Wide, long, or nested data? Reconciling the machine and human viewpoints

Alan Hall, Michel Wermelinger, Tony Hirst
The Open University, UK
{alan.hall, michel.wermelinger, tony.hirst}@open.ac.uk

Santi Phithakkitnukoon
Chiang Mai University,
Thailand
santi@eng.cmu.ac.th

Abstract

Data expressed in tables may be re-arranged in various forms, while conveying the same information. This can create a tension when one form is easier to comprehend by a human reader, but another form is more convenient for processing by machine. This problem has received considerable attention for data scientists writing code, but rather less for end user analysts using spreadsheets. We propose a new data model, the “lish”, which supports a spreadsheet-like flexibility of layout, while capturing sufficient structure to facilitate processing. Using a typical example in a prototype editor, we demonstrate how it might help users resolve the tension between the two forms. A user study is in preparation.

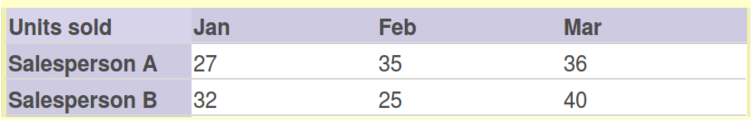
1. Introduction

1.1 Background

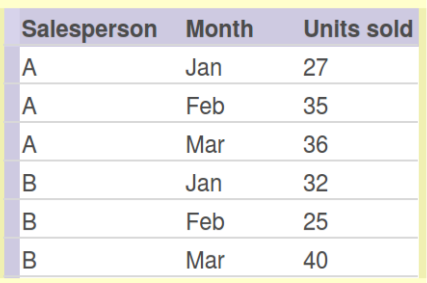
Data in tabular form are everywhere: government statistics, company accounts, scientific results – to name but three examples. But beyond the fact of being arranged in rows and columns, the term “table” covers a multitude of structures, whose choice of layout can affect both efficiency of processing and ease of human comprehension.

Drawing on relational database theory, Wickham (2014) introduced the concept of “tidy” data as a standard that can be used to facilitate preparing tabular data for use with analytical tools. Mount & Zumel (2017) propose “coordinatized” data, performing a similar role: provided a value can be located in some multi-dimensional space, the underlying model can be agnostic to which form of view the user prefers.

A common pattern is when some of the columns in a table form a time series of observations, each row in the table referring to a single subject. A small example is shown in Figure 1(a). This is the *wide* form, which is generally the more human-readable but is an example of an “untidy” layout. The *long* form, where the time series for all subjects are stacked into a single column, is the one more often required by analytical tools, and is shown in Figure 1(b).



