Incrementally structuring freeform tables in spreadsheets

Prabhanshu Gupta, Ravi Chandra Padmala, Prathyush Pramod and Pranav Pramod.
prabhanshu@nilenso.com

ABSTRACT

The cellular grid of spreadsheets is an excellent medium to make tabular structures, which are a powerful problem solving tool. However, formulae are primarily written using cell references which do not capture the larger tabular structures the cells are a part of. This mismatch is known to cause duplication of effort and is a common source of errors. On the other hand, end users eschew formal abstractions due to the upfront cognitive cost they demand with little apparent reward.

In this paper, we design a novel scheme to incrementally structure tabular data in spreadsheets. We first develop an understanding of the visual structure of rich, free-form tables, with the goal of arriving at a simple ruleset to capture it based on visual characteristics alone. We propose a mechanism that allows users to structure and index data across a wide variety of free-form tables using this ruleset. We then present a prototype formula language for querying and transforming this structure. Our approach thus aims to bridge the gap between the flexibility of rich tables and the benefits of structured data.

1. INTRODUCTION

Formula replication in spreadsheets is understood to be a common source of errors. It can be attributed to two underlying issues. Firstly, replicating formulas poses the risk of inconsistencies slipping in. When a formula is updated or new data is added to a sheet, the onus of ensuring that formulas are correctly copy-pasted and include the complete cell range falls on the user. This adds manual effort to maintenance.

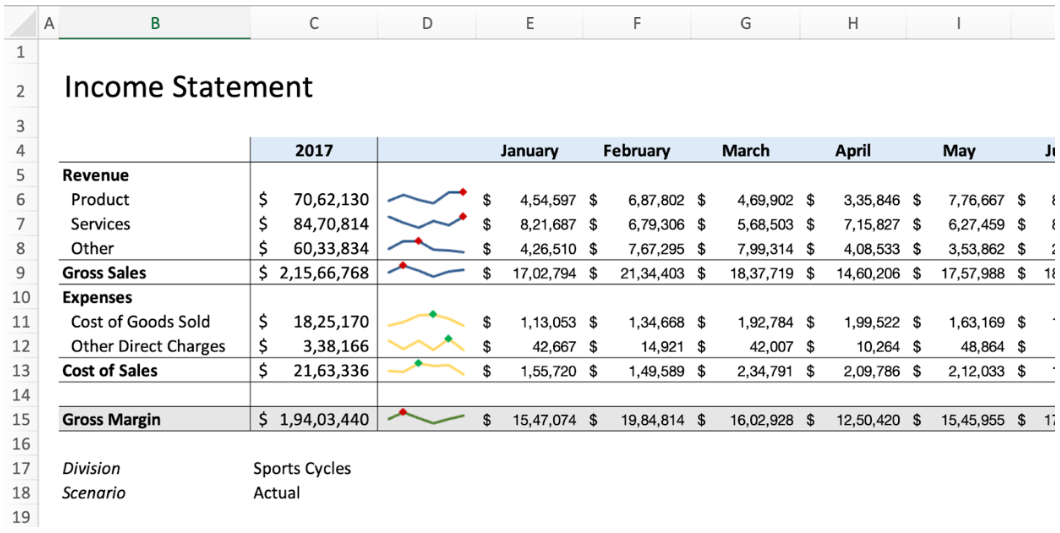
The second, more foundational issue is the inability of formulas written at the cell level to capture user intent, especially when applying them to a range of cells. Spreadsheets are commonly used to work with datasets such as a list of values (a list of phone numbers) or a list of records (a list of employees) – more abstractly these can be thought of as collections such as arrays, sets and lookup tables. When a user replicates a formula across many cells, the presence of an underlying collection is implicit in the action [Bartholomew, 2019].

However, using an explicit data structure to model a problem requires upfront planning and commitment which can pose a hurdle to spreadsheet users. There is a strong preference among end users for methods with lower cognitive load. Learning to use data structures effectively can be challenging and time-consuming [Sarkar & Chalhoub, 2022]. Abstract data structures that are familiar to programmers can increase the cognitive distance between the visual-first, hands-on grid interface and the user [Hermans et al., 2015]. We hence approach the design of a system for structuring and working with collections by building on the ways in which users already conceive of structure in their spreadsheets.

A big part of the success of spreadsheets as a problem-solving tool is their ability to visually organise and present data. The two-dimensional layout acts as a medium in which users can put data values, labels, and annotations. Further, Nardi & Miller note that spreadsheets utilise the strengths of tabular representation for viewing, structuring, and displaying data. Tables are an effective cognitive tool for problem-solving, even though the perceptual reasons behind their effectiveness are not fully understood. They help users juxtapose, draw comparisons and efficiently identify patterns across large datasets [Nardi & Miller, 1990]. The blank grid of spreadsheets provides a structure in which tabular thinking can be cast.

Spreadsheet tables seldom follow a rigorous structure like that of relational tables or matrices. Instead, data is often organised loosely on the grid, resulting in what [Bartram et al., 2021] call "rich tables". Interacting with data in a rich table forms an essential part of a user's sensemaking process. These tables are freeform and incrementally built, with collections and meta-information embedded within their visual layout. The flexibility of the grid enforces minimal constraints on organisation, which allows for different spatial organisations and schemas for the same underlying data.

In order to support users to work with collections, we find ourselves looking at the mechanics of tables as a medium for displaying data and how they capture the notions of storing collections and making summaries. We posit that a mechanism for structuring data needs to embrace the visual nature of rich tables in order to serve spreadsheet users effectively.



