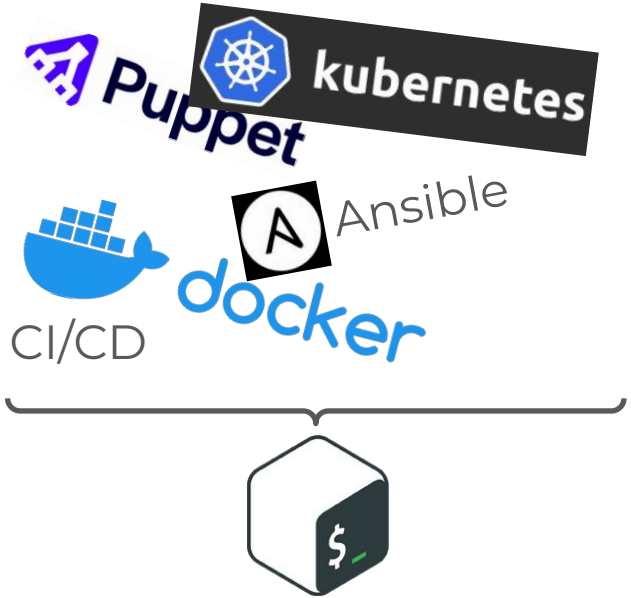




RT: a new type system for the streaming shell

Zekai Li*, Lukas Lazarek*, Evangelos Lamprou, George Kapetanakis,
Konstantinos Mamouras, Nikos Vasilakis

The UNIX shell is everywhere



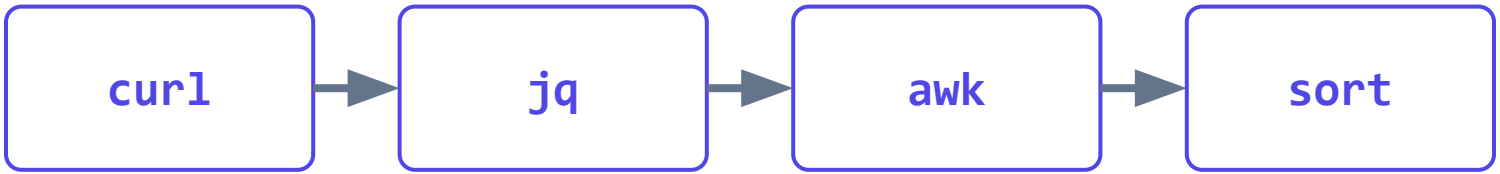
Infrastructure





Agents

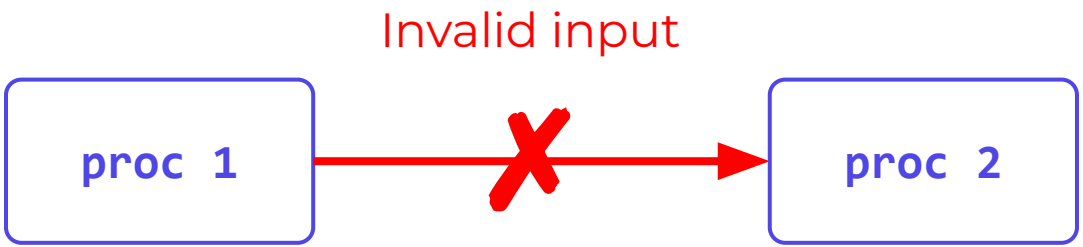
The UNIX ecosystem



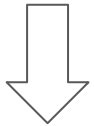
Pipelines

```
curl ... | jq ... | awk ... | sort
```

The Problem: composition mistakes



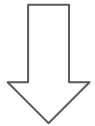
```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```



```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```

du -s ⟹

```
8  /scratch/c
4  /scratch/b
6  /scratch/a
```



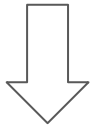
```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```

du -s ⟹

8_/scratch/c
4_/scratch/b
6_/scratch/a

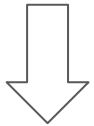
⟹

4_/scratch/b
6_/scratch/a
8_/scratch/c



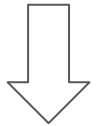
```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```

du -s ⟹ 8_/scratch/c
4_/scratch/b
6_/scratch/a ⟹ 4_/scratch/b
6_/scratch/a
8_/scratch/c ⟹ 8_/scratch/c



```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```

du -s ⟹ 8_/scratch/c
4_/scratch/b
6_/scratch/a ⟹ 4_/scratch/b
6_/scratch/a
8_/scratch/c ⟹ 8_/scratch/c ⟹ /scratch/c

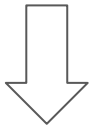


```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```

du -s ⟹ 8_/scratch/c
4_/scratch/b
6_/scratch/a ⟹ 4_/scratch/b
6_/scratch/a
8_/scratch/c ⟹ 8_/scratch/c ⟹ /scratch/c