Units sold	Jan	Feb	Mar
Salesperson A	27	35	36
Salesperson B	32	25	40



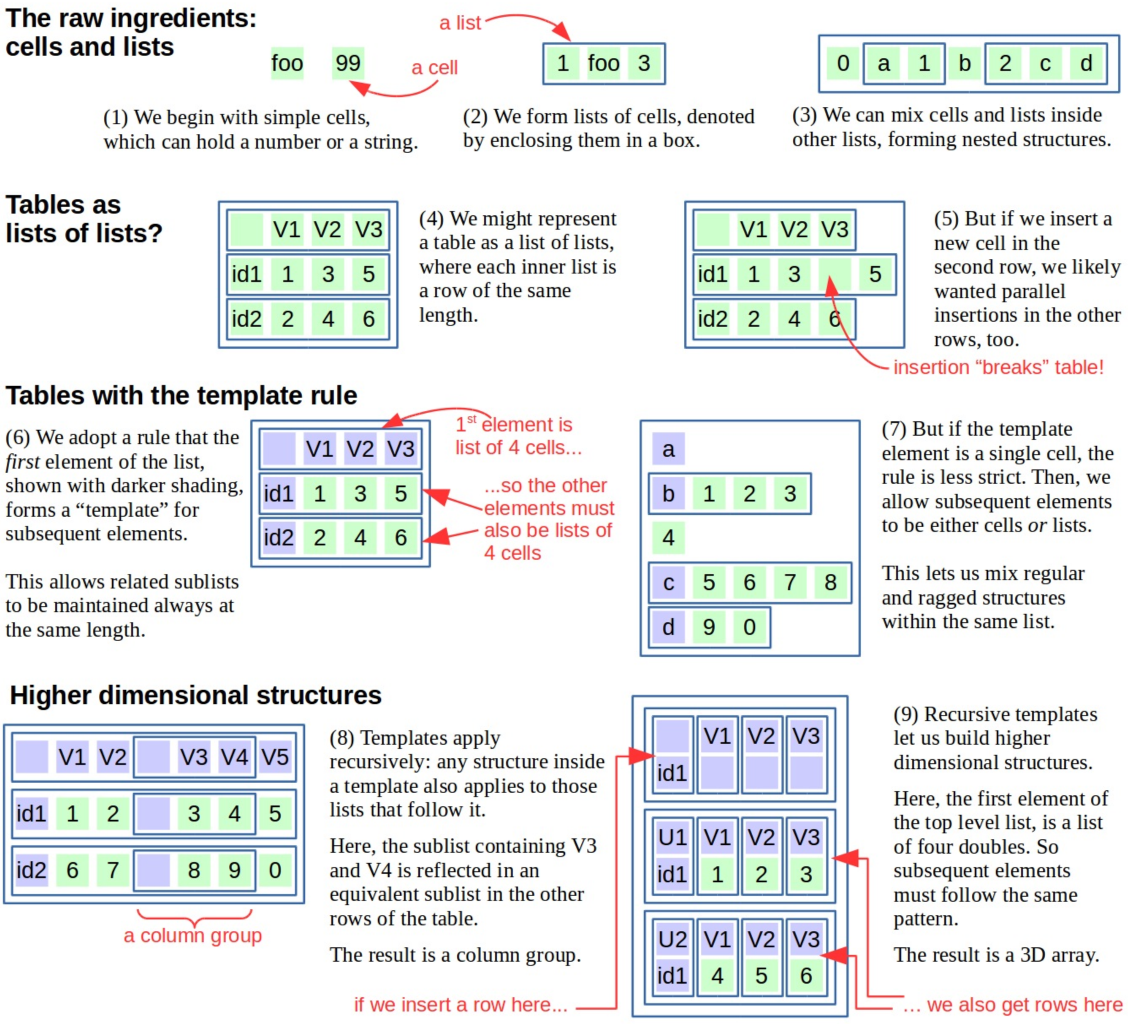
Salesperson	Month	Units sold
A	Jan	27
A	Feb	35
A	Mar	36
B	Jan	32
B	Feb	25
B	Mar	40

Figure 1 – a small dataset in (a) wide form, above; and in (b) long form, right

The work above relates to code-driven analysis of tables. How might these ideas transfer across to interactive analysis, as performed by end users? The predominant end user tool in this space is the spreadsheet, which has achieved its popularity due to its usability (in particular, support for direct manipulation) and its flexibility (Scaffidi, 2016). But in the first author's experience as a professional analyst, it is rare to see even well-designed spreadsheets in long form. It seems a reasonable assumption that this is not a good cognitive fit for the user's view of the data. Referring to the cognitive dimensions of Green & Petre (1996), there is a stronger *closeness of mapping* between the user's mental view of the data structure and the wide form than the long one. This may arise from the enhanced *visibility* of series that are logically continuous or juxtaposed when the two-dimensional space is effectively utilised. The *secondary notation* dimension is also relevant, in that a user can choose to use spacing, shading, gridlines, etc. to help visualise the structure.

Perhaps not surprisingly, previous work on making the spreadsheet “tidier” has taken the direction of making it behave more like a relational database. Bakke & Karger (2016) describe a spreadsheet-like interactive query builder for a backend database, while Chang & Myers (2016) describe a similar tool which takes JSON as input. Cervesato (2007) and Hawkins et al. (2014) both extend the spreadsheet itself with the relation as a native object type, while Mangano et al. (2011) describe a hybrid model in which relational and freeform data can coexist.