Income Statement

	2017		January	February	March	April	May
Revenue							
Product	$ 70,62,130		$ 4,54,597	$ 6,87,802	$ 4,69,902	$ 3,35,846	$ 7,76,667
Services	$ 84,70,814		$ 8,21,687	$ 6,79,306	$ 5,68,503	$ 7,15,827	$ 6,27,459
Other	$ 60,33,834		$ 4,26,510	$ 7,67,295	$ 7,99,314	$ 4,08,533	$ 3,53,862
Gross Sales	$ 2,15,66,768		$ 17,02,794	$ 21,34,403	$ 18,37,719	$ 14,60,206	$ 17,57,988
Expenses							
Cost of Goods Sold	$ 18,25,170		$ 1,13,053	$ 1,34,668	$ 1,92,784	$ 1,99,522	$ 1,63,169
Other Direct Charges	$ 3,38,166		$ 42,667	$ 14,921	$ 42,007	$ 10,264	$ 48,864
Cost of Sales	$ 21,63,336		$ 1,55,720	$ 1,49,589	$ 2,34,791	$ 2,09,786	$ 2,12,033
Gross Margin	$ 1,94,03,440		$ 15,47,074	$ 19,84,814	$ 16,02,928	$ 12,50,420	$ 15,45,955

Division Sports Cycles
Scenario Actual

Fig. 1. An excerpt from a rich table showing a financial model for a company.

In section 2 we look at previous work in understanding the logical structure of drawn tables. We propose a table model in section 4 that can be operationalised in a spreadsheet. A central theme in our work is balancing the tension between freeform and structured approaches to problem exploration. We describe a table parsing mechanism that plays well with the physics of the grid and enables the incremental structuring of rich tables. Section 4.1 describes an extension to the formula language to access and transform the tables so constructed.

The primary motivation for devising a table model is to support users to work with collections of data laid out on the grid. We also wish to augment the expressivity of the formula language by providing a way to use label names while performing higher level tasks such as lookups, filters and aggregations. In section 5 we discuss the range of rich tables our model captures and outliers that need additional consideration.

2. PRIOR ART AND INFLUENCES

2.1 Structuring approaches to spreadsheets

The idea of bridging the gap between the logical level that users operate at and the spreadsheet's computational model has been studied extensively [Tukiainen M., 2001] [Isakowitz et al., 1993]. [Sarkar & Chalhoub, 2022] provides a classification for approaches that address this gap by introducing structuring mechanisms. Our method falls squarely in their category of dynamically allocating regions of the spreadsheet grid to render higher-level data types, such as Excel Tables and Dynamic Arrays.

Methods that require upfront commitment to a structure (such as relational modelling) are understood to be at odds with how spreadsheet users work through a problem incrementally – here we draw on the ideas of gradually introducing structure from [Miller et al., 2016] in order to support the coexistence of free-form and structured modelling. We also draw from studies such as [Nardi et al., 1994] [Kankuzi, 2015] [Hellman, 2008] which propose the usage of a modelling language that doesn't rely on cell references but is closer to human language – we use row and column labels in the extension to the formula language.

2.2 Models for tabular structures

While considerable research has gone into developing tools for extracting data from spreadsheet tables, much of it employs machine learning and heuristics [Eberius, J., 2013] [Thiele et al., 2017]. These approaches provide varying amounts of accuracy and do not provide a simple model that can be applied precisely across a diversity of use cases. Instead, we use the logical model of tables put forth by [Wang X., 1996] and [Hurst M., 2000] as a starting point. Wang notes that a table is composed of two types of cells: data cells and access cells. Hurst notes

"A table is an object which uses linear visual cues to simultaneously describe logical connections between the discrete content entries in the table. A content entry is the basic component of information in the table. A content entry can be any visual symbol. [...] table models break down the content entries in tables into data content entries and heading or label content entries. The idea behind label entries is that they are used as indexes for information in the table."

This description of a table informs much of our table model – Hurst suggests that the process of reading a table starts with following the line of sight from labels and actively trying to classify elements into categories. Labels act as indexes into information regions and hint at possible categorisations one can form. He notes the lack of a well-formed definition for what a category is in this context and develops the notion of categories as hierarchical semantic structures, however we view the labels primarily as indexes and the general idea of *any relation* existing between labels and data suffices for our work. Further, Wang relies on positional sections such as *Headers*, *Box*, *Stub* and so on, however this constrains applicability to freeform tables and we avoid such segmentation of tabular regions.

3. OUR APPROACH

Two features present in Microsoft Excel inform our approach to integrating rich data structures on the grid: Excel Tables and Dynamic array formulas. Excel Tables allow users to demarcate tabular data and use names for tables and columns in formulas. Dynamic array formulas make it easy to write formulas that work on entire ranges. The spillage mechanism sets us up for designing language primitives that can return richer data structures – formulas that return arrays or matrices get to occupy cells underneath, eliminating the need for CSE formula or situating matrices/objects in cells [Blackwell et al., 2004].

Both features however fall short of our needs in other ways. Excel tables require data to be structured relationally (with headers on top) and do not permit freeform structures. Range references can be used without any commitment to structure, however they are fixed and do not expand when new data is appended to the end of a range. Named ranges also suffer from the same problem since they are tied to fixed ranges.

We approach the design of our table model by surveying a wide variety of tables across various print and digital media and noting patterns of information organisation. We divide the problem into two key considerations:

Understanding structure

- How to capture the richness of tables that people put down informally?
- How to minimise the effort in specifying the schema of a table?
- How to access the collections (and constants) embedded in free-form tables in the formula language?

Determining bounds

- When does a cell become part of a collection? How does a collection grow in size?
- Can we determine a collection's bounds from its visual presentation?

4. A MODEL FOR RICH TABLES

[Kohlhase A., 2013] suggests a perception model for the spreadsheet interface based on [Probst G., 1997] et al.'s work, in which glyphs, data, information and knowledge form a hierarchy of increasing complexity and meaning in an information environment. We build on it, considering the grid primarily as a medium to lay out *glyphs* on. Glyphs are any characters denoting text or numbers. Combined with syntax they become *data*, which combined with context from the user's problem domain becomes *information*.

