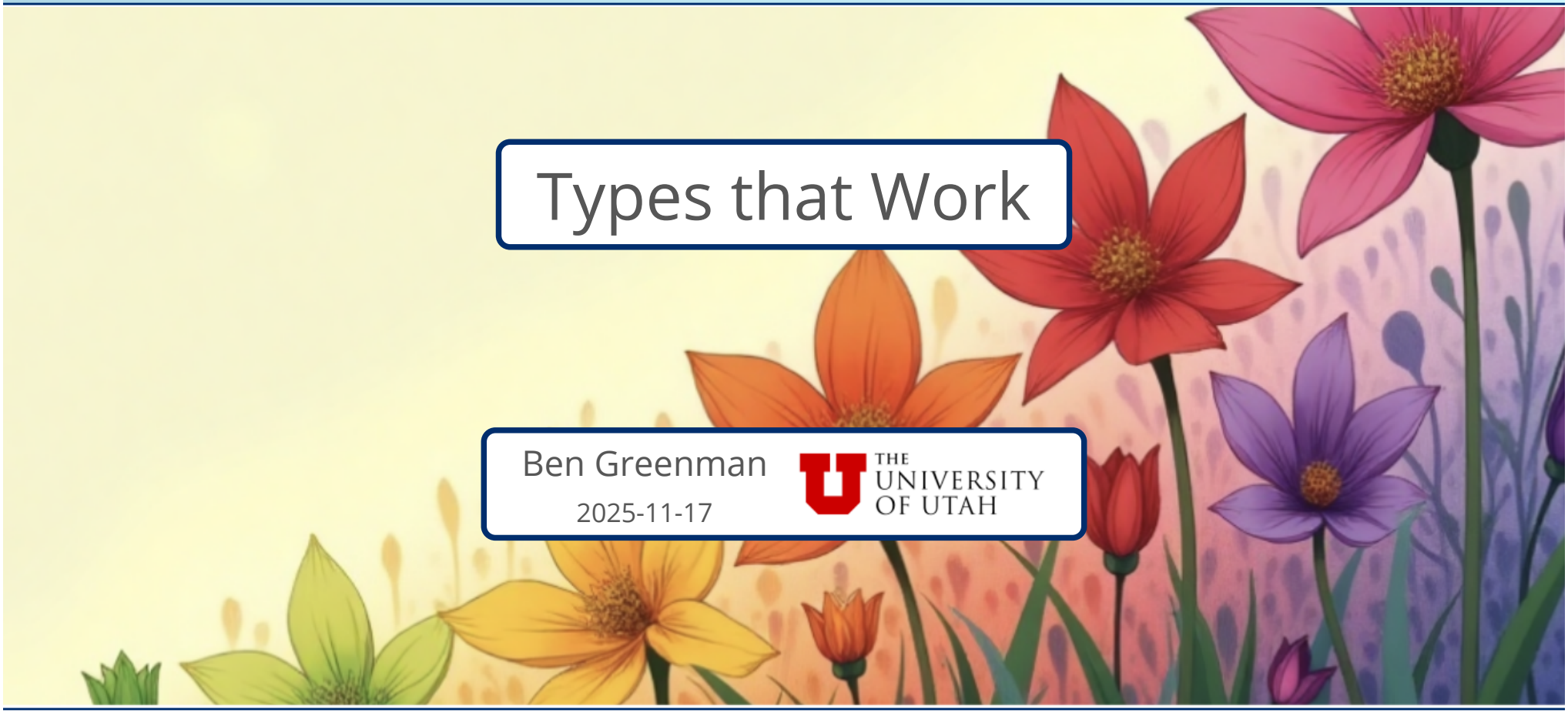
Types that Work

Ben Greenman

THE UNIVERSITY OF UTAH

2025-11-17



Research is when it can fail.

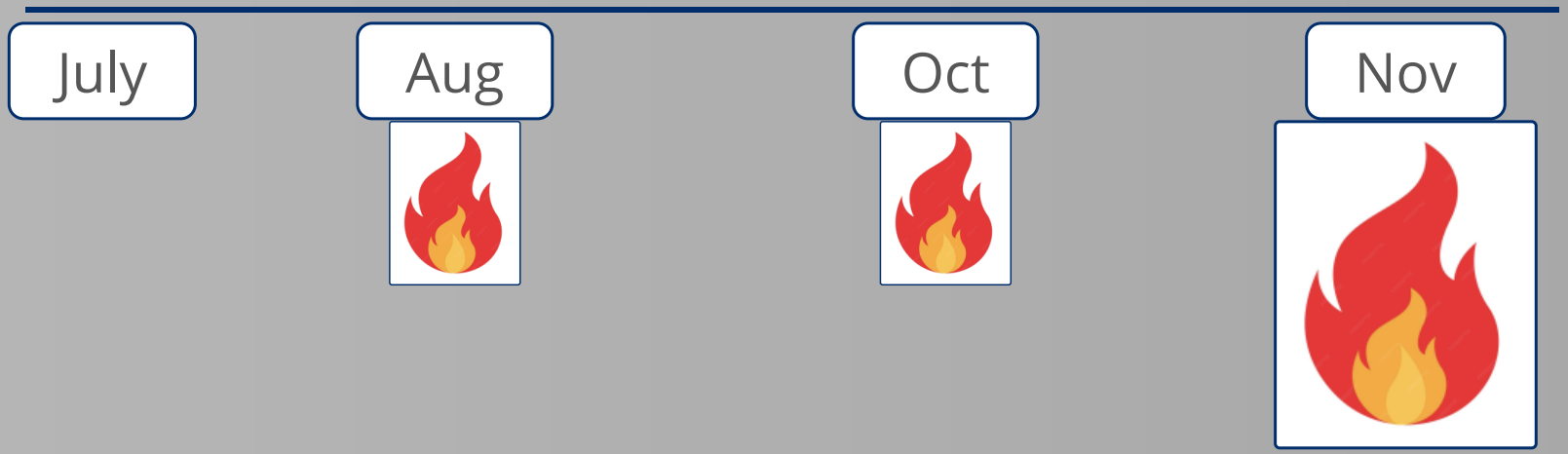

Guy Fieri
@GuyFieri
how it started
how its going
4:13 PM · Oct 12, 2020 · Twitter for iPhone

