

Irulan: Automatic Crash Testing using the GHC API (WIP)

Tristan Allwood
tora@zonetora.co.uk

Susan Eisenbach -
susan.eisenbach@imperial.ac.uk

Imperial College London

What?

Automatically finds expressions that call `error`

Example

```
module Person
( Name, Person, mkPerson, julia
, carla, makeTaller) where

data Name    = ...
data Person = ...

makeTaller :: Person → Person
mkPerson  :: Name → Person
julia    :: Name
carla    :: Name
```

Example

```
> irulan Person
```

```
Person:
```

```
Testing: Person.carla Person.julia
```

```
Person.makeTaller Person.mkPerson
```

```
Results:
```

```
Person.makeTaller ?0 ==> ! TODO - makeTaller
```

Example

```
> irulan -trace Person
Person:
Testing: Person.carla Person.julia
Person.makeTaller Person.mkPerson
  Person.carla ==> .
  Person.julia ==> .
  Person.makeTaller ==> .
  Person.makeTaller ?0 ==> ! TODO - makeTaller
  Person.mkPerson ==> .
  Person.mkPerson ?1 ==> .
Results:
  Person.makeTaller ?0 ==> ! TODO - makeTaller
```

Example

```
> irulan -trace Person
Person:
Testing: Person.carla Person.julia
Person.makeTaller Person.mkPerson
  Person.carla ==> .
  Person.julia ==> .
  Person.makeTaller ==> .
  Person.makeTaller ?0 ==> ! TODO - makeTaller
  Person.mkPerson ==> .
  Person.mkPerson ?1 ==> .
Results:
  Person.makeTaller ?0 ==> ! TODO - makeTaller
```

Example

```
> irulan -trace Person
Person:
Testing: Person.carla Person.julia
Person.makeTaller Person.mkPerson
  Person.carla ==> .
  Person.julia ==> .
  Person.makeTaller ==> .
  Person.makeTaller ?0 ==> ! TODO - makeTaller
  Person.mkPerson ==> .
  Person.mkPerson ?1 ==> .
Results:
  Person.makeTaller ?0 ==> ! TODO - makeTaller
```

Example

```
> irulan -trace Person
Person:
Testing: Person.carla Person.julia
Person.makeTaller Person.mkPerson
  Person.carla ==> .
  Person.julia ==> .
  Person.makeTaller ==> .
  Person.makeTaller ?0 ==> ! TODO - makeTaller
  Person.mkPerson ==> .
  Person.mkPerson ?1 ==> .
Results:
  Person.makeTaller ?0 ==> ! TODO - makeTaller
```


Example

```
Testing: Person1.carla Person1.julia Person1.makeTaller
Person1.mkPerson
Person1.carla ==> .
Person1.julia ==> .
Person1.makeTaller ==> .
Person1.makeTaller ?0 ==> ?0
Person1.mkPerson ==> .
Person1.mkPerson ?4 ==> ?4
Person1.mkPerson Person1.carla ==> .
Person1.makeTaller (Person1.mkPerson Person1.carla) ==> .
Person1.makeTaller (Person1.makeTaller
(Person1.mkPerson Person1.carla)) ==> .
Person1.makeTaller (Person1.makeTaller (Person1.makeTaller
(Person1.mkPerson Person1.carla))) ==> .
Person1.mkPerson Person1.julia ==> .
Person1.makeTaller (Person1.mkPerson Person1.julia) ==> .
Person1.makeTaller (Person1.makeTaller
(Person1.mkPerson Person1.julia)) ==> .
Person1.makeTaller (Person1.makeTaller (Person1.makeTaller
(Person1.mkPerson Person1.julia))) ==> .
```