Similar to laying out information on paper, data elements close to each other appear to be part of the same group and a gap indicates separateness. Spreadsheet users tend to represent collections (sets, lists, lookup tables) on a grid by clustering related data elements together and putting distinct sets farther apart. We refer to these areas of the grid where glyphs (and thereby data elements) are clustered as *information regions*.

These regions are generally tabular in nature and end up forming the rich tables we've discussed before. Rich tables can contain a mix of constants, collections, summaries and notes coexisting with each other. In simpler cases they contain just a single list. Our model's primary concern is establishing the rules for this coexistence and being able to identify individual data elements embedded in rich tables.

First, we touch upon the bounds problem. When new data is appended to a region, we would like for it to expand to contain the new items. A grid can contain multiple information regions, and knowing their boundaries is a prerequisite to determining the size of collections contained in them.

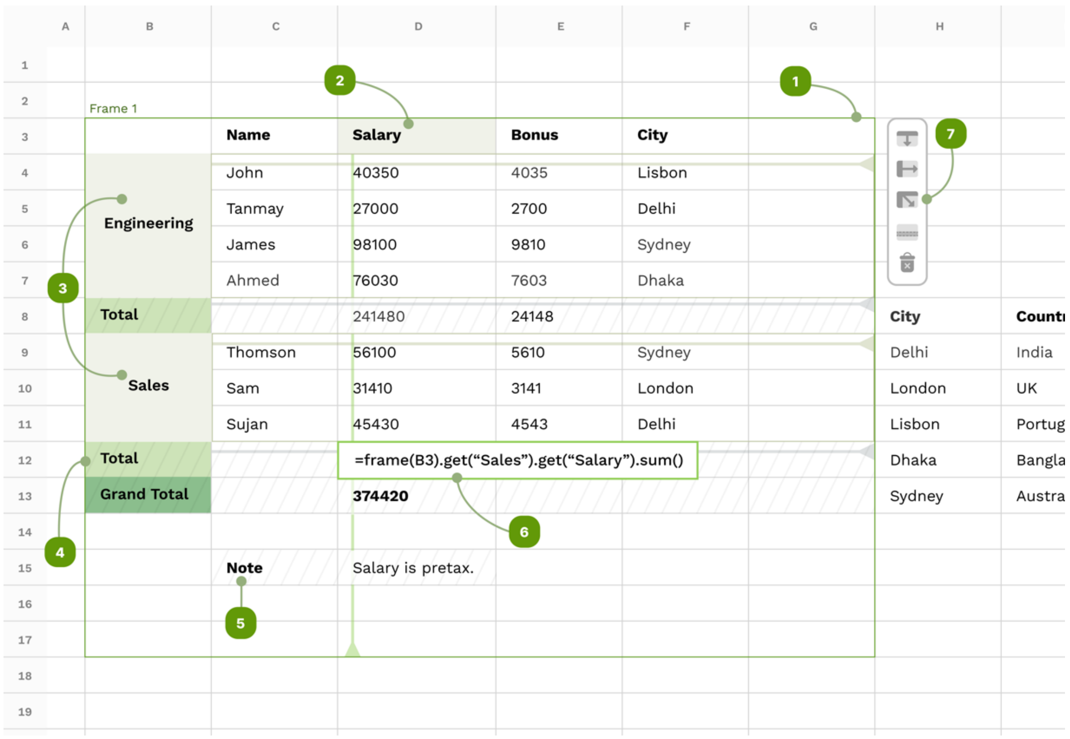
These boundaries are often implicit as the regions are primarily visual clusters defined by the relative spacing between glyphs. We note that users rely on *whitespace* in the form of blank cells to mark separation between two regions in a grid. Information regions are surrounded by a moat of whitespace which hints at the area occupied by them, however users might not explicitly decide on the amount of surrounding whitespace they add and it can vary considerably between spreadsheets.

Whitespace is also used for a variety of other reasons. It can be used inside an information region to make sections; it can be added as a placeholder while working through a problem, or occur temporarily when blank rows/columns are inserted. Correlating the presence of blank cells with their intended purpose is often not straightforward. The many different ways people use them makes it hard to establish a reliable method to find the bounds of a data region by only looking at blank cells. As a solution, we ask the user to mark the bounds of data regions explicitly.

Next, we address the structure problem. Instead of identifying designated areas within a table such as headers, stub, body, cut-ins and so on, we propose that a table's semantic structure emerges mainly from

- the indexing effect of a label on cells that follow it,
- the usage of labels with different spans to build visual hierarchies, and
- the interaction between labels of matching spans to partition the table.

In this framing, labels can exist freely in a table anywhere. With these ideas about boundaries and labels in mind, we now outline the components of our table model.



	A	B	C	D	E	F	G	H	
1									
2		Frame 1							
3			Name	Salary	Bonus	City			
4			John	40350	4035	Lisbon			
5		Engineering	Tanmay	27000	2700	Delhi			
6			James	98100	9810	Sydney			
7			Ahmed	76030	7603	Dhaka			
8		Total		241480	24148			City	Countr
9			Thomson	56100	5610	Sydney		Delhi	India
10		Sales	Sam	31410	3141	London		London	UK
11			Sujan	45430	4543	Delhi		Lisbon	Portug
12		Total		=frame(B3).get("Sales").get("Salary").sum()				Dhaka	Bangla
13		Grand Total		374420				Sydney	Austra
14									
15			Note	Salary is pretax.					
16									
17									
18									
19									

Fig. 2. A sample worksheet containing a company's employee records with an annotated view of (1) a **frame** with its bounds marked with a border; (2) "Salary" marked as a **top label** pointing to a list of salary numbers; (3) employees categorised by department with **left labels**; (4) a **skip label** that cuts across the salary list; (5) comments marked as individual **skip cells**; (6) a **query** that totals the salary for the Sales department and (7) a tool panel for annotating labels. The lines extending from the labels show the range they apply to based on the indexing rules. Note that the Name, Bonus and City columns haven't been labelled by the user yet.