July

Nov

July

Nov


July
Aug
Oct
Nov

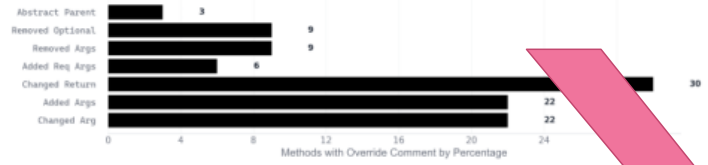


Fig. 2. Subclass Method Overrides from .pyi files

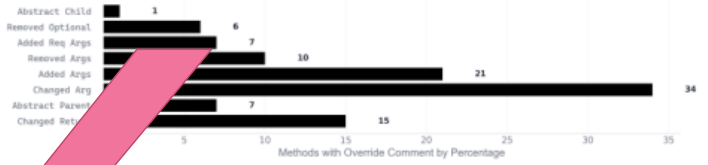


Fig. 3. Subclass Method Overrides from .py files

Fig. 4. Top Tvar Transformations from .py files

Fig. 5. Top TVar Transformations from .pyi files

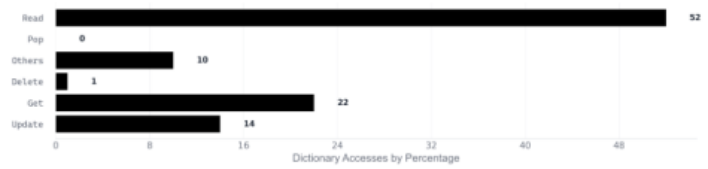


Fig. 6. Dep Dict Access Patterns

Fig. 7. Dep Callable Instances by Context

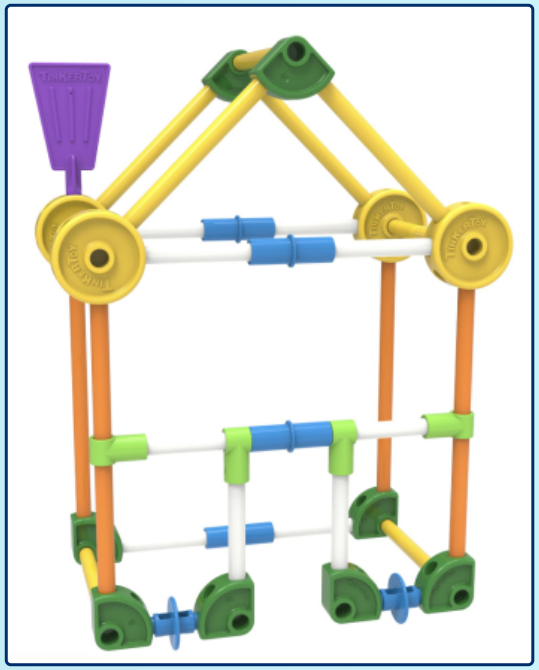
What went wrong?

What went wrong?

Communication!

Software is complicated.
Types help, but ...

… but Types are limiting.

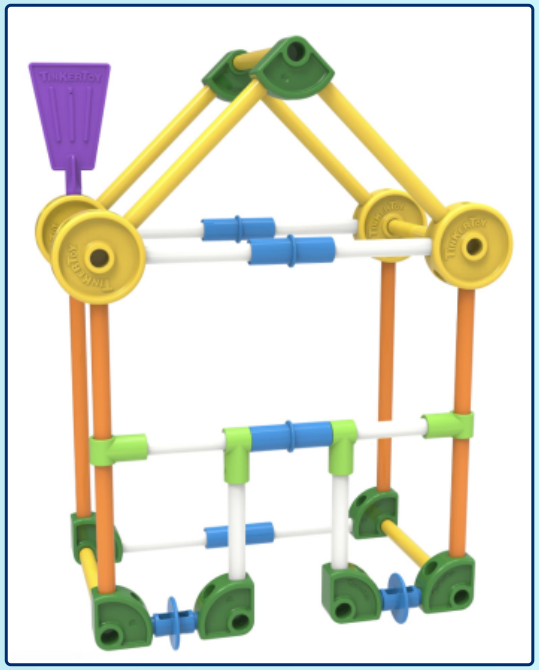


... but Types are limiting.

```
def div(a, b):
  return a / b if b != 0 else "div0"
```

vs.

```
enum DivRes = Ok(int) | Err(str)

def div(a, b) -> DivRes:
  return Ok(a / b) if b != 0 else Err("div0")
```



Gradual Typing

Types where you need them, that's all!

```
def div(a: int, b: int) -> Any:
  return a / b if b != 0 else "div0"
```

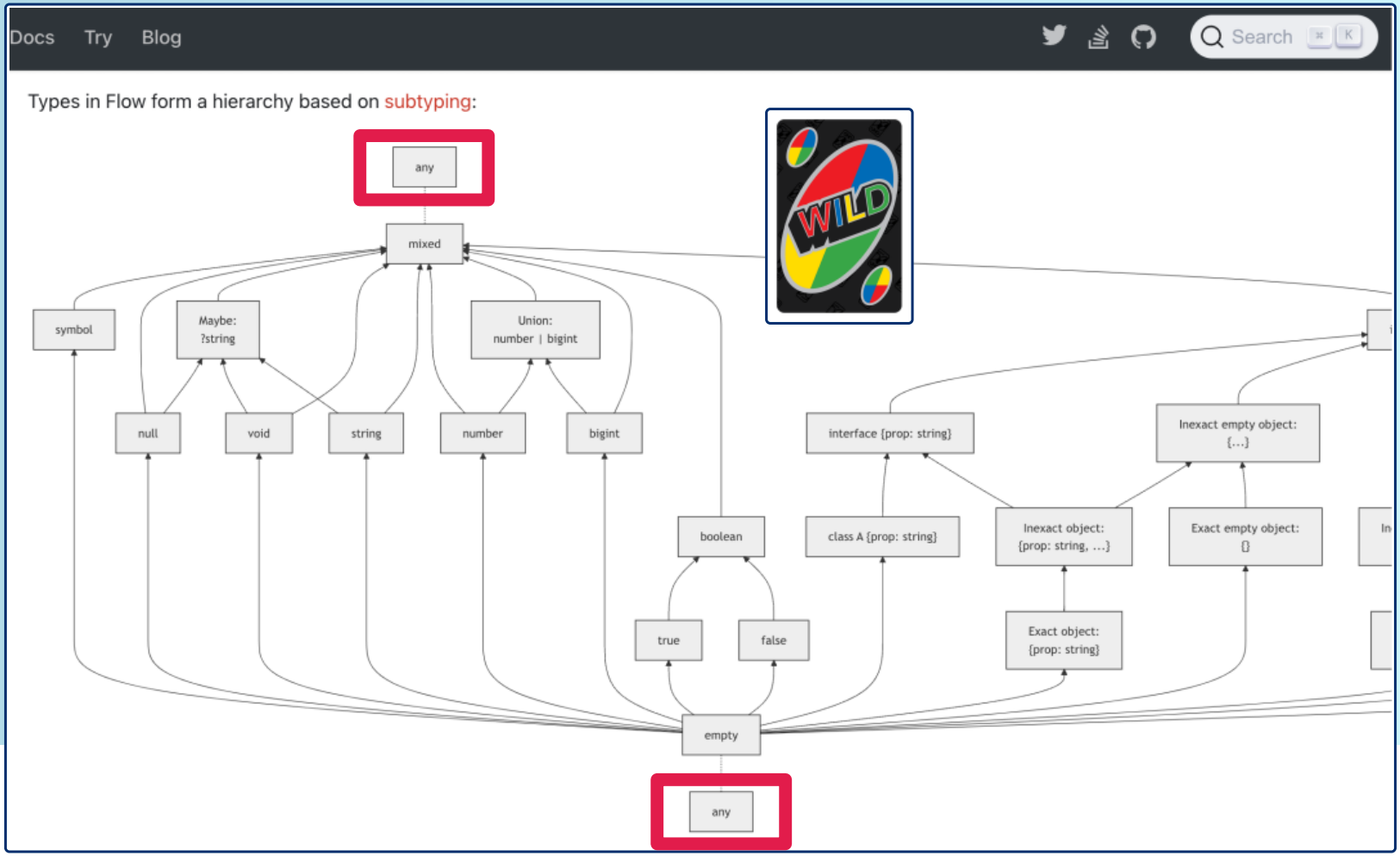
TS

Definitely Typed

34 million clients!

