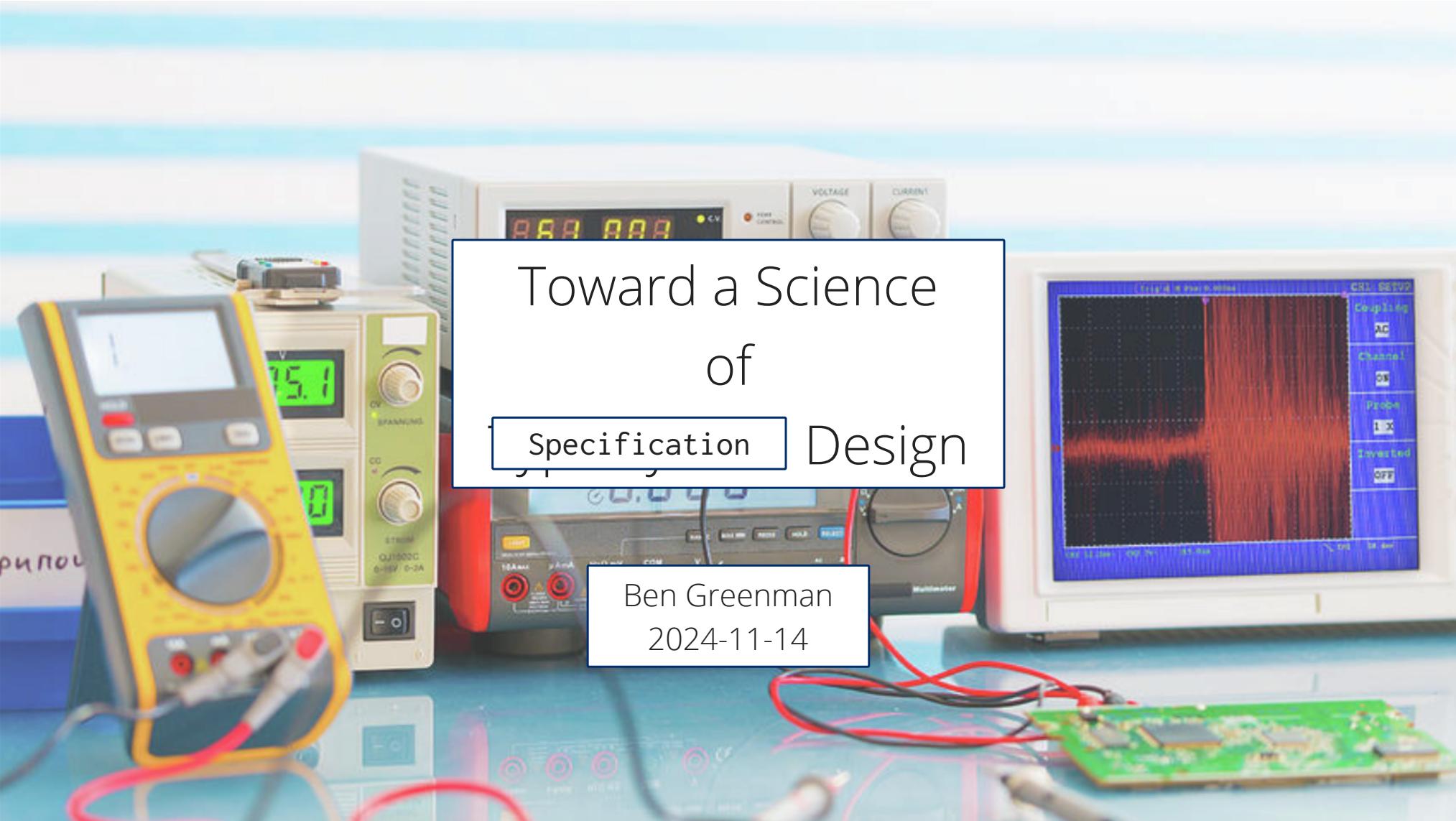


Toward a Science of Type System Design

Ben Greenman
2024-11-14



Toward a Science of Specification Design

Ben Greenman
2024-11-14

Science is ...

Science is ...

Proofs and Refutations

The Logic of
Mathematical Discovery

Imre Lakatos

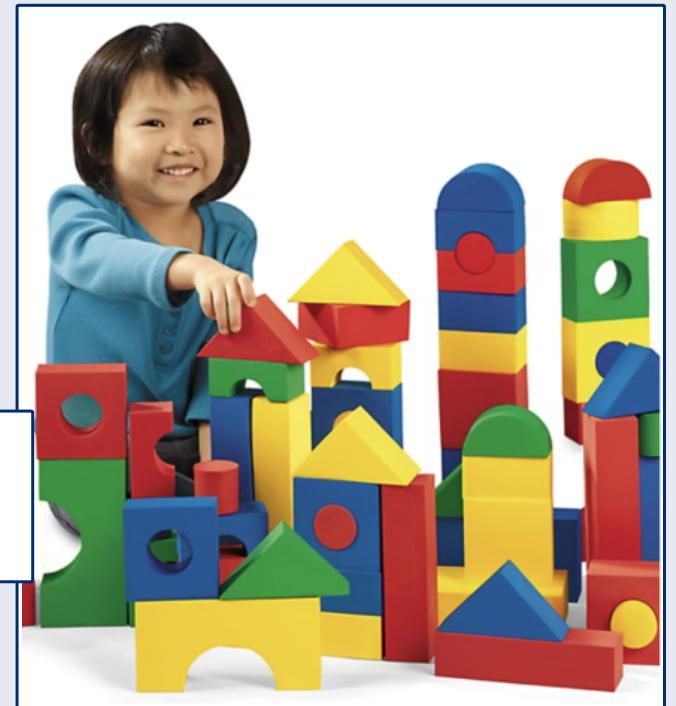
... a search for truth
via **conjectures** and **refutations**.



Programs are ...

Programs are ...

... building blocks,
code says **how** to act





Types are ...

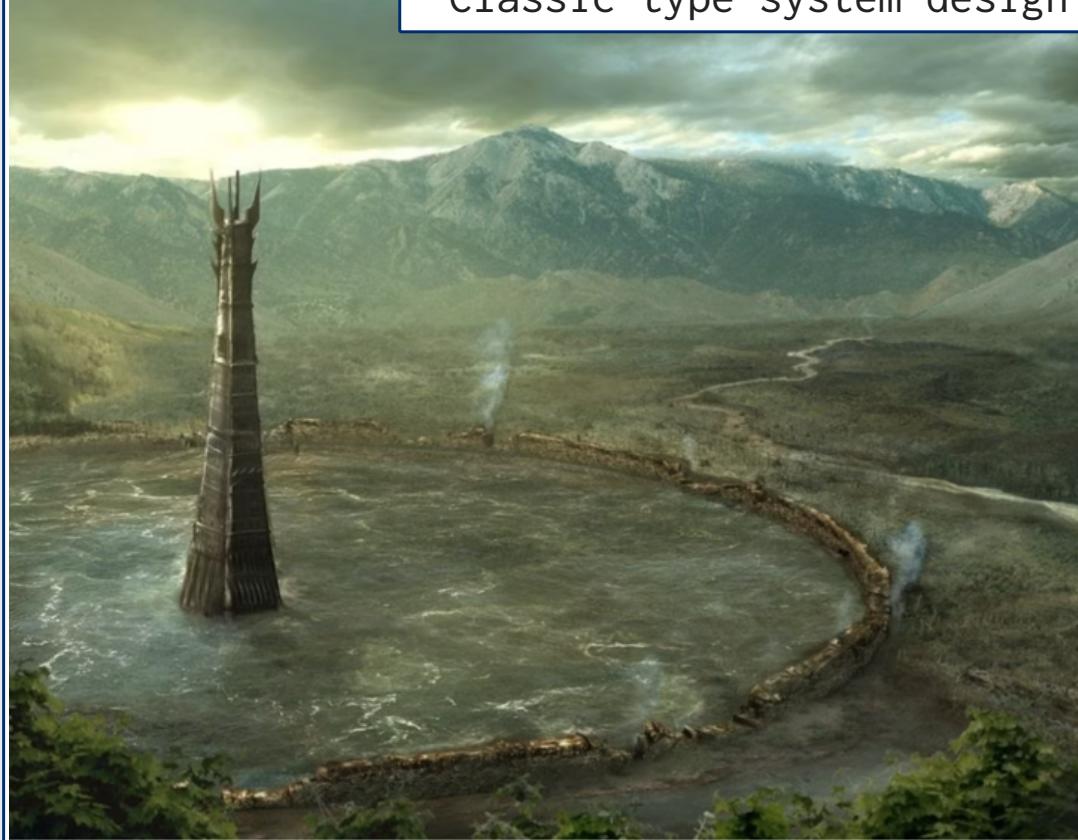
Types are ...

... guardrails,
rules for **what** to do
(or not do!)



Type system design?

Classic type system design

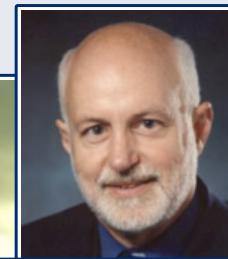




Classic type system design

Robin Milner





Classic type system design

Bob Constable

??