These approaches require the user to shift their mindset from arrangements of cells, to entities with attributes. Even the pivot table (and its more recent counterpart, the unpivot command) forces the user to this kind of dual view of their data. We are exploring whether an alternative data representation might allow the user to remain “untidy”, but still provide the machine with sufficient information to assemble the long form implicitly, behind the scenes. Hence the user may retain the flexibility of layout that is a strength of the spreadsheet, but be relieved of the burden (both mental and mechanical) of maintaining and switching between dual copies of the data, the wide and the long. The machine for its part is able to use the deduced structure to facilitate onward processing and future maintenance of the model.



The raw ingredients: cells and lists

(1) We begin with simple cells, which can hold a number or a string.

(2) We form lists of cells, denoted by enclosing them in a box.

(3) We can mix cells and lists inside other lists, forming nested structures.

Tables as lists of lists?

(4) We might represent a table as a list of lists, where each inner list is a row of the same length.

(5) But if we insert a new cell in the second row, we likely wanted parallel insertions in the other rows, too.

Tables with the template rule

(6) We adopt a rule that the *first* element of the list, shown with darker shading, forms a “template” for subsequent elements.

This allows related sublists to be maintained always at the same length.

V1 V2 V3
id1 1 3 5
id2 2 4 6
1st element is list of 4 cells...
...so the other elements must also be lists of 4 cells
a
b 1 2 3
4
c 5 6 7 8
d 9 0

(7) But if the template element is a single cell, the rule is less strict. Then, we allow subsequent elements to be either cells *or* lists.

This lets us mix regular and ragged structures within the same list.

Higher dimensional structures

(8) Templates apply recursively: any structure inside a template also applies to those lists that follow it.

Here, the sublist containing V3 and V4 is reflected in an equivalent sublist in the other rows of the table.

The result is a column group.

(9) Recursive templates let us build higher dimensional structures.

Here, the first element of the top level list, is a list of four doubles. So subsequent elements must follow the same pattern.

The result is a 3D array.

Figure 2 – the building blocks of the “lish” data model

1.2 The “lish” data model

In an earlier paper (Hall et al., 2017) we have introduced the “lish” as an alternative model for spreadsheet-like data. The lish is a list-based model whose main characteristics are shown diagrammatically in Figure 2. In brief, a lish replaces the usual grid of cells with a list. Each element of a lish can be either a single cell or a further lish, so nesting is possible to any depth. This enables the lish to capture grouped and hierarchical structures more expressively than a grid, but a one-dimensional list is a poor abstraction of a two-dimensional table. So in order to express table- and array-like behaviours, the first element of every lish has a privileged status: it forms a “template”, defining a minimum structure with which subsequent elements must conform. For example, a template that is a list of five cells constrains further elements of that lish to be also lists of five cells. Such a lish could represent a table with five columns and any number of rows. By applying the template rule recursively, we can represent more elaborate or higher dimensional structures. In addition, we developed a typesetting algorithm which ensures that elements associated with a common template are aligned in an intuitive table-like way.

Here, we extend our earlier work by introducing calculations (after the manner of spreadsheet formulae) to the lish, with particular reference to the wide *vs.* long problem. Our goal is to bridge the cognitive gap between a visual layout that accords with the user's mental model of the data, and a machine-friendly layout that will facilitate calculation.

2. Case study

2.1 The scenario

We will illustrate an application of the lish using a small fictitious case study. A local chain of retail outlets has four branches within a town, denoted here simply as North, South, East and West. For budget monitoring purposes they have collected data on monthly footfall and revenue at each outlet. The chain is closed on Sundays, and the number of trading days per month has also been recorded. Using these data, they would like to perform a series of ad hoc calculations, e.g. for total monthly revenue, and revenue per trading day.

The data as entered into our prototype lish editor are shown in Figure 3. This layout is “untidy” – not only is it in wide form, but it has some freeform notes along the bottom of the table which the analyst decided to add, commenting on unusual circumstances in certain months. The grey shaded cells are our templates: they are either the first element in a lish, or part of a sublist that is itself a template. Figure 4 shows the same data in long form; in a normalised database, *month* would be a foreign key in this table, referring to a separate small table holding the trading days per month information.