Bug #1



```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```

du -s ⟹

8_/scratch/c
4_/scratch/b
6_/scratch/a
16_/scratch/d

⟹

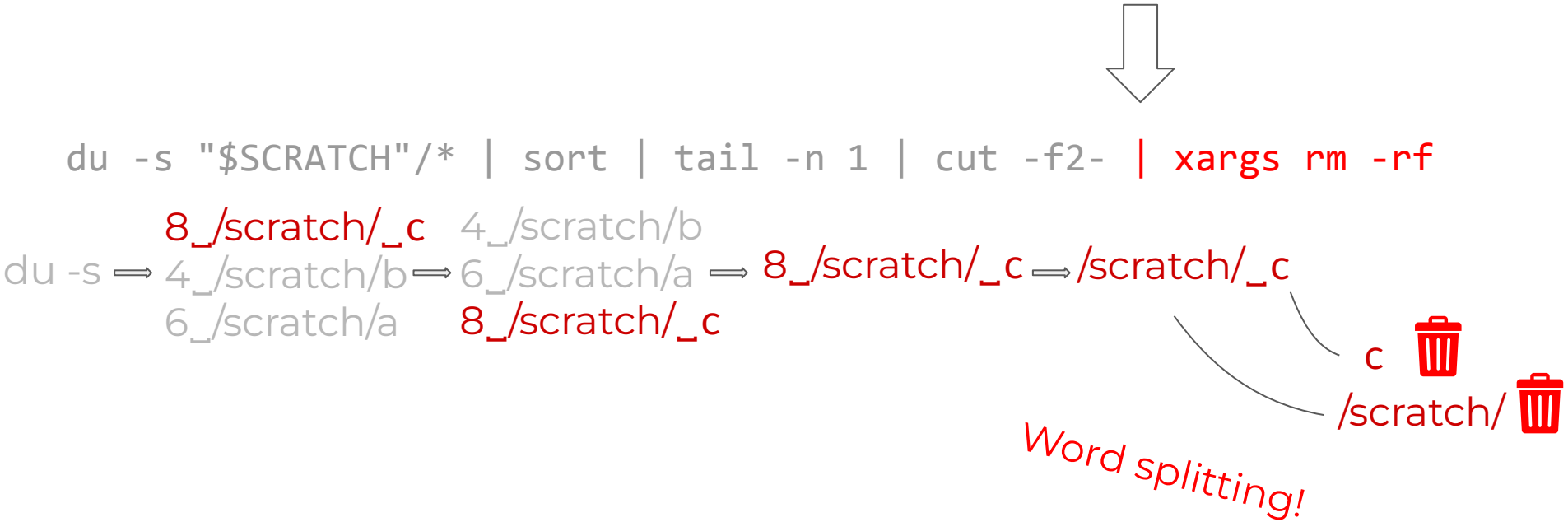
16_/scratch/d
4_/scratch/b
6_/scratch/a
8_/scratch/c

⟹ 8_/scratch/c ⟹ /scratch/c



Lexicographic!

Bug #2


du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
8_/scratch/_c
4_/scratch/b
du -s
4_/scratch/b
6_/scratch/a
6_/scratch/a
8_/scratch/_c
8_/scratch/_c
/scratch/_c
c
/scratch/
Word splitting!

subtle mismatch!

dangerous!

```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```

~~Systems to detect these~~ ~~bugs?~~

The state of practice today is linting

```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```



```
$ shellcheck clean.sh
→ no output.
```

Linter for syntactic code smell patterns

ShellCheck. “ShellCheck – shell script analysis tool.” https://www.shellcheck.net/

`rt`: an overlay type system for the shell

```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```

```
$ shellcheck clean.sh
→ no output.
```




```
$ rt clean.sh
2 type errors:
 1. numeric input to lexicographic `sort`, consider `sort -n`
 2. `xargs rm -rf` will delete unexpected paths containing spaces
```

`rt`: an overlay type system for the shell



Regular types



Built-in types designed for bug finding



Type polymorphism for more expressiveness



Approximations of non-regular data for bug-finding

Ints?
Bools?
Lists?

```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```

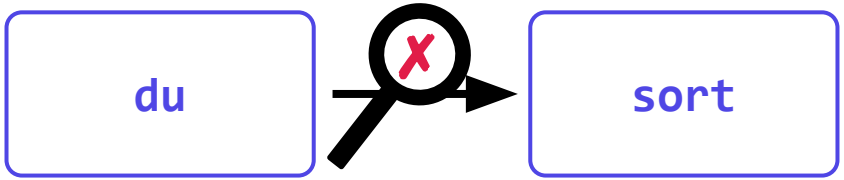
du -s ⟹

8 ./scratch/c
4 ./scratch/b
6 ./scratch/a
16 ./scratch/d

⟹

16 ./scratch/d
4 ./scratch/b
6 ./scratch/a
8 ./scratch/c

Regular structure!
as in regular languages



Type system

Regular types

`du -s` :: $\varepsilon \rightarrow$ [0-9]+␣.+

```
du -s "$SCRATCH"/* | sort
```

```
        8␣/scratch/c
du -s ⟹ 4␣/scratch/b
        6␣/scratch/a
```

Regular types describe structure of lines

① ②

`du -s :: ε → [0-9]+␣.+`

```
du -s "$SCRATCH"/* | sort
```

```
8␣/scratch/c
du -s ⟹ 4␣/scratch/b
```

① Input: ε
nothing

② Output: `[0-9]+␣.+`
lines with 1 or more occurrences of any of the characters 0-9, followed by ␣ and 1 or more occurrences of any characters

Shape of expected input and output **lines**

Line-oriented types for text streams reflect UNIX norms

Regular types for each command

Input that does **not** start with a number

`du -s` :: ε → [0-9]+␣.+ `sort` :: [^0-9].* → [^0-9].*

```
du -s "$SCRATCH"/* | sort
```

Conservatively restrict the space of allowed inputs
→ identify likely bugs

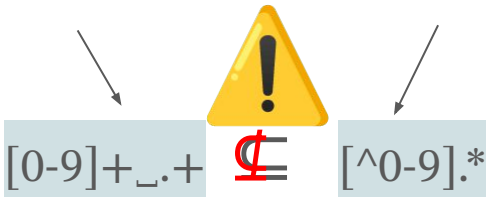



