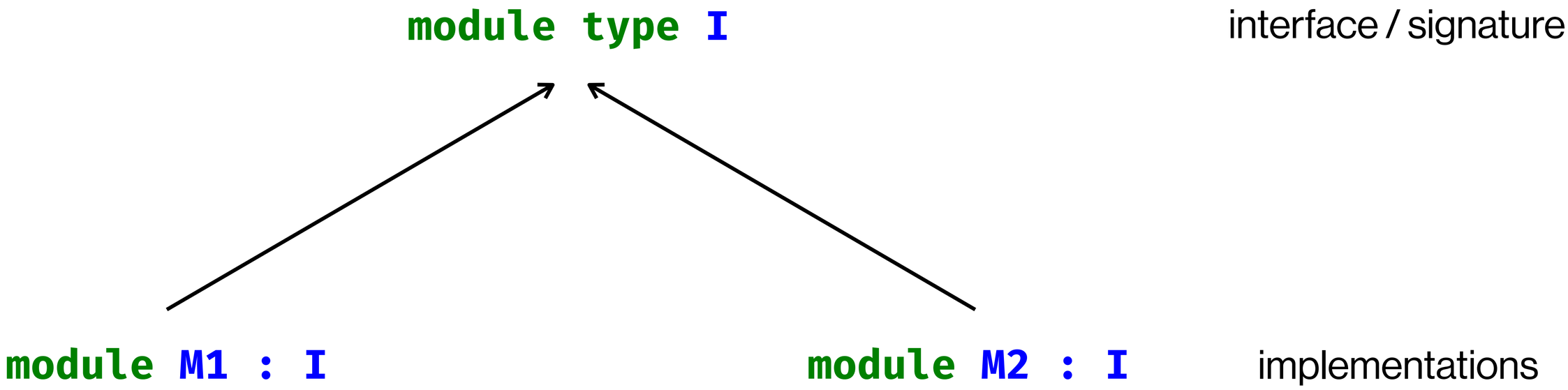
Mica

Automated Differential Testing for OCaml Modules

Ernest Ng
Advised by Harry Goldstein & Benjamin C. Pierce

Representation Independence



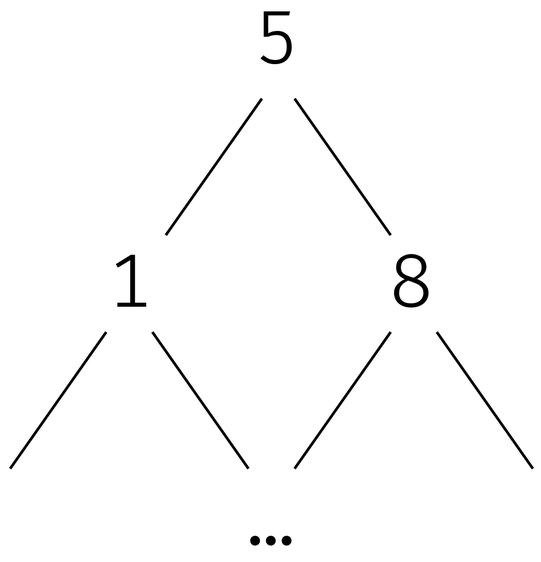
Example: **Finite Sets**

```
module type Set = sig
  type 'a t
  val empty : 'a t
  val add   : 'a → 'a t → 'a t
  val union : 'a t → 'a t → 'a t
  …
end
```

$$\{1,\ 5,\ 8,\ \ldots\} \rightsquigarrow [1;\ 5;\ 8;\ \ldots]$$

```
module ListSet : SetIntf = struct
  type 'a t = 'a list
  (* Invariant : no duplicates in list *)
  …
end
```