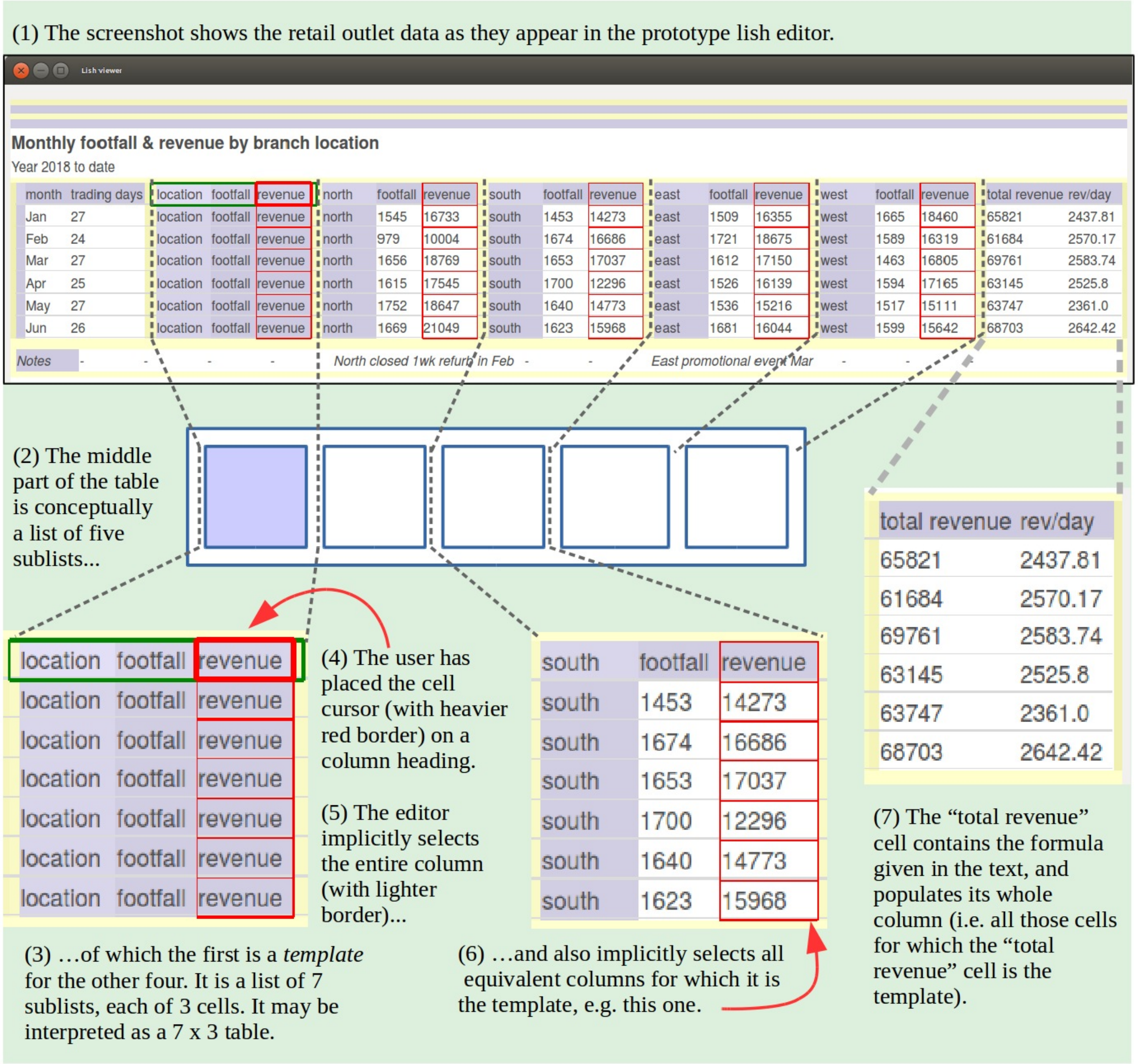
2.2 A grouped aggregation

Let us now consider the task of calculating the total revenue across the four branches, for each month. If using a code-based tool, we should prefer the data to be in the form of Figure 4. The calculation would then proceed e.g. using a `GROUP BY` query in SQL. In a spreadsheet, we could accommodate the same layout using `SUMIF`, but might prefer the wide layout of Figure 3, which plays better to the directness supported by the spreadsheet formula model. Instead of working at the level of the whole table, the user would enter in the first cell of the column a `SUM` formula referring to the individual January revenues for the four outlets, and then copy it down the column.