Type system design needs
methods for making refutations



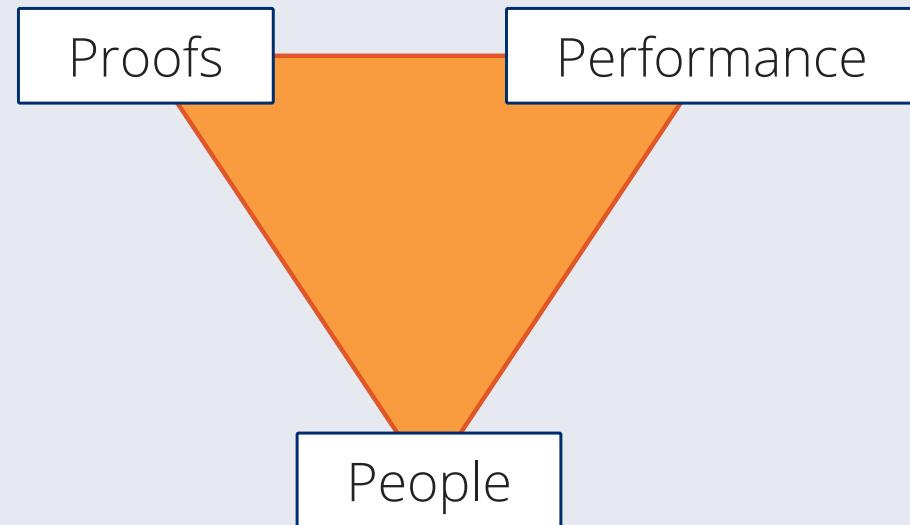


Image credit: Alex Aiken



Gradual Typing

Untyped

Typed



Gradual Typing

Untyped ➤ Typed

```
def join(d0,d1,sort,how):  
    ...
```

DataFrame

bool

Left|Right

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
    -> DataFrame:  
    ...
```

Types where useful, that's all.

Now, what do types mean?

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
-> DataFrame:  
    ...
```

join("hello", ...)

Is **d0** really a data frame?

Now, what do types mean?

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
-> DataFrame:  
    ...
```

join("hello", ...)

Is **d0** really a data frame?

Ideally YES





"The system **lives up to all expectations** that developers have of sound language implementations."

worst case: **2x** runtime overhead





"The system **lives up to all expectations** that developers have of sound language implementations."

worst case: **2x** runtime overhead

"My runtime went from **1 ms to 10 seconds!**"

warning on use trie functions in #lang racket?

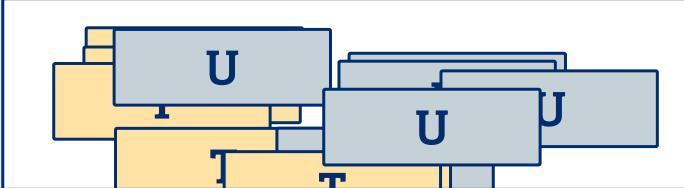


johnbclements

to Racket Users

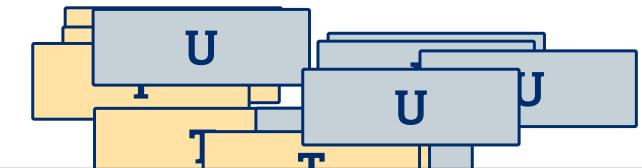
This program constructs a trie containing exactly two keys; ea

mmoon to be nes in the fourth of the tree as described in R6C



Typed Racket

What do **sound types** cost?



Typed Racket

What do **sound types** cost?



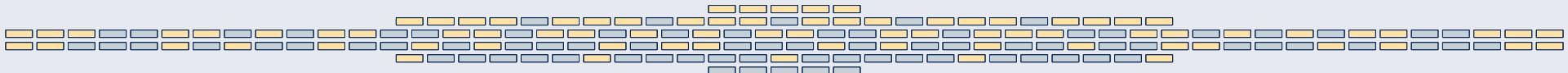
1. Start with a program

```
def join(d0,d1,sort,how):  
    ....
```

2. Add full types

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
    -> DataFrame:  
    ....
```

