Where-Provenance for Bidirectional Editing in Spreadsheets

Jack Williams
Microsoft Research
jack.williams@microsoft.com

Andrew D. Gordon
Microsoft Research
& University of Edinburgh
adg@microsoft.com

***Abstract*—We explore the idea of adding bidirectionality to spreadsheet formulas, so that editing the output can directly affect the input. We introduce portals: a portal is a value paired with its where-provenance, that is, one or more links to its origin. When a portal is the result of a formula in a cell, that cell inherits the capability to edit the locations described by the provenance of the portal. The simplicity of portals makes them amenable to implementation in an existing spreadsheet system. We analyse the list of functions provided by a widely-used commercial spreadsheet system and find that many frequently used functions work with portals with no modification.**

***Index Terms*—spreadsheets, bidirectional editing, provenance**

I. Introduction

In a spreadsheet we take for granted that formulas compute values, and those values cannot be edited because they are computed by the formula. In this paper we explore the idea of adding bidirectionality to spreadsheet formulas, so that editing the output can directly affect the input.

Consider the spreadsheet in Figure 1 where a user writes the formula SORT(UNIQUE(B2:B6)) in D2 and the computed values *spill* into the cells below. The formula reveals that the data is messy with different spellings of the same location.

The user intends to normalise the capitalisations and cell D3 intuitively represents the values in cells B5 and B6. Wouldn't it delight the user if they could fix both mistakes at once simply by changing "england" to "England" in the output of the formula? We tell users that they cannot edit the output of a formula, only its inputs. But with portals, they can!

A *portal* is a value paired with its *where-provenance* [3], [11], that is, one or more links to its origin in the spreadsheet. Users can edit a portal and the change is propagated back to the origin. In our example, the value in cell D3 is the portal, written $\langle\{\mathsf{B5}, \mathsf{B6}\}, \text{“england”}\rangle$, which consists of the value "england" annotated with the where-provenance $\{\mathsf{B5}, \mathsf{B6}\}$. When a portal is the result of a formula in a cell, that cell inherits the capability to edit the locations described by the provenance of the portal. So editing the output at D3 to be "England", does the same to the inputs B5 and B6.

Spreadsheets are touted as a canonical example of a user interface built using functional reactive programming (FRP) [4], [16], [19] or Model-View-Update (MVU), and yet, until now, direct control over the spreadsheet interface was out of reach for users. Our work shows that ideas going back to the view-update problem for databases [17], [20] can bridge the host interface capabilities and the hosted formula language, transforming how users interact with spreadsheets.

@	A	B	C	D	E
1	Name	Location			
2	Francesca	England		Brazil	
3	João	Brazil		england	
4	Ismaïla	senegal		England	
5	Kat	england		senegal	
6	Will	england			

Fig. 1. Normalising Values Using Portals and Formulas

Prior approaches to bidirectional editing often focus on the expressiveness of the updates, rather than how it can be integrated into an existing system. The central idea of this paper is to show that portals, or where-provenance, provides a remarkably simple but effective way to support bidirectional editing in spreadsheets. To support the central idea we make the following contributions:

- We validate the utility of portals by example. Whilst portals are technically modest, we show that they are capable of solving problems faced by hundreds of thousands of spreadsheet users (Section II-B and Section II-C).
- We add portals to a spreadsheet calculus and prove that portals satisfy the core property of where-provenance (Theorem 1).
- We evaluate the viability of where-provenance based bidirectional transformation in spreadsheets. First, we measure all Excel spreadsheet functions that are conceptually capable of propagating where-provenance. Second, we examine the source code of Calc.ts[1] to categorise which functions support our portal technique with no additional modification, some modification, or significant modification. From the Enron spreadsheet corpus [29],

[1]Calc.ts is the client-side calculation engine used by Excel for the Web (https://www.microsoft.com/en-us/garage/wall-of-fame/calc-ts-in-excel-for-the-web/). Screenshots are from PROTO, a TypeScript-based spreadsheet research prototype.

978–1–6654–4592–4/21/$31.00 © 2021 IEEE

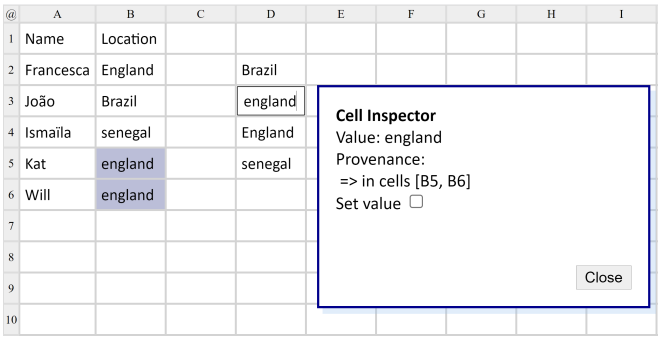

Fig. 2. PROTO Implementation with Cell Inspector

five of the fifteen most used functions support portals with no modification.