Frames
A frame is a bounding box that can be overlaid on an area of interest on the grid (information region) which a user wishes to use as a tabular structure. Since whitespace is used inconsistently for several purposes, we do not rely on it to infer separation between information regions on a worksheet. Frames allows users to demarcate bounds explicitly and mark separation between two adjacent regions. The bounds also determine the size of collections contained within. Frames can grow to accommodate new data.

No constraint is applied on the amount of whitespace required between two frames. It is expected that users would continue using space as they would in the absence of frames. Two frames on a worksheet cannot overlap – each cell on the grid can belong to at most one frame. Frames do not enforce any other constraints. Once drawn, a frame can be freely resized by dragging along its edges.

A frame is allowed to partially overlap with a cell (in the case of a merged cell), although these cells do not functionally belong in the frame. Only cells that are completely within the bounds of the frame are considered to be contained by it.

Cell roles
Cells inside a frame are either a *label cell* or a *data cell*. Any cells that not marked as labels are data cells by default. The role of a cell is usually implicit in visual cues such as formatting, background, relative spacing, and most commonly, domain information embedded in the meaning of words. We ask users to mark label cells in a frame explicitly in order to not rely on these implicit cues. Labels can index data cells based on rules outlined under label indexing.

Cell spans
Merging cells in a spreadsheet allows users to create a single cell that spans across multiple rows and columns. They are often used to create spanning labels which users employ to create visual hierarchy and avoid repeated values across rows and columns. The number of rows occupied by a merged cell is called its row span, and the number of columns its column span. Unmerged cells have row and column spans of 1.

Label direction
Each label inherently suggests a line of sight that a reader follows visually. We call this the *direction of application*, or simply the *direction*. We focus on spreadsheets written in left-to-right languages, which determine the users' reading order; the possible directions for a label are: top-to-bottom, left-to-right, and top-left to bottom-right. We refer to labels with these directions as *top labels, left labels,* and *top-left labels* respectively for brevity.

A label's direction is independent of its position within the table. Commonly top labels will be found on the top edge of a table as headers and left labels on the left edge as stub headers. However this is not always the case. Labels can also be present at the end of the table (in summary rows and columns) and in between (as cut-ins).

A label's direction of application is also influenced by visual and domain-level cues, as with cell roles. We ask the user to specify the direction of each label to avoid implicit inference.

Skip Cells and Skip Labels
A special feature of rich tables is the ability to intersperse summaries and notes within collections. Since labels can categorise collections both horizontally and vertically,

horizontal summarizations can intersect with vertical collections and vice-versa (such as the "Total" rows in the example). Summaries can also partition a table into sub-tables and subsections. We note that users skip over these interspersed cells when reading data collections indexed by labels – in Fig. 2. the Total engineering salary is not part of the Salary list. Inferring the intended role of such cells based on its contents/other heuristics is unreliable since

- Formulas can be present in a range for many reasons, not just summaries.
- They can contain notes or comments which are literal strings. These cannot be distinguished from other labels or data.

We ask users to mark such cells as *skip cells*, which provides a way to add summaries and comments freely, without having to declare sections in a top-down manner. Labels and data cells, both can be marked as skip cells. If a label cell is marked as a skip cell, all the cells indexed by the label are considered to be skipped – we call this label a *skip label*.

Label indexing

Given a frame on a worksheet with some data cells, labels (each with a direction and span) and skip cells, we now describe a mechanism for determining the cells indexed by a label.

A label indexes data cells in its direction of application that fall within its span, the cell adjacent to it until one of two conditions is met:

1. The end of the frame is reached, meaning there are no more cells in that direction.
2. Another label with the same direction and span is encountered, indicating the start of a new category of cells described by the second label.

Multiple labels can apply to a single cell. Skip cells are only indexed by skip labels – they are not indexed by regular labels. An algorithm for indexing is provided in Appendix A.

4.1 Extending the formula language

Given a tabular dataset, users have high-level objectives and questions that stem from their domains of work. For example from Fig. 2:

- Calculate bonus for all employees in a new column, which is 10% of their salary.
- What is James' salary?
- What is the total salary of employees in Sales?
- Given a list of cities and countries in a separate table, which country does each employee live in?

These queries are commonly written using relative references and range references. We propose an addition to the existing spreadsheet formula language that allows users to query and transform data in frames at a structural level – a frame query language. The language enables navigating label indexes and retrieving specific subsets of data at the intersection of multiple labels to support these use cases.