3. Explore all configurations



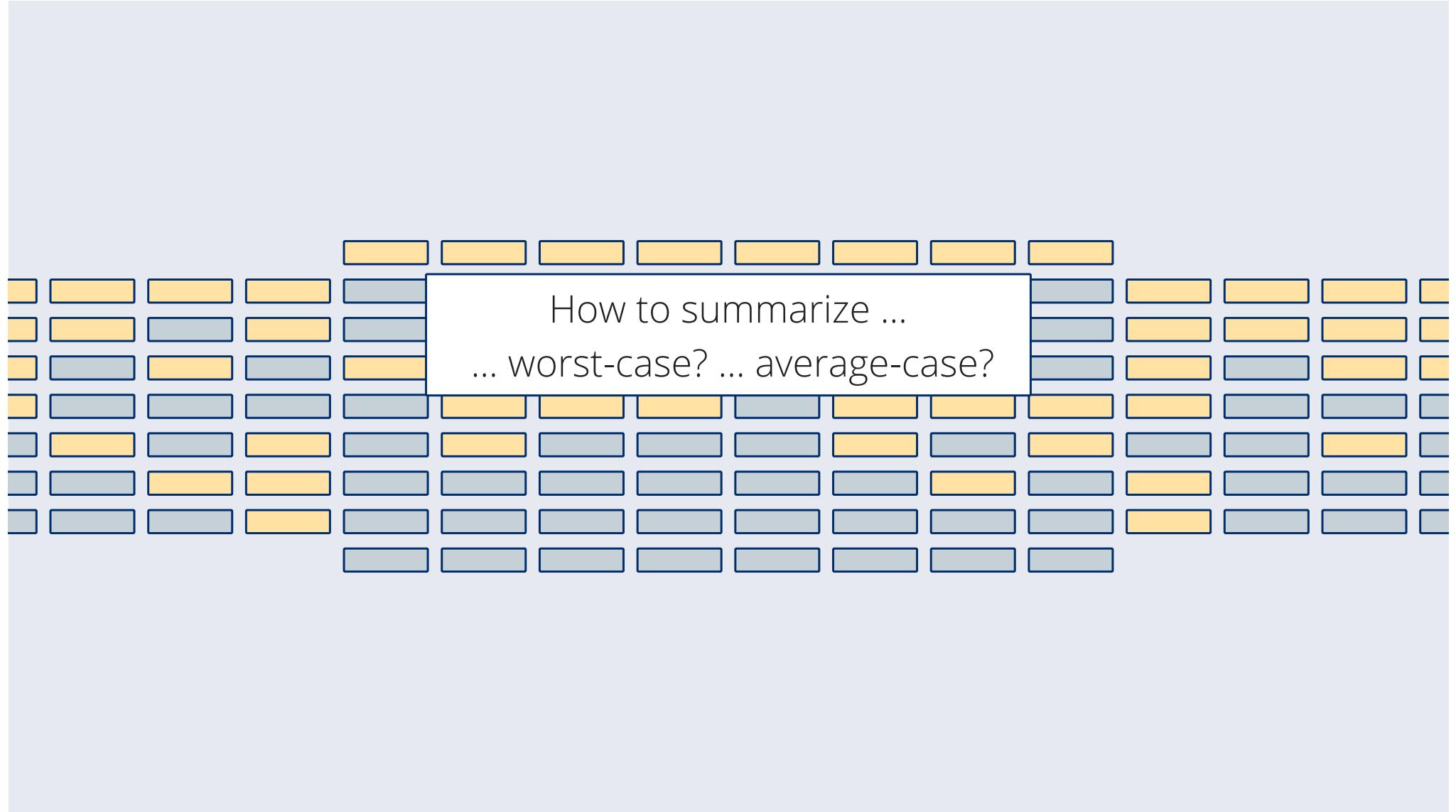


GTP Benchmarks

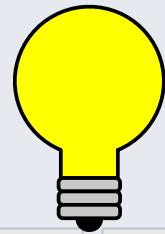
REP'23: 21 programs, +40k combos

Table 1: Benchmarks overview: purpose and characteristics

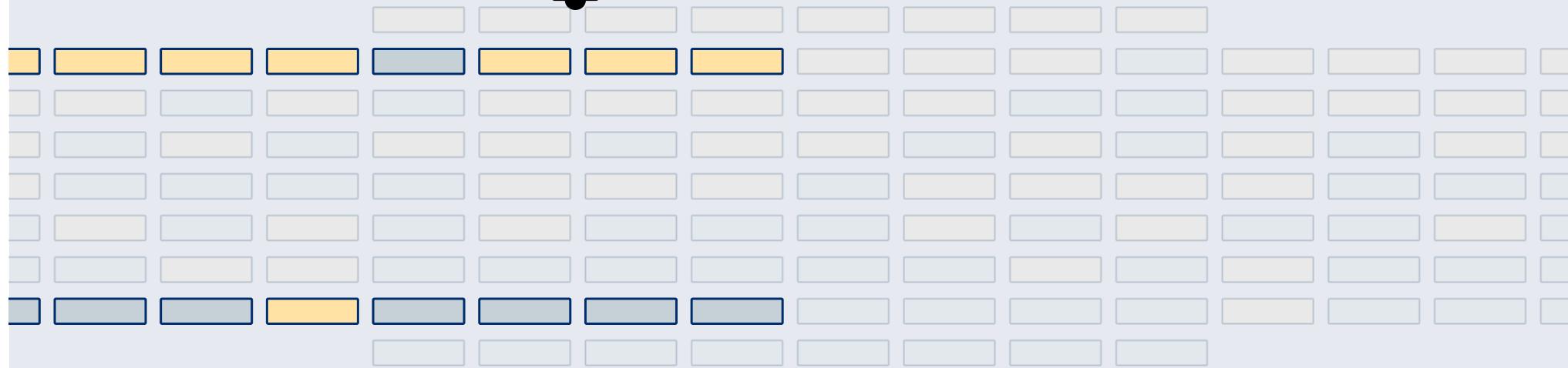
Benchmark	Purpose	T Init	U Lib	T Lib	Adapt	HOF	Poly	Rec	Mut	Imm	Obj	Cls
sieve	<i>prime generator</i>	○	○	○	●	○	○	●	○	●	○	○
forth	<i>Forth interpreter</i> [51]	○	○	○	○	○	○	●	○	●	●	●
fsm	<i>economy simulation</i> [33]	○	○	○	○	○	○	○	●	●	○	○
fsmoo	<i>economy simulation</i> [34]	○	○	○	○	○	○	○	●	●	●	○
mbta	<i>subway map</i>	●	●	○	○	○	○	○	○	○	●	○
morsecode	<i>Morse code trainer</i> [23, 148]	○	○	○	○	○	○	○	●	○	○	○
zombie	<i>HTDP game</i> [151]	○	○	○	●	●	○	●	○	●	○	○
zordoz	<i>bytecode tools</i> [53]	○	●	○	●	●	○	●	●	●	○	○
dungeon	<i>maze generator</i>	○	○	○	○	●	●	●	●	●	●	●
inra	<i>image tools</i> [161]	●	●	●	○	○	○	●	●	●	●	●

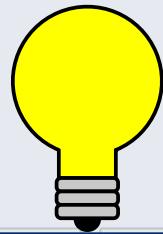


How to summarize ...
... worst-case? ... average-case?

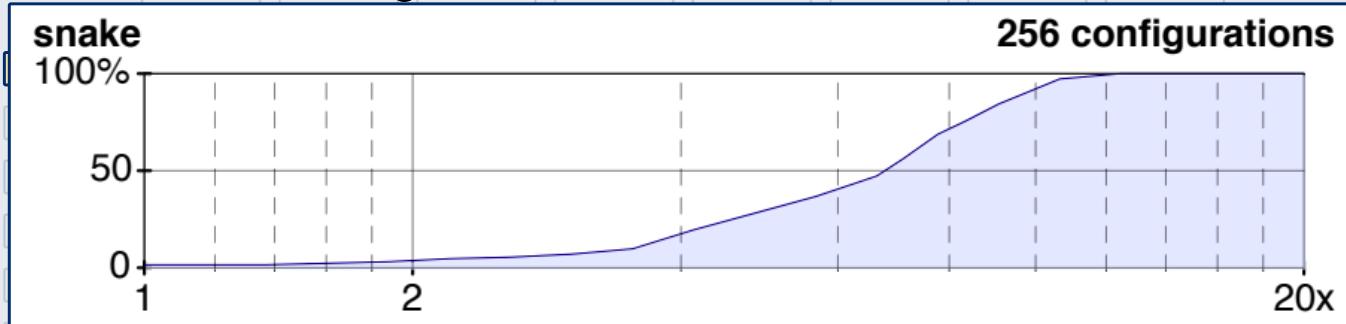


Key: **think like a user**
too slow = useless



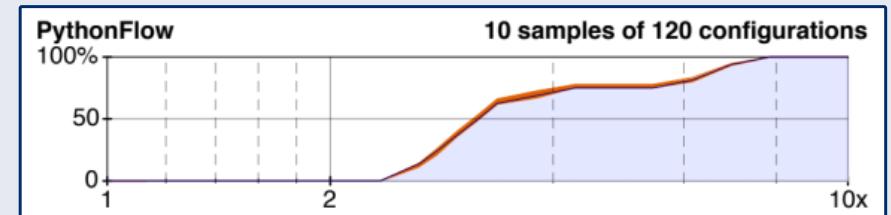
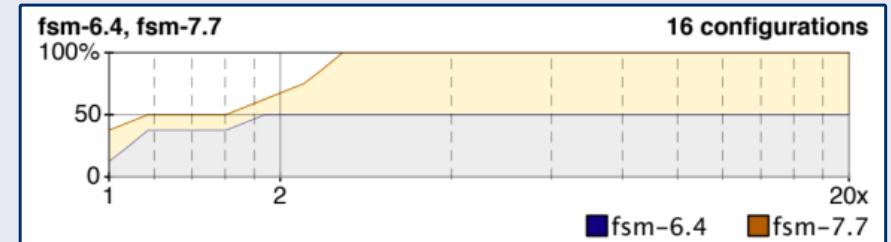
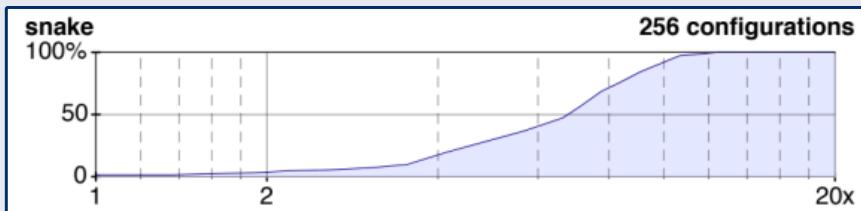


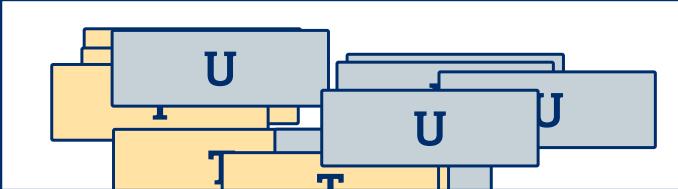
Key: **think like a user**
too slow = useless



x-axis = "too slow" cutoff vs. untyped code (log scale)
y-axis = % useful combos

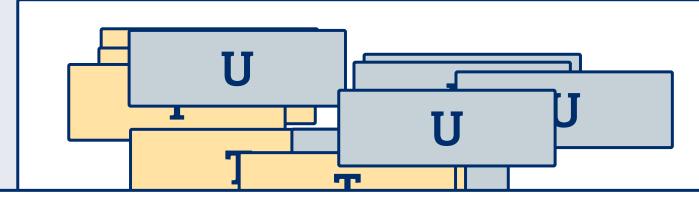
think like a user ==> Now Scalable!





Typed Racket

What do **sound types** cost?

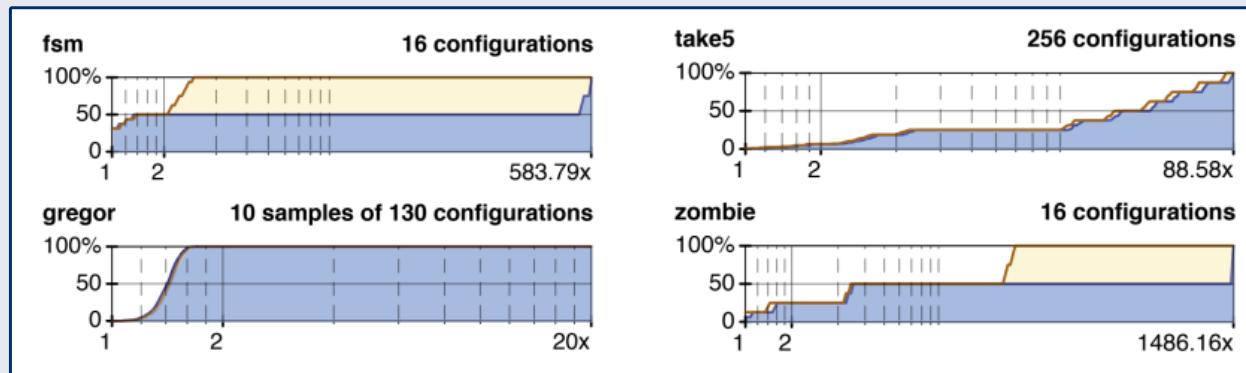


Typed Racket

What do **sound types** cost?

Too much!

OOPSLA'18: A modest optimization, still slow!



Safe and Efficient Gradual Typing

Transient Typechecks are (Almost) Free

Sound Gradual Typing is Nominally Alive and Well

Different behaviors!