```
function div(x: number, y: number): number { return x / y }
```

Type checking with regular types

`du -s` :: ε → [0-9]+␣.+ `sort` :: [^0-9].* → [^0-9].*

```
du -s "$SCRATCH"/* | sort
```



Composition problem!
Incompatible input for `sort`

☼ Tractable (and efficient) of type checking

👩‍💻 Friendly to shell developers versed in regexes

✔ Expressive

The fix type checks

Input starting with numbers

`du -s` :: ε → [0-9]+␣.+ `sort -n` :: [0-9].* → [0-9].*

```
du -s "$SCRATCH"/* | sort -n
```

[0-9]+␣.+ ⊆ [0-9].*

✓

`rt`: an overlay type system for the shell



Regular types

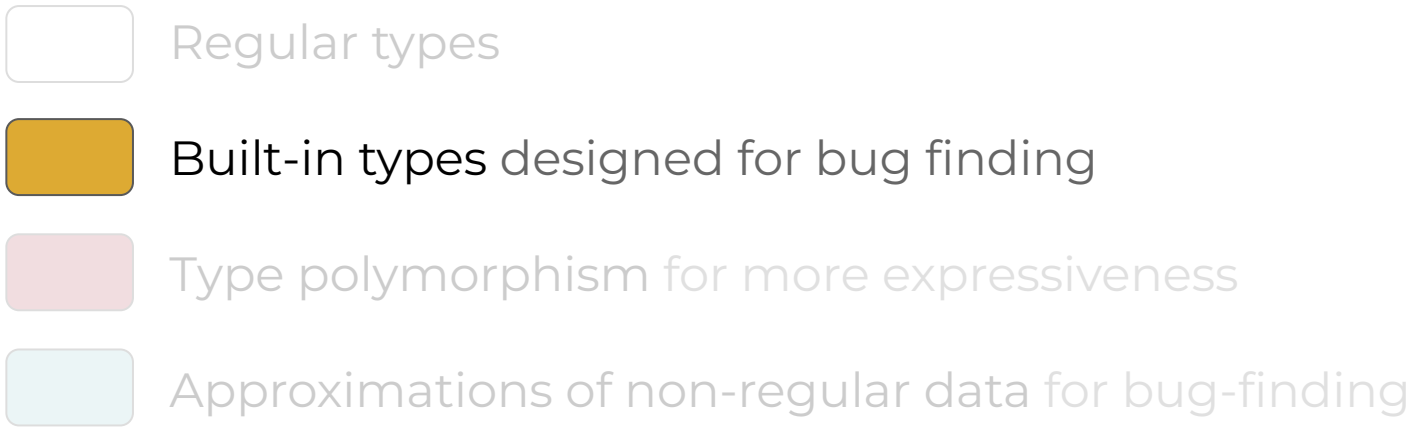
- Types: regular languages
- Commands input and output regular types
- Type checking: language inclusion

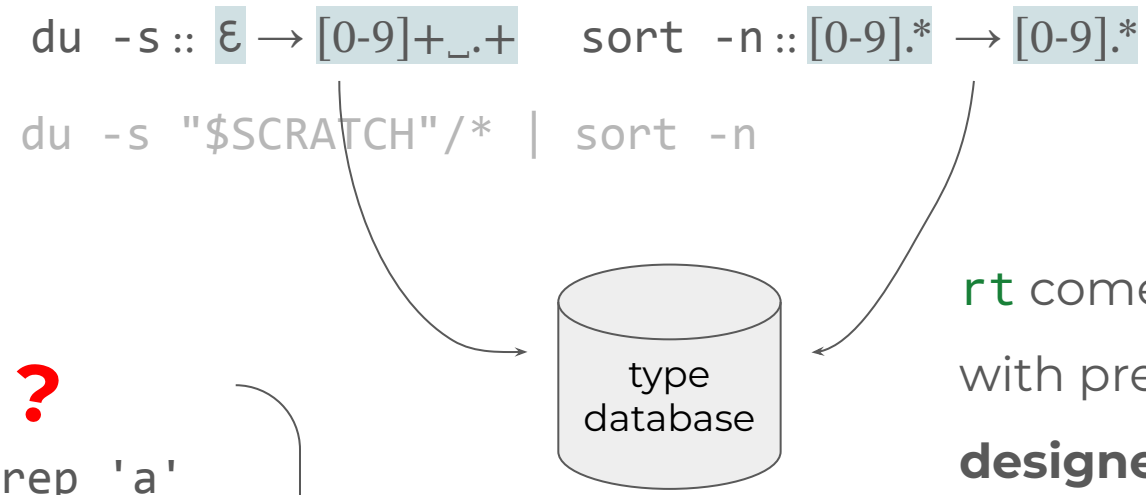
Built-in types designed for bug finding

Type polymorphism for more expressiveness

Approximations of non-regular data for bug-finding

`rt`: an overlay type system for the shell





`rt` comes **batteries-included**

with pre-defined types,

designed for finding bugs

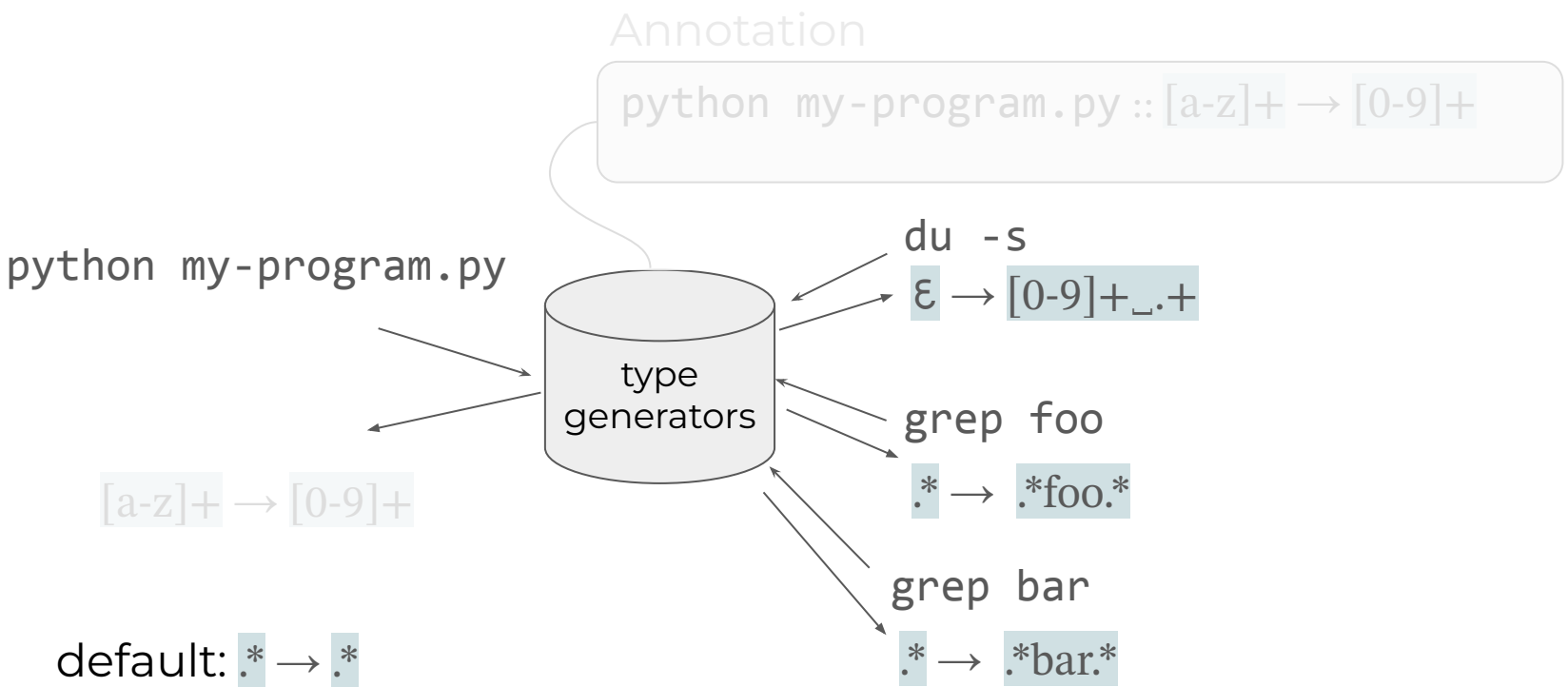
?

