ON THE COST OF TYPE-TAG SOUNDNESS @ PEPM 2018

 Ben Greenman  Zeina Migeed  Northeastern University

RETICULATED

- Gradual typing for Python
- Enforces type-tag soundness

TYPE-TAG SOUNDNESS

If $\vdash e : \tau$ and $\lfloor \tau \rfloor = K$ then either:

- $e \longrightarrow^* v$ and v matches K
- $e \longrightarrow^* \Omega$ (runtime error)
- e diverges

where $\lfloor \tau \rfloor = K$ maps a type to a *type-tag* for its canonical forms, e.g.:

$\lfloor \text{Int} \rfloor = \text{Int}$

$\lfloor \tau \times \tau' \rfloor = \text{Pair}$

$\lfloor \tau \to \tau' \rfloor = \text{Fun}$

RUN-TIME ENFORCEMENT

Reticulated performs a run-time tag check on each dynamically-typed value v that flows into a typed context E expecting type τ :

$E[\, 7 : \text{Int} \,] \longrightarrow E[\, 7 \,]$

$E[\, (1,\text{"NaN"}) : \text{Int} \times \text{Int} \,] \to E[\, (1,\text{"NaN"}) \,]$

$E[\, \text{snd}((1,\text{"NaN"})) : \text{Int} \,] \to$ **Tag Error**

These checks affect performance.

EXPERIMENT

- Evaluated 21 Reticulated programs
  - 18 via exhaustive evaluation
  - 3 via approximate evaluation
- Ran on the *Karst at Indiana University* cluster

CONCLUSIONS

- Worst-case overhead: under 10x
- Best-case overhead: 1x -- 4x
  - Always slower than Python
- Overhead typically increases linearly with the number of type annotations

REFERENCES

- Vitousek, Swords, Siek. *Big Types in Little Runtime:Open World Soundness and Collaborative Blame for Gradual Type* Systems. POPL 2017
- Takikawa, Feltey, Greenman, New, Vitek, Felleisen. *Is Sound Gradual Typing Dead?*. POPL 2016.

QUESTION what is the performance overhead of type-tag soundness in Reticulated?

METHOD

- Add type annotations to Python programs
- Enumerate partially-typed configurations
- Measure performance relative to Python

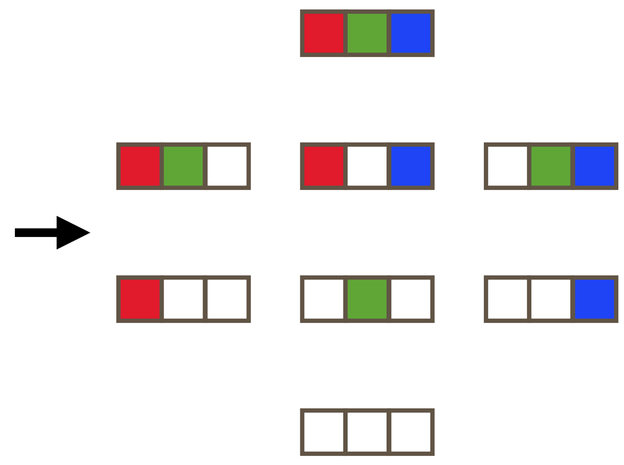
GRANULARITY function `def` and class `@fields`

Example: **1** function + **1** class + **1** method ⟶ 2^3 configurations

```
def add1(n : Int) -> Int:
    return n + 1

@fields({"x" : Int, "y" : Int})
class A:
  x = 0; y = 0;

  def move(self : A, z : Int) -> Void:
    self.x += z
    return None
```

