Gradual Soundness: Lessons from Static Python

Kuang-Chen Lu  Ben Greenman  Carl Meyer  Dino Viehland
Aniket Panse  Shriram Krishnamurthi

BROWN  Meta

〈Programming〉 2023
Static Python
Static Python

Enhanced Python, by Instagram

+2 years running in production
Static Python

Enhanced Python, by Instagram

+2 years running **in production**

Gradually typed

... for some value of **gradual**
What is Gradual Typing?
What is Gradual Typing?

Idea: combine the best parts of typed and untyped code
What is Gradual Typing?

Idea: combine the best parts of typed and untyped code

# Python code

```python
def join(d0,d1,sort,how):
    ....
```

so many parameters!
What is Gradual Typing?

Idea: combine the best parts of typed and untyped code

```python
# Python code

def join(d0, d1, sort, how):
    ....
```

- `DataFrame`
- `bool`
- `Left|Right`
What is Gradual Typing?

Idea: combine the best parts of typed and untyped code

# Python code

def join(d0, d1, sort, how):
    ....

# Python + Types

def join(d0: DataFrame, d1: DataFrame, sort: bool, how: Left|Right):
    -> DataFrame:
        ....
What is Gradual Typing?

Idea: combine the best parts of typed and untyped code

# Python code

```python
def join(d0, d1, sort):
    ....
```

# Python + Types

```python
def join(d0: DataFrame, d1: DataFrame, sort: bool, how: Left | Right):
    -> DataFrame:
    ....
```

Great!

But, **what happens** when typed code and untyped code interact?

Are types sound?
What is Gradual Typing?
What is Gradual Typing?

# Python + Types

```python
def join(d0: DataFrame,
         d1: DataFrame,
         sort: bool,
         how: Left | Right)
    -> DataFrame:
    ....
```

A1.

A2.

A3.
What is Gradual Typing?

A1. Optional static checks, nothing at run-time

A2.

A3.
What is Gradual Typing?

**A1.** Optional static checks, nothing at run-time

```python
# Python + Types
def join(d0: DataFrame,
    d1: DataFrame,
    sort: bool,
    how: Left|Right)
    -> DataFrame:
    ....
```

**A2.**

**A3.**
What is Gradual Typing?

A1. Optional static checks, nothing at run-time

```
join(42, "hola", ...)
```

How to debug?

A2.

A3.
What is Gradual Typing?

A1. Optional static checks, nothing at run-time

How to debug? `join(42, "hola", ...)`

A2. Static types + contracts

A3.
What is Gradual Typing?

A1. Optional static checks, nothing at run-time

How to debug? join(42, "hola", ...)

A2. Static types + contracts

A3.
What is Gradual Typing?

# Python + Types

```python
def join(d0: DataFrame,
         d1: DataFrame,
         sort: bool,
         how: Left | Right)
    -> DataFrame:
        ...
```

A1. Optional static checks, nothing at run-time

How to debug? `join(42, "hola", ...)`

A2. Static types + contracts

Performance? `join(huge0, huge1, ...)`

A3.
What is Gradual Typing?

# Python + Types

def join(d0: DataFrame, d1: DataFrame, sort: bool, how: Left | Right) -> DataFrame:
    ....

## A1. Optional static checks, nothing at run-time

How to debug? `join(42, "hola", ...)`

## A2. Static types + contracts

Performance? `join(huge0, huge1, ...)`

## A3. Progressive static types + tags

Today!
A3. Progressive static types + tags
Experience @ Instagram Web Server

A3. Progressive static types + tags
Experience @ Instagram Web Server

+500 modules with **sound types**
+30k interactions

**A3.** Progressive static types + tags
Experience @ Instagram Web Server

+500 modules with **sound types**
+30k T U interactions

3.9% increase in CPU efficiency

A3. Progressive static types + tags
Experience @ Instagram Web Server

+500 modules with **sound types**

+30k **T**-**U** interactions

3.9% increase in CPU efficiency

**A3.** Progressive static types + tags
Experience @ Instagram Web Server

+500 modules with **sound types**
+30k **UT** interactions

**3.9% increase** in CPU efficiency

A3. Progressive static types + tags
How is Static Python so Fast?
Step 0. Better Compiler & Runtime
Step 0. Better Compiler & Runtime

https://github.com/facebookincubator/cinder
Step 0. Better Compiler & Runtime

https://github.com/facebookincubator/cinder

Cinder Runtime

V Tables
Method-based JIT
...

Step 0. Better Compiler & Runtime

https://github.com/facebookincubator/cinder

Cinder Runtime

- V Tables
- Method-based JIT
- ...

Type-Aware Bytecode

- CALL_FUNCTION: Python default
- INVOKE_METHOD: V Table lookup
- INVOKE_FUNCTION: direct call
Step 1. Fast Soundness Checks
Step 1. Fast Soundness Checks

```
avg(nums)
```

def avg(ns: chklist[int]) -> int:
    ....
Step 1. Fast Soundness Checks

```
avg(nums)
```

```
def avg(ns: chklist[int]) -> int:
    ....
```

Q. How to enforce soundness?

A. **Tag check**

Is `nums` an instance of `chklist[int]`?
Step 1. Fast Soundness Checks

```
def avg(ns:chklist[int]) -> int:
    ....
```

Q. How to enforce soundness?

A. **Tag check**

Is `nums` an instance of `chklist[int]`?