```
grep 'a'
grep 'b'
grep 'c'
grep -v 'a'
grep -o 'a'
...
```

Challenge: infinite command variations

Type Generator

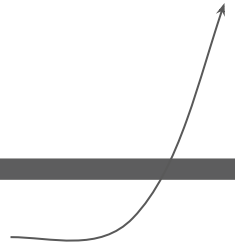
Type Generators enable a finite type DB



`rt` finds bugs early with no developer effort

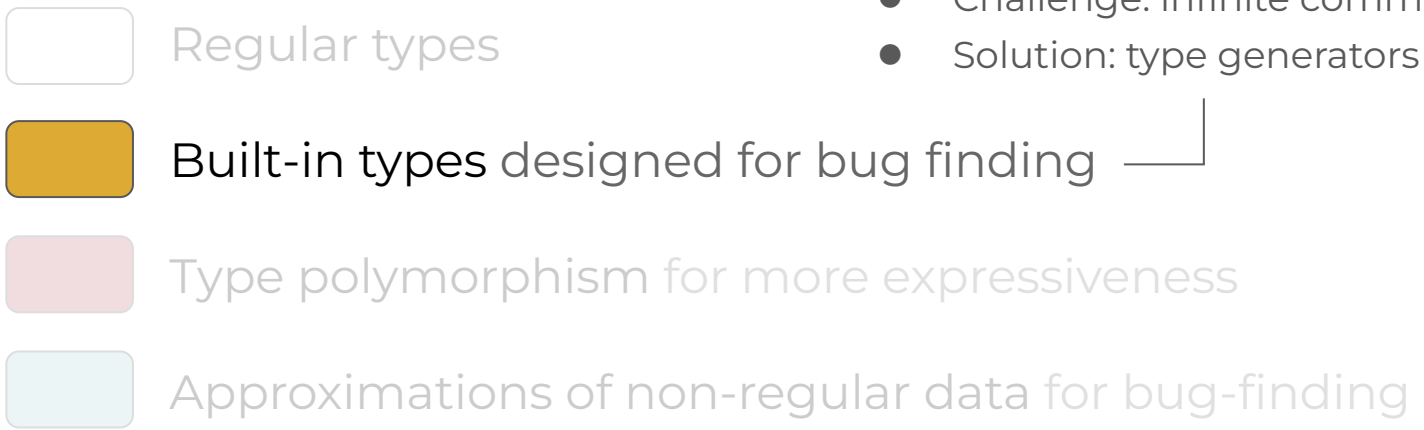
Optional type annotations improve accuracy

```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```



```
$ rt clean.sh
2 type errors:
 1. numeric input to lexicographic `sort`, consider `sort -n`
 2. `xargs rm -rf` will delete unexpected paths containing spaces
```

`rt`: an overlay type system for the shell



- Challenge: infinite command variations
- Solution: type generators

`rt`: an overlay type system for the shell



Type of tail?

