Test-based Solution Filtering for Program Synthesis

Zetten Wang, UC San Diego

Type-based Program Synthesis gives bad results!

`xs: [a] -> (a, a)`	Query

```
1. (,) (head xs) (head xs)
2. ($) (last []) xs
3. last (zip [] xs)
4. head (zip [] xs)
5. last (zip xs [])
6. head (zip xs [])
7. (,) (head (xs++xs)) (head xs)

...

15. (,) (head xs) (last x)
```

Get rid of these uninteresting results to save our desired one!

How to get rid of these uninteresting results?

Manually classify and recognize the patterns

Classification: Invalid Results

```
xs: [a] -> (a, a)
```

Query

```
1. (,) (head xs) (head xs)
2. ($) (last []) xs
3. last (zip [] xs)
4. head (zip [] xs)
5. last (zip xs [])
6. head (zip xs [])
7. (,) (head (xs++xs)) (
...
15. (,) (head xs) (last x)
```

Invalid results can be..

- Always crash e.g. `head []`, `last []`, `fromJust Nothing`
- Always diverge e.g. `last (repeat x)`, `length (repeat x)`

Classification: Duplicate Results

```
xs: [a] -> (a, a)
```

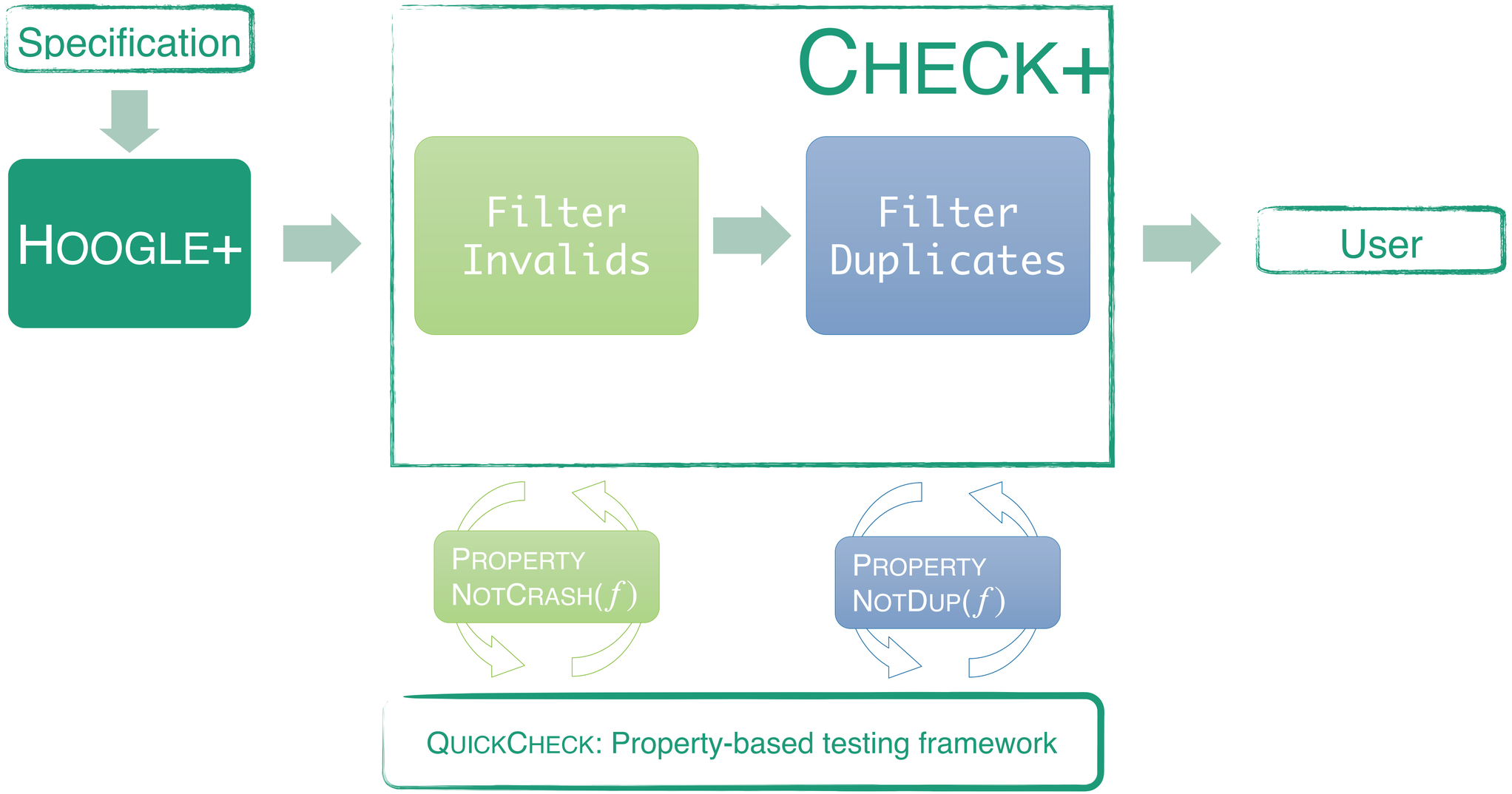
Query

```
1. (,) (head xs) (head xs)
2. ($) (last []) xs
3. last (zip [] xs)
4. head (zip [] xs)
5. last (zip xs [])
6. head (zip xs [])
7. (,) (head (xs++xs)) (head xs)
...
15. (,) (head xs) (last x)
```