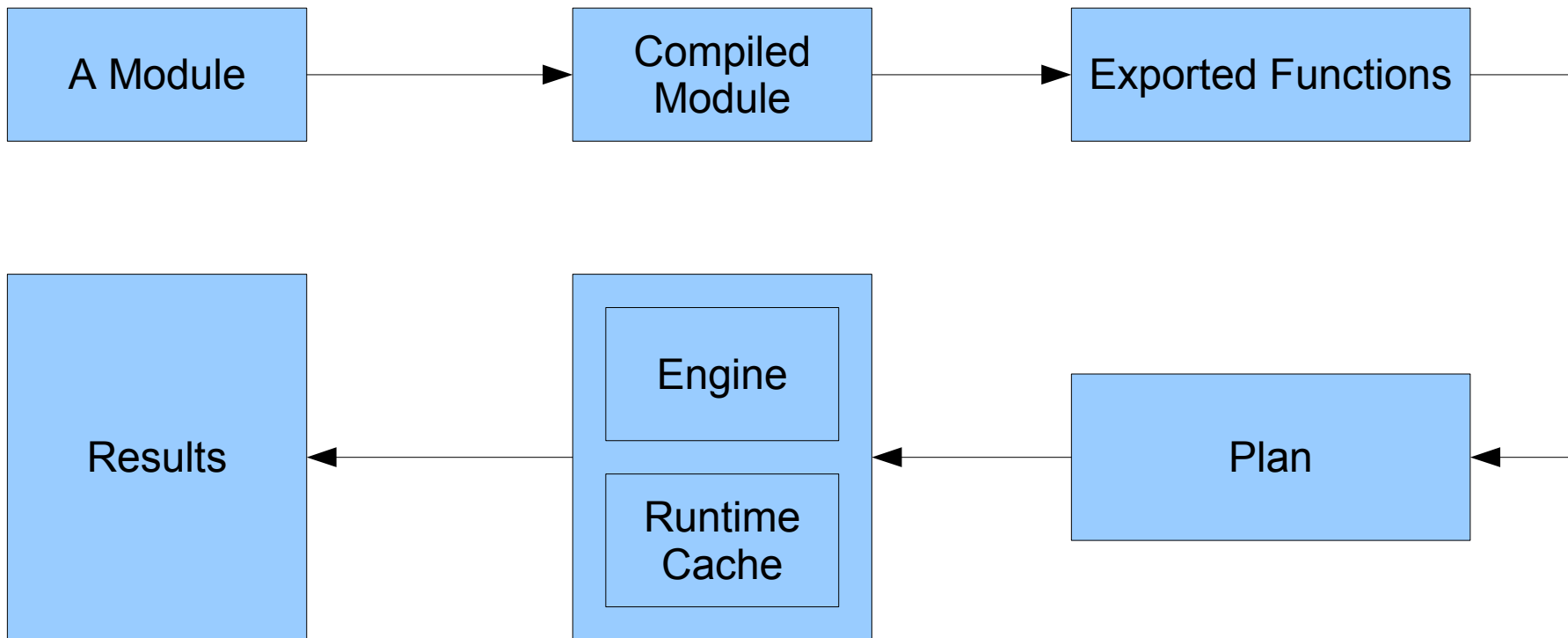
The Story So Far...

- Discovery of exported functions
- Construction of expressions
- Execution of constructed expressions
- Checking whether a user error was called
- Creating arguments to test if expressions are strict in that argument

Laziness

- Similar idea to Lazy Small Check
- Don't provide arguments that aren't needed
- How to tell?
 - `error (uniquePrefix ++ uniqueId)`
- If we catch that error, then generate an appropriate expression

Irulan Overview



Runtime equivalence pruning

```
data Person = Person { getName :: Name, ... }  
julia :: Name  
mkPerson :: Name → Person
```