Problem:

Gradual Types **<** Types


Docs
Try
Blog
Search
Types in Flow form a hierarchy based on subtyping:
any
mixed
symbol
Maybe:
?string
Union:
number | bigint
null
void
string
number
bigint
boolean
true
false
empty
any
WILD
interface {prop: string}
class A {prop: string}
Inexact object:
{prop: string, ...}
Exact object:
{prop: string}
Inexact empty object:
{...}
Exact empty object:
{}

```python
def div(a: int, b: int) -> Any:
  return a / b if b != 0 else "div0"

div(x, y).helloworld()
# No type error!!!
```

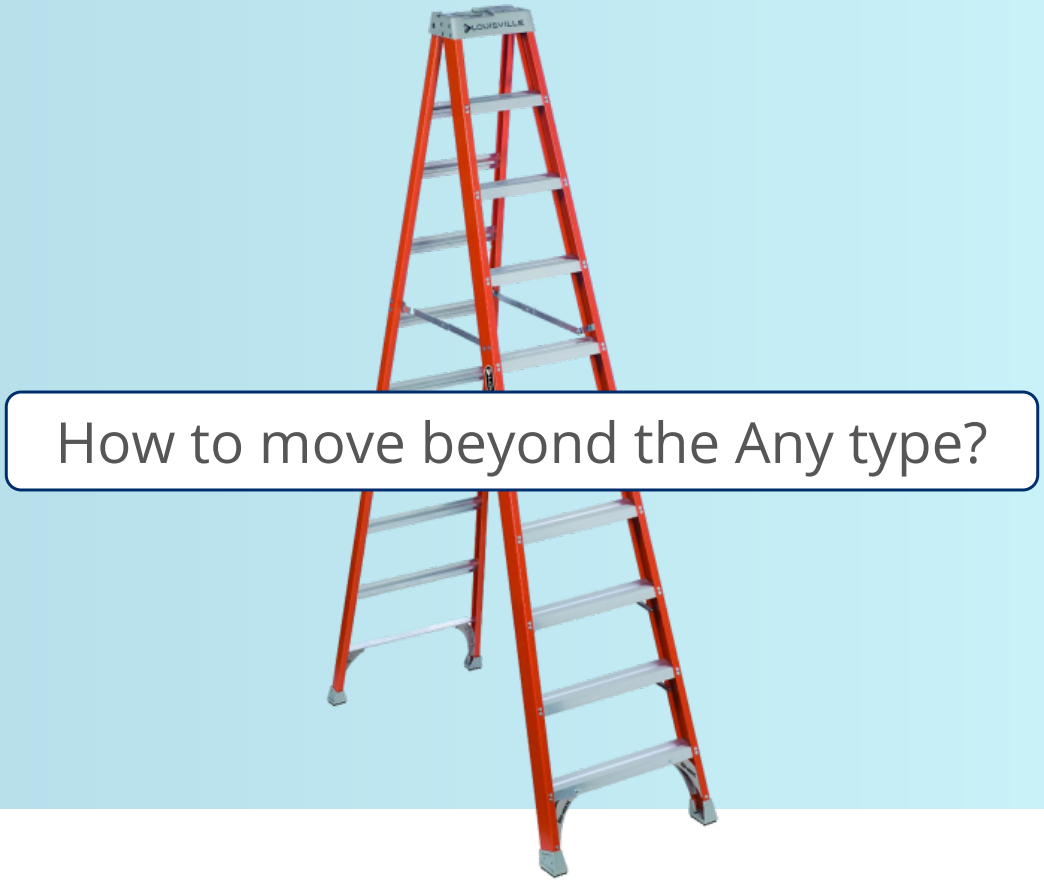
```
def div
  retur                                    y:
                                             "div0"

div(x,
# No ty
```

```
function query(d : DataFrame, row_pos: Int) : Row {

    // typed function, can we optimize the body?
    // NO.

}
```

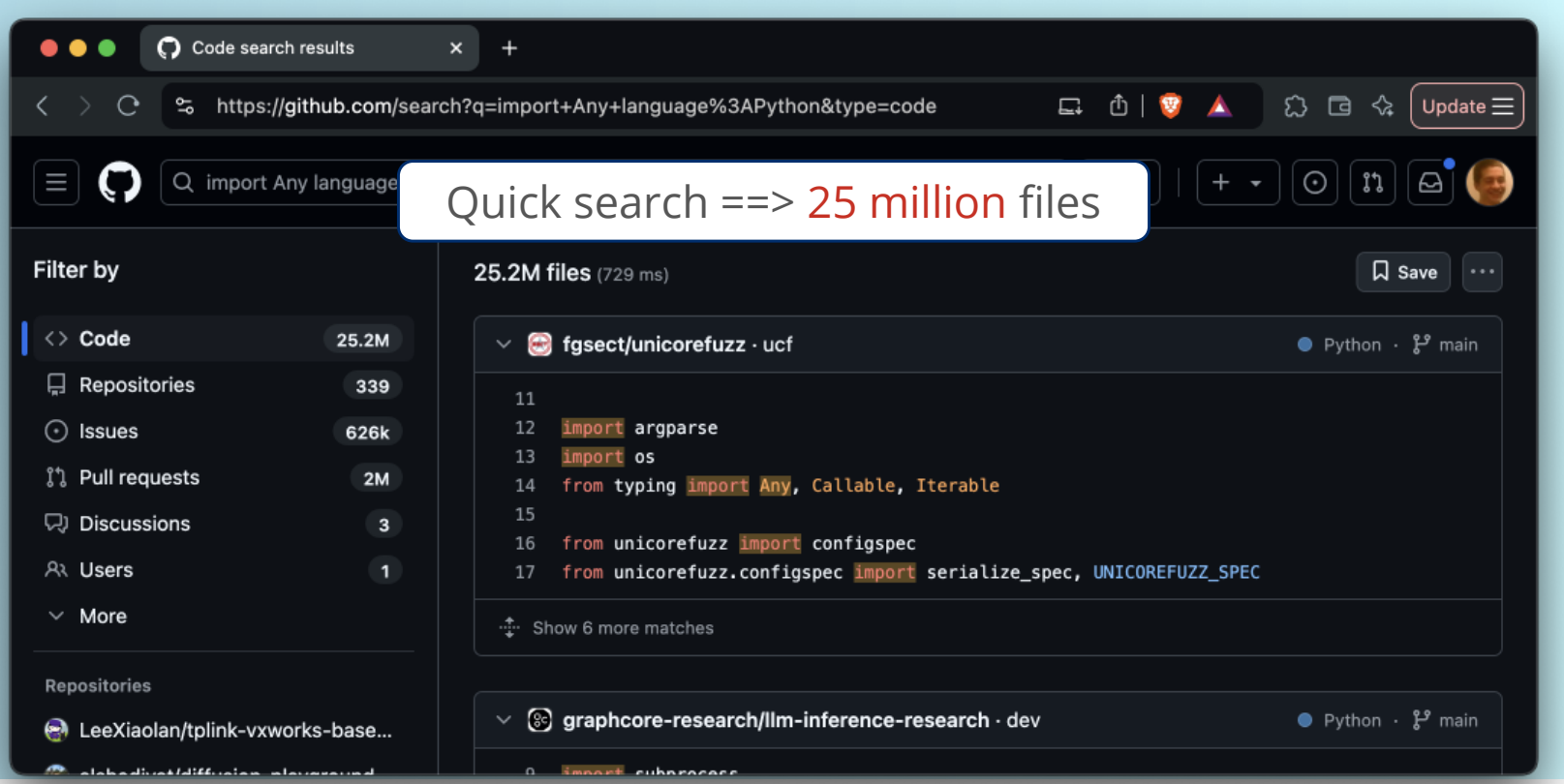

```
function query(d : DataFrame, row_pos: Int) : Row {

    // typed function, can we optimize the body?
    // NO.

}
```