In summary, compared to the state of the art for bidirectional spreadsheets by Macedo et al. [33], our approach is simpler to explain to the user and requires less work per-function to implement, while still covering many use-cases. A detailed discussion of provenance and lenses is given in Section VI.

The commercial adoption of array formulas, partly inspired by the research proposal of arrays in cells [8], exemplifies the idea that spreadsheets are code [28]. In tandem, there is much recent research on spreadsheets as programming languages [5], [9], [10], [37], [40], [44]–[46] or databases [6]. This paper presents and formalises the new programming language concept of spreadsheet portals.

II. A NEW UNIVERSE OF SPREADSHEETS

Bidirectional program manipulation has been explored in different guises including lenses [23], [33], output-directed program manipulation [2], [15], [27], [36], and program synthesis [25], [26]. We implement a spreadsheet-centric approach to bidirectional editing using where-provenance that is capable of transforming how users interact with spreadsheets across many distinct tasks. We validate the versatility of portals with three examples. The first two examples are implemented in PROTO[1] and demonstrate how our technique tightly integrates into the spreadsheet paradigm. The last example extends beyond the features of PROTO and is illustrated using mock-ups. Our examples directly draw on real tasks performed within spreadsheets; Section II-B and Section II-C feature problems addressed in spreadsheet video tutorials with hundreds of thousands of views.

A. Example: Data Cleaning

Commercial spreadsheets have recently undergone a radical transformation that allows a user to write a formula to manipulate and present ranges of data. When a cell's formula computes an array, that array *spills* out of the cell and into the surrounding grid. [2] [3] For a detailed presentation of spilled arrays see the work by [45]. Where previously a user would have to invoke a command to sort or filter a range in-place, now they just write a formula. Formulas and spilled arrays can replace much of the ad-hoc functionality used to process grid data, but without portals they lack the capability to update the input data from the output of a transformation.

Figure 2 presents the same example from Figure 1, but additionally, illustrates how we use where-provenance to explain the effect of an update. The small floating blue rectangle in Figure 1 is the minimised *cell inspector*; when the inspector is clicked it expands and presents information about the active cell, as illustrated in Figure 2. The inspector reveals that the value in D3 is a portal with provenance that links the value to the cells B5 and B6. Portals implement aggregated where-provenance [41], where an annotated value can refer to a set of locations. When UNIQUE finds a pair of portals that point to the same value, the locations are aggregated. In this example, portals $\langle\{\mathsf{B5}\}, \text{“england”}\rangle$ and $\langle\{\mathsf{B6}\}, \text{“england”}\rangle$ are inputs to UNIQUE, and the portal $\langle\{\mathsf{B5}, \mathsf{B6}\}, \text{“england”}\rangle$ is the aggregated value in the output. Aggregated where-provenance lets the user update multiple cells simultaneously. When the user types “England” in cell D3, both B5 and B6 will be updated. The edits applied to each source location are normal spreadsheet edits and trigger recalculation of formula $=$ SORT(UNIQUE(B2:B6)) in D2. There are now three unique values: “brazil”, “England”, and “senegal”, that spill into the cells below. The cell D3 will change to “England”, the cell D4 will change to “senegal”, and the cell D5 will become blank.

A feature of PROTO is that when the inspector is maximised and the active cell contains a portal, the cells described by the portal's provenance are highlighted using coloured foils. Highlighting cells is a familiar pattern to spreadsheets users, typically used to indicate the cells referenced by a formula. In our setting, the foils indicate the origin cells for values in the output of a formula. The coloured foils present a clear explanation of the effect of the bidirectional edit: when a user makes an edit to the active cell, the edit is uniformly applied to all highlighted cells.

B. Example: User Interface Controls

Another aspect of spreadsheets that makes them popular is their capability as a data presentation tool. A single spreadsheet can combine data, calculation, and formatting to build a presentation. With the addition of portals we provide the capability to build interactive presentations that exploit many existing spreadsheet features. Aspects of this example are derived from a spreadsheet video tutorial[4] with over two hundred thousand views; the video shows how to bind spreadsheet values to GUI controls such as buttons.

Figure 3 presents an example calculation (left) spanning range A1:E6, and derived presentational views (middle and right) both spanning range J14:K18. The literal value labelled by (1) is a conversion rate, and the literal value labelled by (2)

[2]https://aka.ms/excel-dynamic-arrays
[3]https://support.google.com/docs/answer/6208276?hl=en

[4]https://web.archive.org/web/20210216032419if_/https://www.youtube.com/watch?v=qMQ0UB6WyKQ