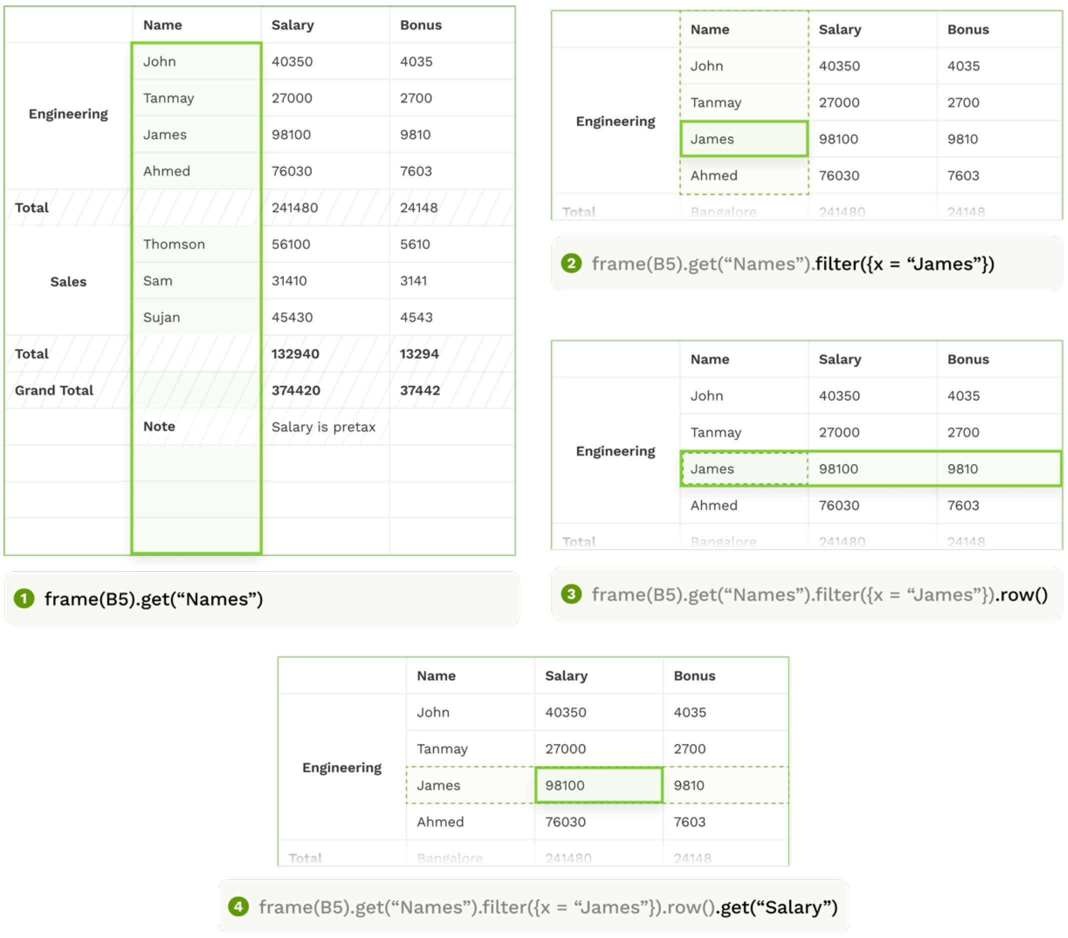
We draw inspiration from path-based query languages like XQuery [Chamberlin D., 2002] and JSONPath [Goessner S, 2007]; we reach for them as they provide a close analogue to how users naturally navigate in a table. By chaining together label lookups and navigational steps, users can traverse a frame declaratively. We mainly borrow the idea of path expressions, steps, function chains, and set operations.

Let us look at the first task as an example. The bonus amount for each employee is 10% of their salary. To calculate the bonuses, we draw a frame around the table and mark cell D3 as a Top label. We can then write the bonus formula as

```
frame(B5).get("Salary") * 0.10
```

A query expression consists of a series of function calls, or steps, chained together using a dot notation. Each step in the function chain takes an *area* as input and returns a transformed area as output, which is then passed as the implicit first argument to the next step in the chain.

An area is a set of cells selected from a frame. The cells that form the area need not be contiguous; they can be spread across the frame. The relative spatial orientation of the cells in the frame is preserved in the area. Areas spill into the grid similar to dynamic arrays. The cell with the smallest row and column number in the set (top-left cell) is considered to be the starting position of the spillage. The second query (finding James' salary) illustrates usage of the query language further (Fig. 3).



	Name	Salary	Bonus
Engineering	John	40350	4035
	Tanmay	27000	2700
	James	98100	9810
	Ahmed	76030	7603
Total		241480	24148
Sales	Thomson	56100	5610
	Sam	31410	3141
	Sujan	45430	4543
Total		132940	13294
Grand Total		374420	37442
	Note	Salary is pretax	

1 frame(B5).get("Names")

	Name	Salary	Bonus
Engineering	John	40350	4035
	Tanmay	27000	2700
	James	98100	9810
	Ahmed	76030	7603
Total	Bangalore	241480	24148

2 frame(B5).get("Names").filter({x = "James"})

	Name	Salary	Bonus
Engineering	John	40350	4035
	Tanmay	27000	2700
	James	98100	9810
	Ahmed	76030	7603
Total	Bangalore	241480	24148

3 frame(B5).get("Names").filter({x = "James"}).row()

	Name	Salary	Bonus
Engineering	John	40350	4035
	Tanmay	27000	2700
	James	98100	9810
	Ahmed	76030	7603
Total	Bangalore	241480	24148

4 frame(B5).get("Names").filter({x = "James"}).row().get("Salary")

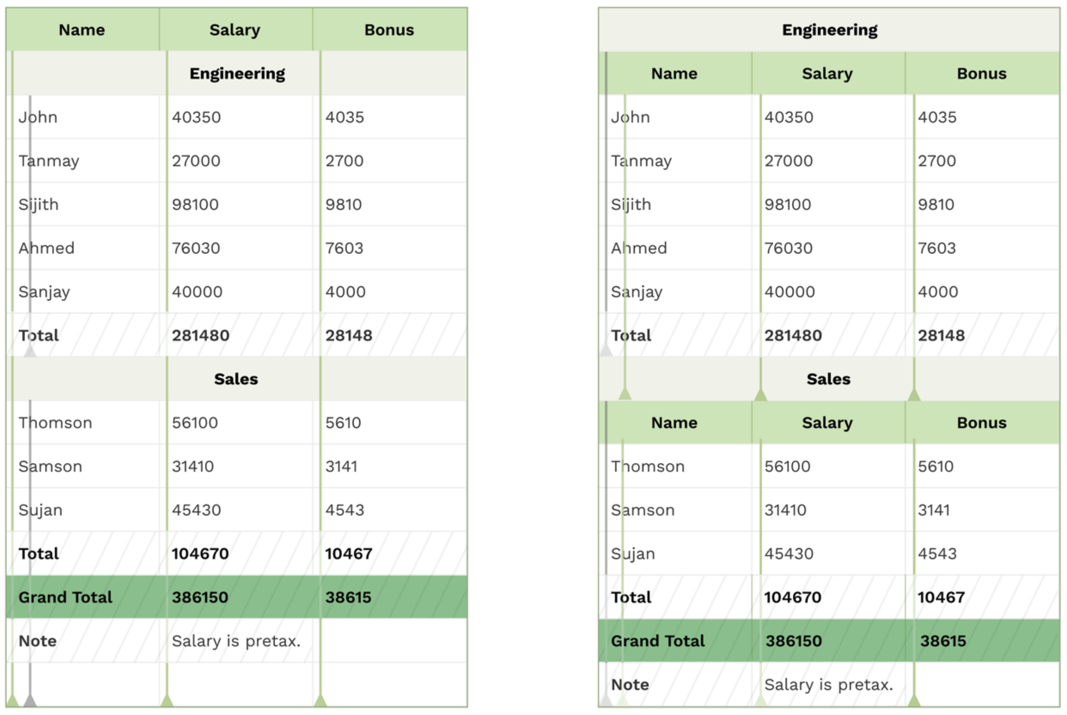
Fig. 3. Queries as stepwise area transformations. The highlighted sections depict the selected area at each step. A single cell containing the salary for James is returned at step 4.