Different behaviors!

```
def join(d0:Array[Int]):  
    ....
```

```
join([0,1,2,...])
```

Different behaviors!

```
def join(d0:Array[Int]):  
    ....
```

```
join([0,1,2,...])
```

- ✓ every element looks good
- ✓ it's an array
- ✓ I don't care
- ✗ it's untyped data

Different behaviors!

```
def join(d0:Array[Int]):  
    ....
```

```
join([0, "XXX", ...])
```

-  bad element
-  it's an array
-  I don't care
-  it's untyped data





Proofs + People



Proofs + People

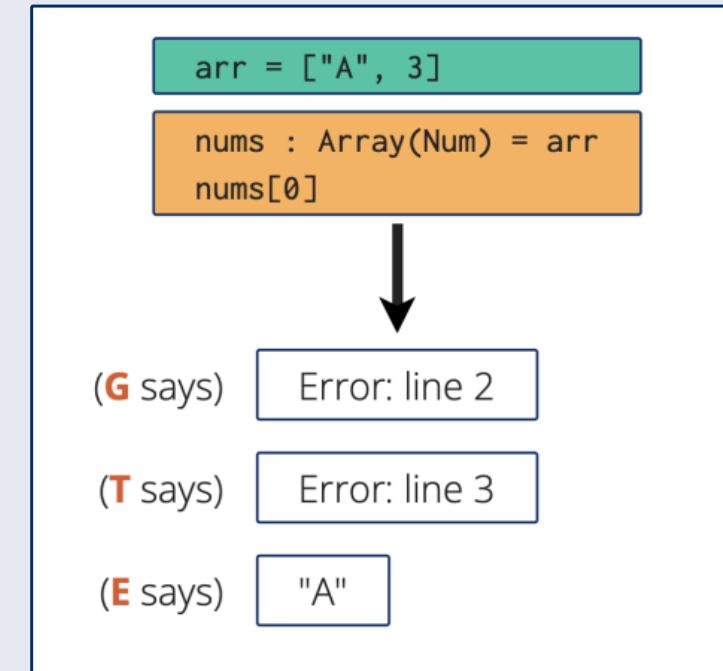


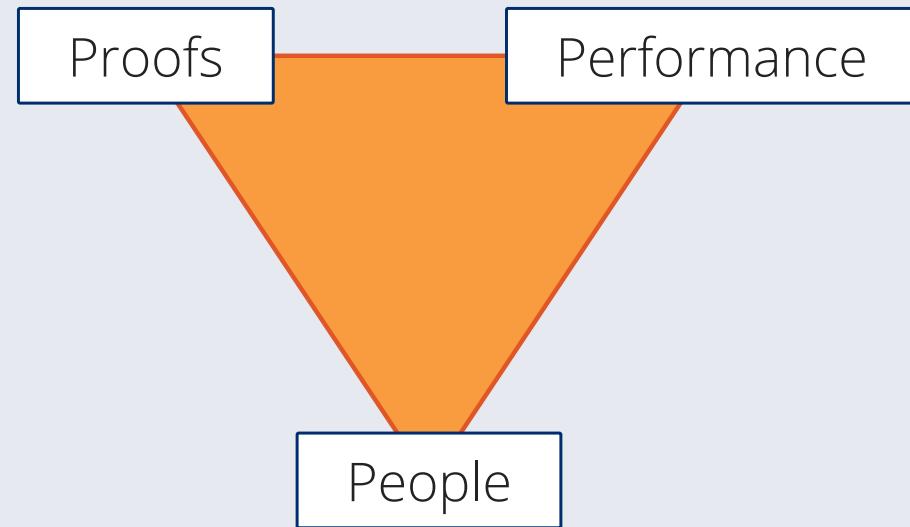
	Guarded	C	F	Transient	A	E
type soundness	✓	✓	✓	y	✓	✗
complete monitoring	✓	✓	✗	✗	✗	✗
blame soundness	✓	✓	✓	h	✓	0
blame completeness	✓	✓	✓	✗	✓	✗

Proofs + People



	Guarded	C	F	Transient	A	E
type soundness	✓	✓	✓	y	✓	✗
complete monitoring	✓	✓	✗	✗	✗	✗
blame soundness	✓	✓	✓	h	✓	o
blame completeness	✓	✓	✓	✗	✓	✗





Gradual Soundness: Lessons from Static Python

[Programming'23]



Sound types in Instagram

Fast in general?

Work in progress:
Measuring 3-tier types



Mrigank Pawagi



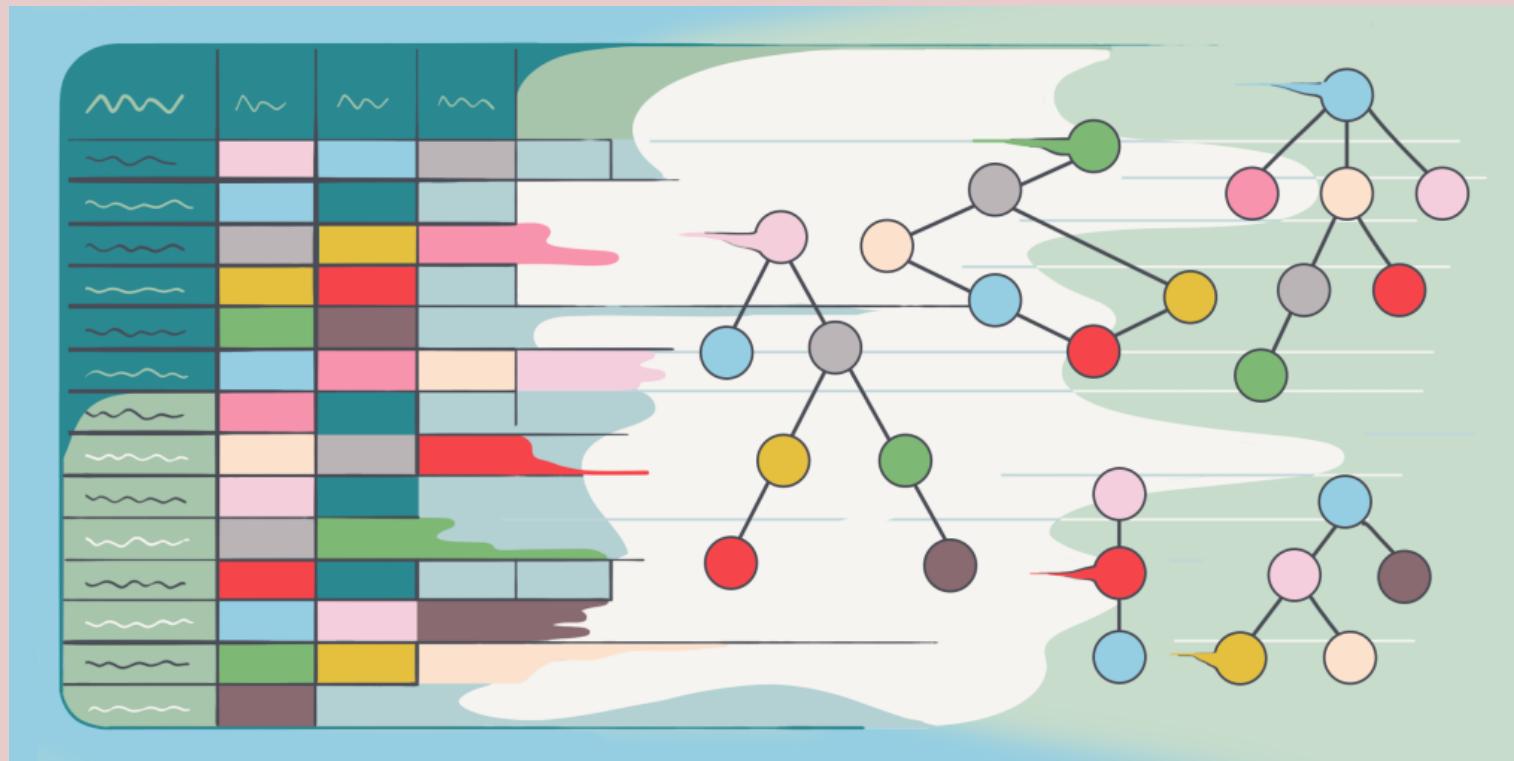
Vivaan Rajesh

lst

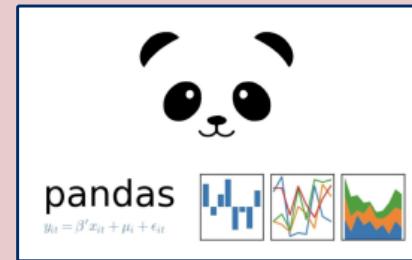
lst: List

lst: List[int]

10 types ==> 49k combos!!



Programming for tables



What about types?

	A	B	C	D
1	YEAR	MONTH	DAY	ANIMAL
2	2004		1	5 Deer
3	2004		1	12 Deer
4	2004		1	21 Deer
5	2004		1	22 Deer
6	2004		1	26 Deer
7	2004		1	27 Turkey
8	2004		1	28 Deer
9	2004		1	29 Coyote
10	2004		1	29 Coyote

Goal: describe table "shapes"
catch wrong programs



tbl.anml



tbl.year + tbl.day

Decades of prior work ...

Remy POPL 1989

Wand I&C 1991

Gaster 1998

Harper++ POPL 1991

Buneman++ TDS 1996

Ohori++ ICFP 2011

Slepak++ ESOP 2014

Vazou++ ICFP 2015

Petricek ECOOP 2017

Kazerounian++ PLDI 2019

Morris++ POPL 2019

...





