Dependently Typed Tables

Joseph Rotella
Advisor: Rob Lewis, Reader: Shriram Krishnamurthi

Overview

Tabular data is ubiquitous:

Most tabular programming frameworks are dynamically typed.

Sources: pandas, Wikimedia Commons, Google, Pyret

Overview

Name	Age	Quiz 1	Quiz 2
Bob	12	6	9
Alice	17		8
Eve	13	7	

Overview

Name	Age	Quiz 1	Quiz 2
Bob	12	6	9
Alice	17		8
Eve	13	7	

Challenges:

- Heterogeneity
- Type-level computation

Overview

Goal: develop and verify a dependently typed tabular programming library in Lean 4 supporting realistic operations.

B2T2

Brown Benchmark for Table Types (Lu et al., 2022):

- Table definition
- Table API
- Example tables and programs
- “Buggy” programs

Typing Tables

Typing Tables

Name String	Age Nat	Quiz 1 Nat	Quiz 2 Nat
Bob	12	6	9
Alice	17		8
Eve	13	7	

Typing Tables

Name String	Age Nat	Quiz 1 Nat	Quiz 2 Nat
Bob	12	6	9
Alice	17		8
Eve	13	7	

Type

Data

Typing Tables

Name String	Age Nat	Quiz 1 Nat	Quiz 2 Nat
Bob	12	6	9
Alice	17		8
Eve	13	7	

Type

Data

- Schema: list of headers (name–type pairs)

Typing Tables

Name String	Age Nat	Quiz 1 Nat	Quiz 2 Nat
Bob	12	6	9
Alice	17		8
Eve	13	7	

Type

Data

- Schema: list of headers (name–type pairs)
- Table(schema): list of rows

Typing Tables

Name String	Age Nat	Quiz 1 Nat	Quiz 2 Nat
Bob	12	6	9
Alice	17		8
Eve	13	7	

Type

Data

- Schema: list of headers (name–type pairs)
- Table(schema): list of rows
- Row(schema): heterogeneous list of cells

Typing Tables

Name String	Age Nat	Quiz 1 Nat	Quiz 2 Nat
Bob	12	6	9
Alice	17		8
Eve	13	7	

Type

Data

- Schema: list of headers (name–type pairs)
- Table(schema): list of rows
- Row(schema): heterogeneous list of cells
- Cell(header): option carrying column information

Tables in Lean

Name	Age
Bob	12
Ana	17
Eve	

```
def studentsEx :
    Table [("Name", String), ("Age", Nat)] :=
Table.mk [
  Row.cons (Cell.val "Bob") (Row.cons (Cell.val 12)
    Row.nil),
  Row.cons (Cell.val "Ana") (Row.cons (Cell.val 17)
    Row.nil),
  Row.cons (Cell.val "Eve") (Row.cons Cell.emp
    Row.nil)
]
```

Tables in Lean

Name	Age	Quiz 1	Quiz 2
Bob	12	6	9
Alice	17		8
Eve	13	7	

```
def students : Table [("Name", String), ("Age",
    Nat), ("Quiz 1", Nat), ("Quiz 2", Nat)] :=
  Table.mk [
    /["Bob"   , 12, 6   , 9   ],
    /["Alice", 17, EMP, 8   ],
    /["Eve"   , 13, 7   , EMP]
  ]
```

Table Operations

Basic Table Operations

Name	Age	Quiz 1	Quiz 2
Bob	12	6	9
Alice	17		8
Eve	13	7	

```
Table [("Name", String), ("Age", Nat), ("Quiz 1", Nat), ("Quiz
2", Nat)]
```

```
dropColumn students "Age"
```

Basic Table Operations

Name	Age	Quiz 1	Quiz 2
Bob	12	6	9
Alice	17		8
Eve	13	7	

```
Table [("Name", String), ("Age", Nat), ("Quiz 1", Nat), ("Quiz 2", Nat)]
```

```
dropColumn students "Age"
```

Name	Quiz 1	Quiz 2
Bob	6	9
Alice		8
Eve	7	

```
Table [("Name", String), ("Quiz 1", Nat), ("Quiz 2", Nat)]
```

Basic Table Operations

Name	Age	Quiz 1	Quiz 2
Bob	12	6	9
Alice	17		8
Eve	13	7	

```
dropColumn students "Quiz"
```

Basic Table Operations

Name	Age	Quiz 1	Quiz 2
Bob	12	6	9
Alice	17		8
Eve	13	7	

```
dropColumn students "Quiz"
```

Could not prove that name "Quiz" is in schema [("Name", String), ("Age", Nat), ("Quiz 1", Nat), ("Quiz 2", Nat)]

Basic Table Operations

Name	Age	Quiz 1	Quiz 2

...

To act on a column, we must prove that it exists.

```
dropColumn students "Age"
```

Basic Table Operations

Name	Age	Quiz 1	Quiz 2
...			

To act on a column, we must prove that it exists.

```
dropColumn students "Age"
```

is actually

```
dropColumn students "Age" (Schema.HasName.tl
    Schema.HasName.hd)
```

Basic Table Operations

Name	Age	Quiz 1	Quiz 2

...