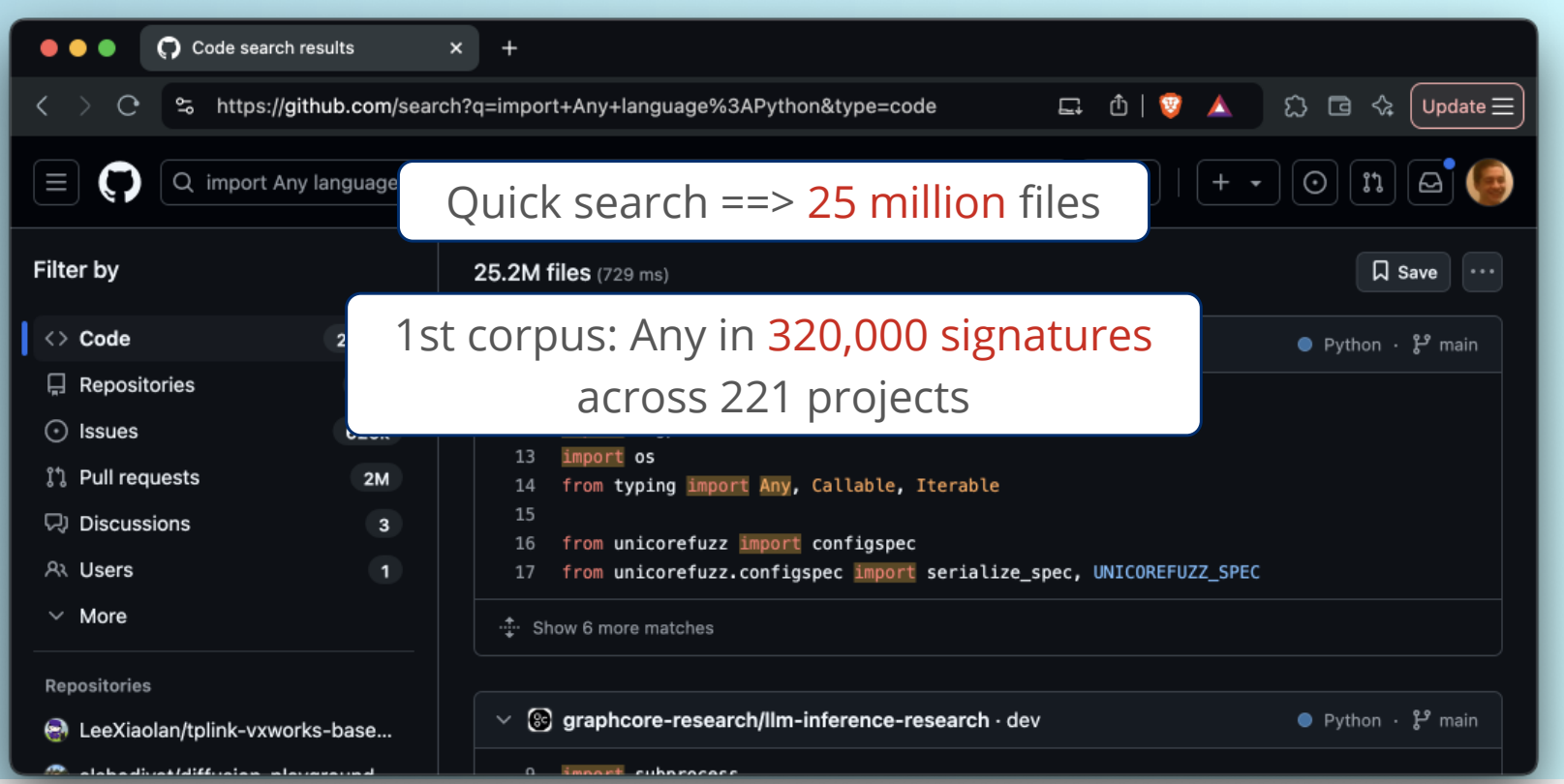

```
query("hello world", 99)  ==>  ??!
```

TypeScript does not protect types.


How to move beyond the Any type?

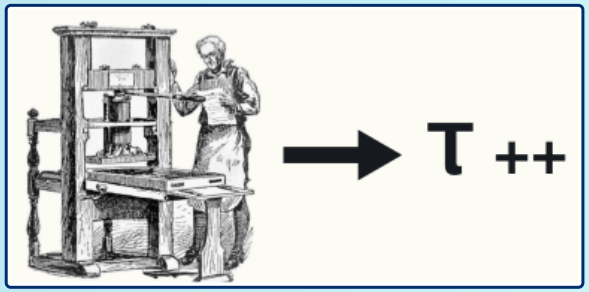

Any
is an opportunity


Code search results
https://github.com/search?q=import+Any+language%3APython&type=code
Update
import Any language
Quick search ==> 25 million files
Filter by
Code 25.2M
Repositories 339
Issues 626k
Pull requests 2M
Discussions 3
Users 1
More
Repositories
LeeXiaolan/tplink-vxworks-base...
25.2M files (729 ms)
Save
fgsect/unicorefuzz · ucf
Python · main
11
12 import argparse
13 import os
14 from typing import Any, Callable, Iterable
15
16 from unicorefuzz import configspec
17 from unicorefuzz.configspec import serialize_spec, UNICOREFUZZ_SPEC
Show 6 more matches
graphcore-research/llm-inference-research · dev
Python · main


Code search results
https://github.com/search?q=import+Any+language%3APython&type=code
Update
import Any language
Quick search ==> 25 million files
Filter by
25.2M files (729 ms)
Save
1st corpus: Any in 320,000 signatures
across 221 projects
Code
Repositories
Issues
Pull requests
2M
Discussions
3
Users
1
More
Python · main
13 import os
14 from typing import Any, Callable, Iterable
15
16 from unicorefuzz import configspec
17 from unicorefuzz.configspec import serialize_spec, UNICOREFUZZ_SPEC
Show 6 more matches
Repositories
LeeXiaolan/tplink-vxworks-base...
graphcore-research/llm-inference-research · dev
Python · main

Goal: Data Science for the Any Type

+ recognize patterns
+ set priorities
+ improve type precision



6 Major Patterns

Self

Dep Dict

Dep Callable

Top TVar

Loose TVar

Override

Self

```
class Shape:
  def move(self: Any, dist: int) -> Any: # imprecise return
    self.position += dist
    return self

class Circle(Shape):
  pass

Circle().move(4) # ideally, a Circle
```

Dep Dict

```python
def add_tax(item: Dict[str, Any]) -> float:
  base = item.get("price", 0) # Any
  return base + (base * 0.10)
```

Dep Callable

```
def outer(fn: Callable) -> Callable: # imprecise
  def inner(self, s1: str, *args, **kwargs) -> Callable:
    s2 = string_to_stat(s1)
    return fn(self, s2, *args, **kwargs)
  return inner
```