`sort -n` :: [0-9].* → [0-9].*

`tail -n 1` :: [0-9].* → [0-9].*

```
du -s "$SCRATCH"/* | sort -n | tail -n 1 | cut -f2- | xargs rm -rf
```

```
echo a | tail -n 1
echo b | tail -n 1
echo c | tail -n 1
echo d | tail -n 1
...
```

Challenge:
typing generic commands

Polymorphism

Instantiated
with actual input

`tail -n 1` :: $\forall\ \alpha\ .\ \alpha \rightarrow \alpha$

`sort -n` :: [0-9].* → [0-9].*

`tail -n 1` :: [0-9].* → [0-9].*

```
du -s "$SCRATCH"/* | sort -n | tail -n 1 | cut -f2- | xargs rm -rf
```

```
echo a | tail -n 1
echo b | tail -n 1
echo c | tail -n 1
echo d | tail -n 1
...
```

Challenge:
typing generic commands

`rt`: an overlay type system for the shell



- Challenge: generic commands
- Solution: polymorphic command types

`rt` finds bugs early with no developer effort

```
du -s "$SCRATCH"/* | sort | tail -n 1 | cut -f2- | xargs rm -rf
```

```
$ rt clean.sh
2 type errors:
 1. numeric input to lexicographic `sort`, consider `sort -n`
 2. `xargs rm -rf` will delete unexpected paths containing spaces
```

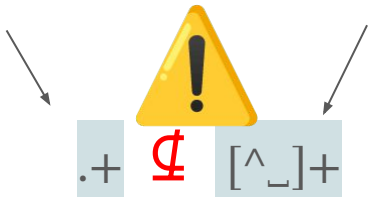
`rt` identifies the second bug

No whitespace

`cut :: [0-9]+␣.+ → .+`

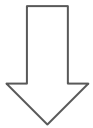
`xargs rm -rf:: [^␣\t\n]+ → ε`

```
du -s "$SCRATCH"/* | sort -n | tail -n 1 | cut -f2- | xargs rm -rf
```



`rt`: an overlay type system for the shell

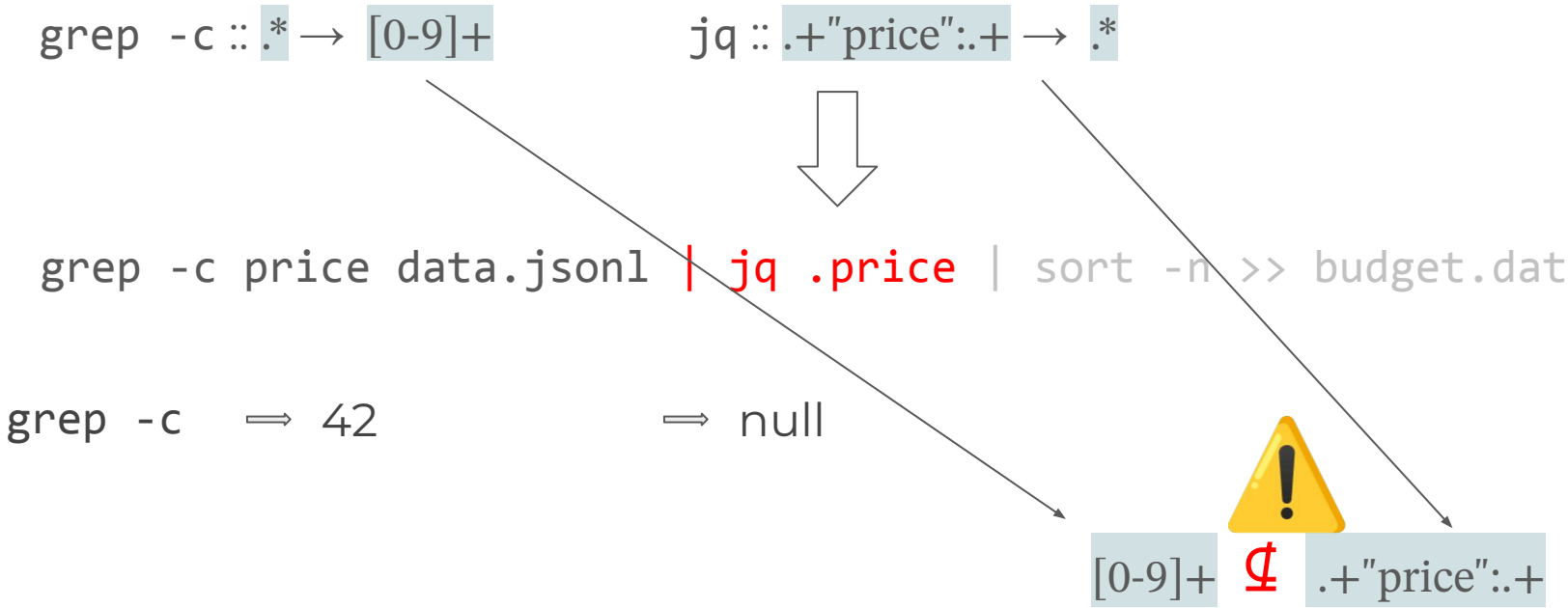




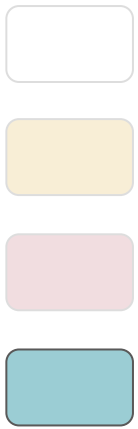
```
grep -c price data.jsonl | jq .price | sort -n >> budget.dat
```