Some of the salient built-in functions in the query language are listed below.

`frame(cell_ref)`	Returns the frame area that cell_ref belongs to.

`get(area, label_name)`	Selects the intersection of the area with the area under the given label.
`row(area)`	Expands the area horizontally to the frame's full horizontal span, excluding labels.
`column(area)`	Expands the area vertically to the frame's full vertical span, excluding labels.
`filter(area, filter_fn)`	Returns an area with cells for which filter_fn evaluates to true.
`union(area1, area2)`	Joins the bounding boxes of the two areas.
`intersect(area1, area2)`	Returns the common overlap between two areas.
`compact(area1)`	Returns the area with all blank cells removed. Non-blank cells are shifted towards the top-left corner.

Fig. 4. demonstrates the ability of the label model to work across different organisations of the same data. Both tables are variations on the same data as contained in Fig. 2. There are subtle differences – the Name, Salary and Bonus headers are repeated in (a) whereas they are only present once in (b). The same query for calculating the Total Sales salary works here as well: `frame(B3).get("Sales").get("Salary").sum()`.



Name	Salary	Bonus
	Engineering	
John	40350	4035
Tanmay	27000	2700
Sijith	98100	9810
Ahmed	76030	7603
Sanjay	40000	4000
Total	**281480**	**28148**
	Sales	
Thomson	56100	5610
Samson	31410	3141
Sujan	45430	4543
Total	**104670**	**10467**
Grand Total	**386150**	**38615**
Note	Salary is pretax.	

	Engineering	
Name	**Salary**	**Bonus**
John	40350	4035
Tanmay	27000	2700
Sijith	98100	9810
Ahmed	76030	7603
Sanjay	40000	4000
Total	**281480**	**28148**
	Sales	
Name	**Salary**	**Bonus**
Thomson	56100	5610
Samson	31410	3141
Sujan	45430	4543
Total	**104670**	**10467**
Grand Total	**386150**	**38615**
Note	Salary is pretax.	

Fig. 4. The Name, Salary, and Bonus headers apply to the entire table in the first layout, however they are repeated in the second example. The model is able to segment both tables appropriately using the indexing rules.

4.2 Workflow

Users select a range of cells on a worksheet with a rich table and invoke a "Mark Frame" command, which creates a frame around the selected cells. The frame is visually indicated by a border, clearly demarcating the boundaries of the information region.

Within the frame, users can mark labels by selecting a cell, invoking the "Mark Label" command, and choosing the label's direction. The system provides visual feedback indicating the indexed cell ranges so they can verify if the inferred structure matches their intention as they build it. In our prototype, labels are displayed as highlights and lines are drawn over ranges indexed by each label.

Using a range of cells in a query can serve as the trigger point for marking a label. Users need not mark all labels at once, only the ones they intend to reference in a query. The effort in marking labels depends on the intricacy of the table structure: more subsections, interspersed cells and comments will require more labelling. However since the structuring scheme is incremental, users have the option to only mark cells contributing to a formula immediately. In most cases a user might not have to markup all labels/skip cells in a frame.

Once a label has been marked, it is available to be used in other queries and contributes to the table structure. When new data is added to the row below the frame, it expands to include the newly added row. Label references in formulas always point to entire collections as long as they are in the frame.

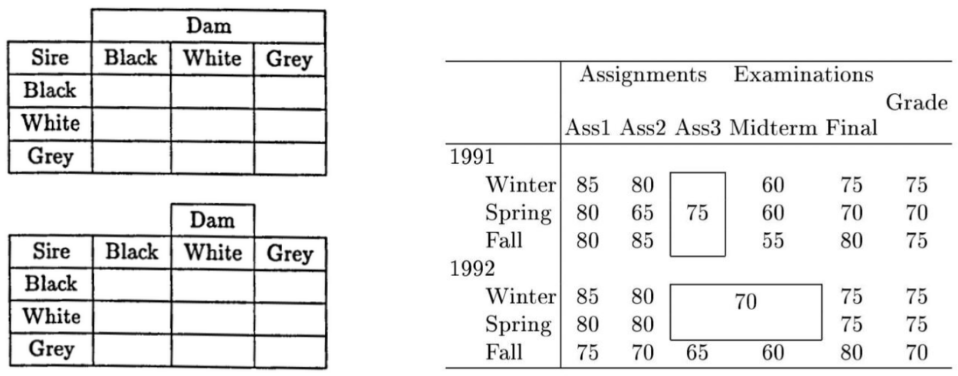
4.3 Coverage

The primary intent of applying our model to a rich table is to segment and index it by means of labels for usage in formulas. Similar to Wang's and Hurst's models, we are able to capture simple constructions starting from lists with a single label, relational tables (similar to Excel Tables) and cross-tables with spanning labels. While Wang and Hurst establish a logical model to derive semantic relations amongst labels and data elements, we do not develop such an understanding. This restricts the model's utility in providing a standard format for information extraction but provides us the flexibility to handle more complex table structures. We note a few areas of improvement – particularly the handling of cut-ins and marginalia such as in Fig. 4.