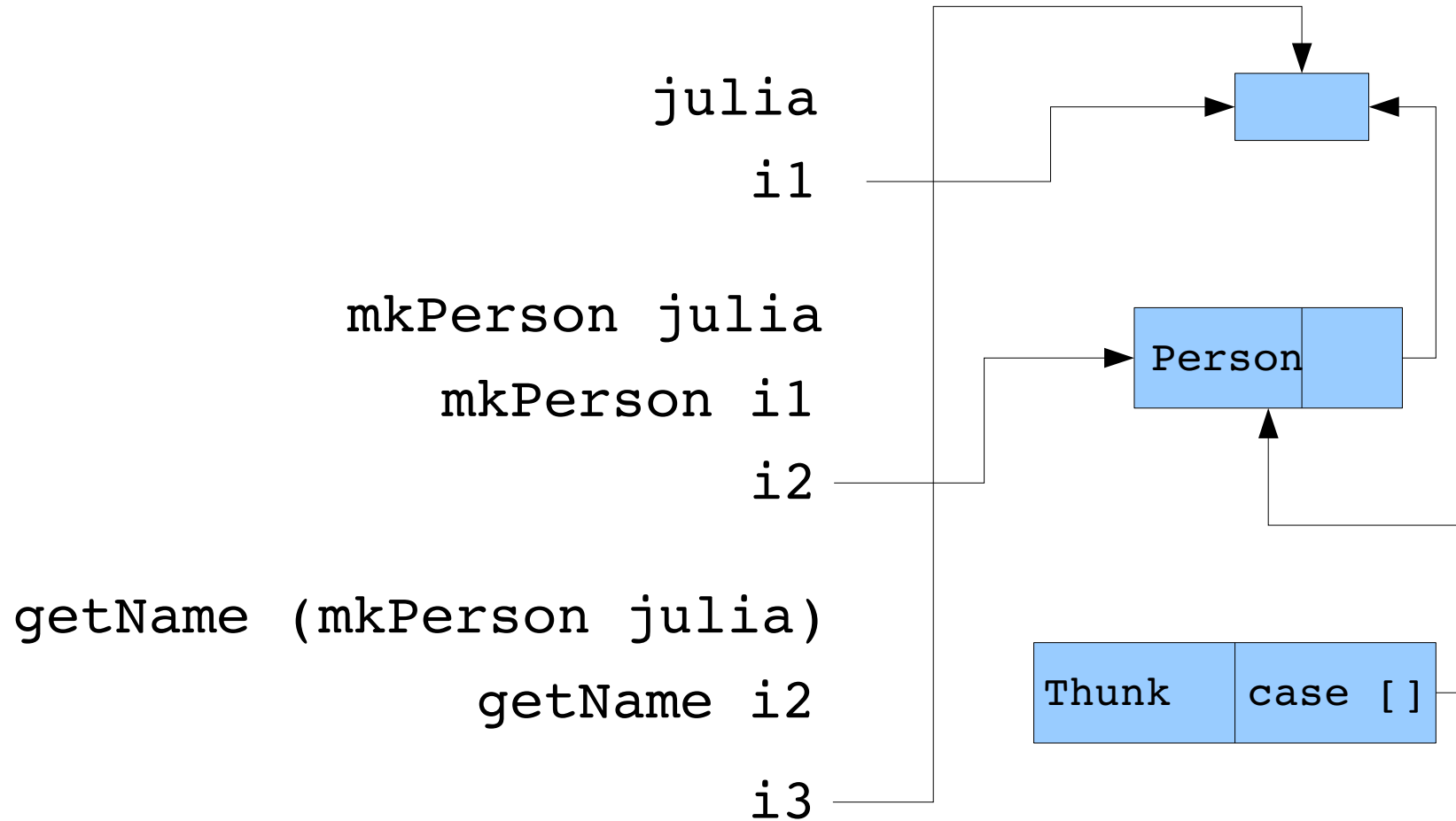
```
> julia  
> getName (mkPerson julia)  
> getName (mkPerson (getName (mkPerson julia)))  
> getName (mkPerson (getName (mkPerson (getName...
```

Runtime equivalence pruning

```
data Person = Person { getName :: Name, ... }
julia :: Name
mkPerson :: Name → Person
```

```
> irulan -trace PeopleShared
Loaded: PeopleShared
PeopleShared:
Testing: PeopleShared.getName PeopleShared.julia PeopleShared.mkPerson
PeopleShared.julia ==> .
PeopleShared.getName ==> .
PeopleShared.getName ?0 ==> ?0
PeopleShared.mkPerson ==> .
PeopleShared.mkPerson ?1 ==> ?1
PeopleShared.mkPerson PeopleShared.julia ==> .
PeopleShared.getName (PeopleShared.mkPerson PeopleShared.julia)
  ==> # == PeopleShared.julia
Results:
```