`grep -c` ⟹ 42

Approximate key properties for bug-finding



`rt`: an overlay type system for the shell



Regular types

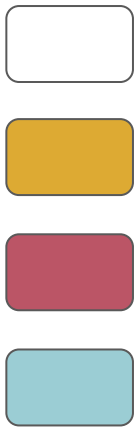
Built-in types designed for bug finding

Type polymorphism for more expressiveness

Approximations of non-regular data for bug-finding

- Regular approximations still find bugs

`rt`: an overlay type system for the shell



Regular types

Built-in types designed for bug finding

Type polymorphism for more expressiveness

Approximations of non-regular data for bug-finding

Evaluation: bug-finding effectiveness

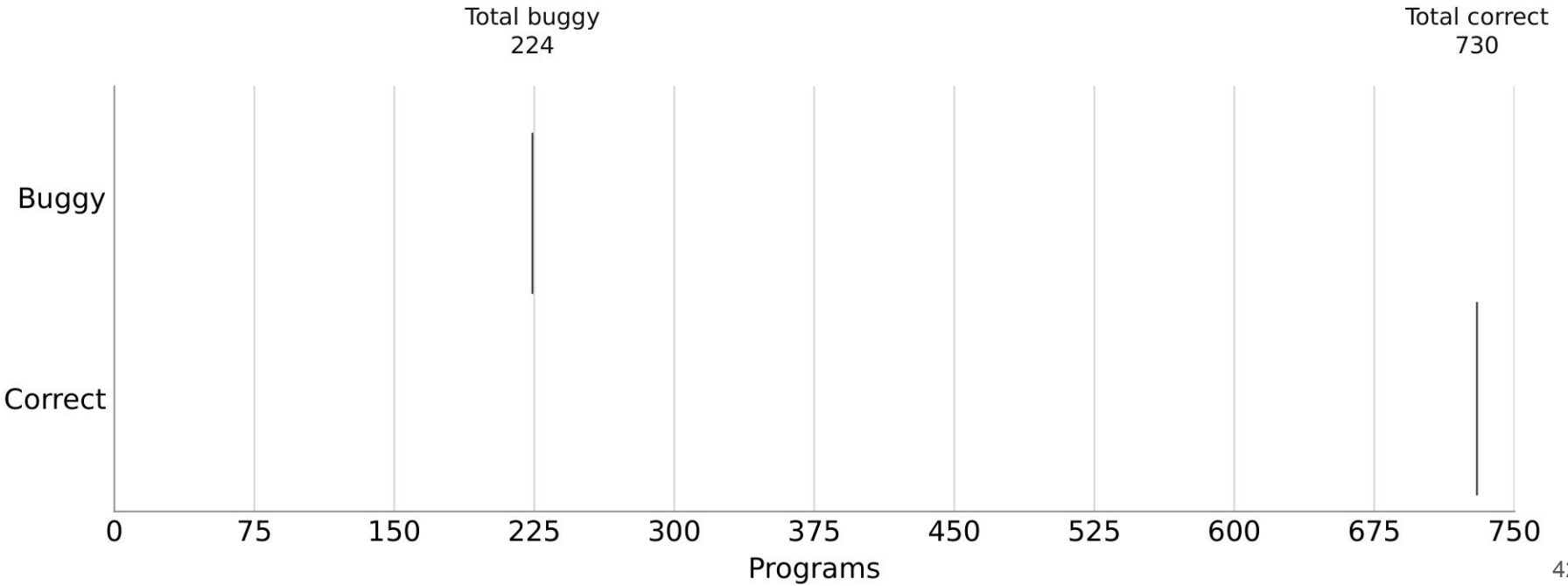
Evaluation Question:
What is the bug finding effectiveness of RT?
→ Does `rt` warn about real bugs?
→ Does `rt` inaccurately warn about correct programs?

Benchmarks: **954** programs (224 buggy)

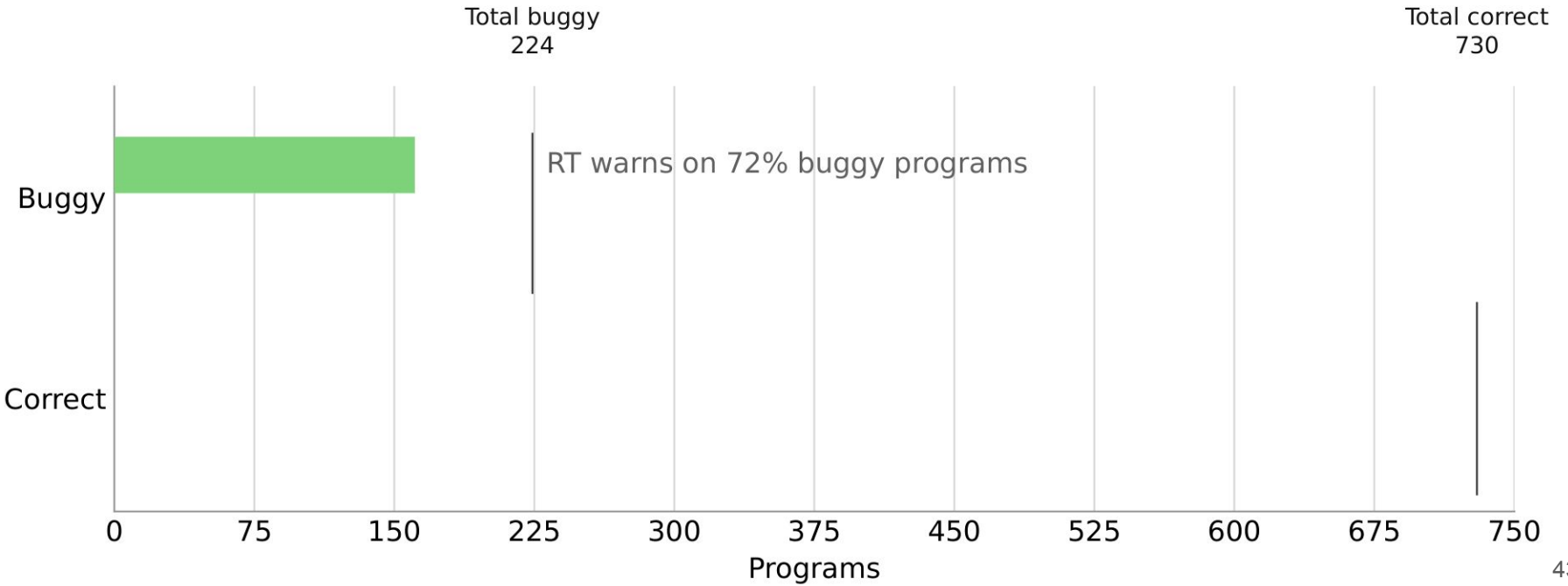
Q&A forums: StackOverflow, ...
Open source: Github,
Benchmarks: Koala (ATC'25),
Intercode (Yang et al. '23),
and more!