There is a problem with the wide spreadsheet layout, however: although it makes the form of the data more comprehensible, and formulae more direct, the approach just mentioned would not scale well to the addition of extra retail outlets. In general, spreadsheet formulae involving cells that are logically related but not physically adjacent on the sheet are more fiddly to construct and error-prone than those involving the selection of a single contiguous range.

To produce the monthly totals, we must specify two things: which cells are to be summed (here, all of the revenue cells), and across what dimensions to sum them (here, we want the row sums – other options might be the column sums, or the grand total). The lish can help on both counts.

First, the editor supports a cell selection method whereby navigating to any template cell implicitly selects *all* those cells for which it is a template, recursively. Go to a column label, and you implicitly select the column; go to the top left cell in a table, and you select the whole table; go to the very first cell in the outermost lish, and you "select all". The lish in Figure 3 has been arranged such that a single sublist within the main table represents the four smaller tables for the individual branch outlets, labelled North, South, East and West respectively. The 7 x 3 region of grey cells beginning with "location", near the left hand side, is the template for this sublist; it could be visualised as the base plane upon which identically shaped tables, one for each outlet, are to be stacked. So in the figure, selecting the column heading for "revenue" has selected that column, which in turn has selected the equivalent columns in all of the outlets. Although we have "wide" columns, this selection has unfolded a "long" column that was hidden in plain sight in their midst.



(1) The screenshot shows the retail outlet data as they appear in the prototype lish editor.

Monthly footfall & revenue by branch location

Year 2018 to date

month	trading days	location	footfall	revenue	north	footfall	revenue	south	footfall	revenue	east	footfall	revenue	west	footfall	revenue	total revenue	rev/day
Jan	27	location	footfall	revenue	north	1545	16733	south	1453	14273	east	1509	16355	west	1665	18460	65821	2437.81
Feb	24	location	footfall	revenue	north	979	10004	south	1674	16686	east	1721	18675	west	1589	16319	61684	2570.17
Mar	27	location	footfall	revenue	north	1656	18769	south	1653	17037	east	1612	17150	west	1463	16805	69761	2583.74
Apr	25	location	footfall	revenue	north	1615	17545	south	1700	12296	east	1526	16139	west	1594	17165	63145	2525.8
May	27	location	footfall	revenue	north	1752	18647	south	1640	14773	east	1536	15216	west	1517	15111	63747	2361.0
Jun	26	location	footfall	revenue	north	1669	21049	south	1623	15968	east	1681	16044	west	1599	15642	68703	2642.42
Notes	-	-	-	-	North closed 1wk refurb in Feb			-		-	East promotional event Mar			-	-	-		

(2) The middle part of the table is conceptually a list of five sublists...

location	footfall	revenue
location	footfall	revenue
location	footfall	revenue
location	footfall	revenue
location	footfall	revenue
location	footfall	revenue
location	footfall	revenue

(4) The user has placed the cell cursor (with heavier red border) on a column heading.

(5) The editor implicitly selects the entire column (with lighter border)...

south	footfall	revenue
south	1453	14273
south	1674	16686
south	1653	17037
south	1700	12296
south	1640	14773
south	1623	15968

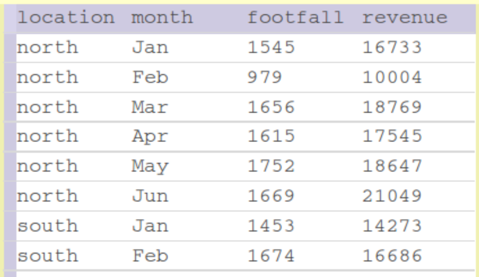
total revenue	rev/day
65821	2437.81
61684	2570.17
69761	2583.74
63145	2525.8
63747	2361.0
68703	2642.42

(7) The "total revenue" cell contains the formula given in the text, and populates its whole column (i.e. all those cells for which the "total revenue" cell is the template).

