Nitpick: A Counterexample Generator for Higher-Order Logic based on a Relational Model Finder

Jasmin C. Blanchette & Tobias Nipkow
Technische Universität München

Quickcheck

Berghofer & Nipkow, SEFM 2004

inspired by Haskell tool

based on random testing

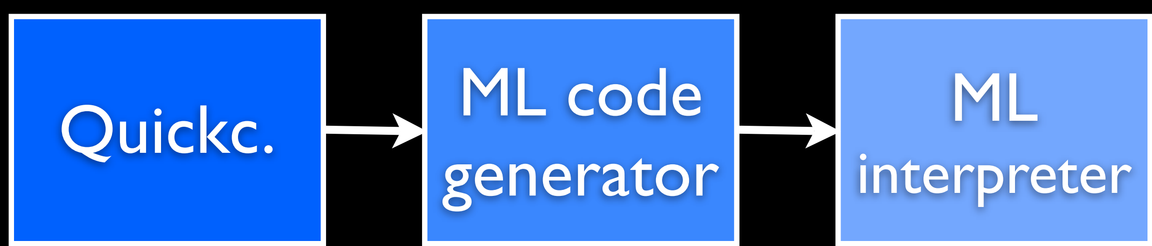
+ sound (no spurious counterexs.)

+ fast

– requires executability

Quickcheck

Berghofer & Nipkow, SEFM 2004



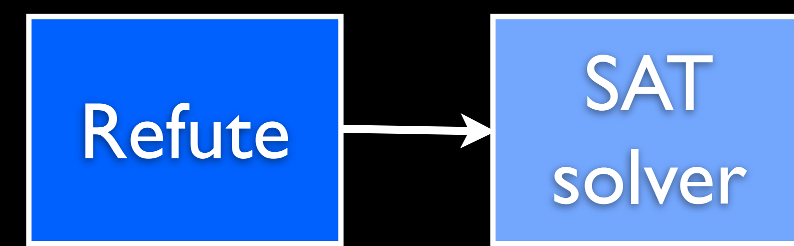
inspired by Haskell tool

based on random testing

+ sound (no spurious counterexs.)
+ fast
– requires executability

Refute

Weber, PDPAR 2004



SAT-based

finite approx. of infinite types

+ general-purpose
– unsound infinite types
– doesn't scale very well

Nitpick

second iteration of Refute

based on Kodkod (Alloy's backend)

handles definitional principles specially

optimizes common idioms

+ sound
+ general-purpose
+ scales better than Refute
– slower than Quickcheck

Kodkod's Logic:

First-Order Relational Logic (FORL)

universe: finite set of atoms

term: n-ary relation (set of atom n-tuples)

Kodkod's Logic:

First-Order Relational Logic (FORL)

universe: finite set of atoms

term: n-ary relation (set of atom n-tuples)

var pigeons = $\{a_1, \ldots, a_{30}\}$

var holes = $\{a_{31}, \ldots, a_{59}\}$

var $\varnothing \subseteq$ nest $\subseteq \{a_1, \ldots, a_{30}\} \times \{a_{31}, \ldots, a_{59}\}$

solve ($\forall$p $\in$ pigeons: one p.nest)

$\wedge$ ($\forall$h $\in$ holes: lone nest.h)

Basic Translation

Basic Translation

- finite first-order is easy:
  - scalars → singletons
  - functions → relations

Basic Translation

- finite first-order is easy:
  - scalars → singletons
  - functions → relations
- finite higher-order is also easy:
  - λ-abstractions → set comprehensions
  - $\sigma \to \tau$ argument → $|\sigma|$ arguments of type τ

Infinite Types and Partiality

Infinite Types and Partiality

- ★ **considers finite subsets of types**
  e.g. {0, 1, ..., K} for nat

Infinite Types and Partiality

- considers finite subsets of types
  e.g. {0, 1, ..., K} for nat
- { } = unknown value

Infinite Types and Partiality

- ★ considers finite subsets of types
  e.g. {0, 1, ..., K} for nat
- ★ { } = unknown value
- ★ functions may be partial
  e.g. Suc K gives { }

Infinite Types and Partiality

- considers finite subsets of types
  e.g. {0, 1, ..., K} for nat
- { } = unknown value
- functions may be partial
  e.g. Suc K gives { }
- f({ }) = { }

Infinite Types and Partiality

- considers finite subsets of types
  e.g. {0, 1, …, K} for nat
- { } = unknown value
- functions may be partial
  e.g. Suc K gives { }
- f({ }) = { }
- but: { } ∨ true = true

Definitional Principles

Inductive Predicates	**Coinductive Predicates**
Inductive Datatypes	**Coinductive Datatypes**
Recursive Functions	**Corecursive Functions**

Definitional Principles

Inductive Predicates	Coinductive Predicates
❶ Inductive Datatypes	Coinductive Datatypes
Recursive Functions	Corecursive Functions

Definitional Principles

2 Inductive Predicates	Coinductive Predicates
1 Inductive Datatypes	Coinductive Datatypes
Recursive Functions	Corecursive Functions

Definitional Principles

② Inductive Predicates	③ Coinductive Predicates
① Inductive Datatypes	Coinductive Datatypes
Recursive Functions	Corecursive Functions

Definitional Principles

2 Inductive Predicates	3 Coinductive Predicates
1 Inductive Datatypes	4 Coinductive Datatypes
Recursive Functions	Corecursive Functions