Baselines: ShellCheck *Linter*, not type system;
state of practice today: ~70% of top github shell projects

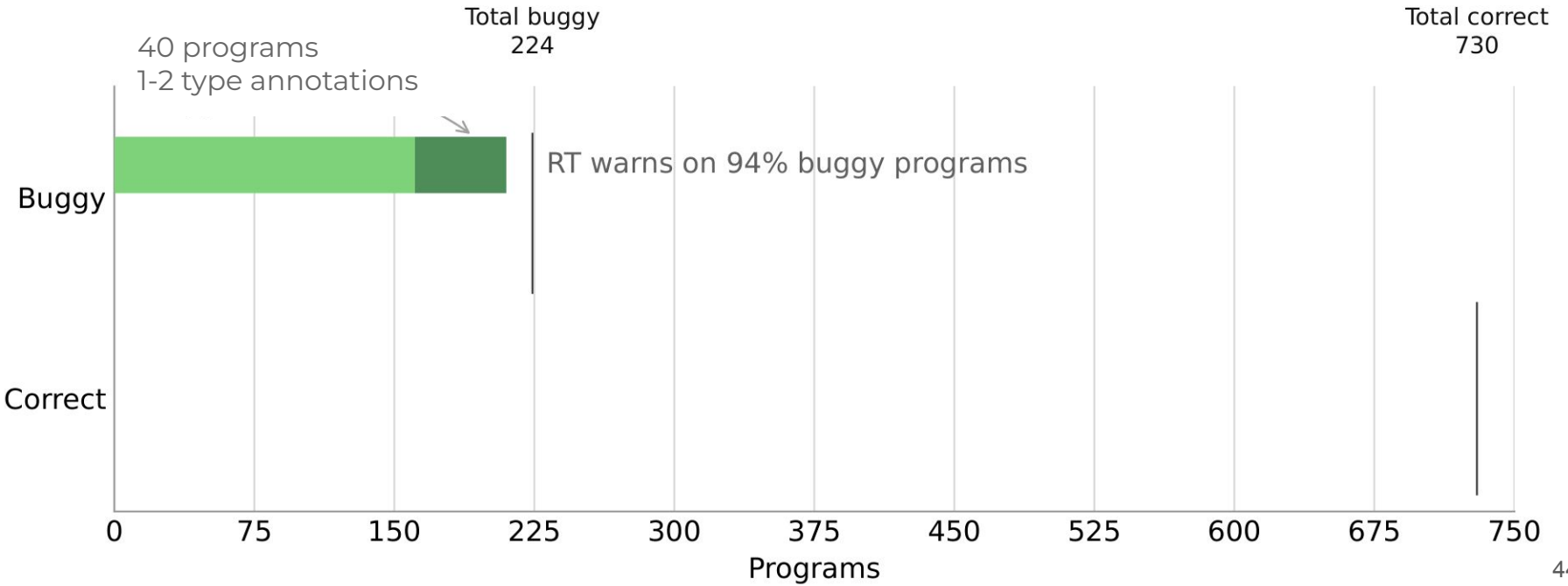
Does RT accurately detect bugs?