{1, 5, 8, …} ⇝



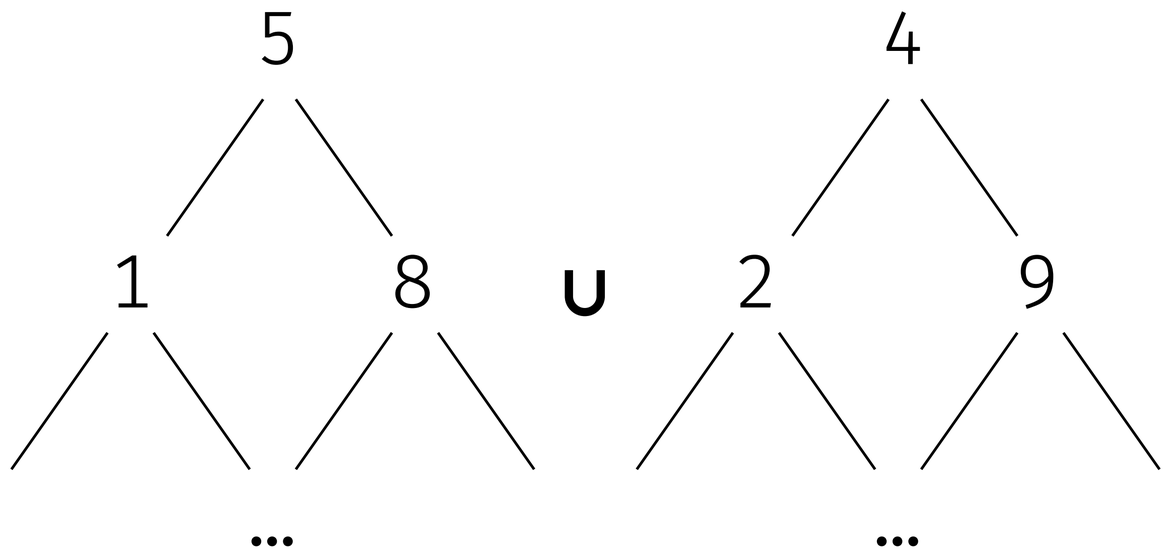

```
type 'a tree =
  | Empty
  | Node of 'a tree * 'a * 'a tree
```

```
module BSTSet : SetIntf = struct
  type 'a t = 'a tree
  (* BST invariants *)
  …
end
```

Are these equivalent?

{1, 5, 8} ∪ {2, 4, 9}

```
[1; 5; 8] ++ [2; 4; 9]
```



Observational equivalence

equivalent ***inputs*** $\longmapsto$ equivalent ***outputs***

How do we test for

observational equivalence ?

Property-Based Testing (PBT)

Property-Based Testing (PBT)

1. Write a ***property***

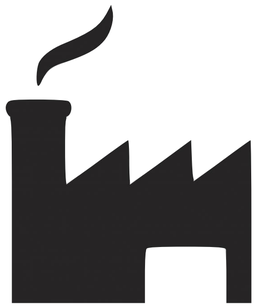
executable spec
describing
desired behavior

Property-Based Testing (PBT)

1. Write a ***property***

executable spec
describing
desired behavior

2. Generate ***random inputs***



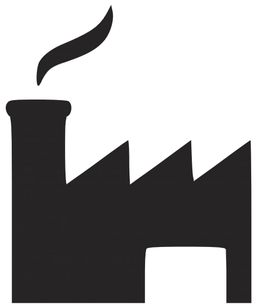
$\rightsquigarrow$ x_1 x_2 … x_n

Property-Based Testing (PBT)

1. Write a ***property***

executable spec
describing
desired behavior

2. Generate ***random inputs***



$\rightsquigarrow$ x_1 x_2 … x_n

3. Test if random inputs satisfy property

Property-Based Testing (PBT)