(3) …of which the first is a *template* for the other four. It is a list of 7 sublists, each of 3 cells. It may be interpreted as a 7 x 3 table.

(6) …and also implicitly selects all equivalent columns for which it is the template, e.g. this one.

Figure 3 – The retail outlet data in wide form.



location	month	footfall	revenue
north	Jan	1545	16733
north	Feb	979	10004
north	Mar	1656	18769
north	Apr	1615	17545
north	May	1752	18647
north	Jun	1669	21049
south	Jan	1453	14273
south	Feb	1674	16686

Figure 4 – The same data as Figure 3, but in long form (first few cases only)

Second, the editor can deduce from the structure, in just the same way, that the `total_revenue` cell labels a single column. So it "knows" that any summary statistics requested from this cell are to be calculated row-wise. The long column selected in the previous paragraph retains the memory of its fold points, so can be segmented correctly to produce the required monthly totals.

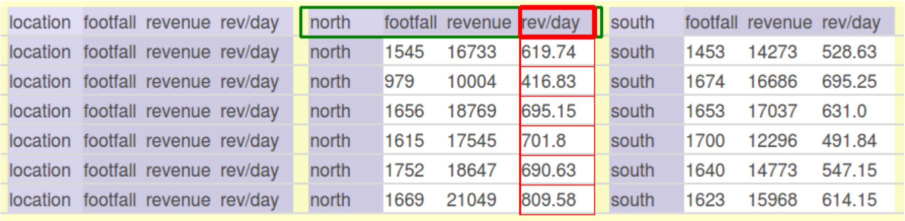
The formula for total revenue is `sum($location.revenue)` and is defined only once, in the head of the `total_revenue` column. From here, it populates the entire column. The dollar in the syntax denotes a labelled location in the lish (as opposed to a function name, like `sum`). The dot, in the expression `location.revenue`, avoids potential ambiguity by designating the intended `revenue` label to be the one within the `location` lish. That expression could have been typed verbatim, but was filled in automatically by the machine: the editor supports a spreadsheet-like interactive mode for building formulae, in which the user may navigate to a cell in the midst of formula construction and have that cell's label inserted. So both components of the specification – what to sum, and across what dimension to sum it – have been specified entirely visually, and in each case by pointing to a single cell. We don't need an explicit `GROUP BY` as with the long form. Nor do we have to reference individually all the columns to be summed, as with the wide form.

2.3 A binary operation

The lish in Figure 3 contains another calculated column (at the far right hand side) in which the revenue per day has been calculated for each month. The formula for this is `$total_revenue / $trading_days`. Both the numerator and denominator are one-dimensional lists of the same length, so the division is carried out pairwise between their corresponding elements. The result is a third list of the same length.

Now let us add a slightly more complicated calculation. Suppose we would like revenue per trading day, as before, but this time broken down by outlet as well as by month. To achieve this, the user first created an extra column in the original 7 x 3 sublist that is the template for the individual outlets. Since this sublist is a template, the editor automatically added a similar column for each individual outlet. Then, the user created a formula, `$revenue / $trading_days`, at the head of the new column in the template (the `location` qualifier for `revenue` is not needed this time, as unlike in subsection 2.2 the `revenue` column is local to the sublist that contains the formula). An excerpt from the resulting lish is shown in Figure 5.

Once again, pairwise divisions have been carried out between the elements of `revenue` and the elements of `trading_days`, but this time there is not a one-to-one correspondence between the two. As in the previous subsection, the machine "knows" from the nested structure that, for instance, the single value of 25 trading days in April, occurring in the denominator, is the counterpart to each of the four revenue values (one per location) for April, where they occur in the numerator. In the next section, we provide an overview of how such deductions are made.



location	footfall	revenue	rev/day	north	footfall	revenue	rev/day	south	footfall	revenue	rev/day
location	footfall	revenue	rev/day	north	1545	16733	619.74	south	1453	14273	528.63
location	footfall	revenue	rev/day	north	979	10004	416.83	south	1674	16686	695.25
location	footfall	revenue	rev/day	north	1656	18769	695.15	south	1653	17037	631.0
location	footfall	revenue	rev/day	north	1615	17545	701.8	south	1700	12296	491.84
location	footfall	revenue	rev/day	north	1752	18647	690.63	south	1640	14773	547.15
location	footfall	revenue	rev/day	north	1669	21049	809.58	south	1623	15968	614.15