- Fast! No traversal, no wrapper
- Rejects built-in Python lists
Step 2. Progressive Types
Step 2. Progressive Types
Step 2. Progressive Types
Step 2. Progressive Types

```
list
chklist[int]
```
Step 2. Progressive Types

**Shallow types** for Python value-shapes

- list
- dict
  - string
  - bool
- int

**Concrete types** for sound generics

- chklist[int]
- chkdict[string, int]
- chklist[T]
Step 2. Progressive Types

**Shallow types** for Python value-shapes
- list
- dict
  - string
  - bool
- int

**Concrete types** for sound generics
- chklist[int]
- chkdict[string, int]
- chklist[T]

**Primitive types** for C values
- int64
- Array[float32]
Step 3. Limited Dyn Type
Step 3. Limited Dyn Type

GT Theory

Untyped code \(=\) Dyn-Typed code
Step 3. Limited Dyn Type

**GT Theory**

Untyped code $==$ Dyn-Typed code

**Static Python**

Untyped code $!=$ Dyn-Typed code
Step 3. Limited Dyn Type

**GT Theory**

Untyped code \(=\) Dyn-Typed code

Types enable arbitrary migrations (gradual guarantees)

**Static Python**

Untyped code \(!=\) Dyn-Typed code

Types enable **optimizations**
Step 3. Limited Dyn Type

GT Theory

Untyped code == Dyn-Typed code

Types enable arbitrary migrations (gradual guarantees)

Static Python

Untyped code != Dyn-Typed code

Types enable optimizations

Shallow ~ dispatch

Concrete ~ fast checks

Primitive ~ unboxing
Step 3. Limited Dyn Type

GT Theory

- Untyped code $\equiv$ Dyn-Typed code

Types enable arbitrary migrations (gradual guarantees)

Static Python

- Untyped code $\neq$ Dyn-Typed code

Types enable optimizations

```
class A:
def f(self)->int:

class B(A):
def f(self):
    # Type Error

Shallow ~ dispatch

Concrete ~ fast checks

Primitive ~ unboxing
```
Step 3. Limited Dyn Type

GT Theory

Untyped code == Dyn-Typed code

Types enable arbitrary migrations (gradual guarantees)

Static Python

Untyped code != Dyn-Typed code

Types enable optimizations

class A:
    def f(self)->int:

    # Type Error

class B(A):
    def f(self):
        # Type Error

Shallow ~ dispatch

Concrete ~ fast checks

Primitive ~ unboxing

x:int64 = 42
y = x
# Type Error
Step 3. Limited Dyn Type

| GT Theory | Static Python |
|-----------|---------------|
| Untyped code == Dyn-Typed code | Untyped code != Dyn-Typed code |

Types enable arbitrary migrations (gradual guarantees) << Types enable optimizations

```
class A:
    def f(self) -> int:

class B(A):
    def f(self):
        # Type Error

x: int64 = 42
y = x
# Type Error
```

```
def avg(ns: chklist[dyn]):
    ....

def avg(chklist[int](1,2)):
    # Runtime Error
```

Shallow ~ dispatch

Concrete ~ fast checks

Primitive ~ unboxing
Step 4. Limited Scope
Step 4. Limited Scope

Focus on high-payoff **optimizations** rather than feature-completeness

- `eval`
- `first-class class`
- `multiple inheritance`
- `Callable[T0, T1]`
- `Setof[T]`
- `Union[T0, T1, T2]` => defer to Python

==>

Pyre logo
How is Static Python so Fast?

0. Better Compiler & Runtime
1. Fast Soundness Checks
2. Progressive Types
3. Limited Dyn Type
4. Limited Overall Scope
How is Static Python so Fast?

0. Better Compiler & Runtime
1. Fast Soundness Checks
2. **Progressive Types**
   - 3. Limited Dyn Type
   - 4. Limited Overall Scope

Types gradually enable optimizations

**Gradual Soundness**
More Experience

- Shallow
- Concrete
- Primitive
More Experience

Instagram, March 2023:

- 959 typed modules
- 10 with Concrete (fast reads)
- 16 with Primitives (unboxed math)
More Experience

Instagram, March 2023:

- **959 typed** modules
- **10 with Concrete** (fast reads)
- **16 with Primitives** (unboxed math)

Microbenchmarks

1x = Python, **lower** is faster
Takeaways
Takeaways

GT Researchers

Guarantees vs. Performance?
Takeaways

Guarantees vs. Performance?

Qs for Concrete:
- migrating list to chklist[T] etc.
- fast tags for Union[T0, T1, T2]
Takeaways

Practitioners

Why not your language?

- Shallow
- Concrete
- Primitive
Takeaways

Language Designers

Redex model found:
5 critical soundness bugs
16 correctness issues
Takeaways
The End

τ λ

New research directions

Who's next?

Model found:
5 soundness + 16 other issues

Static Python

Shallow
Concrete
Primitive