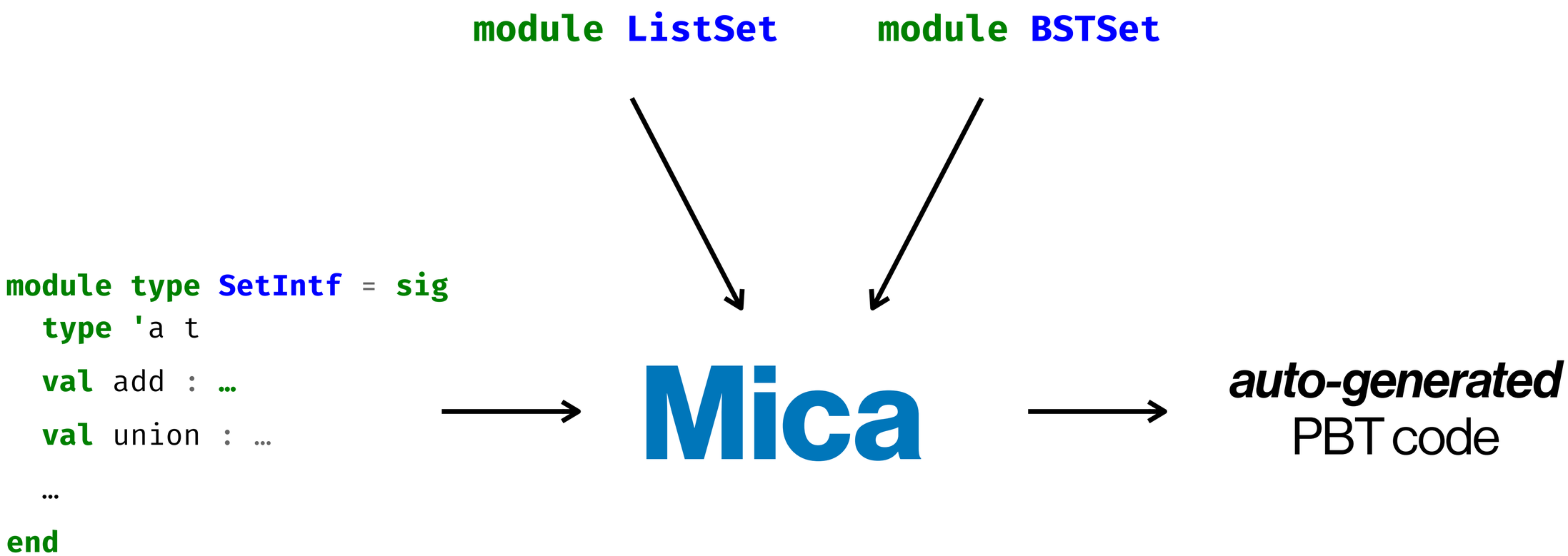
Popularised by **QuickCheck**

[Claessen & Hughes 2000]

Problem :

Writing PBT code for different modules requires ***significant programmer effort***!

Solution : **Mica**


module ListSet
module BSTSet
module type SetIntf = sig
type 'a t
val add : …
val union : …
…
end
Mica
auto-generated
PBT code

All of the following code is
automatically generated by Mica

Symbolic Expressions

```
module type Set = sig
  type 'a t                                  ⟷   type expr =
  val empty : 'a t                           ⟷     | Empty

  val add : 'a → 'a t → 'a t                 ⟷     | Add of int * expr

  val union: 'a t → 'a t → 'a t              ⟷     | Union of expr * expr
  …                                                 …
end
```

Types & Values

```
type ty =
   | Bool
   | Int
   | T
```

```
type value =
   | ValBool of bool
   | ValInt of int
   | ValT of int M.t
```

Mica automatically produces:

QuickCheck Generator
for **well-typed** symbolic expressions

```
val gen_expr : ty → expr Generator.t
```

`gen_expr` **T**

well-typed symbolic expressions
of type **T**

`gen_expr` **T**

well-typed symbolic expressions of type **T**

```
Intersect (Add 2 Empty) Empty
```

✓

`gen_expr` **T**

well-typed symbolic expressions of type **T**

```
Intersect (Add 2 Empty) Empty
```

✓

```
Is_empty (Size Empty)
```

✗

Mica automatically produces:

Interpreter for symbolic expressions

```
val interp : expr → value
```

Mica automatically produces :

Executable for testing observational equivalence

CORE_QUICKCHECK

Jane Street

Generator

generate ***random***
symbolic expressions

```
(Is_empty
    (Intersect (Rem 8 (Add 7 Empty))
               (Union (Add 2 Empty)
                      (Union Empty Empty))
```

Generator

generate ***random***
symbolic expressions

```
(Is_empty
    (Intersect (Rem 8 (Add 7 Empty))
               (Union (Add 2 Empty)
                      (Union Empty Empty))
```

Interpreter

interpret expressions
over modules

… → **module BSTSet** → …
… → **module ListSet** → …

Generator

generate ***random***
symbolic expressions

```
(Is_empty
    (Intersect (Rem 8 (Add 7 Empty))
               (Union (Add 2 Empty)
                      (Union Empty Empty))
```