Top TVar

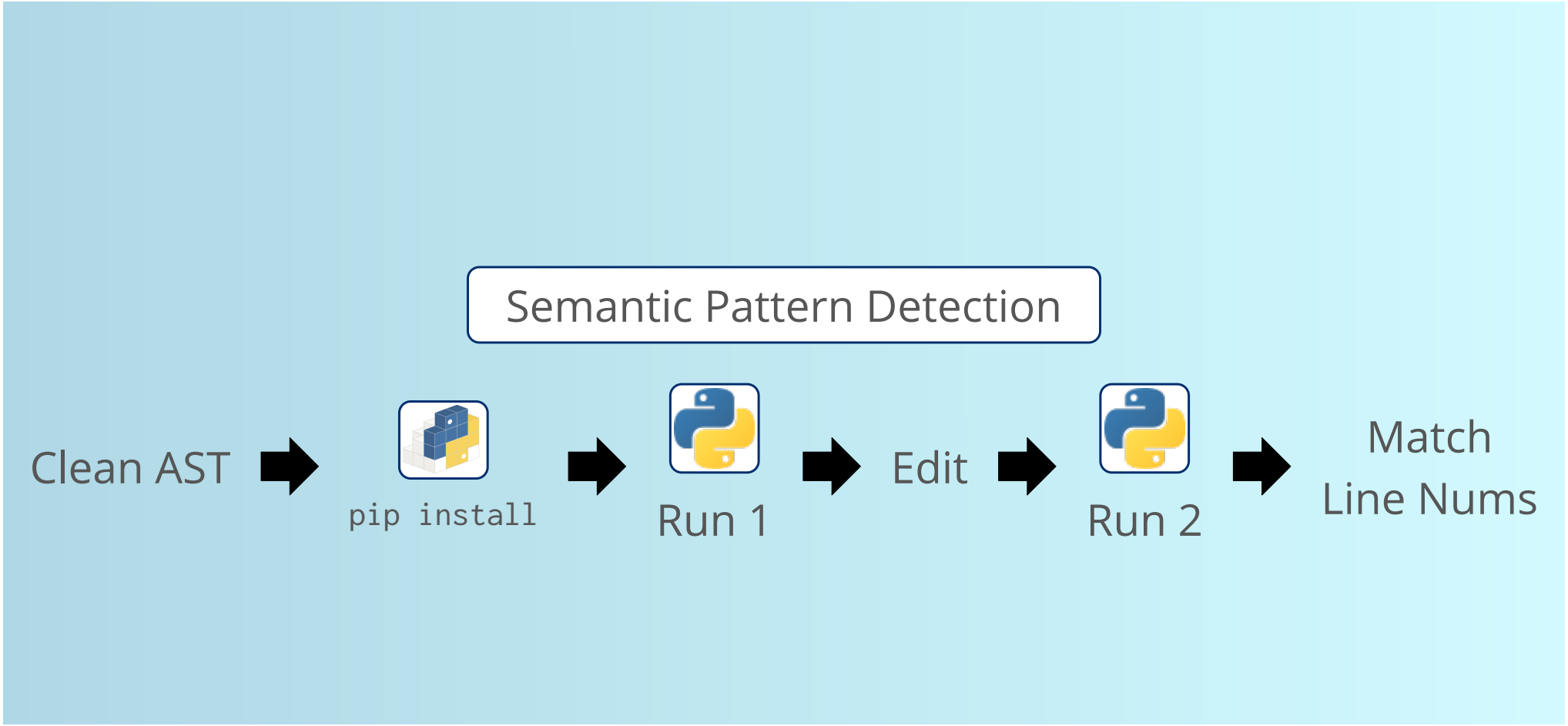
```
def eq(a: Any, b: Any) -> bool:
  return a == b
```

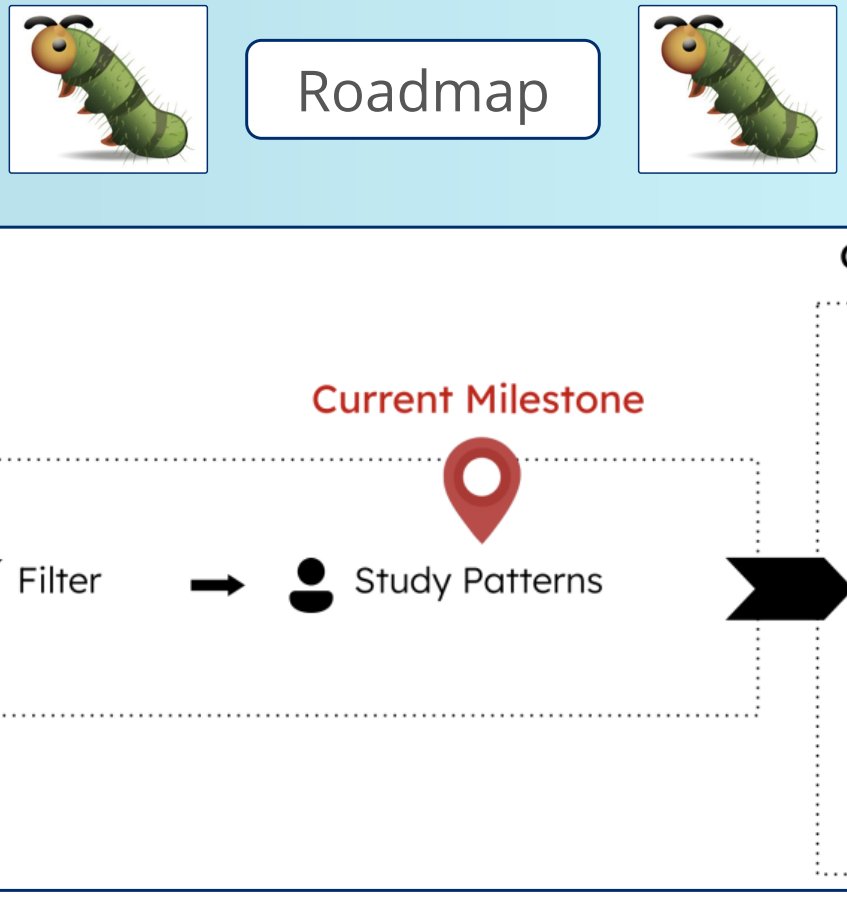
Loose TVar

```
Car = TypeVar('Car') # TVar without bound

def add_car(x: List[Car], car: Car):
  x.append(car)

add_car(["FJ40", "Baja Buggy"], 5) # NOT a type error
```

Override

```
class BinaryIO(IO[bytes]):
  def write(self, s: bytes | bytearray) -> int:
    pass

class FileOpener(BinaryIO):
  def write(self, s: bytes) -> int: # type: ignore[override]
    return self.fdesc.write(s)
```


Semantic Pattern Detection
Clean AST
pip install
Run 1
Edit
Run 2
Match
Line Nums


Roadmap
Corpus Study
79000
REPOS
Pre-corpus Study
Current Milestone
221
REPOS
Filter
Study Patterns
Analysis Engine
τ++

Early Results for 2,917 Projects

Pattern	Projects	Instances
Self	126	252
Dep Dict	1,295	41,671
Dep Call	496	1,296
Top TV	1,968	110,505
Loose TV	1,129	8,981
Override	368	3,474

Over-Counting Top TVars ...

9% garbage, T -> _

50% questionable, _ -> T

Pattern	Projects	Instances
Self	126	252
Dep Dict	1,295	41,671
Dep Call	496	1,296
Top TV	1,968	110,505
Loose TV	1,129	8,981
Override	368	3,474

Under-Counting Loose TVars ...

Generic functions and classes can be parameterized by using type parameter syntax:

```
from collections.abc import Sequence

def first[T](l: Sequence[T]) -> T:  # Function is generic over the TypeVar "T"
    return l[0]
```

Or by using the TypeVar factory directly:

We support only **TVar**

Challenge: Typing Control Flow

```
if x is an Int:
  x + 1
```

Rainfall Problem

Compute average rainfall from a list of possibly-faulty weather reports.

Ignore data that is non-numeric, too high (>999), or too low (<0).

Rainfall Problem

Compute average rainfall from a list of possibly-faulty weather reports.