Inductive Datatypes

❶ Inductive Datatypes

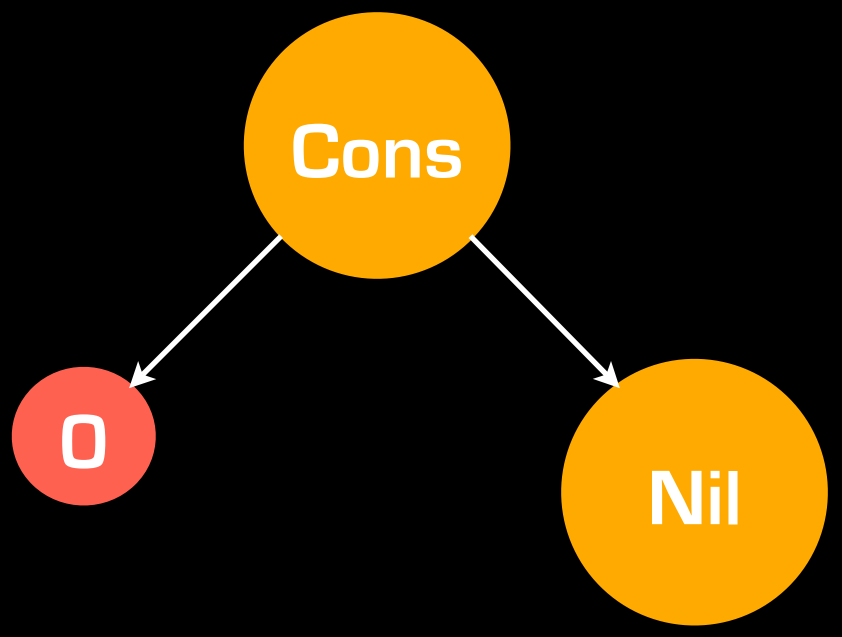
Based on Kuncak & Jackson, ESEC/FSE 2005