We have tested the model empirically against a diverse set of real-world spreadsheet tables sourced from various corpuses. We have not conducted a formal evaluation against a standardised dataset yet. We note a few cases that our model does not capture, due to two reasons primarily.

First, the indexing rules we propose might be unable to correctly segment a layout as a user would expect – there is a mismatch between their visual perception and the capabilities of our model. Fig. 5(a) shows a common example where spanning labels are not created using merged cells. Since Dam has the same span as White, the Dam label does not apply beyond the second row and does not form an hierarchy. In Fig. 5(b), the merged cell with value 70 spans multiple cells and is not indexed by Winter, Spring or Midterm (similarly the merged cell with 75).

More generally, establishing the ground truth of complex tables can be challenging [Hu, J., 2001]. Users can have differing interpretations of the same table and such tables can be inherently ambiguous. Fig. 6 illustrates an ambiguous table where a reader familiar with the domain might not categorise Sheep and Goat under the animal type Swine, and treat them as separate categories. Indexing such a table with the current mechanism can be erroneous one way or the other.



	Dam		
Sire	Black	White	Grey
Black			
White			
Grey			

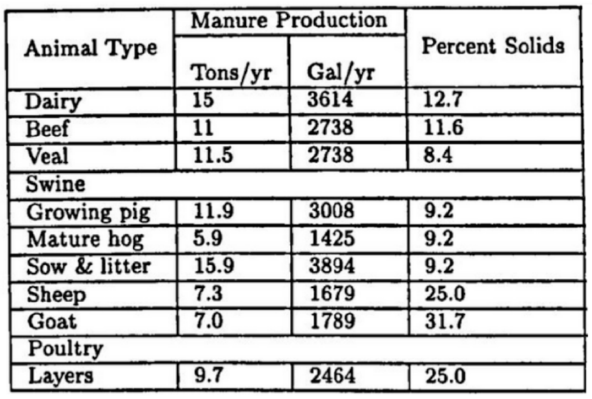
		Dam	
Sire	Black	White	Grey
Black			
White			
Grey			

(a)

	Assignments			Examinations		
	Ass1	Ass2	Ass3	Midterm	Final	Grade
1991						
Winter	85	80	75	60	75	75
Spring	80	65		60	70	70
Fall	80	85		55	80	75
1992						
Winter	85	80	70		75	75
Spring	80	80			75	75
Fall	75	70	65	60	80	70

(b)

Fig. 5. The label indexing rules fail to apply to these tables as a user might expect.



Animal Type	Manure Production		Percent Solids
	Tons/yr	Gal/yr	
Dairy	15	3614	12.7
Beef	11	2738	11.6
Veal	11.5	2738	8.4
Swine			
Growing pig	11.9	3008	9.2
Mature hog	5.9	1425	9.2
Sow & litter	15.9	3894	9.2
Sheep	7.3	1679	25.0
Goat	7.0	1789	31.7
Poultry			
Layers	9.7	2464	25.0

Fig. 6. An ambiguous categorisation (Sheep and Goat).

We also note that collections can exist without any labels, which are usually lists of single items. Unlabelled collections are not covered by the model, however it is possible to add functions to the proposed query language to capture such collections.

5. DISCUSSION

We have proposed a model for rich tables that provides users a way to incrementally structure freeform tables while introducing a small set of new concepts. It has a few advantages: first, it provides a deterministic way to infer the bounds of data elements in a table based on user-specified labels, rather than relying on heuristics or visual cues. Second, it allows users to index these data elements by name and use them in formulas with the query language. Third, this lets users refer to collections that can expand as new data is added around them, which allows users to avoid formula replication.

We have built a small prototype application as a proof of concept demonstrating these [Bean, 2024]. We are planning a user study to evaluate the model's effectiveness with respect to cognitive load and usability. Further, we are studying improvements to the query language's ergonomics and exploring richer use cases such as creating pivot tables and aggregations. We are also planning on benchmarking the applicability of the model against popular open spreadsheet corpuses.

5.1 Limitations

We note a few limitations to our current approach. We have only focussed on spreadsheets with left-to-right languages and it remains to be seen if a different set of label directions will be required to serve languages with other writing directions. The applicability across different table layouts needs to be closely studied and the model tweaked appropriately to allow for further variation if needed. Another key challenge is the efficient recalculation of frame queries when cells are updated. In our current prototype frame queries are not re-evaluated reactively, which requires further development work.

Lastly, we have not explored a logical model for establishing relationships between labels and data. An underlying semantic model would allow us to normalise rich tables to known structures such as relational tables or graphs.

5.3 Increase in user effort

Using the model requires some manual steps: marking frames and declaring labels. This poses a challenge in the model's uptake. We have noted that the effort required doesn't necessarily depend on the sheer number of rows/columns; rather, it increases with the complexity of the table structure. Simple tables are easy to label and use, whereas tables with more subsections, interspersed cells and comments require more work. The incremental nature of the model requires users to only mark labels needed to be used in a formula. Users are not required to mark unimportant or far-off cells. This optimises effort on the part of the user by focussing their attention on just the relevant areas.

In early demos we have also noticed that users require guidance to understand the model's workings. Picking the appropriate frame size requires some familiarity with the dynamics of label indexing: A user needs to know the rough size of ranges they want indexed by each label. Frames require little commitment and can be freely resized, so we think users should be able to develop this familiarity with experimentation. One line of research we would like to pursue is providing assistance through good interface design and suggestions/prompts around marking frames and labels.