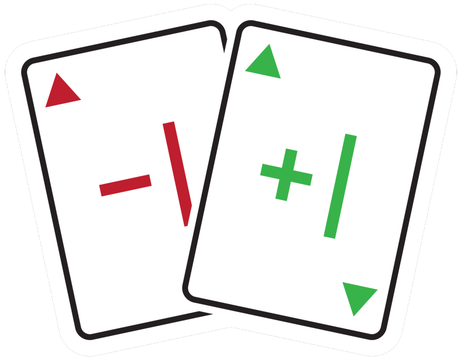
Interpreter

interpret expressions
over modules

… → **module BSTSet** → …
… → **module ListSet** → …

Executable

test for
observational
equivalence



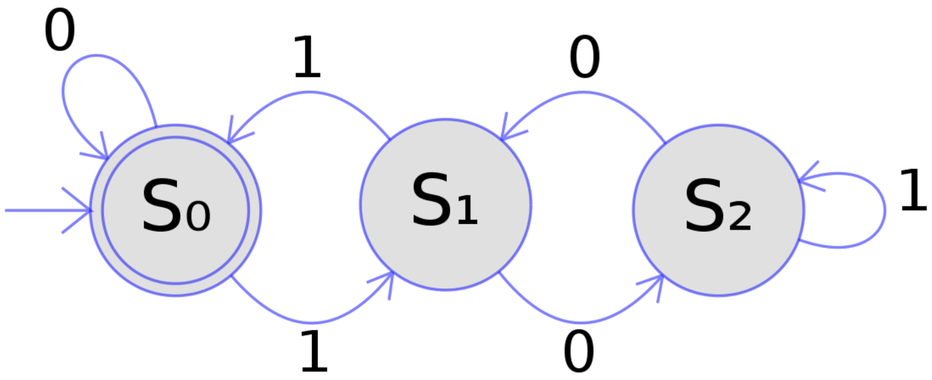
Case studies

Regex matching

Brzozowski derivatives

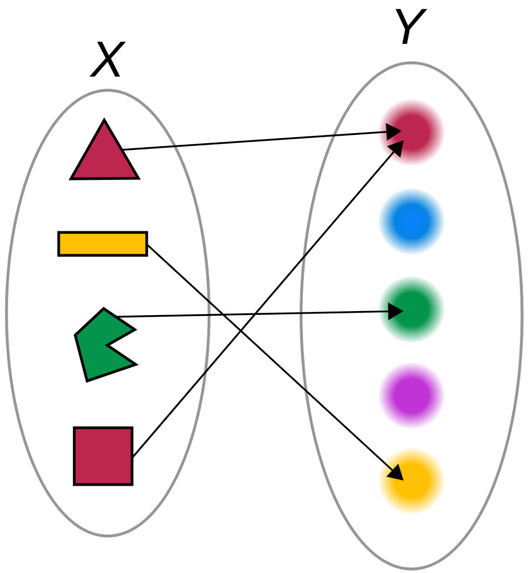
$$u^{-1}S = \{v \in \Sigma^* \mid uv \in S\}$$