To act on a column, we must prove that it exists.

```
dropColumn students "Age"
```

is actually

```
dropColumn students "Age" (Schema.HasName.tl
    Schema.HasName.hd)
```

Proofs are indices.

Iterated Table Operations

What if we want to drop multiple columns?

Iterated Table Operations

What if we want to drop multiple columns?

```
def dropColumnsNaive
    (t : Table schema)
    (nms : List (CertifiedName schema)) :=
  nms.foldl dropColumn t
```

Iterated Table Operations

What if we want to drop multiple columns?

```
def dropColumnsNaive
    (t : Table schema)
    (nms : List (CertifiedName schema)) :=
  nms.foldl dropColumn t
```

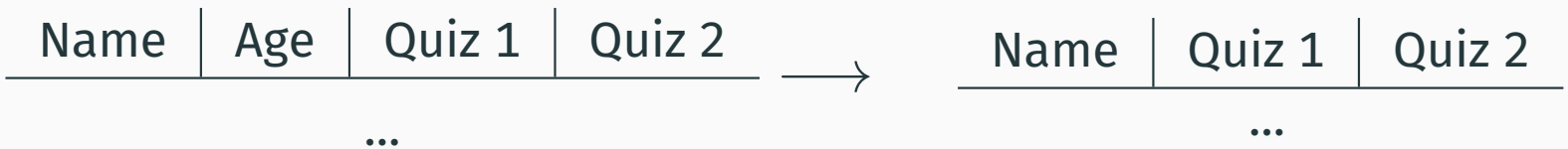
```
dropColumnsNaive students [⟨"Age", pf⟩, ⟨"Age", pf⟩]
```

Iterated Table Operations

What if we want to drop multiple columns?

```
def dropColumnsNaive
    (t : Table schema)
    (nms : List (CertifiedName schema)) :=
  nms.foldl dropColumn t
```

```
dropColumnsNaive students [⟨"Age", pf⟩, ⟨"Age", pf⟩]
```

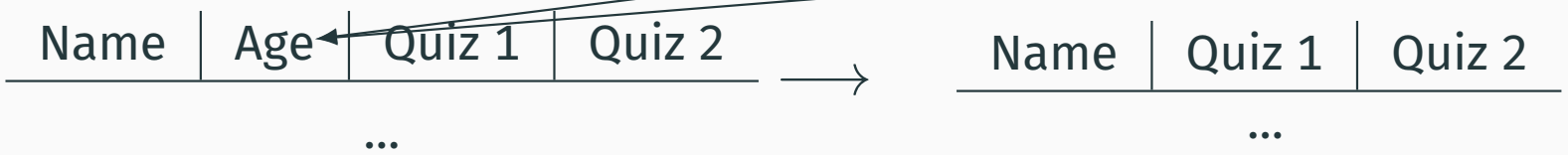


Iterated Table Operations

What if we want to drop multiple columns?

```
def dropColumnsNaive
    (t : Table schema)
    (nms : List (CertifiedName schema)) :=
  nms.foldl dropColumn t
```

```
dropColumnsNaive students [⟨"Age", pf⟩, ⟨"Age", pf⟩]
```

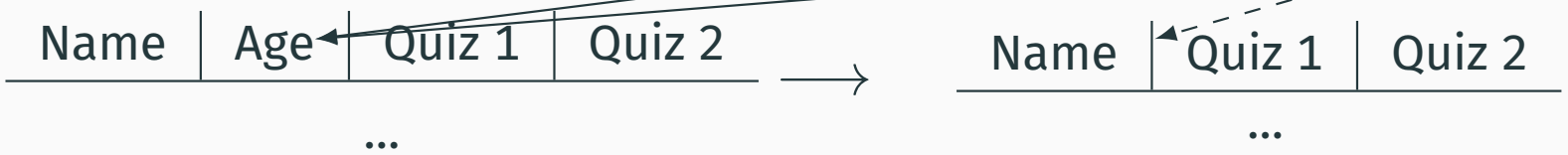


Iterated Table Operations

What if we want to drop multiple columns?

```
def dropColumnsNaive
    (t : Table schema)
    (nms : List (CertifiedName schema)) :=
  nms.foldl dropColumn t
```

```
dropColumnsNaive students [⟨"Age", pf⟩, ⟨"Age", pf⟩]
```



Iterated Table Operations

```
dropColumns T₀ ⟨a₀, a₁, a₂⟩
```

Iterated Table Operations

```
dropColumns T₀ ⟨a₀, a₁, a₂⟩
```

How can we represent this sequence?

Iterated Table Operations

```
dropColumns T₀ ⟨a₀, a₁, a₂⟩
```

How can we represent this sequence?

- Lists: no heterogeneity

Iterated Table Operations

```
dropColumns T₀ ⟨a₀, a₁, a₂⟩
```

How can we represent this sequence?