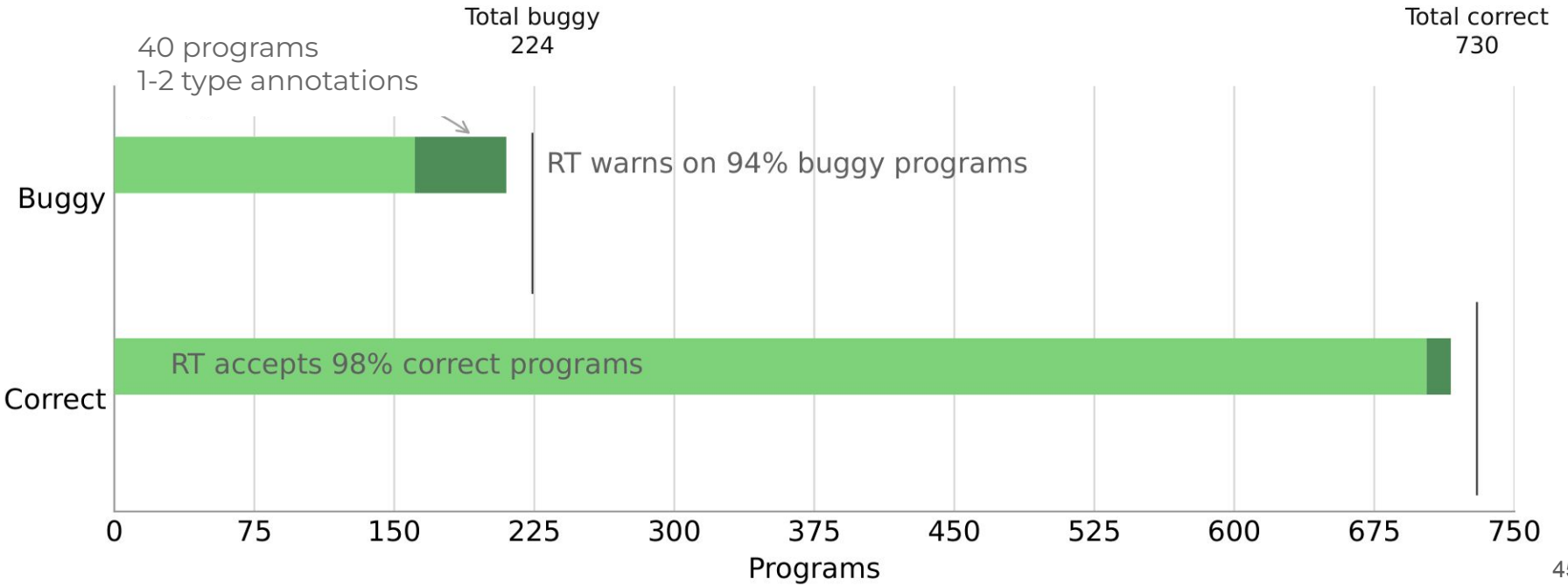
Does RT accurately detect bugs?



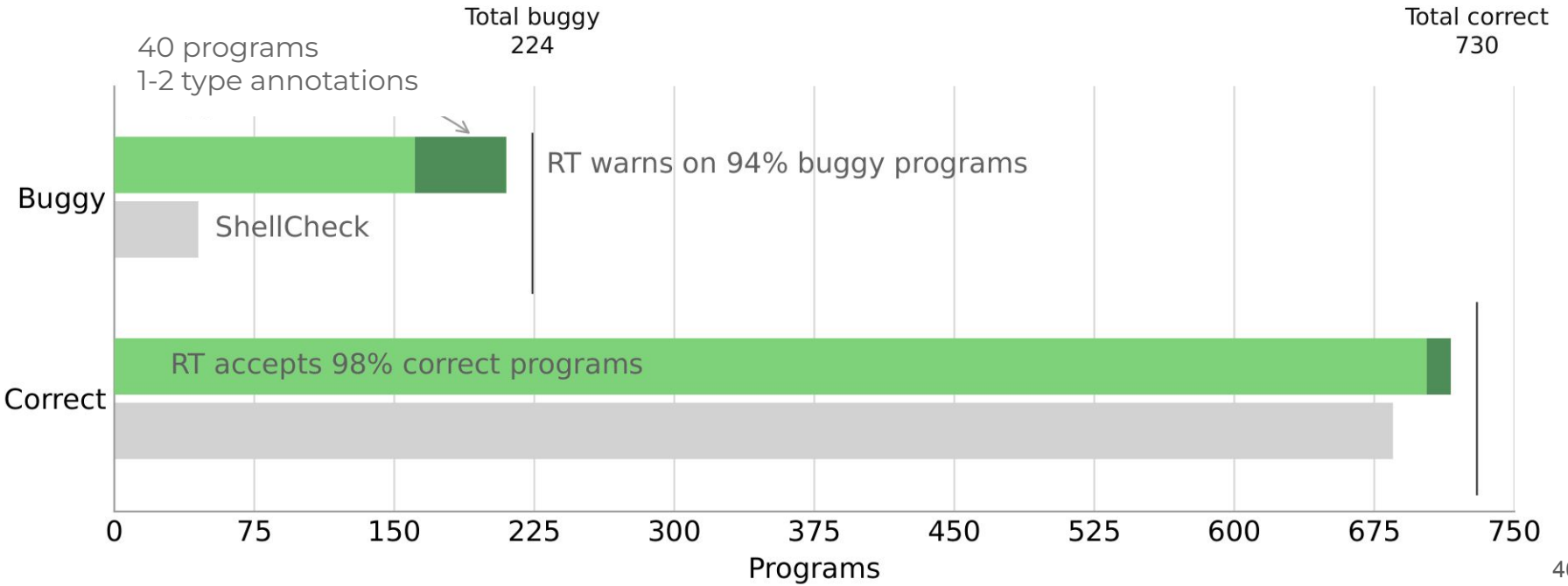
Does RT accurately detect bugs?



Does RT accurately detect bugs?



Does RT accurately detect bugs?



More in the paper

1. Syntactic sugar for common patterns: `filepath, float, date, url`

2. Improved precision via type DSL based on FSTs:
   e.g., the type of `sed "s/^[^ ]* //"`

3. Example-based error reporting

4. Additional evaluation (performance, feature ablation, and more)

RT: a new type system for the streaming shell

Zekai Li, Lukas Lazarek, Evangelos Lamprou, George Kapetanakis, Konstantinos Mamouras, Nikos Vasilakis



Problem

Process composition mistakes

Solution

RT: Regular-language Type system for the shell

http://atlas.cs.brown.edu

Results

94% of buggy pipelines identified

ahead of execution, with few false positives