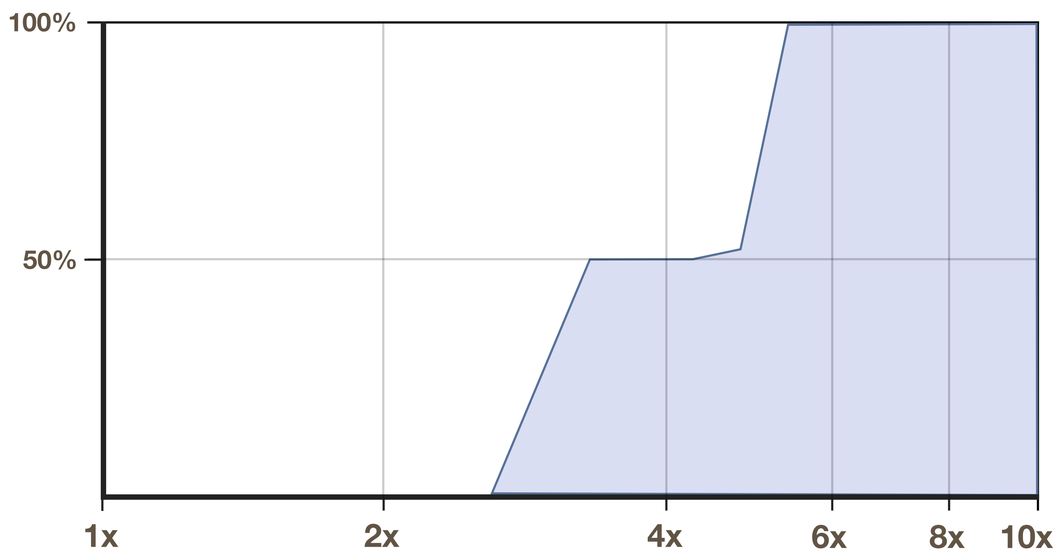


EXHAUSTIVE EVALUATION for small programs

- Measure all configurations
- Count the % of configurations that run at most **D** times slower than Python

Example: for **D** between 1x and 10x.

espionage **4,096 configurations**



All configurations are **6**-deliverable

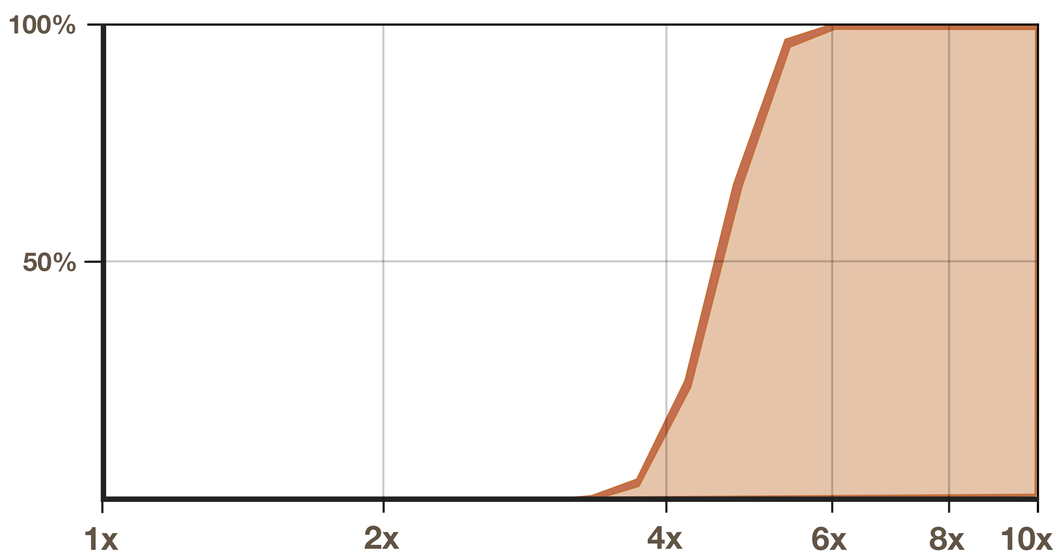
50% of all configurations are **4**-deliverable

APPROXIMATE EVALUATION for large programs

- Measure **R** samples of **S** configurations drawn uniformly at random
- Build a confidence interval for the true % of **D**-deliverable configurations

Example: for a program with 2^{34} configurations.

aespython **10 samples of 340 configurations**



All configurations are **7**-deliverable in all samples

With 95% confidence, 8% - 10% of all configs. are **4**-deliverable