B2T2

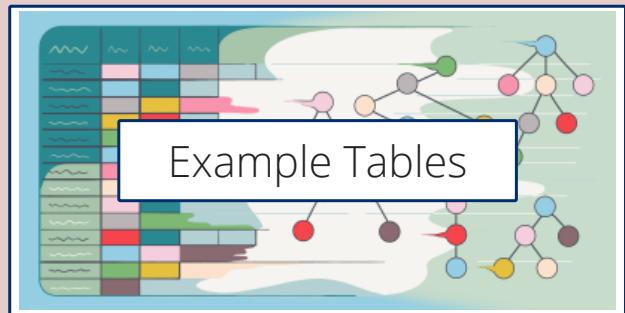
add-column
API
subtable ...



BROWN Benchmark for Table Types

Example Programs

Example Errors



Example Tables

B2T2



BROWN Benchmark for Table Types

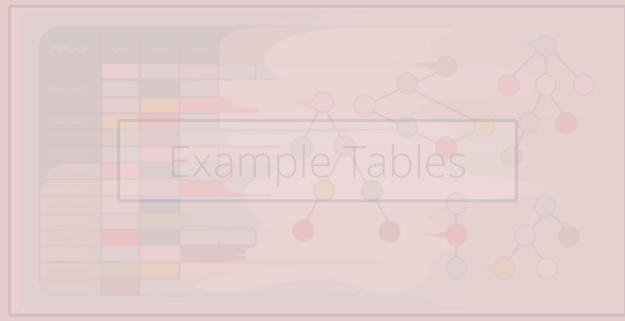
add-column

API

subtable ...

Example Programs

Example Errors



B2T2



BROWN Benchmark for Table Types

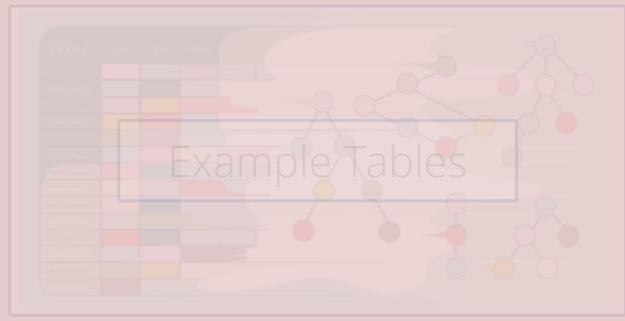
add-column

API

subtable ...

Example Programs

Example Errors



B2T2

add-column
API
subtable ...



BROWN Benchmark for Table Types

Example Programs

Example Errors



B2T2



BROWN Benchmark for Table Types

add-column

API

subtable ...

Example Programs

Example Errors

Example Program:

Average columns that **start with "quiz"**

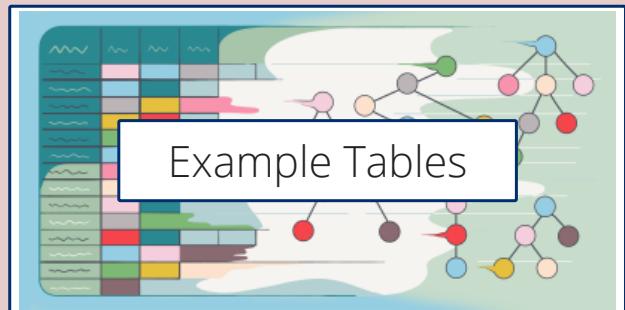
```
buildColumn(gradebook, "avg",
  function(row):
    let quizColnames =
      filter(header(row),
        function(c):
          startsWith(c, "quiz"))
    let scores = map(quizColnames,
      function(c):
        getValue(row, c))
    sum(scores) / length(scores))
```

Example Error:

Task: find participants who ate black and white jellybeans

```
filter(jellybeanTable,  
       function(r):  
         getValue(r, "black and white") == true)
```

Error: no column "black and white"



Example Tables

B2T2

add-column

API

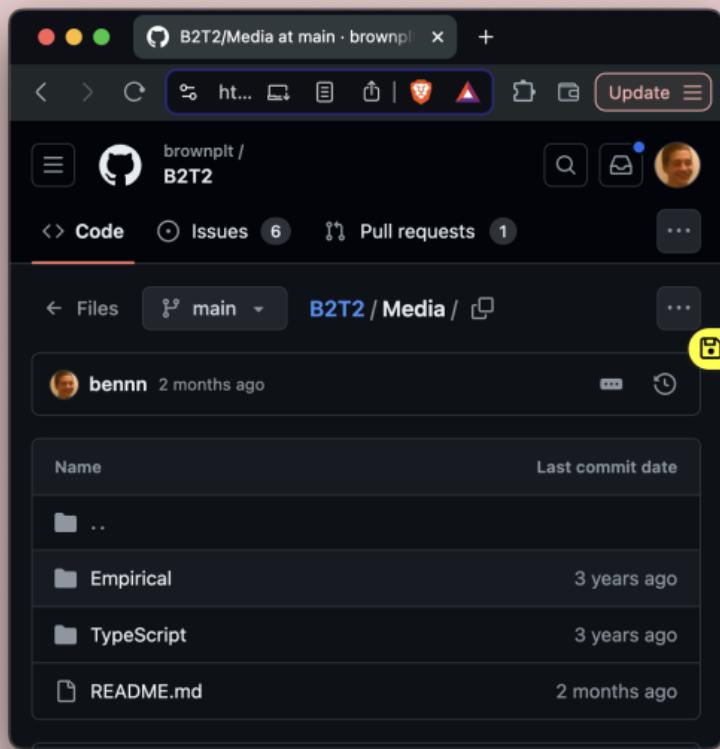
subtable ...



BROWN Benchmark for Table Types

Example Programs

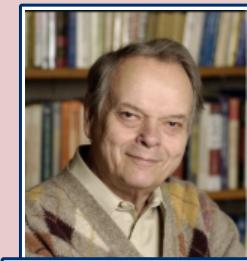
Example Errors



Work in progress:
Type-Narrowing Benchmark



"some account should be taken of
the premises in conditional expressions"



John Reynolds

```
if e1:  
    e2 # with refined types
```

```
def first(c : tuple[object, object]):  
    if type(c[0]) is int:  
        return c[0] + 1
```

```
def add_one(a : object):  
    tmp = type(a) is int  
    if tmp:  
        return a + 1
```

```
def parent_score(n : Node):  
    if n.parent is not None:  
        total += n.parent.wins + n.parent.losses
```

```
def first(c : tuple[object, object]):  
    if type(c[0]) is int:  
        return c[0] + 1
```

```
def add_one(a : object):  
    tmp = type(a) is int  
    if tmp:  
        return a + 1
```

```
def parent_score(n : Node):  
    if n.parent is not None:  
        total += n.parent.wins + n.parent.losses
```

```
def first(c : tuple[object, object]):  
    if type(c[0]) is int:  
        return c[0] + 1
```

```
def add_one(a : object):  
    tmp = type(a) is int  
    if tmp:  
        return a + 1
```

```
def parent_score(n : Node):  
    if n.parent is not None:  
        total += n.parent.wins + n.parent.losses
```

```
def first(c : tuple[object, object]):  
    if type(c[0]) is int:  
        return c[0] + 1
```

```
def add_one(a : object):  
    tmp = type(a) is int  
    if tmp:  
        return a + 1
```

```
def parent_score(n : Node):  
    if n.parent is not None:  
        total += n.parent.wins + n.parent.losses
```



```
def first(c : tuple[object, object]):  
    if type(c[0]) is int:  
        return c[0] + 1
```



```
def add_one(a : object):  
    tmp = type(a) is int  
    if tmp:  
        return a + 1
```



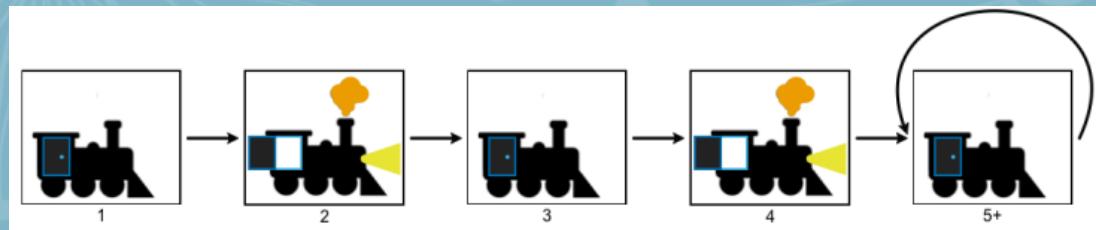
```
def parent_score(n : Node):  
    if n.parent is not None:  
        total += n.parent.wins + n.parent.losses
```

Benchmark	Description
positive	refine when condition is true
negative	refine when condition is false
alias	track test results assigned to variables
connectives	handle logic connectives
custom_predicates	allow programmers define their own predicates
predicate_2way	custom predicates refines both positively and negatively
predicate_strict	perform strict type checks on custom predicates
predicate_multi_args	predicates can have more than one arguments
object_properties	refine types of properties of objects
tuple_whole	refine types of the whole tuple
tuple_elements	refine types of tuple elements
subtyping	refine supertypes to subtypes
subtyping_structural	refine structural subtyping