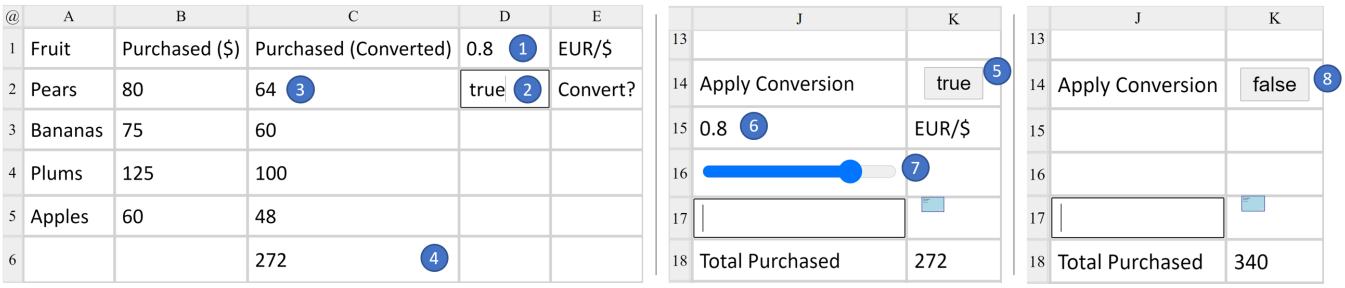

Fig. 3. Calculation and Interactive GUI Controls: Calculation (left), GUI controls (middle and right)

is used to enable a currency conversion. The cell labelled by 3 holds the formula $= \mathsf{B2{:}B5} * \mathsf{IF}(\mathsf{D2}, \mathsf{D1}, 1)$. The formula multiplies each cell in the range $\mathsf{B2{:}B5}$ by the conversion rate when enabled, or 1 otherwise. The output is an array of converted values which spill down. The cell labelled by 4 holds the formula $= \mathsf{SUM}(\mathsf{C2{:}C5})$.

Suppose the user wants to construct an interactive summary of the data, as depicted by the grid in the middle of Figure 3. Without portals, the summary must span the same inputs used by the calculation (D1 and D2) otherwise there is no mechanism to update the inputs, hence the logic and visualisation are coupled. Systems such as Excel allow users to create floating controls bound to cells, permitting some separation, however the controls are not first-class values and therefore the interface cannot be controlled formulas. With portals and formatting, a user can create a decoupled and interactive visualisation. We place the visualisation controls in a separate range on the same spreadsheet, but they could also be on a separate spreadsheet.

The cell labelled by 5 holds the formula $=$ D2 and is formatted using a *button format*. A button format renders the cell as a button, and when the button is clicked, an edit is made to the formatted cell. For example, clicking the button in the depicted state is equivalent to writing false into the cell.

The cell labelled by 6 holds the formula $= \mathsf{IF}(\mathsf{D2}, \mathsf{D1:E1}, \text{""})$. The cell labelled by 7 holds the formula $= \mathsf{IF}(\mathsf{D2}, \mathsf{SLIDER}(\mathsf{D1}, 0, 1, 0.01), \text{""})$. The formula $\mathsf{SLIDER}(\mathsf{D1}, 0, 1, 0.01)$ takes the portal obtained by evaluating D1 and labels the portal with a *slider format*, with minimum 0, maximum 1, and step 0.01.

When the user moves the slider the format writes numbers to the cell, and those edits are then propagated back to D1 via the portal. When the user clicks the button, as shown by 8, false is propagated back to D2 via the portal. The formatted controls are oblivious to portals, and the effect of the slider controlling D1 is achieved through composition of edits. First, the control applies an edit to the formatted cell, and then the portal propagates that edit back. After clicking the button, the spreadsheet recalculates and the formulas labelled 6 and 7 evaluate to blank values, hiding the controls for updating the rate. The user has created a interactive visualisation with conditional controls, using only formulas and formatting.

C. Example: Lightweight Database

Spreadsheets are frequently used as lightweight databases. This example is derived from a spreadsheet video tutorial[5] with over one million views; the video shows how to build an inventory manager using a spreadsheet. The manager includes a small form that lets the user add new records to a table in the sheet. Implementing the form requires additional code written in Visual Basic. We show how portals and a single formula can implement the same behaviour.

Spreadsheets such as Excel allow *tables* in the grid. When a table is created a name is defined that spans the table range and users can refer to this name in formulas. The following spreadsheet called Data (we display the sheet name in the top left of the grid) includes a table in the range $\mathsf{A1{:}C4}$; if the table is named Sales, then the formula =Sales evaluates to the range $\mathsf{A1{:}C4}$. A table will *expand* as items are added to the cells immediately below the table. When a user enters a value into cell A5 the table's range will expand, including the range associated with name Sales.

Data	A	B	C	D
1	**Customer**	**Product**	**Qty**	
2	Stuff Ltd.	Widget	13	
3	Robots Inc.	Gizmo	10	
4	Items Plc.	Dongle	15	
5				
6				

Using portals, any region of a spreadsheet can implement a form for adding rows. In a separate sheet called Form a user can write:

Form	A	B	C
1	Customer	Product	Qty
2	$= \mathsf{OFFSET}(\mathsf{Sales}, \mathsf{ROWS}(\mathsf{Sales}), 0, 1)$		

The formula $\mathsf{OFFSET}(\mathsf{Sales}, \mathsf{ROWS}(\mathsf{Sales}), 0, 1)$ computes the range for the row immediately below the table. In our example the formula evaluates to the range Data!A5:C5 which will spill portals into the grid:

Form	A	B	C
1	Customer	Product	Qty
2	⟨{Data!A5}, “”⟩	⟨{Data!B5}, “”⟩	⟨{Data!C5}, “”⟩

[5]https://web.archive.org/web/20201024233423/https://www.youtube.com/watch?v=-1N0L-FDWCs

ColumnName	N	::=	$\mathrm{A} \mid \ldots \mid \mathrm{Z} \mid \mathrm{AA} \mid \mathrm{AB} \mid \ldots$
	m, n	$\in$	$\mathbb{N}_1$
Address	a, b	::=	Nm
Range	r	::=	$a_1{:}a_2$
Literal	L	::=	$c \mid \mathsf{ERR} \mid \{L_{i,j}{}^{i\in 1..m, j\in 1..n}\}$
Formula	F	::=	$L \mid r \mid \mathsf{IF}(F_1, F_2, F_3) \mid f_n(F_i{}^{i\in 1..n})$
Sheet	$\mathcal{S}$	::=	$[a_i \mapsto F_i{}^{i\in 1..n}]$
Access		=	$\text{Address} \cup (\mathbb{N}_1 \times \mathbb{N}_1)$
Path	π	$\in$	Access^* (lists of Access)
PathSet	α	$\subseteq$	Path
PortalValue	$\hat{P}$	::=	$c \mid \mathsf{ERR} \mid \{P_{i,j}{}^{i\in 1..m, j\in 1..n}\}$
Portal	P	::=	$\langle \alpha, \hat{P} \rangle$
Scalar	S	::=	$\epsilon \mid c \mid \mathsf{ERR}$
ComputedValue	V, W	::=	$S \mid \{V_{i,j}{}^{i\in 1..m, j\in 1..n}\} \mid P$
Grid	γ	::=	$[a_i \mapsto V_i{}^{i\in 1..n}]$

Fig. 4. Syntax for Core Calculus

When the user edits the cells containing the portals, the edits will be redirect to the end of the table, the table will expand to include the new content, and the formula in A2 will evaluate to the next empty row. The table will expand as soon as any portal is updated; for best results, the user would manually trigger recalculation once all fields are entered. In a similar way, we can write formulas to build a form to lookup a row by customer name and edit the contents.

III. A Core Calculus with Portals

In this section we present a new spreadsheet calculus with *portals* by augmenting and extending the calculus of [45]. We show that portals implement where-provenance.

A. Syntax

The syntax of the core calculus is presented in Figure 4.

Let a and b range over addresses written in A1-style. An address is composed from a column N and row m, where a column index is a base-26 numeral written using the letters [A-Z], and a row index is a decimal numeral. We number columns left-to-right and rows top-to-bottom, so that the origin cell A1 is at the top-left of the grid. We use the terms *address* and *cell* as synonyms. Let r denote a range which is a pair of addresses $a_1{:}a_2$. A range represents a rectangle in the spreadsheet. We assume that ranges are normalised such that a_1 is the top-left corner and a_2 is the bottom-right corner. Let L range over literals. A literal is either a constant c, an error literal ERR, or an array literal $\{L_{i,j}{}^{i\in 1..m, j\in 1..n}\}$. If E is a piece of syntax, we use $E^{i\in 1..n}$ to denote n expressions from syntax class E. Let F range over formulas. A formula is either a literal, a range, a conditional expression, or a function application. Let f range over operators such as $+$, and worksheet functions such as SORT. We write f_n when function f has arity n. Let $\mathcal{S}$ range over sheets. A sheet is a finite, hence partial, map from addresses to formulas. Spreadsheet implementations make an internal distinction between cells that contain literals and cells that contain formulas. In our calculus we do not make these two classes disjoint, but we do refer to an address that maps to a formula consisting of a literal as a *literal address*, and otherwise refer to the address as a *formula address*.

We now describe the first significant extension to the core calculus. Let Access be the set of *access elements*, where an element is an address a or pair of indices (m, n). Let Path be the set of lists of access elements. Intuitively, a path π describes some part of a literal in a sheet. Consider the spreadsheet:

	A	B	C
1	10	$\{5, 6\}$	$=$A1 $*$ B1

Path A1 describes literal 10, path B1 describes literal $\{5, 6\}$, and path B1.$(1, 2)$ describes literal 6. We use nil to represent the empty path and $\pi.\pi'$ to represent the concatenation of two paths. Concatenation is associative with nil as the identity element. Let α range over sets of paths.

Our last class of syntax is concerned with evaluation, and where we introduce portals. Let P range over portals and $\hat{P}$ range over portal values. A portal is a pair of a path set and a portal value; a portal value is a constant, error, or array of portals. Portals are literals that are recursively annotated with path sets, and a portal is never immediately nested within another portal. Let S range over scalar values, which can be blank ϵ, a constant c, or an error ERR. Let V range over computed values, which can be a scalar, an array of computed values, or a portal. Let γ range over grids. A grid is a partial and finite map from addresses to computed values.