A duplicate result is syntactically distinct to but has the same behavior as other results.

- `[]` vs. `(zip xs [])`
- `(head xs)` vs. `(head (xs++xs))`

Overview of Check+



Property: Invalid Result

A synthesized result is invalid if it throws an exception or diverges on all tested inputs.

Prop. 1 passed ⇒ Invalid & Reject!

✳Build Prop. 1: Result f either fails or diverges.

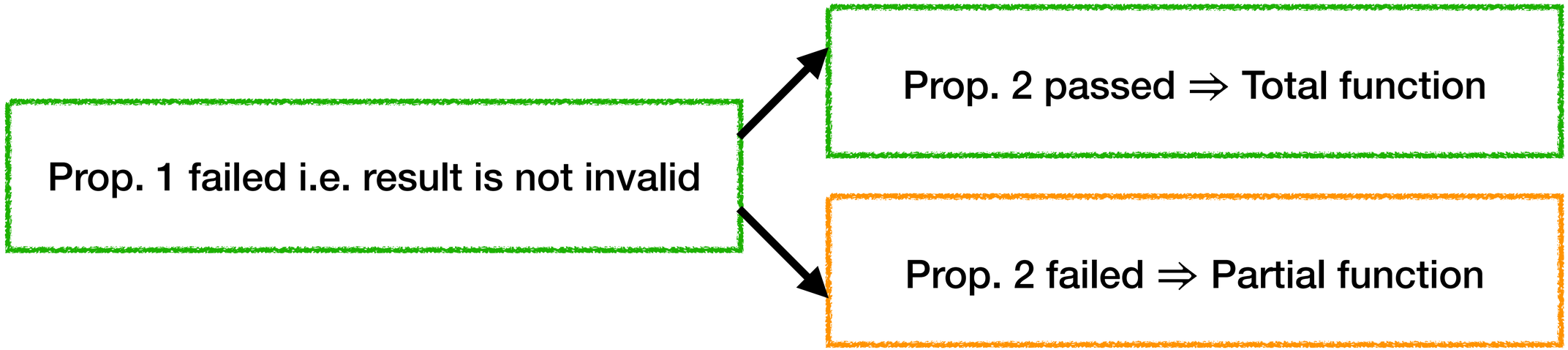
✳Build Prop. 2: Result f terminates with meaningful outputs.

Property: Invalid Result

A synthesized result is invalid if it throws an exception or diverges on all tested inputs.