❶ Inductive Datatypes

Based on Kuncak & Jackson, ESEC/FSE 2005

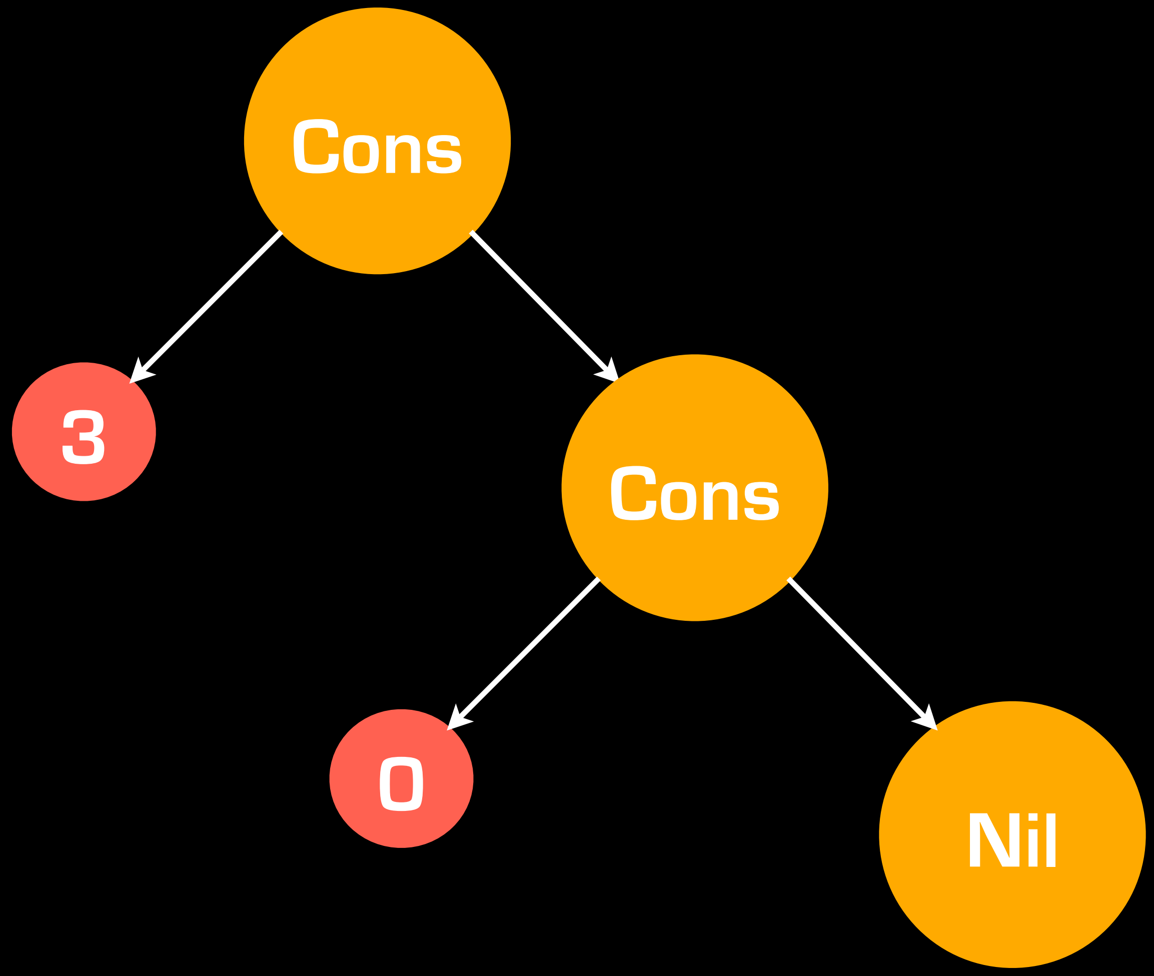
❶ Inductive Datatypes

Based on Kuncak & Jackson, ESEC/FSE 2005



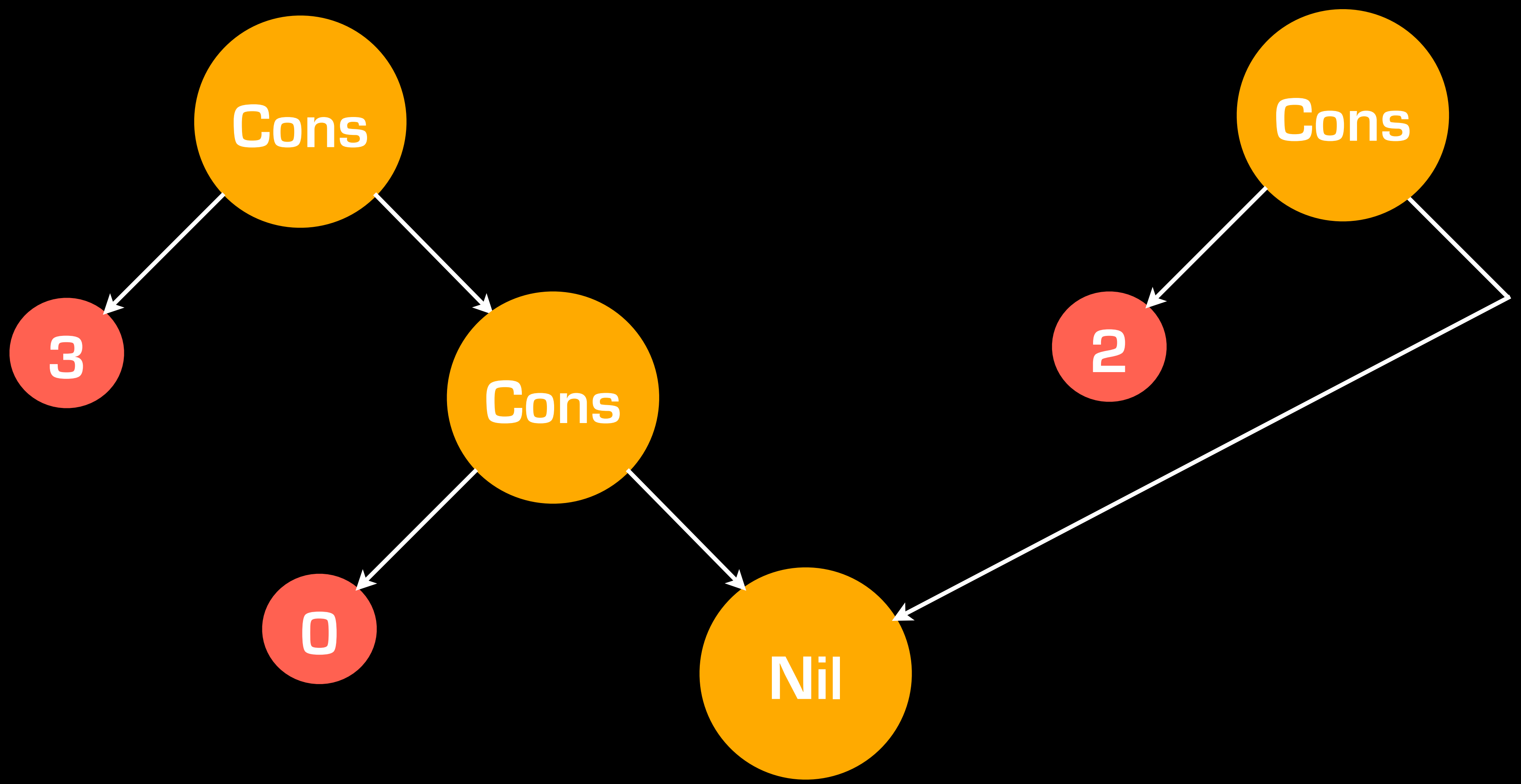
❶ Inductive Datatypes

Based on Kuncak & Jackson, ESEC/FSE 2005



❶ Inductive Datatypes

Based on Kuncak & Jackson, ESEC/FSE 2005



{[], [0], [3, 0], [2]}

❶ Inductive Datatypes

FORL axioms:

Selector

Uniqueness

Acyclicity

❶ Inductive Datatypes

FORL axioms:

Selector

Uniqueness

Acyclicity

S: $\neg\ \mathrm{null}\ xs \Rightarrow \mathrm{one}\ hd(xs) \wedge \mathrm{one}\ tl(xs)$

U: $(hd(xs), tl(xs)) = (hd(ys), tl(ys)) \Rightarrow xs = ys$

A: $(xs, xs) \notin tl^{+}$

Inductive Predicates

inductive even where

even 0

even n ⇒ even (Suc (Suc n))

fixpoint eq.: (overapprox.)

$$\text{even } x = (x = 0 \lor \exists n.\ x = \text{Suc } (\text{Suc } n) \land \text{even } n)$$

unrolled eq.: (underapprox.)

$$\text{even}_0\ x = \mathit{False}$$

$$\text{even}_{k+1}\ x = (x = 0 \lor \exists n.\ x = \text{Suc } (\text{Suc } n) \land \text{even}_k\ n)$$

❷ Inductive Predicates

inductive even **where**

even 0

even n ⇒ even (Suc (Suc n))

fixpoint eq.: (overapprox.)

$$\text{even } x = (x = 0 \lor \exists n.\ x = \text{Suc } (\text{Suc } n) \land \text{even } n)$$

unrolled eq.: (underapprox.)

$$\text{even}_0\ x = \mathit{False}$$

$$\text{even}_{k+1}\ x = (x = 0 \lor \exists n.\ x = \text{Suc } (\text{Suc } n) \land \text{even}_k\ n)$$

❷ Inductive Predicates

inductive even where

even 0

even n ⇒ even (Suc (Suc n))

fixpoint eq.:
(overapprox.)

even x =

(x = 0 ∨ ∃n. x = Suc (Suc n) ∧ even n)

❷ Inductive Predicates

inductive even **where**

even 0

even n ⇒ even (Suc (Suc n))

fixpoint eq.:
(overapprox.)

$$\text{even } x = (x = 0 \lor \exists n.\ x = \text{Suc } (\text{Suc } n) \land \text{even } n)$$

unrolled eq.:
(underapprox.)

$$\text{even}_0\ x = \textit{False}$$

$$\text{even}_{k+1}\ x = (x = 0 \lor \exists n.\ x = \text{Suc } (\text{Suc } n) \land \text{even}_k\ n)$$

	Pos.	Neg.
WF	fixp.	fixp.
NWF	unroll	fixp.

inductive

	Pos.	Neg.
WF	fixp.	fixp.
NWF	unroll	fixp.

inductive

	Pos.	Neg.
WF	fixp.	fixp.
NWF	fixp.	unroll

coinductive

❸ Coinductive Predicates

	Pos.	Neg.
WF	fixp.	fixp.
NWF	unroll	fixp.

inductive

	Pos.	Neg.
WF	fixp.	fixp.
NWF	fixp.	unroll

coinductive

❸ Coinductive Predicates

	Pos.	Neg.
WF	fixp.	fixp.
NWF	unroll	fixp.

Termination provers!