Our presentation of literals and computed values is explicit. Literals and formulas are things users write, hence literals do not include blank (ϵ). Computed values are computed by formulas, or returned by dereferencing an address. It is possible to simplify our presentation by assuming all computed values are portals, where a scalar is a portal with an empty path set. We do not take this approach to be faithful to our implementation where there is a distinction between computed values with provenance, and computed values without. We refer to computed values as values for short.

B. Operational Semantics

The operational semantics for the core calculus is presented in Figure 5. The extension to the calculus of [45] is minor; we highlight the changes to support portals using a grey box.

Write $\mathcal{S} \vdash F \Downarrow V$ to show that in sheet $\mathcal{S}$ formula F evaluates to value V. A literal evaluates to itself. A conditional expression evaluates the condition and extracts a non-portal value using enter which is defined in Figure 5; the corresponding branch is then evaluated. Our semantics is defined when the condition evaluates to a boolean, while spreadsheet implementations typically used relaxed notions of *truthy* and *falsy*. We omit these definitions because they are orthogonal to portals. A function application evaluates each argument and then applies the portal-aware semantics of f, written $[\![\cdot]\!]^{\mathsf{P}}$ and defined in Figure 5, to the arguments. A

Formula evaluation $\boxed{\mathcal{S} \vdash F \Downarrow V}$

$$\frac{}{\mathcal{S} \vdash L \Downarrow L}$$

$$\frac{\mathcal{S} \vdash F_1 \Downarrow V_1 \qquad \mathsf{enter}(V_1) = \mathsf{TRUE} \qquad \mathcal{S} \vdash F_2 \Downarrow V_2}{\mathcal{S} \vdash \mathsf{IF}(F_1, F_2, F_2) \Downarrow V_2}$$

$$\frac{\mathcal{S} \vdash F_1 \Downarrow V_1 \qquad \mathsf{enter}(V_1) = \mathsf{FALSE} \qquad \mathcal{S} \vdash F_3 \Downarrow V_3}{\mathcal{S} \vdash \mathsf{IF}(F_1, F_2, F_2) \Downarrow V_3}$$

$$\frac{\forall i \in 1..n.\ \mathcal{S} \vdash F_i \Downarrow V_i \qquad [\![f_n]\!]^{\mathsf{p}}(V_i^{\,i \in 1..n}) = V}{\mathcal{S} \vdash f_n(F_i^{\,i \in 1..n}) \Downarrow V}$$

$$\frac{\mathcal{S} \vdash a\,!\,V}{\mathcal{S} \vdash a{:}a \Downarrow V} \qquad \frac{a_1 \neq a_2 \qquad \mathsf{size}(a_1{:}a_2) = (m, n) \qquad \forall i \in 1..m, j \in 1..n.\ \mathcal{S} \vdash (a_1 + (i,j))\,!\,V_{i,j}}{\mathcal{S} \vdash a_1{:}a_2 \Downarrow \{V_{i,j}{}^{\,i \in 1..m, j \in 1..n}\}}$$

Portal-aware interpretation $\boxed{[\![f_n]\!]^{\mathsf{p}}}$

$$[\![f_n]\!]^{\mathsf{p}} = \lambda x_i^{\,i \in 1..n}.\ [\![f_n]\!](\mathsf{enter}(x_i^{\,i \in 1..n}))$$

Address dereferencing $\boxed{\mathcal{S} \vdash a\,!\,V}$

$$\frac{\mathcal{S}(a) = L}{\mathcal{S} \vdash a\,!\,\mathsf{path}(a, L)}$$

$$\frac{\mathcal{S}(a) = F \qquad F \neq L \qquad \mathcal{S} \vdash F \Downarrow V}{\mathcal{S} \vdash a\,!\,V} \qquad \frac{a \notin \mathrm{dom}(\mathcal{S})}{\mathcal{S} \vdash a\,!\,\epsilon}$$

Sheet evaluation $\boxed{\mathcal{S} \Downarrow \gamma}$

$$\mathcal{S} \Downarrow \gamma \quad \stackrel{\mathsf{def}}{=} \quad \forall a \in \mathrm{dom}(\mathcal{S}).\ \mathcal{S} \vdash a\,!\,\gamma(a)$$

Functions and operators

$\mathsf{size} : \mathrm{Range} \to \mathbb{N}_1 \times \mathbb{N}_1$
$\mathsf{size}(N_1 m_1{:}N_2 m_2) = (m_2 - m_1 + 1, N_2 - N_1 + 1)$

$(+) : \mathrm{Address} \times (\mathbb{N}_1 \times \mathbb{N}_1) \to \mathrm{Address}$
$Nm + (i,j) = (N + j - 1)(m + i - 1)$

$\mathsf{path} : \mathsf{Path} \times \mathsf{Literal} \to \mathsf{Portal}$
$\mathsf{path}(\pi, \{L_{i,j}{}^{\,i \in 1..m, j \in 1..n}\}) =$
$\quad \langle \{\pi\}, \{\mathsf{path}(\pi.(i,j), L_{i,j})^{\,i \in 1..m, j \in 1..n}\} \rangle$
$\mathsf{path}(\pi, L) = \langle \{\pi\}, L \rangle$ if L scalar