Ignore data that is non-numeric, too high (>999), or too low (<0).

```
def rainfall(data : List[JSON]) -> Number:
  let total = 0, count = 0
  for report in data:
    if report is Object:
      let val = report["rainfall"]
      if val is Number and 0 <= val <= 999:
        total += val
        count += 1
  return total / count  # assuming count >0
```

```
def rainfall(data : List[JSON]) -> Number:
  let total = 0, count = 0
  for report in data:
    if report is Object:
      let val = report["rainfall"]
      if val is Number and 0 <= val <= 999:
        total += val
        count += 1
  return total / count
```

Type Tests

Aliasing!

Data Structure

Type Narrowing

```python
def fst(a : tuple[object, object]) -> int:
  if type(a[0]) is int:
    return a[0] + 1
```

```python
def filter_nums(bs: list[object]) -> list[int]:
  return [b for b in bs if type(b) is int]
```

```python
class Node:
  parent : Node | None
  ...

def score(n: Node) -> int:
  if node.parent is not None:
    return node.parent.wins + node.parent.losses
  ...
```

Type Narrowing

```python
def fst(a : tuple[object, object]) -> int:
  if type(a[0]) is int:
    return a[0] + 1
```

```python
def filter_nums(bs: list[object]) -> list[int]:
  return [b for b in bs if type(b) is int]
```

```python
class Node:
  parent : Node | None
  ...

def score(n: Node) -> int:
  if node.parent is not None:
    return node.parent.wins + node.parent.losses
  ...
```

Type Narrowing

```python
def fst(a : tuple[object, object]) -> int:
  if type(a[0]) is int:
    return a[0] + 1
```

```python
def filter_nums(bs: list[object]) -> list[int]:
  return [b for b in bs if type(b) is int]
```

```python
class Node:
  parent : Node | None
  ...

def score(n: Node) -> int:
  if node.parent is not None:
    return node.parent.wins + node.parent.losses
  ...
```

Type Narrowing

```python
def fst(a : tuple[object, object]) -> int:
  if type(a[0]) is int:
    return a[0] + 1
```

```python
def filter_nums(bs: list[object]) -> list[int]:
  return [b for b in bs if type(b) is int]
```

```python
class Node:
  parent : Node | None
  ...

def score(n: Node) -> int:
  if node.parent is not None:
    return node.parent.wins + node.parent.losses
  ...
```

Design Space of Type Narrowing

Set-Theoretic Types

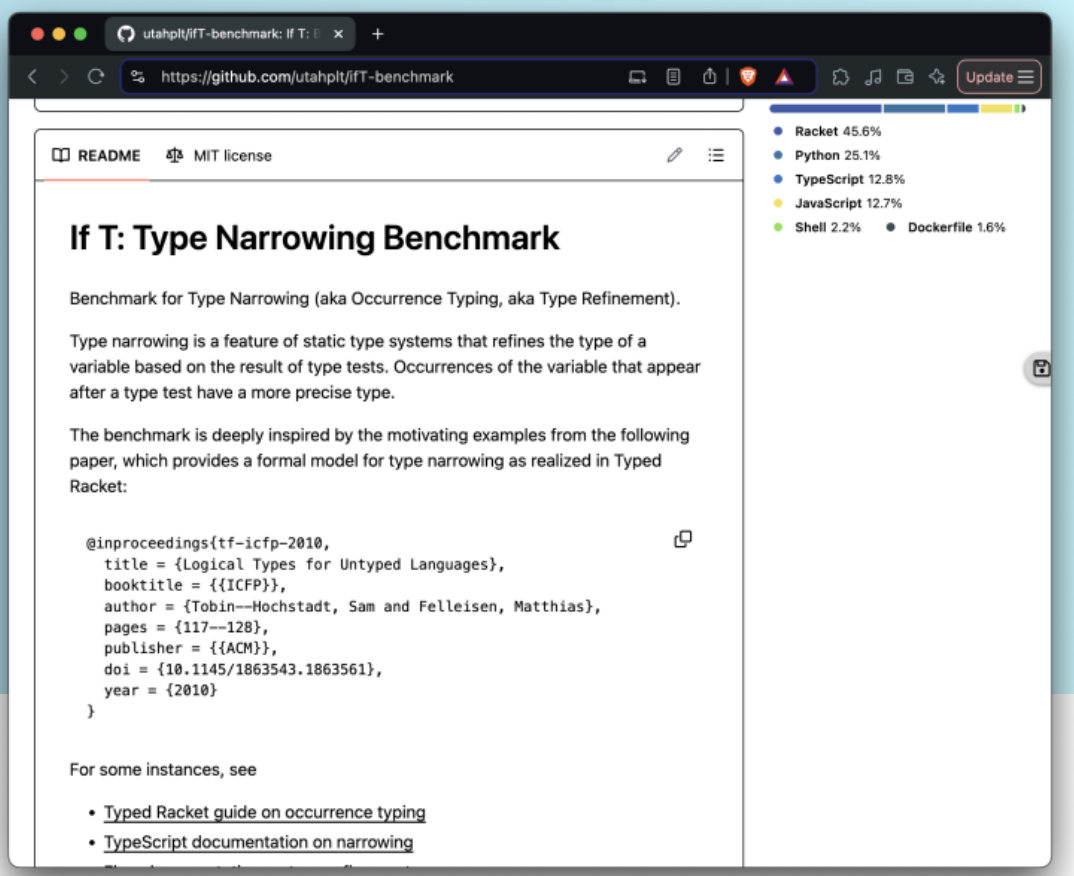
$T := T \cup T \mid T \cap T \mid \neg T$

Occurrence Typing

$\Gamma \vdash e : \tau;\ \varphi^{+} \mid \varphi^{-};\ o$

??

github.com/utahplt/ift-benchmark



`github.com/utahplt/ift-benchmark`

4 core dimensions
  * 13 benchmark items
    pass & fail tests

+ practical examples
+ datasheet

	Basic Narrowing:	
1.	positive	Refine when condition is true
2.	negative	Refine when condition is false
3.	connectives	Handle logic connectives: not, or, and
4.	nesting_body	Conditionals nested within branches
	Compound Structures:	
5.	struct_fields	Refine (immutable) structure fields
6.	tuple_elements	Refine tuple elements
7.	tuple_length	Refine based on tuple size
	Advanced Control Flow:	
8.	alias	Track logical implication through variables
9.	nesting_condition	Conditionals nested within conditions
10.	merge_with_union	Correctly merge control-flow branches
	Custom Predicates:	
11.	predicate_2way	Custom predicates that narrow positively and negatively
12.	predicate_1way	Custom predicates that narrow only positively
13.	predicate_checked	Typecheck the body of custom predicates

	Basic Narrowing:	
1.	positive	Refine when condition is true
2.	negative	Refine when condition is false
3.	connectives	Handle logic connectives: not, or, and
4.	nesting_body	Conditionals nested within branches
	Compound Stru	
5.	struct_fields	s
6.	tuple_elements	
7.	tuple_length	
	Advanced Contr	
8.	alias	variables
9.	nesting_condition	Conditionals nested within conditions
10.	merge_with_union	Correctly merge control-flow branches
	Custom Predicates:	
11.	predicate_2way	Custom predicates that narrow positively and negatively
12.	predicate_1way	Custom predicates that narrow only positively
13.	predicate_checked	Typecheck the body of custom predicates

Intentionally left out:

- Subtyping
- Mutation
- Concurrency

If-T Pseudocode

nesting_condition: Success

```
define f(x: Top, y: Top) -> Number:
  if (if x is Number: y is String else: false):
    return x + String.length(y)
  else:
    return 0
```

nesting_condition: Failure

```
define f(x: Top, y: Top) -> Number:
  if (if x is Number: y is String else: y is String):
    return x + String.length(y)
  else:
    return 0
```