Runtime equivalence pruning



Runtime Equivalence Pruning

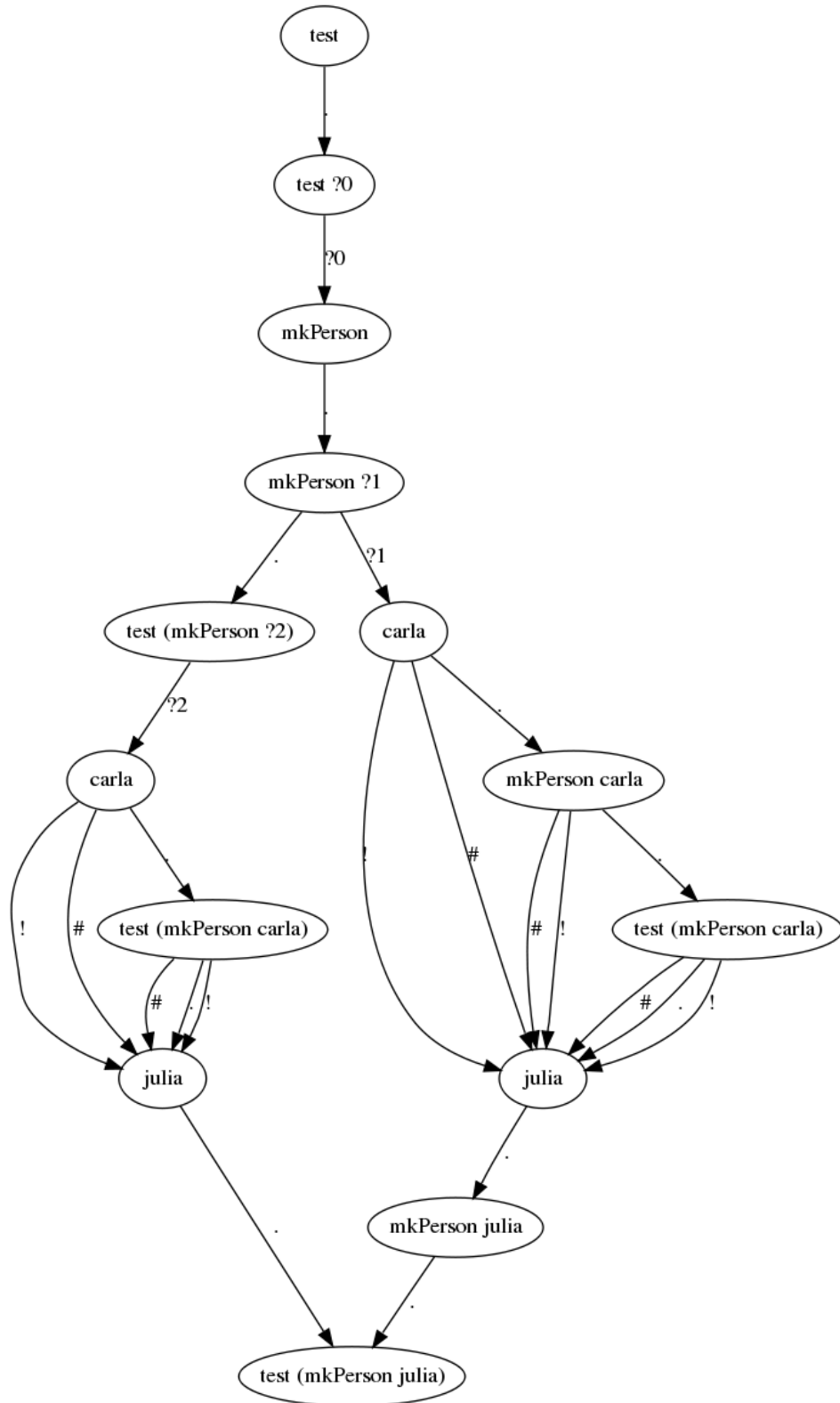
- Remember and reuse already calculated expressions
- `StableName` library for equivalence check