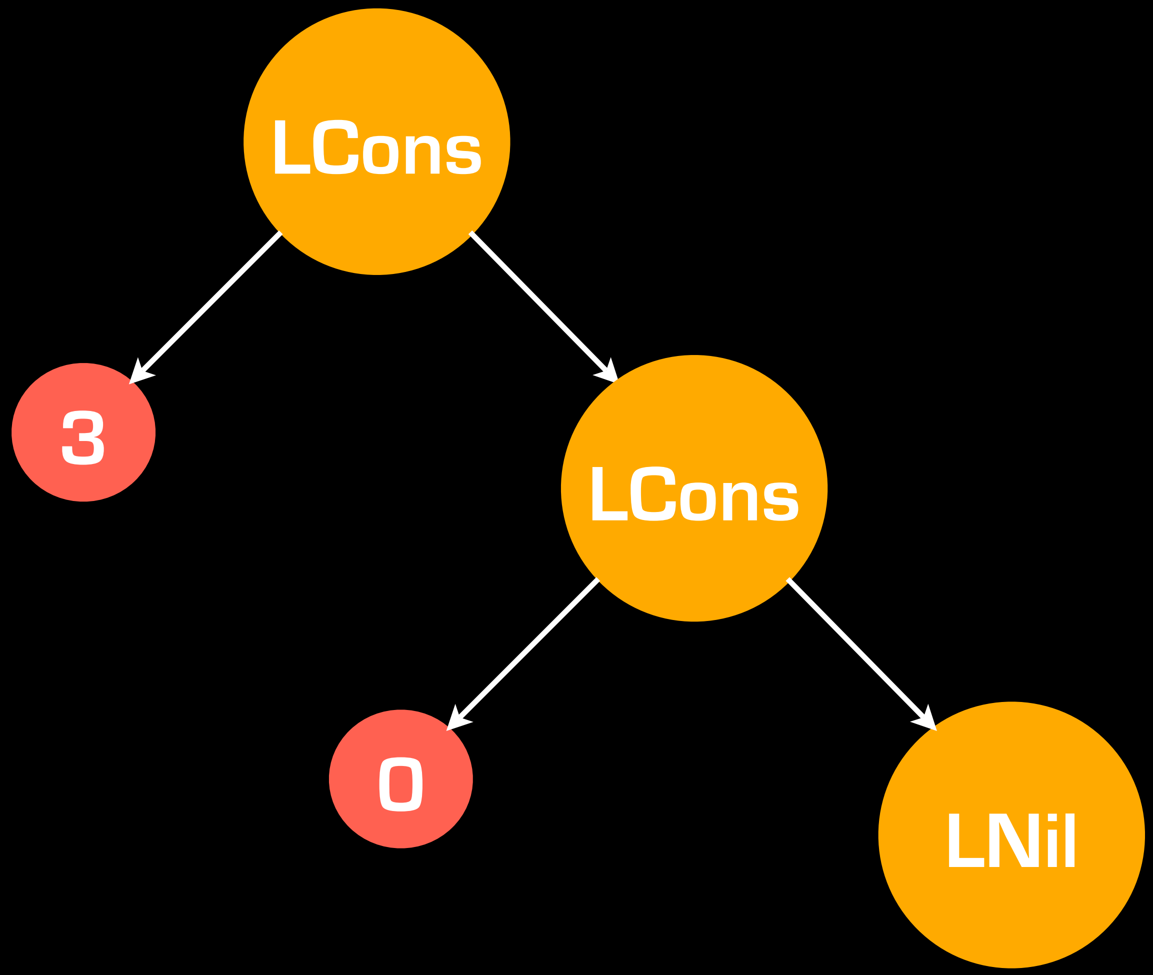
inductive

	Pos.	Neg.
WF	fixp.	fixp.
NWF	fixp.	unroll

coinductive

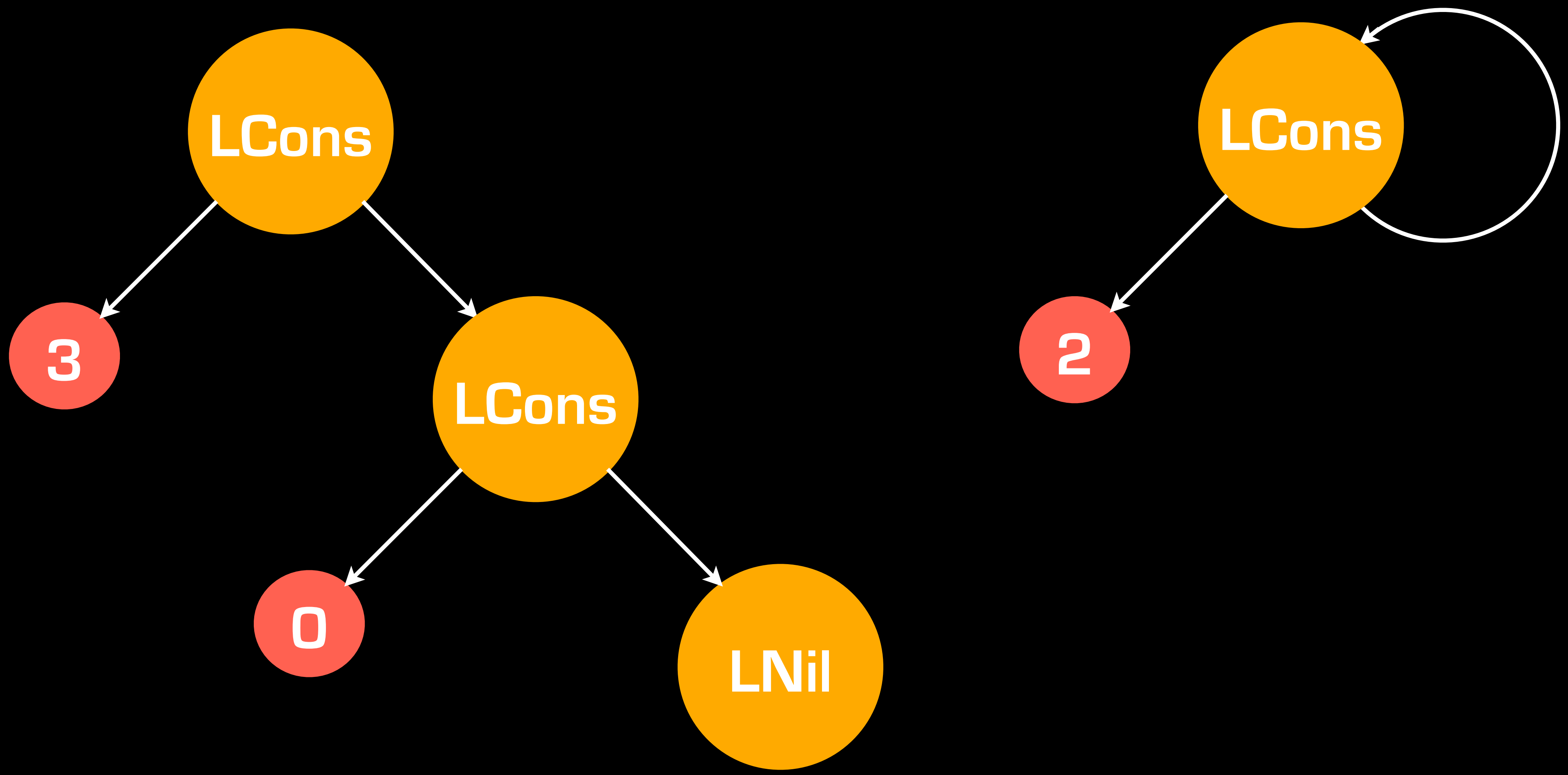
Coinductive Datatypes

④ Coinductive Datatypes



{[], [0], [3, 0], [2, 2, 2, ...]}

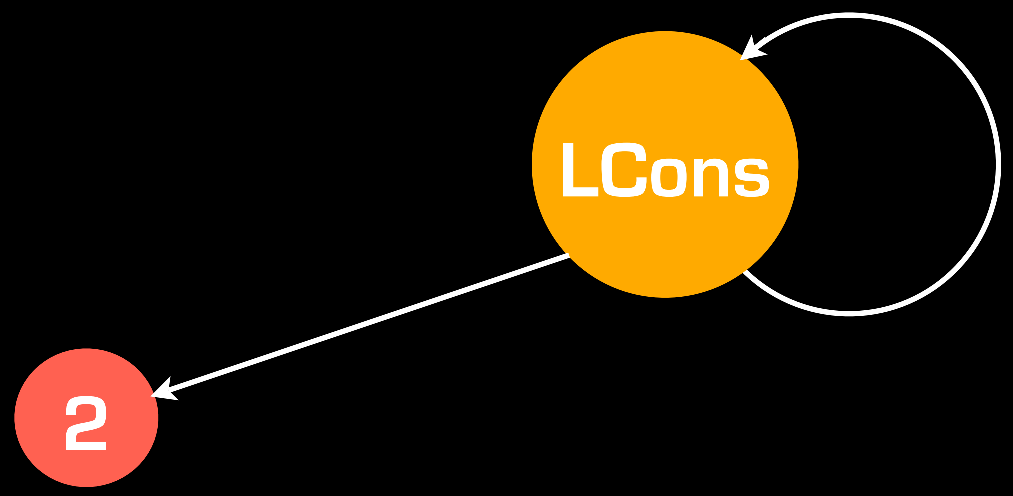
4 Coinductive Datatypes



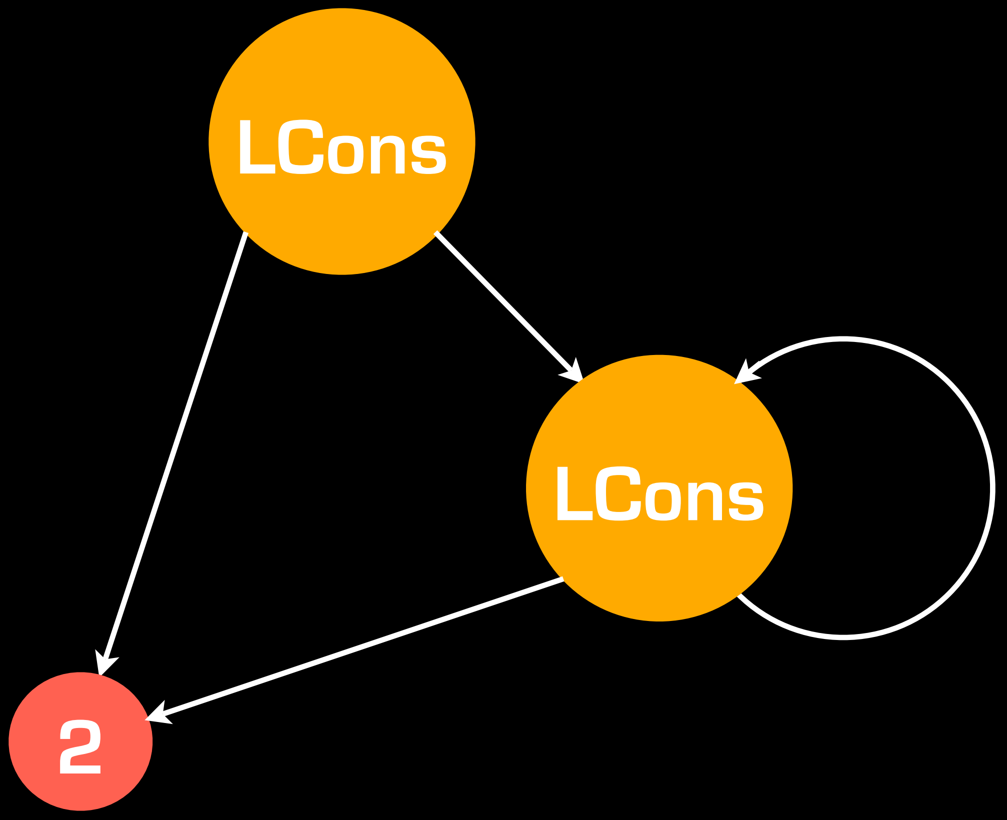
{[], [0], [3, 0], [2, 2, 2, ...]}