Table 2 Benchmark Results: ● = passed, × = failed imprecisely, × = failed unsoundly

		Typed Racket	TypeScript	Flow	Mypy	Pyright
	Basic Narrowing:					
1.	positive	●	●	●	●	●
2.	negative	●	●	●	●	●
3.	connectives	●	●	●	●	●
4.	nesting_body	●	●	●	●	●
	Compound Structures:					
5.	struct_fields	●	●	●	●	●
6.	tuple_elements	●	●	●	●	●
7.	tuple_length	×	●	●	●	●
	Advanced Control Flow:					
8.	alias	●	●	×	×	●
9.	nesting_condition	●	×	×	×	×
10.	merge_with_union	●	●	●	×	●
	Custom Predicates:					
11.	predicate_2way	●	●	●	●	●
12.	predicate_1way	●	×	●	●	●
13.	predicate_checked	●	×	●	×	×

Custom Predicates

```
function is_num(x: any)
 : x is number
```

```
def is_num(x) -> TypeIs(int)
```

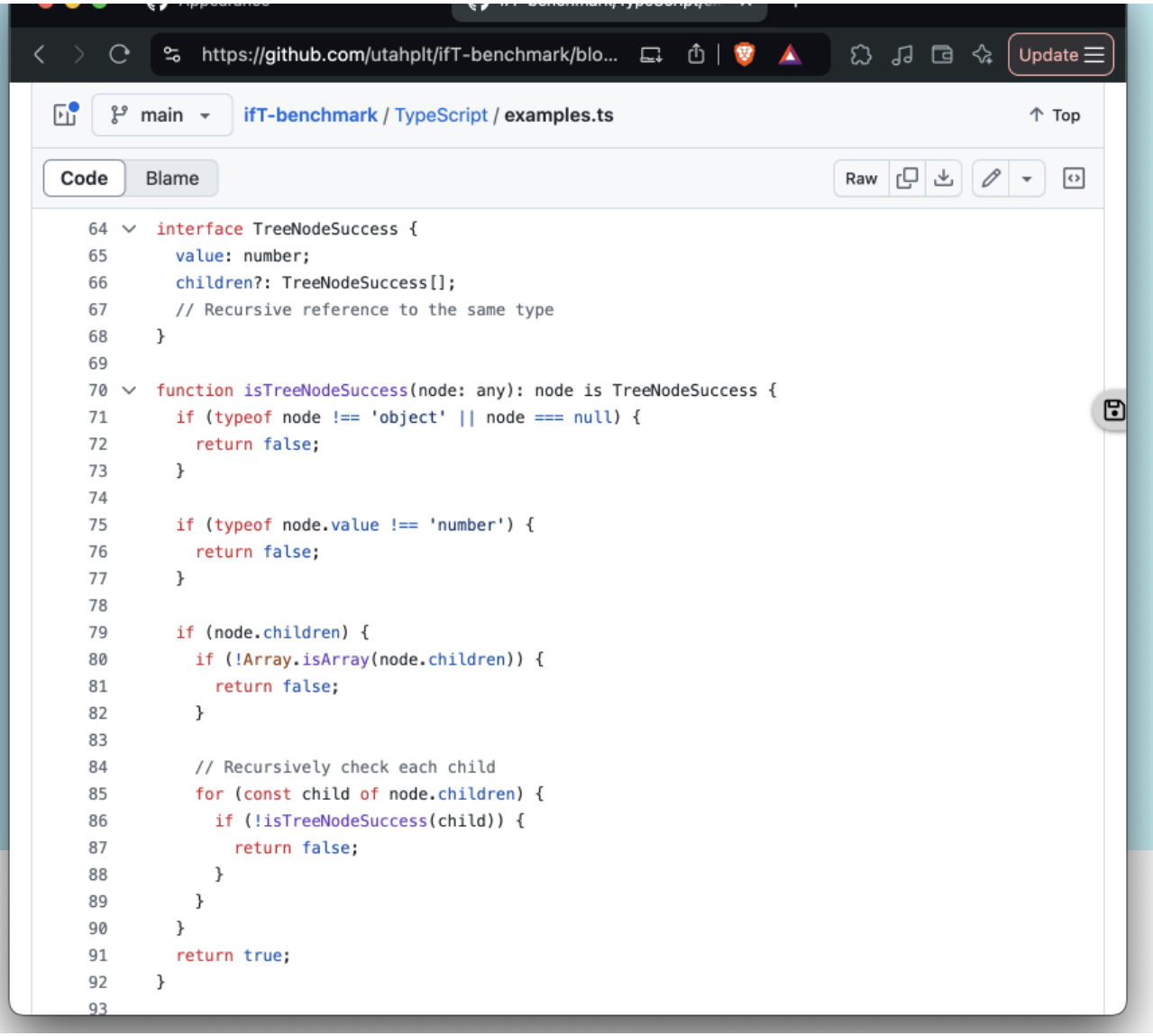
```
function is_even(x: any)
 : implies x is number
 // Flow, not TypeScript
```

```
def is_even(x) -> TypeGuard(int)
```

Custom Predicates

```
opt-proposition =
                | : type
                | : pos-proposition
                    neg-proposition
                  object

pos-proposition =
                | #:+ proposition ...

neg-proposition =
                | #:- proposition ...

         object =
                | #:object index
```

```
proposition = Top
            | Bot
            | type
            | (! type)
            | (type @ path-elem ... index)
            | (! type @ path-elem ... index)
            | (and proposition ...)
            | (or proposition ...)
            | (implies proposition ...)

  path-elem = car
            | cdr

      index = positive-integer
            | (positive-integer positive-integer)
            | identifier
```