Figure 5. An excerpt from the retail outlet data after adding a revenue per day *column*

3. A brief sketch of lish calculus

In the previous section, we saw examples of the machine using the structure of nested lists and templates which make up a particular lish to reason intelligently about the calculations required. The results accord intuitively with our notion of tabular structures, even though the primitive operations concerned are not defined on those structures: aggregation functions (like summation) are defined on one-dimensional arrays, and binary operators (like division) are defined on pairs of scalars.

Lish calculus is the name we give to our collection of algorithms that extend the definition of these operations to the lish. It draws heavily upon the notion of vectorised arithmetic, as defined by the R programming language (R Core Team, 2018). That language is itself based upon the usual mathematical definitions of vector and matrix operations, but with the novel addition of a "recycling rule", which provides greater flexibility by accommodating operands that may be vectors of different lengths.

In order to operate upon lishes, we modify vectorised arithmetic in two main ways. The first is fairly trivial: since the first element of each lish is a template and does not contain ordinary data, it must be omitted from any arithmetic. For example, if we sum a lish, we do not attempt to include the first element within the sum. The second modification is more complicated: we need a version of the recycling rule that will accommodate operands that may be not just of different lengths, but different depths as well. For example in subsection 2.3, `trading_days` was a one dimensional list of cells, whereas `revenue` was a list of lists of cells. The details of binary operators as defined over lishes are quite intricate, due to their recursive nature, and are beyond the scope of this paper. But the principle that underpins them is very simple: we perform pairwise matching between those sublists that were derived from the *same* template list. In our implementation, we cache the template with each sublist to improve the efficiency of this matching. A similar principle applies to aggregation functions, like `sum`: we compare those templates that appear within the lish being summed to those within the lish that is destined to hold the result. Templates that appear in the former but not in the latter identify which dimensions are to be summed over.

4. Discussion

The case study showed that the lish embodies more of the structure of the data than a flat grid, and that the machine can use this structure in ways that may help the user. But there is an obvious objection: when entering the data in lish form, did the user not have to do some extra work up front in order to define the structure in the first place? If so, perhaps on balance they are no better off than entering the data in a spreadsheet and performing the wide to long transformation explicitly. We are planning a user study which should provide some empirical evidence to this question. In the meantime, we make here a common sense argument based on relative costs and benefits.

Starting to build a model as a lish has some costs compared to building the same model in a spreadsheet. A blank lish is just a one-dimensional sequence of empty cells, in which tables only appear once the user starts enclosing sublists from among those cells. Our critical assumption is that the nested structure of the lish mirrors the way the user visualises the natural hierarchies in their data (*closeness of mapping*, again), so that forming a sublist requires minimal mental effort, as well as minimal mechanical effort (one key-press).

On the credit side, the lish repays the user with some early gratification once a basic structure is in place. We saw in the case study that formulae can be fewer and simpler than in a spreadsheet, and instantly populate all their relevant cells. A similar benefit occurs when cells are inserted or deleted in a template: all dependent tables are grown or shrunk to match accordingly. Just as the time-proven spreadsheet model gives the user a feeling of direct control by instantly updating *values*, so the lish complements that desirable attribute by instantly updating *structure*.

As an alternative to having the user explicitly lay out the structure, by grouping cells into sublists, we might consider *inferring* it by assuming certain spreadsheet conventions (such as column labels forming the first row of a table). This approach has been successfully applied before: e.g. by Hermans et al. (2011) to mapping data flow at varying granularity; by Hodnigg & Pinzger (2015), to parsing a worksheet into cognitive units; and by Kankuzi (2017), to abstracting formula calculations in narrative form. It has the advantage that it can be applied to a conventional spreadsheet, but would appear to be more challenging with higher dimensional structures such as those occurring in our case study. There is also the distinction that explicit structures like the lish produce a clearly deterministic result, whereas inference relies to some extent on the machine making a correct interpretation of flexible conventions.