❹ Coinductive Datatypes

❹ Coinductive Datatypes

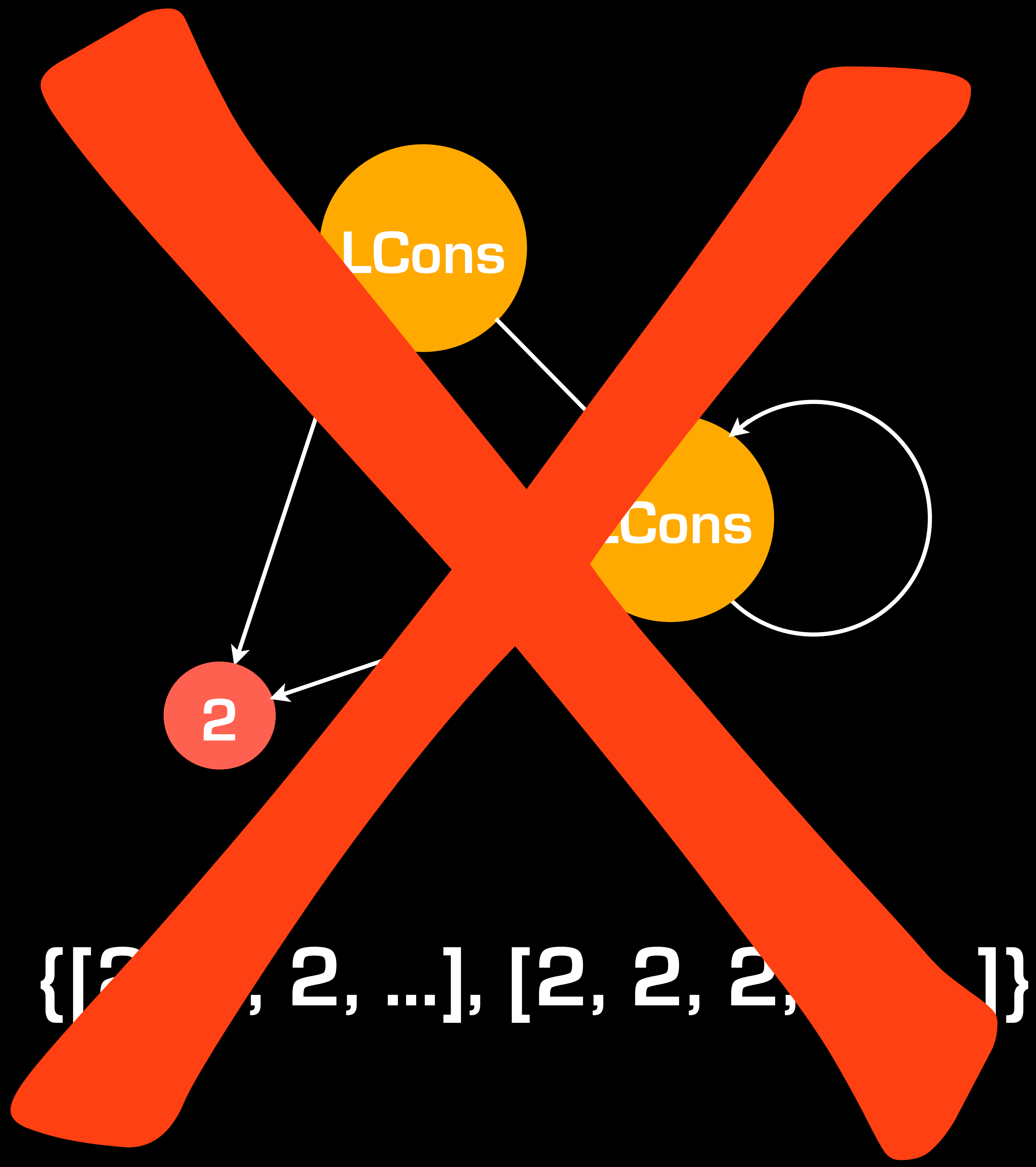


❹ Coinductive Datatypes



{[2, 2, 2, ...], [2, 2, 2, 2, ...]}

❹ Coinductive Datatypes



❹ Coinductive Datatypes

FORL axioms:

Selector

Uniqueness

Acyclicity

S: $\neg\ \text{null}\ xs \implies \text{one}\ hd(xs) \wedge \text{one}\ tl(xs)$

U: $(hd(xs), tl(xs)) = (hd(ys), tl(ys)) \implies xs = ys$

A: $(xs, xs) \notin tl^{+}$

❹ Coinductive Datatypes

FORL axioms:

Selector

Uniqueness

~~Acyclicity~~

S: $\neg\ \mathsf{null}\ xs \;\Rightarrow\; \mathsf{one}\ hd(xs) \wedge \mathsf{one}\ tl(xs)$

U: $(hd(xs), tl(xs)) = (hd(ys), tl(ys)) \;\Rightarrow\; xs = ys$

~~A: $(xs, xs) \not\subseteq tl^{+}$~~

❹ Coinductive Datatypes

FORL axioms:

Selector

Uniqueness

~~**A**cyclicity~~

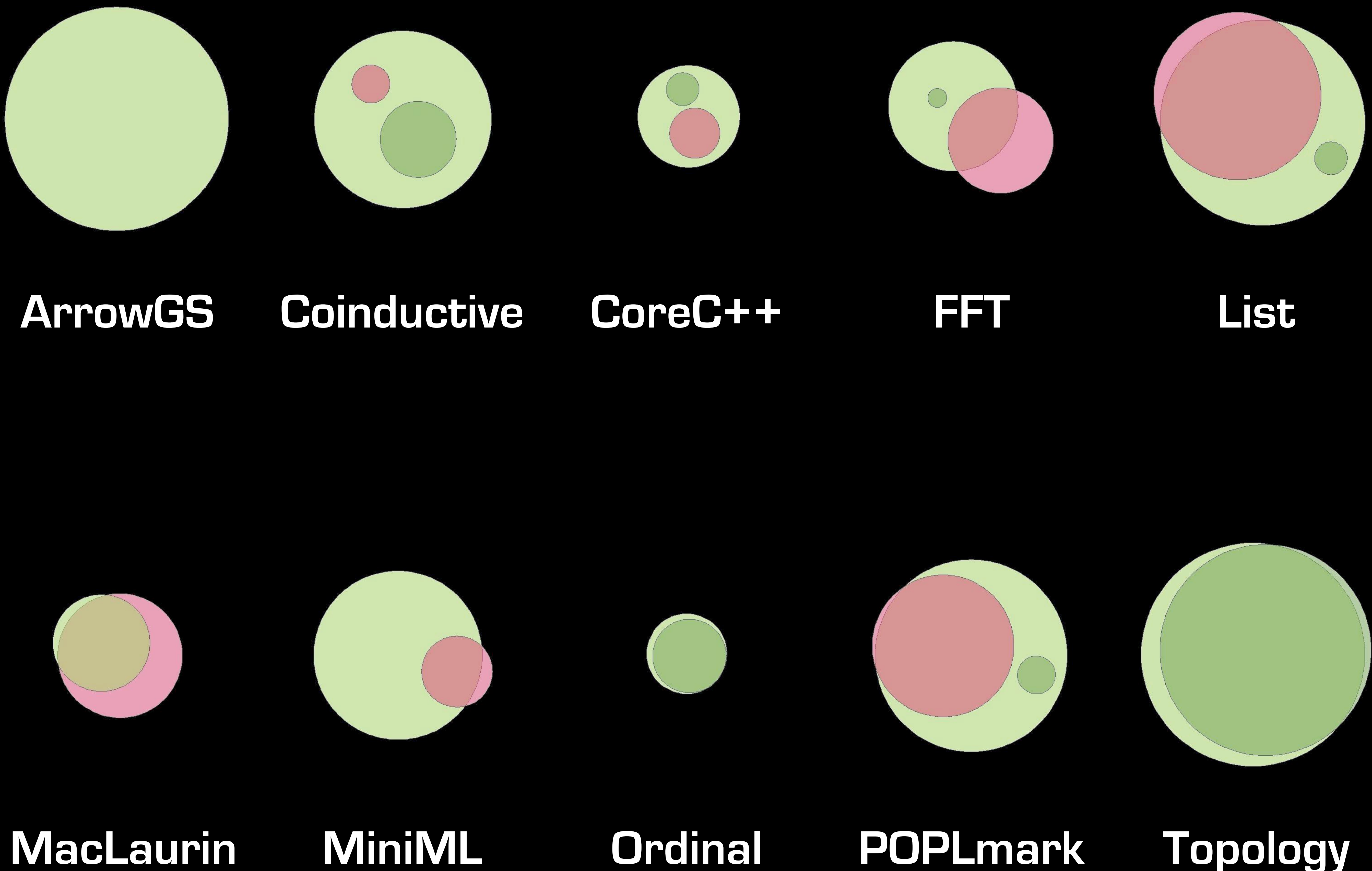
Bisimilarity

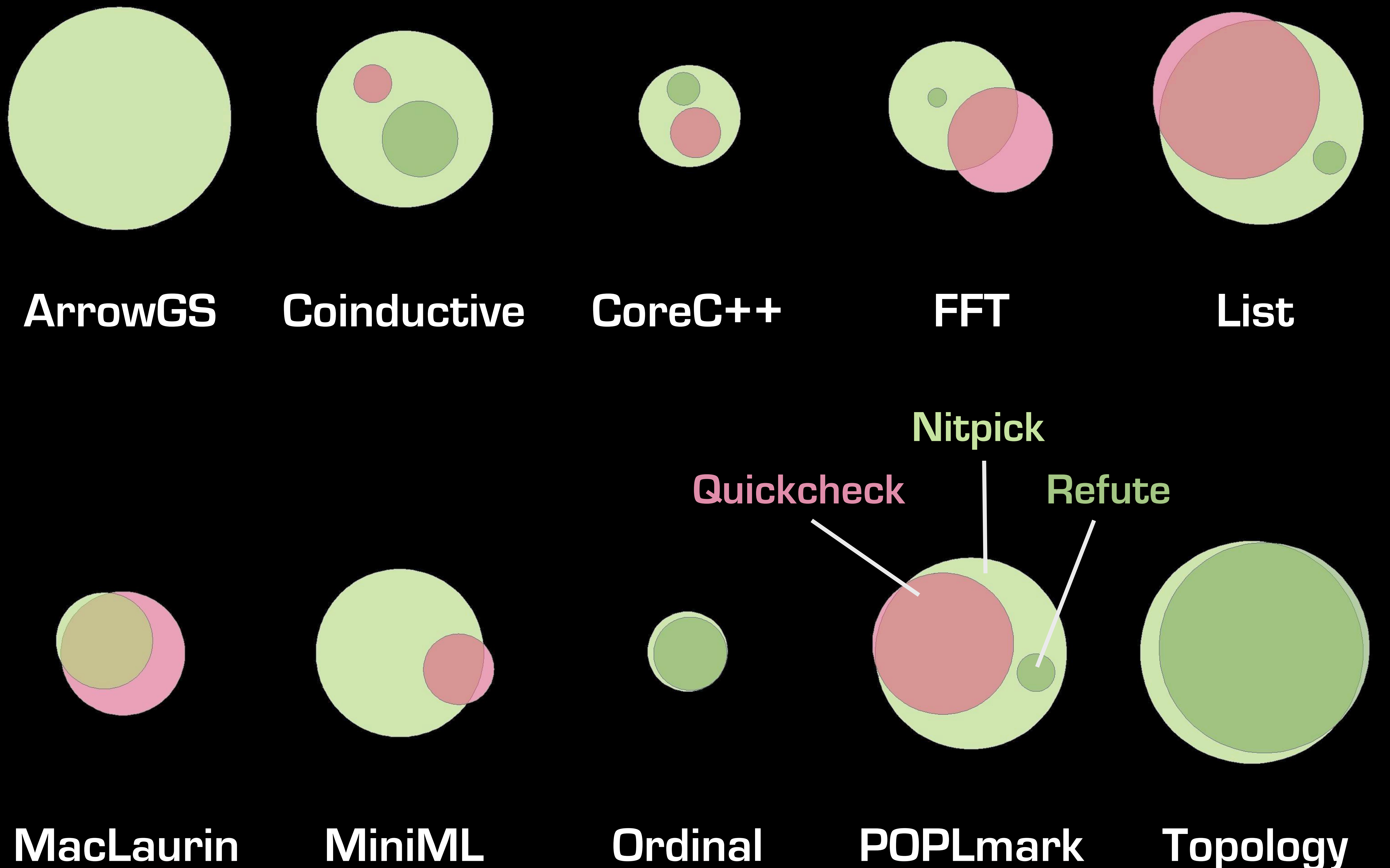
S: $\neg$ null xs $\Rightarrow$ one hd(xs) $\wedge$ one tl(xs)