Linear Temporal Logic (LTL) Misconceptions

Linear Temporal Logic (LTL) Misconceptions



LTL : for changes over time

Prop. Logic
and or ==> not

+

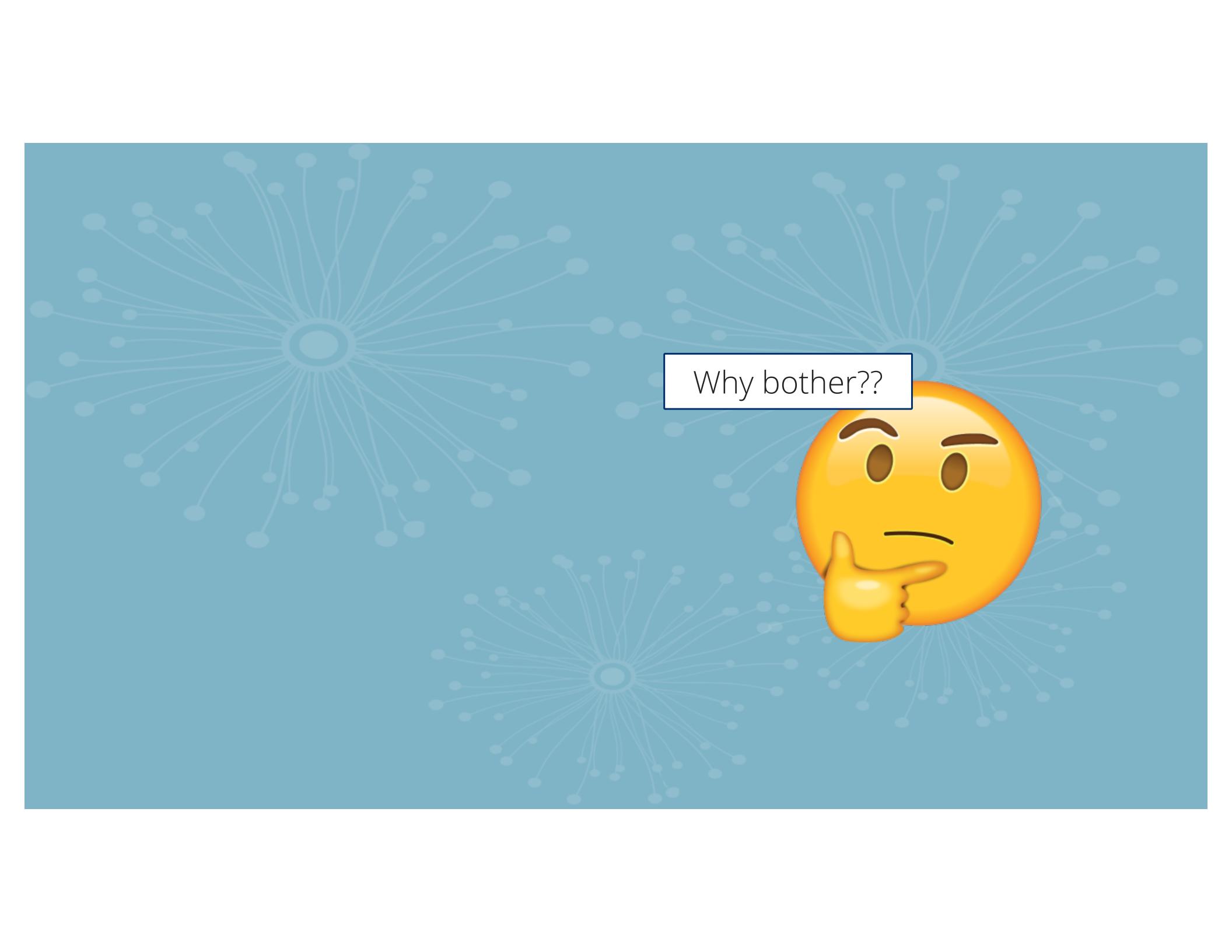
Temporal Operators
always eventually next until

- 
- ✓ Expressive
 - ✓ Small
 - ✓ Good decision procedures

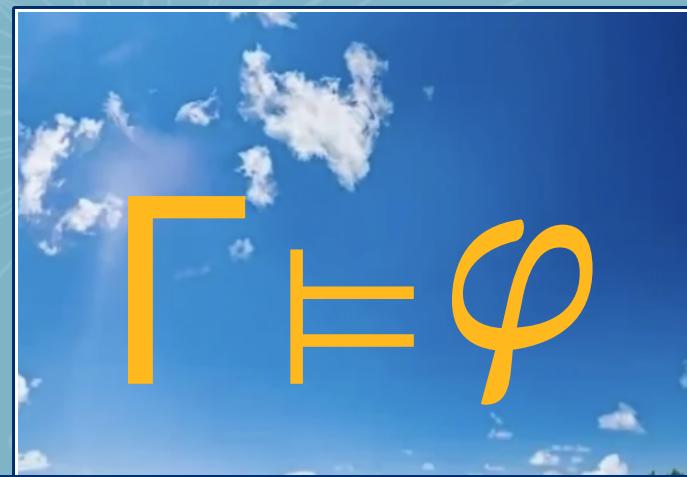
.... and easy to learn?

In what ways is LTL difficult to use?





Why bother??



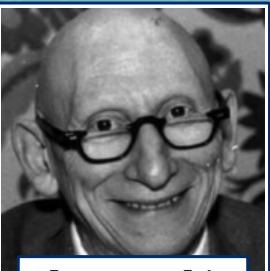
Logics have precise semantics!



Why bother??



"One cannot proceed from the **informal**
to the formal by purely formal means"



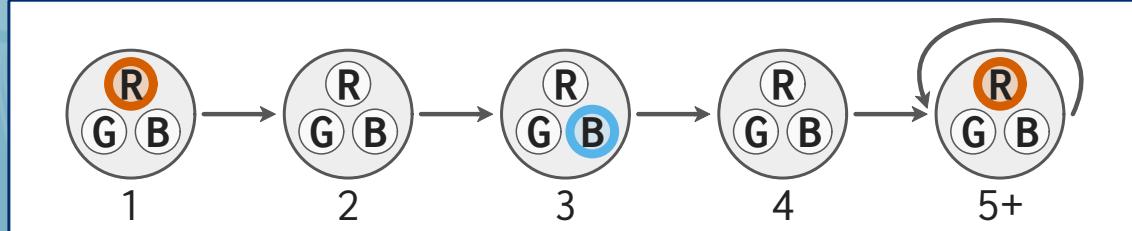
Alan Perlis

Misconceptions get in the way!

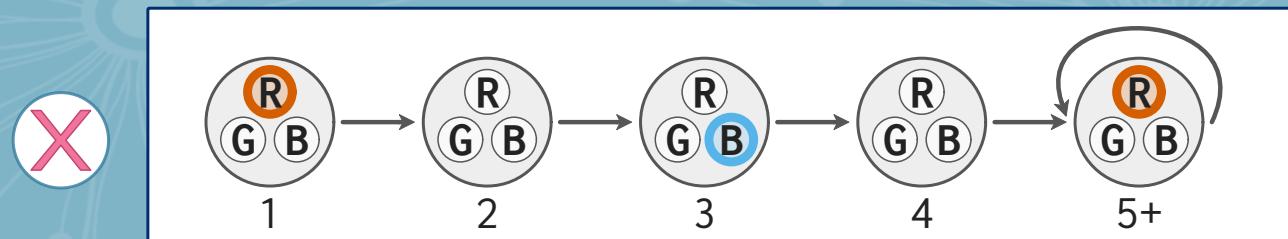


Q. Do the traces below satisfy this formula?
{eventually Red} and {eventually Green}

Q. Do the traces below satisfy this formula?
{eventually Red} and {eventually Green}

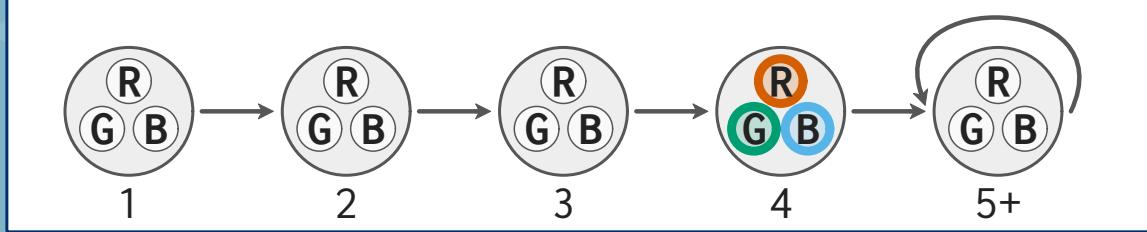
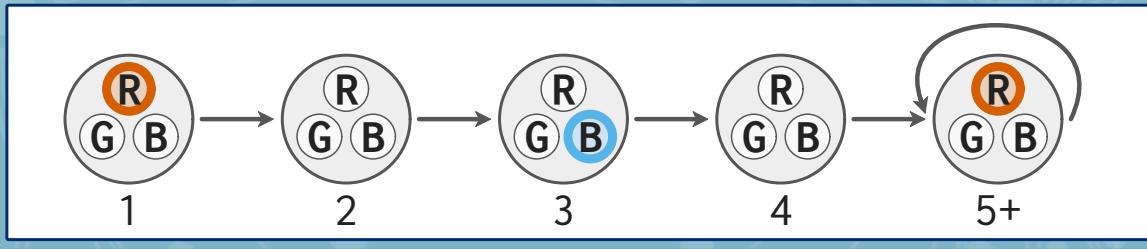