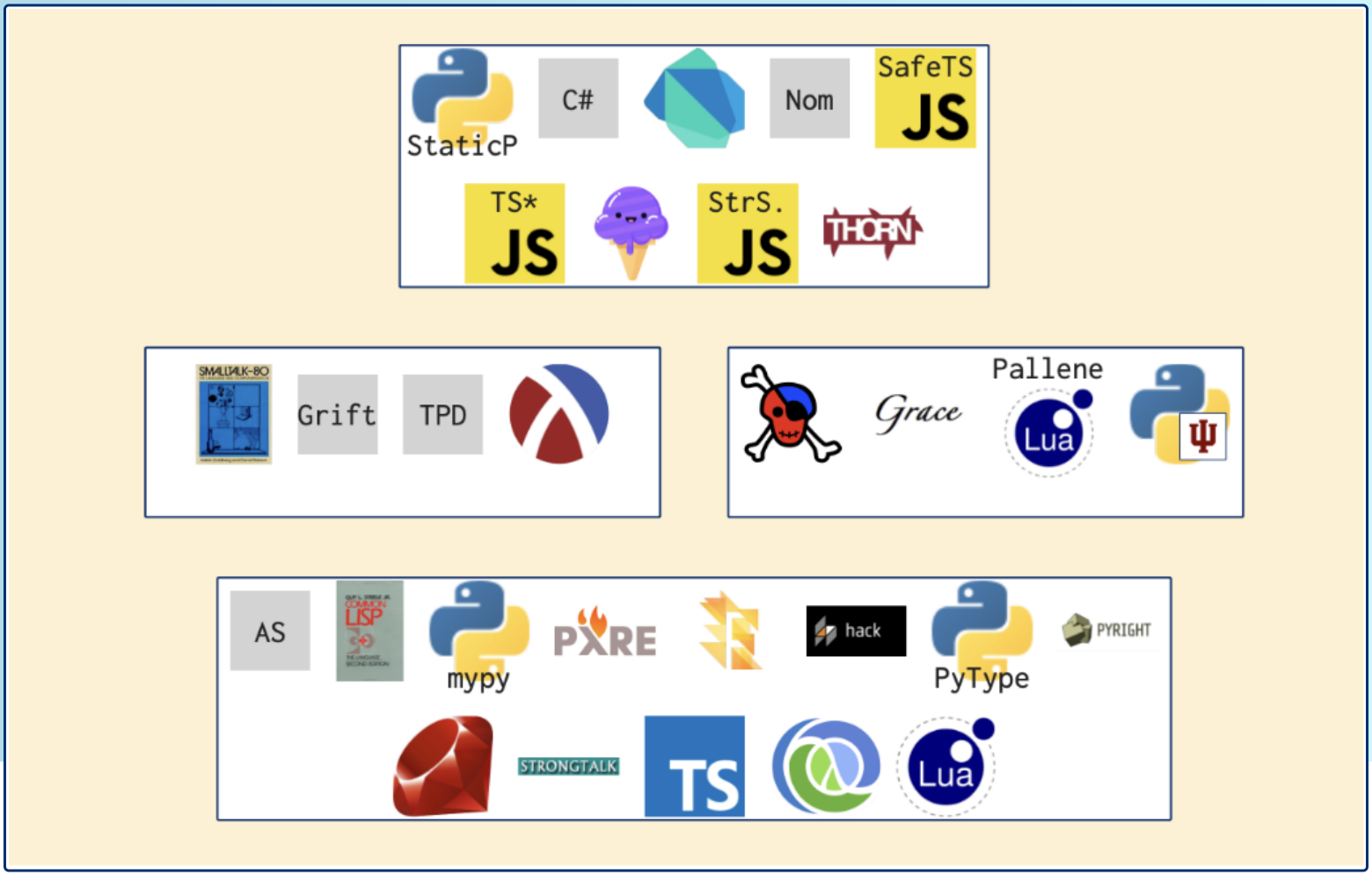



```typescript
interface TreeNodeSuccess {
  value: number;
  children?: TreeNodeSuccess[];
  // Recursive reference to the same type
}

function isTreeNodeSuccess(node: any): node is TreeNodeSuccess {
  if (typeof node !== 'object' || node === null) {
    return false;
  }

  if (typeof node.value !== 'number') {
    return false;
  }

  if (node.children) {
    if (!Array.isArray(node.children)) {
      return false;
    }

    // Recursively check each child
    for (const child of node.children) {
      if (!isTreeNodeSuccess(child)) {
        return false;
      }
    }
  }
  return true;
}
```

Table 2 Benchmark Results: ● = passed, × = failed imprecisely, × = failed unsoundly

		Typed Racket	TypeScript	Flow	Mypy	Pyright
	Basic Narrowing:					
1.	positive	●	●	●	●	●
2.	negative	●	●	●	●	●
3.	connectives	●	●	●	●	●
4.	nesting_body	●	●	●	●	●
	Compound Structures:					
5.	struct_fields	●	●	●	●	●
6.	tuple_elements	●	●	●	●	●
7.	tuple_length	×	●	●	●	●
	Advanced Control Flow:					
8.	alias	●	●	×	×	●
9.	nesting_condition	●	×	×	×	×
10.	merge_with_union	●	●	●	×	●
	Custom Predicates:					
11.	predicate_2way	●	●	●	●	●
12.	predicate_1way	●	×	●	●	●
13.	predicate_checked	●	×	●	×	×

Whence gradual types?

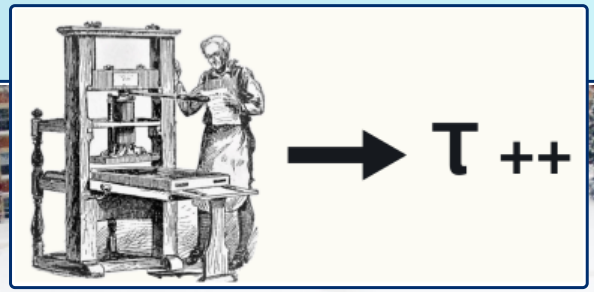

StaticP
C#
Nom
SafeTS
JS
TS*
JS
StrS.
JS
THORN
SMALLTALK-80
Grift
TPD
Grace
Pallene
Lua
AS
COMMON LISP
mypy
PXRE
hack
PyType
PYRIGHT
STRONGTALK
TS
Lua

Whence gradual types?

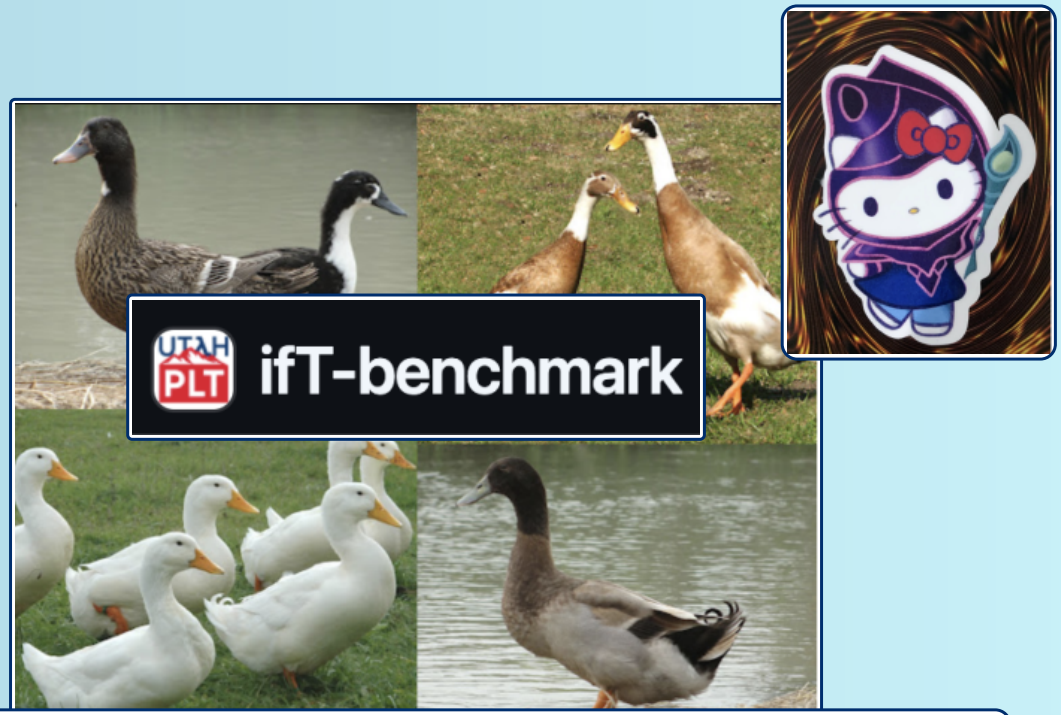
Two leading strategies:

1. +Dynamic

2. Wizardry



Understanding why Dynamic appears



Tools for fair cross-language comparison

PLT works on advanced programming languages, including language design, formal semantics, language implementation, and programming environments. Its members are spread throughout the world.

Utah PLT is part of the Kahlert School of Computing at the University of Utah.
GitHub: https://github.com/utahplt

⟩ People

Utah PLT's current and past team members.

⟩ Publications

Technical publications from PLT at Utah.