$\mathsf{enter} : \mathrm{ComputedValue} \to \mathrm{ComputedValue} \setminus \mathrm{Portal}$
$\mathsf{enter}(\langle \alpha, \hat{P} \rangle) = \hat{P}$
$\mathsf{enter}(V) = V$ if $V \neq P$

$(\sqcup) : (\mathrm{Portal}^2 \rightharpoonup \mathrm{Portal}) \cap (\mathrm{PortalValue}^2 \rightharpoonup \mathrm{PortalValue})$
$c \sqcup c = c$
$\mathsf{ERR} \sqcup \mathsf{ERR} = \mathsf{ERR}$
$\{\hat{P}_{i,j}{}^{\,i \in 1..m, j \in 1..n}\} \sqcup \{\hat{P'}_{i,j}{}^{\,i \in 1..m, j \in 1..n}\} =$
$\quad \{\hat{P''}_{i,j}{}^{\,i \in 1..m, j \in 1..n}\}$
$\quad$ if for all $i \in 1..m, j \in 1..n. \hat{P}_{i,j} \sqcup \hat{P'}_{i,j} = \hat{P''}_{i,j}$
$\langle \alpha_1, \hat{P}_1 \rangle \sqcup \langle \alpha_2, \hat{P}_2 \rangle = \langle \alpha_1 \cup \alpha_2, \hat{P}_3 \rangle \quad$ if $\hat{P}_1 \sqcup \hat{P}_2 = \hat{P}_3$

Fig. 5. Operational Semantics for Core Calculus

singleton range evaluates by dereferencing the address. A non-singleton range evaluates by dereferencing each address in the range and constructing an array of the resulting values.

Interpretation $[\![f_n]\!]^{\mathsf{p}}$ is defined by *entering* any argument portals to extract the portal value, and then appealing to the portal-oblivious interpretation $[\![f_n]\!]$ that defines the underlying semantics. The effect of $[\![f_n]\!]^{\mathsf{p}}$ is to forget path information for scalar arguments but preserve path information within array arguments. The assumption is that functions derive new data from scalar values, rather than copying, but may permute, filter, or duplicate array elements. We discuss array functions in Section III-C.

Write $\mathcal{S} \vdash a\,!\,V$ to show that in sheet $\mathcal{S}$ address a dereferences to value V. There are three classifications for an address with respect to a sheet: an address may be a literal address, a formula address, or missing. Each classification has a single rule. A literal address dereferences to a portal. The operator path traverses the literal and promotes each sub-literal to a portal with the relative path. For example, given $\mathcal{S}$ below,

	A	B	C
1	10	$\{5, 6\}$	$=$A1 $*$ B1

then $\mathcal{S} \vdash \mathsf{B1}\,!\,\langle\{\mathsf{B1}\}, \{\langle\{\mathsf{B1}.(1.1)\}, 5\rangle, \langle\{\mathsf{B1}.(1,2)\}, 6\rangle\}\rangle$. A formula address dereferences by evaluating the bound formula. A missing address dereferences to the blank value ϵ. A portal value is only constructed when we copy a literal from the current sheet, rather than computing or synthesizing a value. As a consequence, a portal is always able to point to *where* in the sheet its literal originated.

Write $\mathcal{S} \Downarrow \gamma$ to show that sheet $\mathcal{S}$ evaluates to grid γ. A sheet evaluates to a grid if every address in the sheet dereferences to the corresponding value in the grid.

C. Sort, Filter, and Unique

A motivation for portals is that an existing language needs only minor modifications to support portals. The implementation of most functions is unchanged, only using $[\![\cdot]\!]^{\mathsf{p}}$ to uniformly remove portals. We assume our calculus supports the array functions SORT, FILTER, and UNIQUE that are available in spreadsheets. We omit their definitions but outline their necessary extensions to support portals. In Section IV-A we present a full account of the spreadsheet functions that are compatible with portals.

a) Sort: To implement SORT we only need to modify value comparison to accommodate portals. Portal path sets have no effect on ordering and can be removed before the comparison; wherever previously we used a comparison function $[\![\mathsf{compare}_2]\!]$, we now use $[\![\mathsf{compare}_2]\!]^{\mathsf{p}}$.

b) Filter: To implement FILTER we only need to modify the predicate to accommodate portals. Most spreadsheets lack first-class functions and therefore FILTER is implemented by passing a boolean column vector that specifies the rows to retain. The only necessary change is to ignore portals within the column vector, which is achieved using $[\![\cdot]\!]^{\mathsf{p}}$. Assume operator isTrue_1, where $[\![\mathsf{isTrue}_1]\!] = \lambda x.x = \mathsf{TRUE}$; wherever previously we used $[\![\mathsf{isTrue}_1]\!]$, we now use $[\![\mathsf{isTrue}_1]\!]^{\mathsf{p}}$.