Runtime Equivalence Pruning

- `StableName HValue` with newtypes
- Can't safely cache values with ?s in them

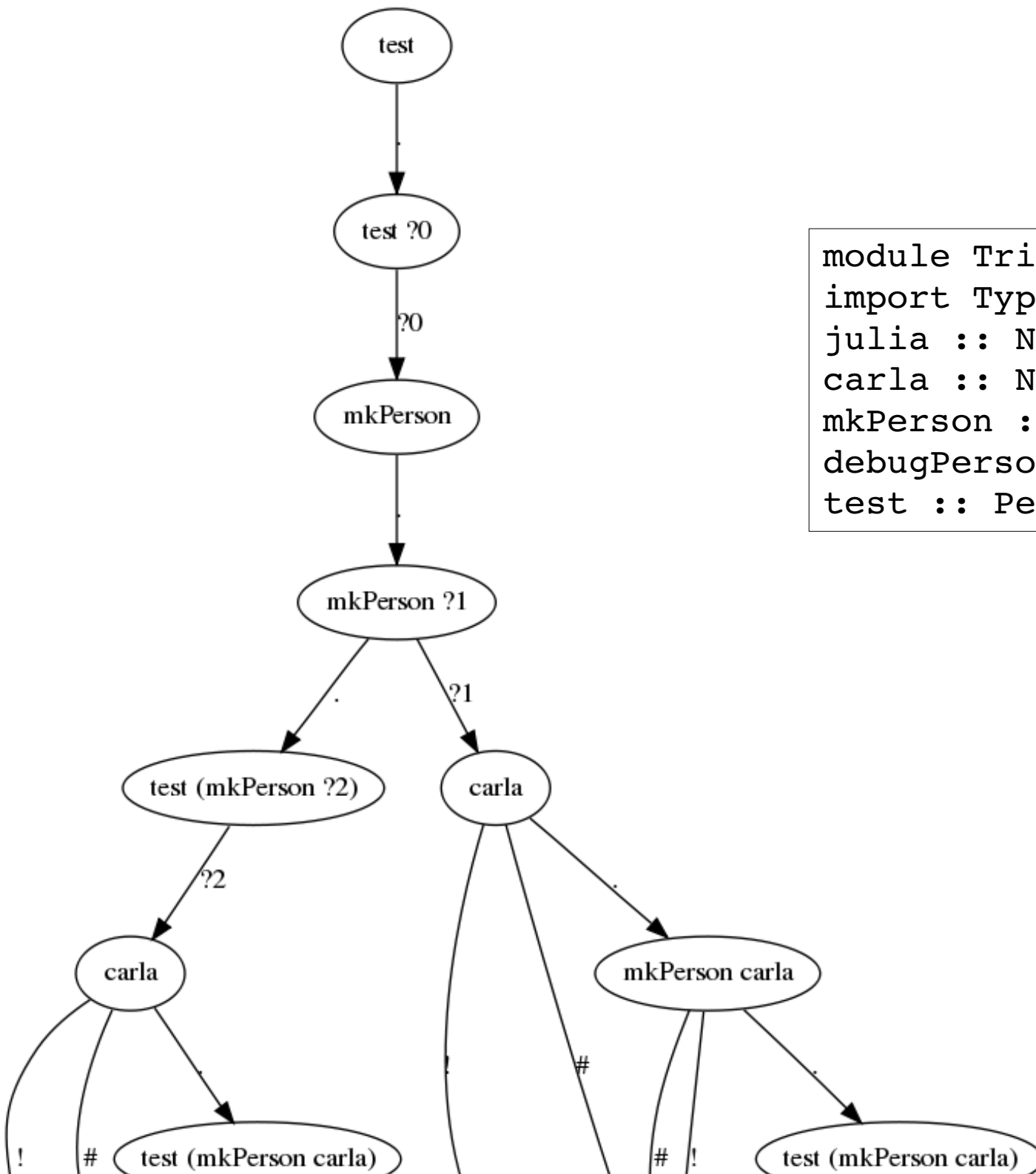
Plans

```
module Trivial1 where
import Types
julia :: Name
carla  :: Name
mkPerson :: Name → Person
debugPerson :: Person → String
test :: Person → Bool
```