A limitation of the lish is that it has no concept of a foreign key relationship. Since it is a nested structure it can express relationships of a form where one type of object “owns” a fixed or variable number of some other type of object. Hence, it can represent a limited class of 1:*n* relationships without the need for a foreign key as such. But it is not intended to be a replacement for the relational model itself. Its domain of application lies rather in a middle ground, as exemplified by our retail outlets case study, where multi-dimensional data or families of similar tables present scaleability problems for the spreadsheet, but a relational database would feel over-engineered.

5. Conclusions and future work

We have seen that there can be a tension between data representations that are easily comprehended by humans and those that are more conveniently processed by machine. A case in point is the choice between “wide” and “long” tabular data, often occurring where a time series is involved.

The “lish” provides a data model that supports the more human-oriented form for visualisation, while enabling the machine-oriented form to be accessed readily for calculation. We have shown in our case study of retail outlet data how an existing model may be expressed in lish form, and how the use of template cells and implicit data selections can assist the end user in constructing an analysis.

Our next step will be to conduct a user study to evaluate these ideas. First we will seek to verify our assumption that users comprehend tabular data in the way that we have assumed, and that the lish representation of “chunks within chunks” does indeed map closely to the user's mental model of a sequence of tables. We will also seek to evaluate the usability aspect. In particular, we would like to assess users' perceptions of the relative costs and benefits of constructing a model in lish form – and if necessary, what we might do to make this balance more favourable.

References

Bakke, E. and Karger, D.R. (2016) Expressive query construction through direct manipulation of nested relational results. Proceedings of the 2016 International Conference on Management of Data, 1377-1392.

Cervesato, I. (2007) Nexcel, a deductive spreadsheet. The Knowledge Engineering Review, 22(3), 221–236.

Chang, K. and Myers, B. (2016) Using and exploring hierarchical data in spreadsheets. Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, 2497-2507.

Green, T. and Petre, M. (1996) Usability analysis of visual programming environments: a ‘cognitive dimensions’ framework. Journal of Visual Languages & Computing, 7(2), 131–174.

Hall, A., Wermelinger, M., Hirst, T. and Phithakkitnukoon, S. (2017) Structuring spreadsheets with the “lish” data model. Proceedings of the 2017 European Spreadsheet Risk Interest Group (EuSpRIG) Conference.

Hawkins, T., Lemon, A. and Gibson, A. (2014) Introducing Morphit, a new type of spreadsheet technology. Proceedings of the 2014 European Spreadsheet Risk Interest Group (EuSpRIG) Conference.

Hermans, F., Pinzger, M., and Van Deursen, A. (2011) Supporting professional spreadsheet users by generating leveled dataflow diagrams. Proceedings of the 2011 33rd International Conference on Software Engineering (ICSE), 451–460.

Kankuzi, B. (2017) Dynamic Translation of Spreadsheet Formulas to Problem Domain Narratives. Proceedings of the 2017 Annual Workshop of the Psychology of Programming Interest Group (PPIG), 86-95.

Hodnigg K. and Pinzger, M. (2015) XVIZIT: Visualizing cognitive units in spreadsheets. Proceedings of the 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT), 210-214.

Mangano, N., Ossher, H., Simmonds, I., Callery, M., Desmond, M. and Krasikov, S. (2011) Blending freeform and managed information in tables. Proceedings of the 2011 33rd International Conference on Software Engineering (ICSE), 840-843.

Mount, J. and Zumel, N. (2017) Coordinatized data: a fluid data specification. Technical report, Win-Vector LLC. Available at: http://www.win-vector.com/blog/2017/03/coordinatized-data-a-fluid-data-specification/

R Core Team (2018) R Language Definition v 3.5.0, section 3.3, “Elementary arithmetic operations”. Available at: https://cran.r-project.org/doc/manuals/r-release/R-lang.html

Scaffidi, C. (2016) The impact of human-centric design on the adoption of information systems: A case study of the spreadsheet. Proceedings of the 2016 11th Iberian Conference on Information Systems and Technologies (CISTI).

Wickham, H. (2014) Tidy data. Journal of Statistical Software, 59(10).