$$|\cdot| : \text{Portal} \cup \text{PortalValue} \to \text{Literal}$$
$$|\langle \alpha, \hat{P} \rangle| = |\hat{P}|$$
$$|\{{P_{i,j}}^{i \in 1..m, j \in 1..n}\}| = \{|P_{i,j}|^{i \in 1..m, j \in 1..n}\}$$
$$|\hat{P}| = \hat{P} \text{ if } \hat{P} \text{ scalar}$$

$$\mathsf{occur} : \text{Sheet} \cup \text{ComputedValue} \to \mathcal{P}(\text{Path} \times \text{Literal})$$
$$\mathsf{occur}(\mathcal{S}) = \bigcup\{\mathsf{occur}(\mathsf{path}(a, L)) \mid a \in \mathrm{dom}(\mathcal{S}), \mathcal{S}(a) = L\}$$
$$\mathsf{occur}(\langle \alpha, \hat{P} \rangle) = \{(\pi, |\hat{P}|) \mid \pi \in \alpha\} \cup \mathsf{occur}(\hat{P})$$
$$\mathsf{occur}(\{{V_{i,j}}^{i \in 1..m, j \in 1..n}\}) = \bigcup\{\mathsf{occur}(V_{i,j}) \mid \forall i \leq m, j \leq n\}$$
$$\mathsf{occur}(S) = \varnothing$$

Fig. 6. Where-provenance Definitions

c) Unique: We could implement UNIQUE similarly to SORT by interpreting equality using $[\![\cdot]\!]^{\mathsf{P}}$, but while correct, this would remove the capability to aggregate equal literals and update them simultaneously. Each portal in the output would only describe one location from the set of locations with equal literals. Instead, we want to *aggregate* provenance. [41] defines a *Unique* function on where-provenance annotated bags that returns a set of tuples by removing duplicates after merging annotations. Our goal is to construct a UNIQUE function like [41], although we map arrays to arrays. Write $P \sqcup P'$ and $\hat{P} \sqcup \hat{P}'$ to be the partial *join operator* on portals and portal values; the join aggregates provenance for equivalent inputs modulo provenance. We tacitly assume that for non-portal value V, $P \sqcup V$ and $V \sqcup P$ promotes V to a portal where the path set is $\varnothing$. Two portals are equal iff $P \sqcup P'$ is defined. To implement UNIQUE we use $\sqcup$ to determine the equality of two portals, and the result of $\sqcup$ is used in the output. By using $\sqcup$, the combined provenance of two equal portals appears in the output.

D. Portals as Where-provenance

To a user, a portal represents an "exact copy" of one or more input literals; in this section we formalise this intuition by showing that portals implement where-provenance.

Where-provenance was first introduced by [11] in the context of databases. Elements in an output relation are annotated with locations from the input relation to identify where the element originated from. Where-provenance has since been adapted to programming languages. [3] develop a calculus of provenance for a functional language TML which can be instantiated to implement where-provenance. A TML program evaluates and produces a *trace* which can be interpreted with where-provenance. [21] extend Links with *language-integrated-provenance*. Query expressions for a database can be augmented to dynamically propagate where-provenance. Our presentation draws on both systems. Like [3], we allow provenance to annotate arbitrary literals in the program context rather than databases. Like [21], we dynamically propagate where-provenance rather than producing a trace. Unlike both systems, we support aggregated where-provenance.

Across all systems the key result of where-provenance is analogous, and when stated in the context of spreadsheets says that the annotated literals in an evaluated formula are a subset of the annotated literals from the input sheet.

(A) No Additional Changes
SORT, SORTBY, FILTER, IF, IFERROR, IFNA, IFS, OFFSET, TRANSPOSE, CHOOSE, HLOOKUP, VLOOKUP, LOOKUP, SWITCH, XLOOKUP, INDEX, INDIRECT

(B) Minor Changes
UNIQUE, MIN, MINA, MINIFS, MAX, MAXA, MAXIFS, MODE, MEDIAN, MODE.SNGL, MODE.MULT, DMIN, DMAX, AGGREGATE, SMALL, LARGE, SUBTOTAL

(C) Significant Semantic Changes
LEFT, RIGHT, TRIM, MID, CONCAT, REPLACE, REPT, SUBSTITUTE

Fig. 7. Spreadsheet Functions that Interact with Where-provenance

Figure 6 presents the definitions for where-provenance. Write $|\cdot|$ for the portal erasure operator. Write occur, analogous to the definition by [3], for the operator that extracts (π, L) provenance tuples from a value or sheet.

We now state our result for where-provenance: the set of provenance tuples $\mathsf{occur}(V)$ obtained by evaluating any address in sheet $\mathcal{S}$ is a subset of the provenance tuples $\mathsf{occur}(\mathcal{S})$ for $\mathcal{S}$.

Theorem 1 (Where provenance)**.**
If $\mathcal{S} \vdash a\,!\,V$ *then* $\mathsf{occur}(V) \subseteq \mathsf{occur}(\mathcal{S})$.

We omit the details of the full proof, which is an induction on the derivation of $\mathcal{S} \vdash a\,!\,V$.