Q. Do the traces below satisfy this formula?
{eventually Red} and {eventually Green}

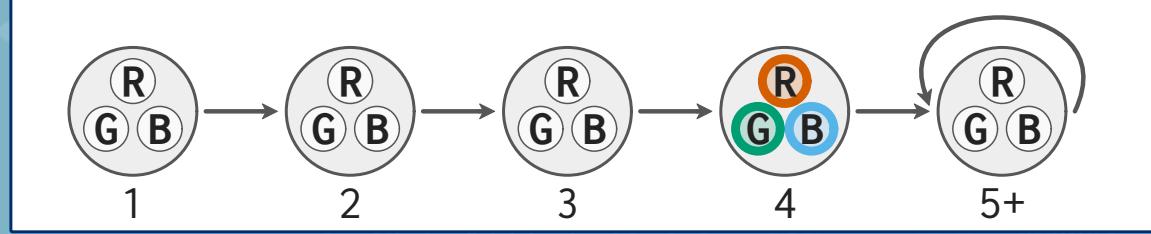
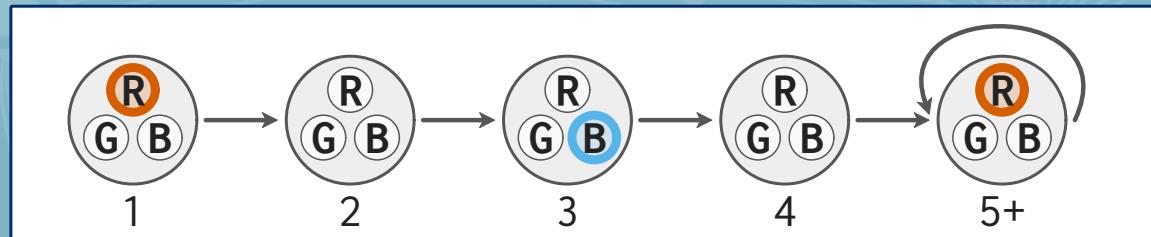


Q. Do the traces below satisfy this formula?
{eventually Red} and {eventually Green}

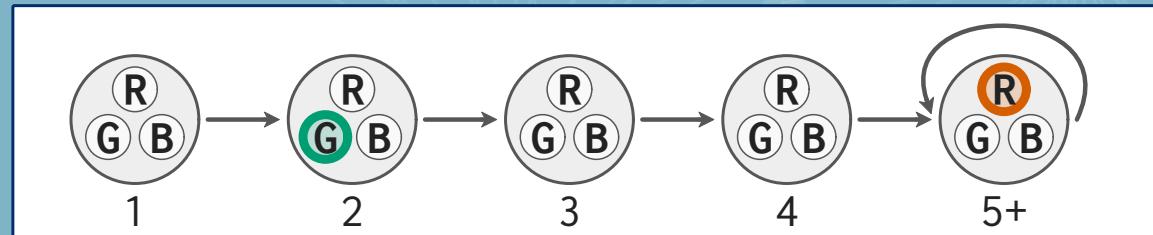
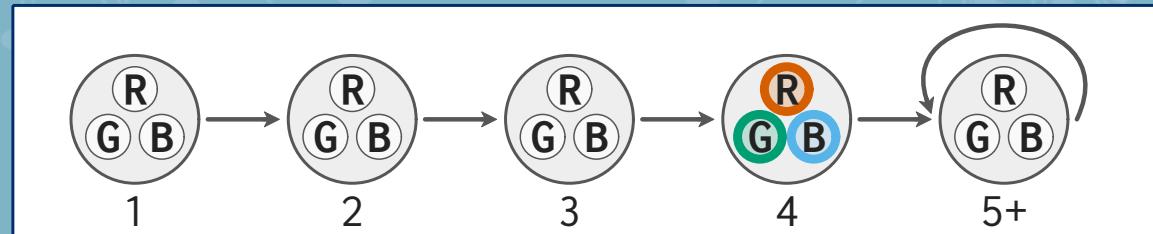
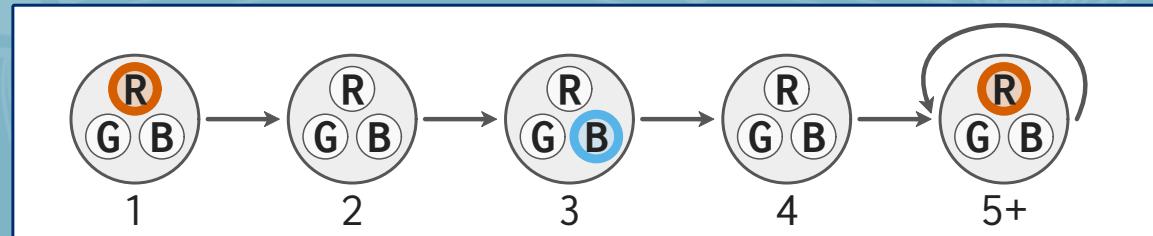
X



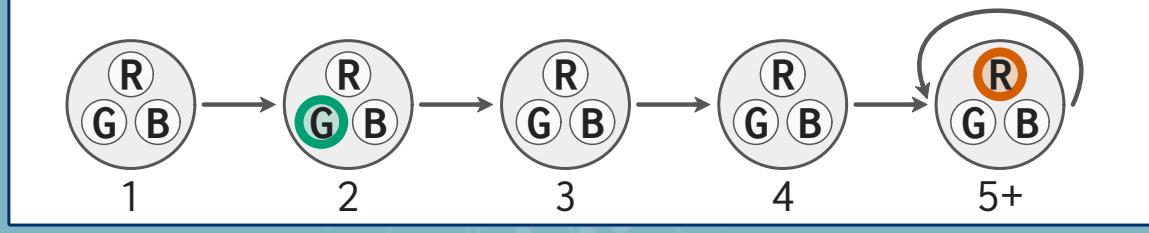
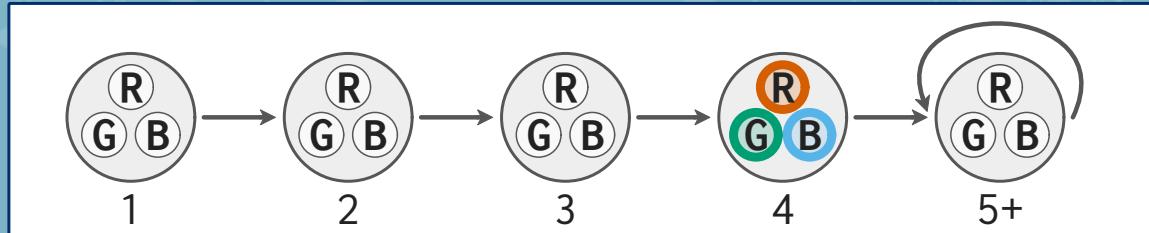
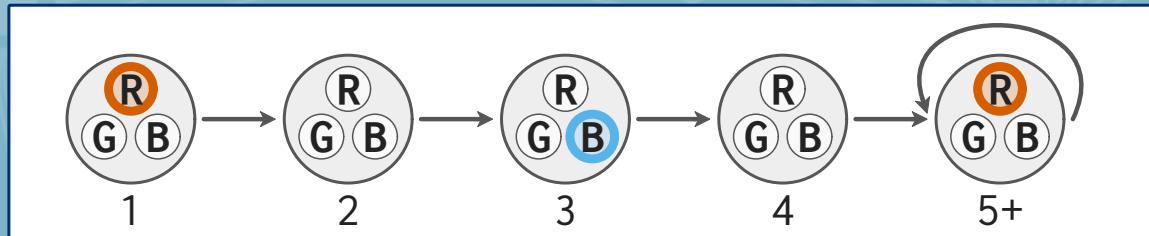
Q. Do the traces below satisfy this formula?
{eventually Red} and {eventually Green}



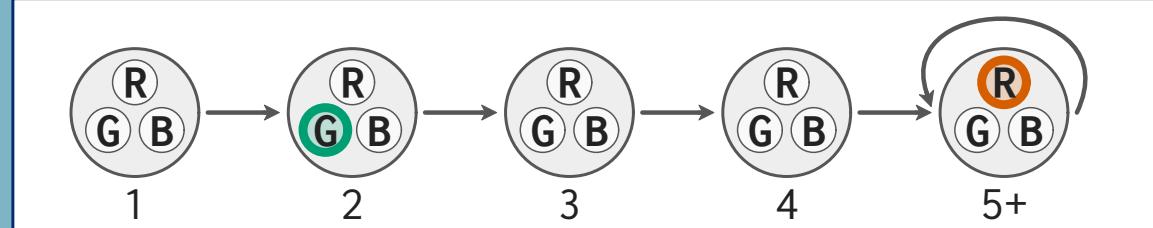
Q. Do the traces below satisfy this formula?
{eventually Red} and {eventually Green}