- Lists: no heterogeneity
- HLists: no inter-element type dependence

Iterated Table Operations

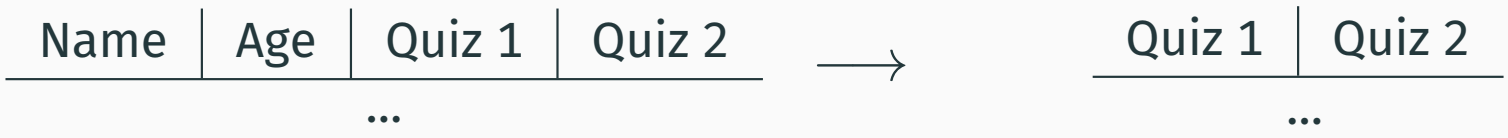
```
dropColumns T₀ ⟨a₀, a₁, a₂⟩
```

How can we represent this sequence?

- Lists: no heterogeneity
- HLists: no inter-element type dependence

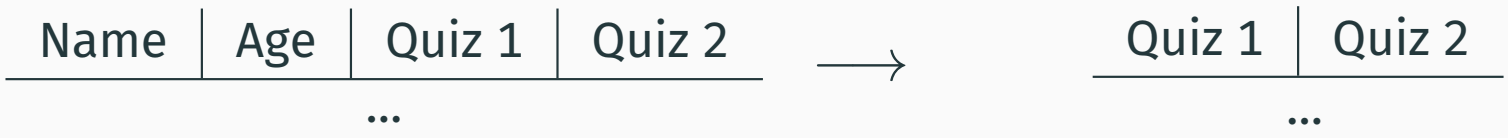
Action lists: dependent heterogeneous lists

Iterated Table Operations




```
dropColumns students (ActionList.cons ⟨"Name", .hd⟩
    (ActionList.cons ⟨"Age", .hd⟩ ActionList.nil))
```

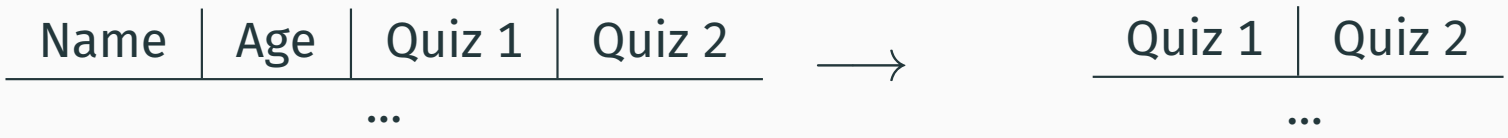
Iterated Table Operations




```
dropColumns students (ActionList.cons ⟨"Name", .hd⟩
    (ActionList.cons ⟨"Age", .hd⟩ ActionList.nil))
```

```
dropColumns students A["Name", "Age"]
```

Iterated Table Operations




```
dropColumns students (ActionList.cons ⟨"Name", .hd⟩
    (ActionList.cons ⟨"Age", .hd⟩ ActionList.nil))
```

```
dropColumns students A["Name", "Age"]
```

Consistent, adaptive notation:

```
sortByColumns students A["Name", "Age"]
```

Miscellany

Pivots:

Name	Age	Quiz 1	Quiz 2
Alice	17	6	8
Bob	12	8	9

`pivotLonger` ⇄ `pivotWider`

Name	Age	Test	Score
Alice	17	Quiz 1	6
Alice	17	Quiz 2	8
Bob	12	Quiz 1	8
Bob	12	Quiz 2	9

Miscellany

Pivots:

Name	Age	Quiz 1	Quiz 2
Alice	17	6	8
Bob	12	8	9

`pivotLonger` ⇄ `pivotWider`

Name	Age	Test	Score
Alice	17	Quiz 1	6
Alice	17	Quiz 2	8
Bob	12	Quiz 1	8
Bob	12	Quiz 2	9

Groupings:

Dept	Code	Subject
CS	150	Programming
Math	1630	Analysis
CS	1570	Algorithms
Math	1530	Algebra

`groupBySubtractive` →

Key	Groups	
	Code	Subject
CS	150	Programming
	1570	Algorithms
	Code	Subject
Math	1630	Analysis
	1530	Algebra

Miscellany

Pivots:

Name	Age	Quiz 1	Quiz 2
Alice	17	6	8
Bob	12	8	9

pivotLonger ⇄ pivotWider

Name	Age	Test	Score
Alice	17	Quiz 1	6
Alice	17	Quiz 2	8
Bob	12	Quiz 1	8
Bob	12	Quiz 2	9

Groupings:

Dept	Code	Subject
CS	150	Programming
Math	1630	Analysis
CS	1570	Algorithms
Math	1530	Algebra

groupBySubtractive →

Key	Groups	
	Code	Subject
CS	150	Programming
	1570	Algorithms
	Code	Subject
Math	1630	Analysis
	1530	Algebra

Verification:

```
theorem groupBySubtractive_spec4 :
  ∀ (t : Table sch) (c : η) (hc : sch.HasCol (c, τ)),
  List.NoDuplicates (getColumn2 (groupBySubtractive t c)
    "key")
```