Programming with Dependently Typed Tables in Lean

Joseph Rotella, Robert Y. Lewis

Lean Together 2025
January 15, 2025

Motivation

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

Tabular data are ubiquitous in data science, databases.

Generally in purpose-built or dynamically-typed languages.

Goal: implement type-safe tabular programming in Lean.

Brown Benchmark for Table Types

The Brown Benchmark for Table Types (B2T2) (Lu et al., 2022):

- Table definition
- Table API
- Example (valid and invalid) programs

Original implementation in TypeScript; recently also in Empirical and Idris 2 (Wright et al., 2022).

This project:

- Implemented the B2T2 API and examples in Lean
- Formally verified this implementation w.r.t. the B2T2 specification

Encoding Tables in Lean

Anatomy of a Table

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

Type (schema)

Data (rows of cells)

- Arbitrary number of independently typed columns
- Columns are both typed and named
- Cell emptiness is primitive (B2T2)

Encoding 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)
]
```

Encoding Tables in Lean

Name	Age
Bob	12
Ana	17
Eve	

```
def students :
    Table [("Name", String), ("Age", Nat)] :=
  Table.mk [
    /["Bob", 12],
    /["Ana", 17],
    /["Eve", EMP]
  ]

def students' :=
  Table.mk [
    /["Name" := "Bob", "Age" := 12],
    /["Ana", 17],
    /["Eve", EMP]
  ]
```

Computing with Tables

Dropping a Column

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)]
```

Table functions compute on types *and* data.

Dropping a Column

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)]

Column-Existence Proofs

To act on a column, prove that it exists.

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

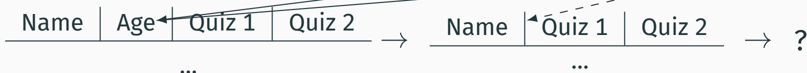
- Compile-time: Catch column-reference errors
- Runtime: Proofs are (data-valued) indices

Iterated Table Operations

What about dropping multiple columns?

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

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



Iterated Table Operations

`dropColumns` T_0 ⟨a_0, a_1, a_2⟩

What is the type of this sequence?

- `List (f sch)`: no heterogeneity
- `HList` [f_0 `sch`, …, f_n `sch`]: no inter-element type dependence

Action lists: dependent heterogeneous lists

Action List Notation

With a single argument, auto-params suffice:

```
dropColumn students "Age" (by name)
```

But action lists bundle data with proofs:

```
dropColumns students (.cons ⟨"Age", by name⟩ (.cons
    ⟨"Quiz 1", by name⟩ .nil))
```

Use custom notation to detect and synthesize needed proofs:

```
dropColumns students A["Age", "Quiz 1"]
```

```
renameColumns students A[("Name", "Nom"),
                         ("Age", "Edat")]
```

Pivots

Pivots: type–data interchange

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

```
pivotWider t "Test" "Score" :
  sch.removeHeaders A["Test", "Score"] ++
  (t.getColumn2 "Test").somes.unique.map (·, Nat)
```

Verification

B2T2 Specifications

- Schematic specifications

```
def pivotLonger_spec3 {τ : Type u_η}
  (t : Table sch)
  (cs : ActionList (Schema.removeTypedName τ)
    sch)
  (c1 : η) (c2 : η) :
  (pivotLonger t cs c1 c2).schema.HasCol (c1, η)
```

- Behavioral specifications

```
theorem count_spec4 {τ} [DecidableEq τ]
  (t : Table sch) (nm : η)
  (hnm : sch.HasCol (nm, τ)) :
  nrows (count t nm hnm) =
  (getColumn2 t nm hnm).unique.length
```

Deviations from B2T2

Formally verified all contracts from the B2T2 specification, with the following exceptions:

- We do not mandate schematic uniqueness.
  - Can optionally be enforced as a subtype:
    `{t : Table sch // sch.Unique}`
- We diverge from the B2T2 spec where elements of action lists are applied sequentially rather than in parallel:
  - E.g., the following is valid:
    `renameColumns t A[("A", "B"), ("B", "C")]`

Takeaways

Verification takeaways:

- Since B2T2 is inspired by dynamically typed languages, many proofs are immediate from type signatures.
- Extensive use of data-valued proofs challenges some built-in tactics (e.g., `induction`) and the pattern-matcher.
- Schematic specifications represent an avenue for more robust automation.