Q. Do the traces below satisfy this formula?
{eventually Red} and {eventually Green}

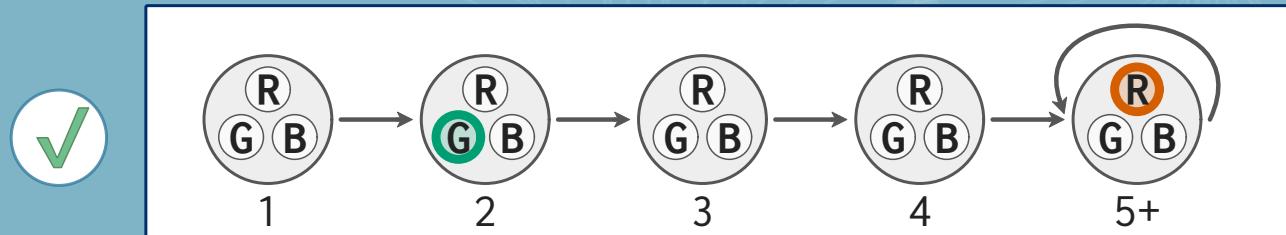


Q. Do the traces below satisfy this formula?
{eventually Red} and {eventually Green}



Q. Do the traces below satisfy this formula?
{eventually Red} and {eventually Green}

Not satisfied, because Green comes before Red
Bad Prop misconception





LTLf = finite-trace LTL

LTLf = finite-trace LTL

LTL LTLf

$\neg X(e) == X(\neg e)$ ✓ ✗

$F(G(e)) == G(F(e))$ ✗ ✓

Catalog

14 categories of LTL and LTLf Errors

Catalog

14 categories of LTL and LTLf Errors

Length

Last

Cycle G

Implicit Prefix

Trace Split U

Spreading X

Bad Prop

Bad State Index

Implicit F

Implicit G

Other Implicit

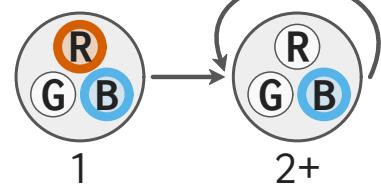
Exclusive U

Bad State Quantification

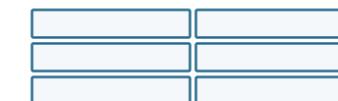
Weak U

Reusable Pipeline

Instruments

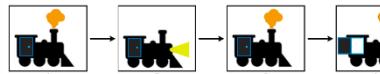


Catalog



3 Survey Instruments

Example Question: Is the formula
always (Engine or Light)
satisfied by this trace?



Example Answer: Yes, because either the engine headlight is on in each state.

Does the example make sense to you?*

Yes

No (please explain)

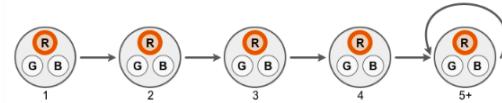
Example Question:

$G(X \text{ (Red)})$

Example Answer:

- *LTL* description: The Red light is on in every state.
- *LTLf* description: Every state must be followed Red on. No finite traces satisfy the formula.

satisfied by this trace?*



Yes

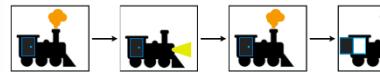
No

3 Survey Instruments

Q. Formats:

- LTL --> English
- English --> LTL
- Trace Matching
- Explain Mismatches
- Check Equations

Example Question: Is the formula always (Engine or Light) satisfied by this trace?



Example Answer: Yes, because either the engine headlight is on in each state.

Does the example make sense to you?*

Yes

No (please explain)

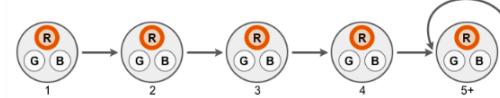
Example Question:

G (X (Red))

Example Answer:

- *LTL* description: The Red light is on in every state.
- *LTLf* description: Every state must be followed Red on. No finite traces satisfy the formula.

satisfied by this trace?*



Yes

No

<https://ltl-tutor.xyz>

LTL Tutor [Version 1.1.1] Logged in as anon-user-BwlkcG

Tutor Dashboard LTL Syntax Generate Exercise Instructor Dashboard Profile Log Out

Exercise

Does this trace satisfy the following LTL formula? Question 1 of 7

$(\neg (\text{F} p))$

$\text{!p \& a \& !d} \leftrightarrow \text{!p \& a \& !d}$

Yes

No

Check Answer Next Question



Siddhartha Prasad

<https://ltl-tutor.xyz>

The image shows a screenshot of the LTL Tutor web application. At the top, there is a URL bar with the address <https://ltl-tutor.xyz>. Below the URL bar, there are two browser windows side-by-side.

Left Window (Exercise View):

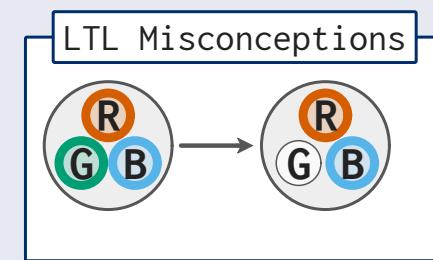
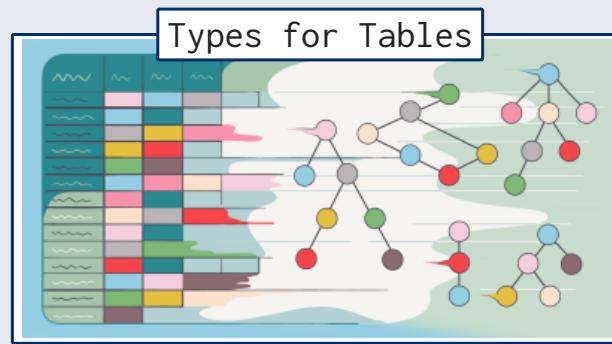
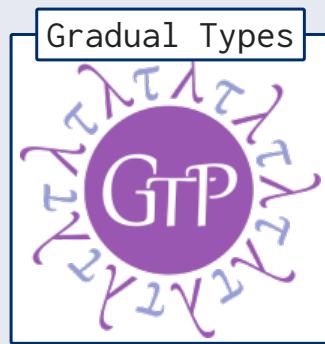
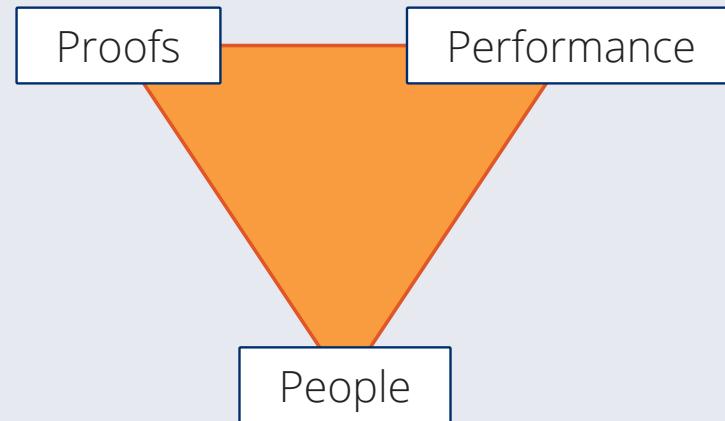
- Header: "Tutor Dashboard" and "LTL Syntax".
- Section: "Exercise".
- Text: "Does this trace satisfy the formula $(\neg (\mathbf{F} p))$?".
- Diagram: A state transition diagram with states $\neg p$, p , $\neg a$, and a . Transitions are labeled $\neg p \& a \& \neg d$ and $\neg p \& a \& d$.
- Buttons: "Check Answer" and "Next Question".

Right Window (Tutor Dashboard View):

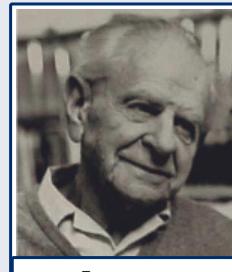
- Header: "Tutor Dashboard" and "LTL Syntax".
- Buttons: "Check Answer" and "Next Question".
- Text: "That's not correct 😞 Don't worry, keep trying! The correct answer is highlighted in green (i.e: $(X (p \rightarrow (X a)))$)".
- Text: "Your selection is more permissive than the correct answer. Here is a trace that satisfies your selection, but not the correct answer:"
- Diagram: A state transition diagram with states $\neg p$, p , $\neg a$, and a . Transitions are labeled $\neg p \rightarrow p$, $p \rightarrow \neg a$, $\neg a \rightarrow a$, and $a \rightarrow \neg p$.
- Text: "Alt Trace: $\neg p; p; \neg a; \text{cycle}\{1;1\}$ ".
- Diagram: A Venn diagram with two overlapping circles. The left circle is green and labeled "Correct answer". The right circle is red and labeled "Your answer".



Ratha Prasad



Some theories are more **testable** than others;
they take, as it were, greater risks."



Karl Popper