DFAs

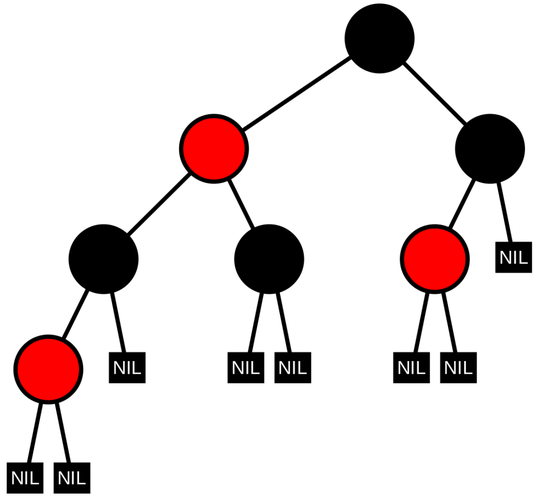


Functional maps

Association lists



Red-Black Trees

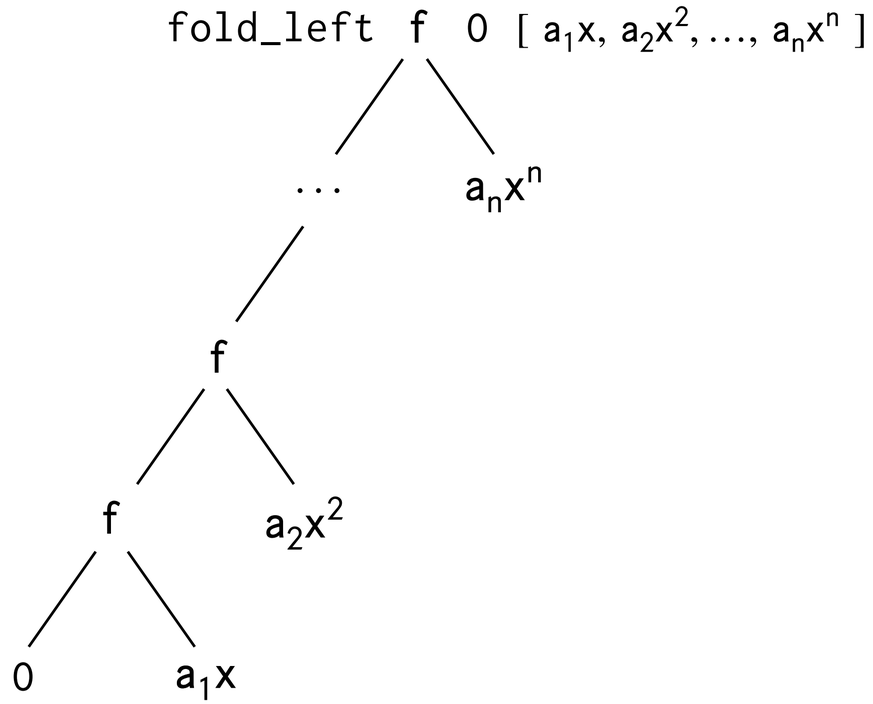


Polynomials

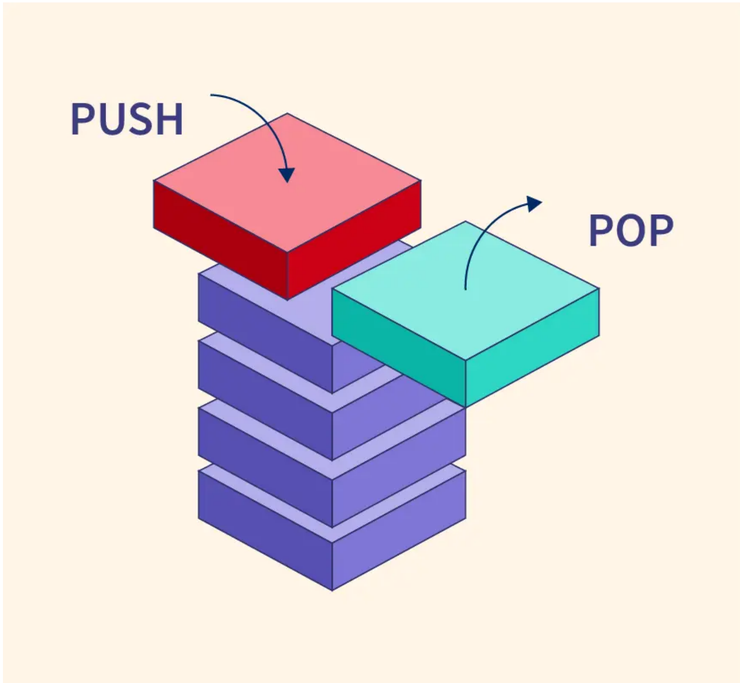
Horner's algorithm

$$
\begin{aligned}
p(x_0) &= a_0 + x_0\Big(a_1 + x_0\big(a_2 + \cdots + x_0(a_{n-1} + b_n x_0)\cdots\big)\Big) \\
&= a_0 + x_0\Big(a_1 + x_0\big(a_2 + \cdots + x_0 b_{n-1}\big)\Big) \\
&\;\;\vdots \\
&= a_0 + x_0 b_1 \\
&= b_0 .
\end{aligned}
$$

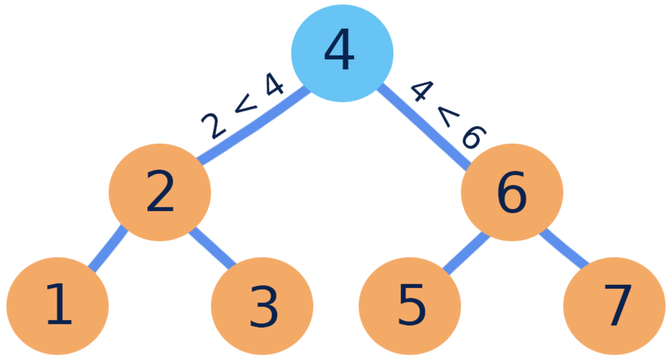
List fold



Stacks



Sets (BSTs, lists)



Takeaways

Takeaways

1. Checking observational equivalence requires significant programmer effort

Takeaways

1. Checking observational equivalence requires significant programmer effort
2. **Mica** can automate this process via PBT !

Thanks!

ngernest/**mica**

ngernest@seas.upenn.edu