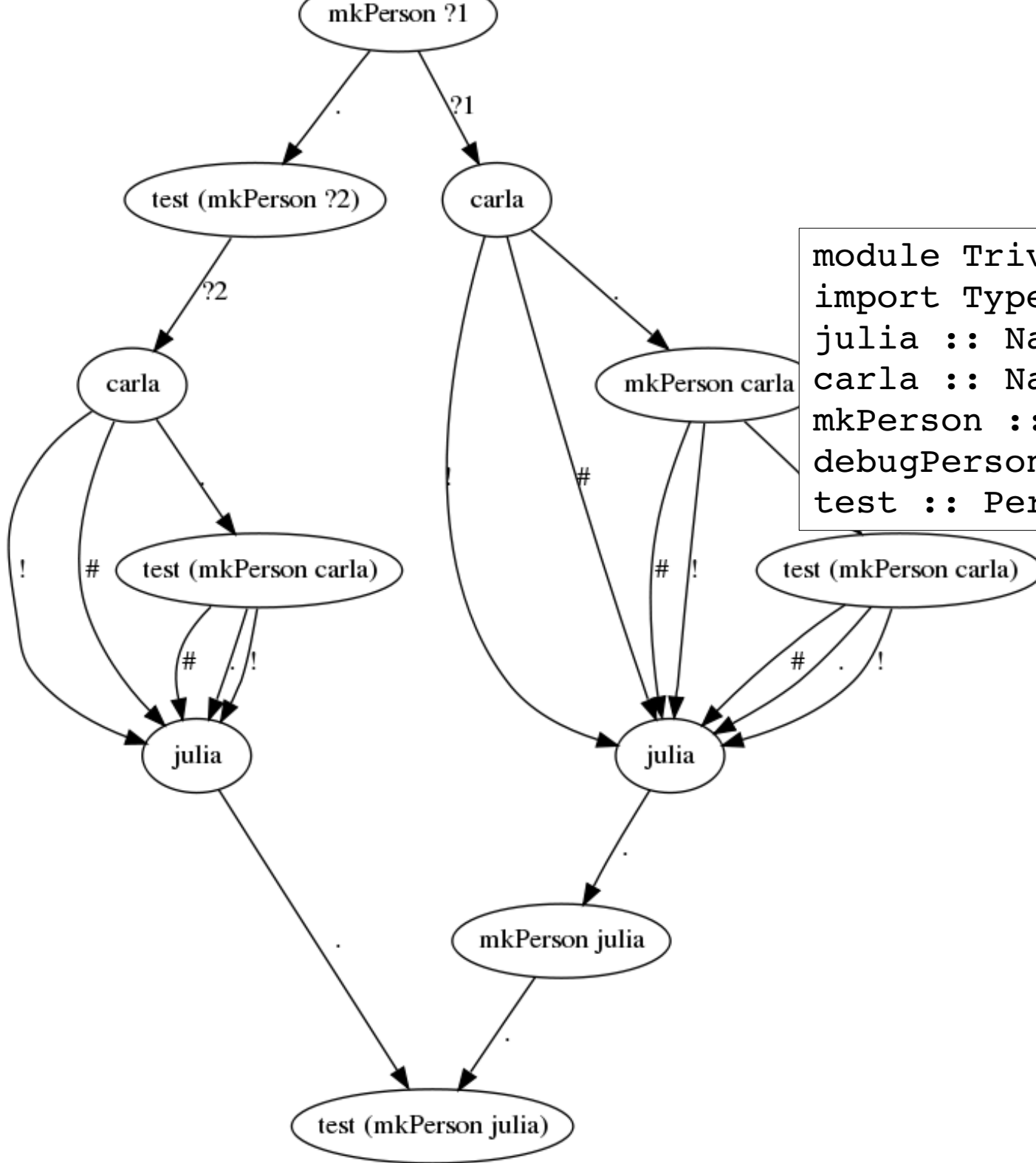


Plans

```
module Trivial1 where
import Types
julia :: Name
carla :: Name
mkPerson :: Name → Person
debugPerson :: Person → String
test :: Person → Bool
```



Plans



```
module Trivial1 where
import Types
julia :: Name
carla :: Name
mkPerson :: Name -> Person
debugPerson :: Person -> String
test :: Person -> Bool
```

TODO TODO TODO

- Understand polymorphism correctly
- Memory usage
- Expressions that fail to terminate (timeout)
- Chase down imports to build bigger sets of support
- Optimise
- Record selector functions
- Make safe for human consumption
- Continue having fun!

Cool crazy ideas

- Source code analysis
 - (ala Catch? Lazy Narrowing?)
- Make use of HPC / code coverage
- Analysing finite-sized data structures

Thank you for listening!

Tic-Tac-Toe



Tic-Tac-Toe

```
module Board(...) where
data Board deriving Eq
data Player deriving Eq
data Location deriving Eq
data GameOver = Win Player | Draw deriving Eq

emptyBoard :: Board
placePiece :: Player → Location → Board → Board
getPiece :: Location → Board → Maybe Player
hasWon :: Board → Maybe GameOver

noughts, crosses :: Player
t1, tm, tr, ml, mm, mr, bl, bm, br :: Location
```

Tic-Tac-Toe

- After placing a piece on an empty square, placing any other pieces should not change the piece on that square

```
prop_placePiece_fixed :: Board → Player → Location
                        → [(Player, Location)] → Bool
prop_placePiece_fixed inBoard player1 location1 otherPlaces
= isNothing (getPiece location1 inBoard) ==>
  (getPiece location1
   . flip (foldr (uncurry placePiece)) otherPlaces
   . placePiece player1 location1 $ inBoard
  ) == Just player1
```

Thoughts

- An example program with assertions
- Quick/Small/LS check motivation
- Need to add code
- Need to make instances
- But we have an API already!
- Or... irulan