✳Build Prop. 1: Result f either fails or diverges.

✳Build Prop. 2: Result f terminates with meaningful outputs.



Property: Duplicate Result

A synthesized result is duplicate if it is syntactically distinct to but has the same behavior as other results.

Build Prop. 1: $f \neq f', f' \in \mathcal{S}$ where $\mathcal{S}$ is the set of all previous synthesized results.

Challenges

- Infinite Data Structures

  Consider `\x -> (repeat x, head [])`

  Does it ever fail?

```
>> print (f x)
([x, x, x, x, x, x, x, x,
x, x, x, x, x, x, x, …
```

Challenges

- Higher-order Function

  Consider `\f -> f (head [])` with `f = {_ -> 0}`

  Does it ever fail?

```
>> g _ = 0
>> f g
0
```

Updates to Check+

- Random Generation vs. Enumeration: from QuickCheck to SmallCheck
- Function-arguments are enumerated based on size
- Input-output pairs captured from stdout

Recall — Property: Invalid Result

A synthesized result is invalid if it throws an exception or diverges on all tested inputs.

Random vs. Enumeration

- Random testing with shrinking enabled
  - First, generate an input example i with a pre-defined seed and size.
  - Test it against function f, until $f(i)$ fails. Then i is a counterexample to the property that f always holds.
  - Shrink the size of i to get i', where $f(i')$ fails too.

Random vs. Enumeration

- Random testing with shrinking enabled
  - First, generate an input example i with a pre-defined seed and size.
  - Test it against function f, until $f(i)$ fails. Then i is a counterexample to the property that f always holds.
  - Shrink the size of i to get i', where $f(i')$ fails too.

Random vs. Enumeration

- Input generation with Enumeration

```
Test.QuickCheck> generate (resize 3 arbitrary) :: IO [String]

["\222119v","s?5",""]

Test.QuickCheck> generate (resize 3 arbitrary) :: IO [Int]

[0,-2]
```

Updates to Check+

- Random Generation vs. Enumeration: from QuickCheck to SmallCheck
- Function-arguments are enumerated based on size
- Input-output pairs captured from stdout

Example 1

```
xs: [a] -> (a, a)
```

Query

```
1. (,) (head xs) (head xs)
```

```
[0, 1] -> (0, 0)
[1, 0] -> (1, 1)
[] -> error
```

```
2. (,) (last xs) (last xs)
```

```
[0, 1] -> (1, 1)
[1, 0] -> (0, 0)
[] -> error
```

```
3. (,) (head xs) (last xs)
```

```
[0, 1] -> (0, 1)
[1, 0] -> (1, 0)
[] -> error
```

Discarded results

```
1) last (zip [] xs)
2) head (zip [] xs)
3) last (zip xs [])
4) head (zip xs [])
```

Example 2

```
f: (a -> b) -> g: (a -> c) -> x: a -> (b, c)
```

Query

```
1. (,) (f x) (g x)
```

```
{_->0} {_->0} 0 -> (0, 0)*
```

Discarded results

```
1. (,) (f (fromJust Nothing)) (g x)
2. (,) (g (head [])) (f x)
3. (,) (g (last [])) (f x)
```

***Note: in CLI given as \x -> case x of 0 -> 0; 1 -> 0; -1 -> 0; 2 -> 0; ... \x -> case x of 0 -> 0; 1 -> 0; -1 -> 0; 2 -> 0; ... 0 ==> (0, 0)**

Example 3

```
f: (a -> Either b c) -> xs: [a] -> ([b], [c])
```

Query

```
1. partitionEithers (map f xs)
```

```
{_->Left 0} [] -> ([], [])
```

```
2. partitionEithers (repeat (f (head xs)))
```

```
{_->Right 0} [] -> ([], [0, 0, ...])
```

Discarded results

```
1. partitionEithers (repeat (f (last xs)))
2. curry (last []) xs f
```