Lastly, keeping the model's learning curve gentle has been an important consideration in our design choices. To that end, we believe that the visual nature of the model builds on familiar patterns of grid usage. The query language attempts to map well to the visual representation. Nonetheless, it is a significant departure from spreadsheet formula syntax, so we expect a fair amount of refinement to make it simple enough to use.

Since learning new spreadsheet features like frames and labels does require some effort, we expect advanced users working with large, complex sheets to find them more appealing than novice users with simpler sheets. For advanced users, the maintainability and reduction in errors offered by such features may well justify the additional work. We plan on conducting user studies to evaluate how experience level affects their preferences.

ACKNOWLEDGMENTS

We would like to thank Akshatha H for helping review drafts of this paper. We are also grateful to nilenso software for supporting our work.

REFERENCES

Bartholomew, P. (2019), "Will Dynamic Arrays finally change the way Models are built?", Proceedings of the EuSpRIG 2019 Conference, pages 149-159.

Bartram, L., Correll, M., and Tory M. (2021), "Untidy Data: The Unreasonable Effectiveness of Tables", IEEE Transactions on Visualization and Computer Graphics, vol. 28, no. 1, pages 686-696.

Bean, https://github.com/nilenso/bean, 11:30am 13/4/2024

Blackwell, A. F., Burnett, M. M., and Jones, S. P. (2004), "Champagne Prototyping: A Research Technique for Early Evaluation of Complex End-User Programming Systems", IEEE Symposium on Visual Languages - Human Centric Computing, Rome, Italy, pages 47-54.

Chamberlin, D. (2002), "XQuery: An XML query language", IBM Systems Journal, Vol. 41, No. 4, pages 597-615.

Eberius, J., Werner, C., Thiele, M., Braunschweig, K., Dannecker, L., and Lehner, W. (2013), "DeExcelerator: a framework for extracting relational data from partially structured documents", 22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, pages 2477-2480.

Goessner, S. (2007), https://goessner.net/articles/JsonPath/index.html#e2, 12pm 13/4/2024

Hermans, F. and Storm, T. (2015), "Copy-Paste Tracking: Fixing Spreadsheets Without Breaking Them", Proceedings of the First International Conference on Live Coding.

Hu, J., Kashi, R., Lopresti, D., Nagy, G., and Wilfong, G. (2001), "Why Table Ground-Truthing is Hard", Sixth International Conference on Document Analysis and Recognition, pages 129-133.

Hurst, M. (2000), The Interpretation of Tables in Texts, Ph.D. thesis, University of Edinburgh.

Isakowitz, T., Schocken, S., and Lucas Jr., H. C. (1993), "Toward a Logical/Physical Theory of Spreadsheet Modelling", Working Paper Series, Stern School of Business.

Kankuzi, B. (2015), "Deficiencies in Spreadsheets: A Mental Model Perspective", Publications of the University of Eastern Finland, Dissertations in Forestry and Natural Sciences No 183.

Koci, E., Thiele, M., Romero, O., and Lehner, W. (2017), "Table Identification and Reconstruction in Spreadsheets", 29th International Conference on Advanced Information Systems Engineering, CAiSE'17.

Kohlhase, A. (2013), "Human-Spreadsheet Interaction", INTERACT 2013, Part IV, p 571–578.

Miller, G., Hermans, F., and Braun, R. (2016), "Gradual Structuring: Evolving the Spreadsheet Paradigm for Expressiveness and Learnability".

Nardi, B. and Miller, J. (1990), "The spreadsheet interface: A basis for end-user programming", Human-Computer Interaction: INTERACT '90.

Nardi, D. and Serrecchia, G. (1994), "Automatic Generation of Explanations for Spreadsheet Applications", Proceedings of the Tenth Conference on Artificial Intelligence for Applications.

Probst, G., Raub, S., and Romhardt, K. (2003), “Wissen managen”, 4th edition, Gabler Verlag.

Sarkar, A. and Chalhoub, G. (2022), "It's Freedom to Put Things Where My Mind Wants: Understanding and Improving the User Experience of Structuring Data in Spreadsheets", CHI Conference on Human Factors in Computing Systems, paper 585.

Tukiainen, M. (2001), "Comparing two spreadsheet calculation paradigms: an empirical study with novice users", Interacting with Computers, Volume 13, Issue 4, 1 April 2001, pages 427-446.

Wang, X. (1996), “Tabular Abstraction, Editing and Formatting, Ph.D. thesis”, Univ. of Waterloo.

Hellman Z. (2008), "Breaking Out of the Cell: On The Benefits of a New Spreadsheet User Interaction Paradigm", Proceedings of EuSpRIG 2005, page 113.

APPENDIX

Algorithm The label indexing algorithm, described only for top labels for brevity. Indexing rules for other directions can be written similarly.

```
function BLOCKING_LABEL_TOP(table, label)
    for label' in table.labels.sort_by(label.row) do
        if (label.column == label'.column) and
           (label.dirn == label'.dirn) and
           (label.span == label'.span) and
           (label.row < label'.row ) then
            return label'
        end if
    end for
    return None
end function

function LABEL_CELLS(table, label)
    blocking_row = blocking_label_top(label, table.labels).row
    end_row = blocking_row is None ? table.end_row : blocking_row - 1
    end_column = min(label.end_column, table.end_column)
    end_cell = cell(row = end_row, column = end_column)
    return area(start = label, end = end_cell) - table.labels
end function

function SKIPPED_CELLS(table)
    skip_labels = table.skip_labels()
    cells = set(skip_labels)
    for label in skip_labels do
        cells = cells.union(label_cells(table, label))
    end for
    return cells
end function

function LABEL_LOOKUP_BY_NAME(table, label_name)
    labels = table.labels()
    matching_labels = filter_labels_by_name(label_name, labels)
    cells = set()
    for label in matching_labels do
        cells = cells.union(top_label_cells(table, label))
    end for
    return cells - skipped_cells(table)
end function
```