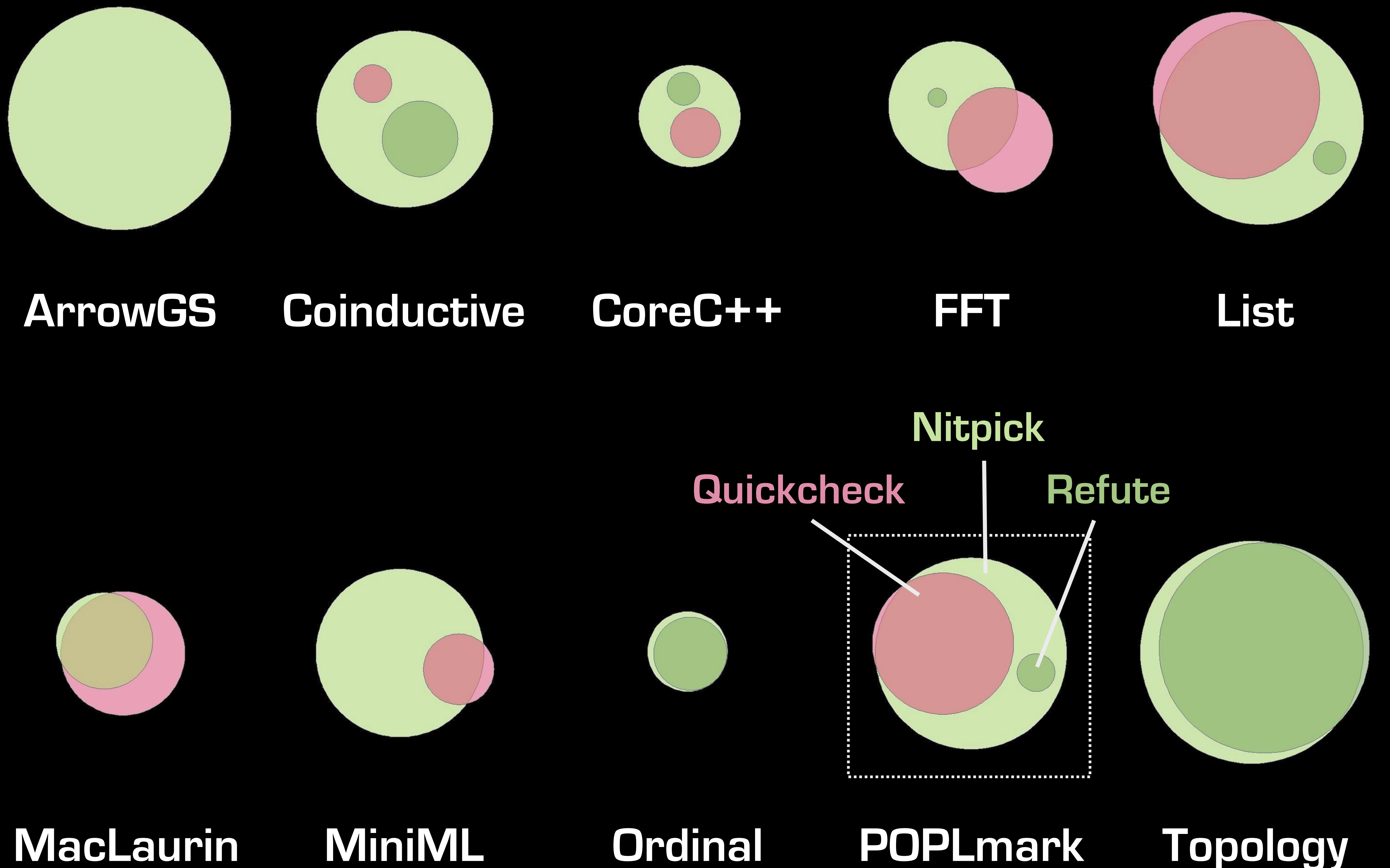
U: (hd(xs), tl(xs)) = (hd(ys), tl(ys)) $\Rightarrow$ xs = ys

~~A: (xs, xs) $\not\subseteq$ tl$^+$~~

B: xs ~ ys $\Rightarrow$ xs = ys


ArrowGS
Coinductive
CoreC++
FFT
List
MacLaurin
MiniML
Ordinal
POPLmark
Topology


ArrowGS
Coinductive
CoreC++
FFT
List
Nitpick
Quickcheck
Refute
MacLaurin
MiniML
Ordinal
POPLmark
Topology



ArrowGS
Coinductive
CoreC++
FFT
List
Nitpick
Quickcheck
Refute
MacLaurin
MiniML
Ordinal
POPLmark
Topology

Conclusion

Conclusion

SAT-based like Refute,
but benefits from Kodkod's optimizations

Conclusion

SAT-based like Refute,
but benefits from Kodkod's optimizations

Efficient and precise SAT coding of
(co)inductive predicates & datatypes

Conclusion

SAT-based like Refute,
but benefits from Kodkod's optimizations

Efficient and precise SAT coding of
(co)inductive predicates & datatypes

Saves time
and encourages playful exploration

"We are currently trying out the new Nitpick tool and it works very nicely with some of our theories."

"What a fast tool."

"Nitpick sparte mir letzte Woche einige Stunden ein."

"Nitpick rocks! Otherwise I would have actually written code just to enumerate stupid finite relations. Now I just had to write down the properties."