IV. Evaluating Where-provenance

In this section we evaluate where-provenance as a practical basis for implementing bidirectional spreadsheets. Our operational semantics demonstrate that only a minor change to a formula language is required to support portals, but we omit detailed discussion about the built-in functions which form a core part of spreadsheets. Here, we analyse the full breadth of spreadsheet functions[6] to measure how many could propagate where-provenance, and hence support bidirectional updates, and what modifications to each function are required.

A. Where-provenance Supported Spreadsheet Functions

For our evaluation we first classify the functions that could, in theory, propagate where-provenance. Next, we analyse the spreadsheet implementation Calc.ts to evaluate the extent to which the functions we identify must be modified to support our approach. We illustrate the need for a conceptual and pragmatic analysis by example. There are functions such as MAX, whose mathematical definition, or implementation in another language such as Haskell or Python, would work with where-provenance. However, when we analyse the implementation of MAX in Calc.ts, we find that the aggregated maximum value is

[6]https://support.microsoft.com/en-us/office/excel-functions-alphabetical-b3944572-255d-4efb-bb96-c6d90033e188

specialised to a number, rather than being the "largest" value according to a generic comparator. Given our semantics in Section III, this would result in the erasure of provenance. In contrast, the implementation of TRANSPOSE propagates where-provenance with no modification.

Figure 7 presents the list of functions available in spreadsheet application Excel that we identify as potentially capable of propagating where-provenance. The functions are split into three classes based on the analysis of Calc.ts: (A) functions that need no further changes beyond those we describe in Section III, (B) functions that need additional minor changes, and (C) functions that require significant semantic changes.

Class (A) consists of polymorphic array functions such as TRANSPOSE and INDEX, lookup functions such as VLOOKUP and HLOOKUP, control-flow functions such as IF and SWITCH, and reference manipulation functions such as INDIRECT and OFFSET.

Class (B) consists of functions that require further minor changes, where we define minor changes as anything that is confined to the core definition of the function. UNIQUE is unique; changes are required to aggregate provenance for equivalent values as we describe in Section III. The remaining functions all extract values from a collection, but are included here because they are specialised to aggregate numbers. Addressing the limitation requires portal-aware code to be added to each function implementation to retain provenance; in most cases the change is uniform across all the affected functions.

Class (C) consists of functions that are conceptually compatible with where-provenance propagation but require substantial changes—every function is a string manipulation function. Functions LEFT and RIGHT are string specific implementations of the classic array-processing functions *take* and *drop* respectively. If a string was represented as an array of characters, which was then promoted to an array of portals referring to characters, then *take* or *drop* would naturally preserve the where-provenance of each character. However, in Calc.ts strings are a primitive type and the interface does not expose array-like functionality. Supporting where-provenance at a character level would require changes to the function implementations and the implementation of strings.

B. Function Usage in Practice

We now turn to existing research on spreadsheet usage to understand whether the functions we identify are frequently used in practice. We refer to the study by [31] that compares two spreadsheet corpora: Enron [29] and EUSES [22]. From Enron, seven of the fifteen most frequently used functions are present in Figure 7, and five of the compatible functions are from Class (A). From EUSES, six of the fifteen most frequently used functions are present in Figure 7, and three of the compatible functions are from Class (A). The functions SORT, FILTER, and UNIQUE which we highlight in our examples, were not available when Enron and EUSES were curated. Omitted from the table are operators, which are significantly more frequent than functions. In Enron, 71.4% of all formulas contain an operator; in EUSES, 58.5% of all formulas contain an operator [31]. However, whilst operators are used more frequently, this does not suggest making operators bidirectional is more useful to spreadsheet users. From our observations, many practical examples feature the functions we describe in Figure 7—even back in 2004 [7] and 2008 [8] users were asking to directly edit the output of a VLOOKUP formula! The variety of functions present in Figure 7 suggests that where-provenance based bidirectional spreadsheets provide a surprising amount of utility.

V. Implementation

Our implementation Proto[1] extends the user interface and calculation engine of an existing TypeScript spreadsheet prototype. Here we describe key aspects of our implementation and discuss important remaining questions. There are two primary areas of implementation: formula evaluation and the spreadsheet interface. We extend formula evaluation with tagged values to represent portals. Function dispatch is modified to strip tags from input values. The only function implementation we modify is UNIQUE. A departure from our formal presentation is that we do not eagerly annotate array elements with provenance, instead we do so lazily. When provenance is stripped from an array, we add provenance to the elements.

Our modification to the underlying interface infrastructure is minor. The existing implementation provides a setCell function for modifying the model. We augment this function to inspect the computed value of the target cell, and if the computed value is a portal, we redirect the operation. One interface challenge is disambiguating whether a user wants to perform a portal update, or modify a cell's formula. For example, in Figure 1, if the user edits D2, does the edit apply to the formula or to the value "Brazil" in B3? Proto uses a checkbox to distinguish editing the value from the formula, however controls always edit the value. We envisage a more sophisticated interface could employ a dedicated *value editor*.

Our core calculus and Proto promotes every value read from the grid to a portal. We do not measure the overhead introduced by portals but we expect that it is not feasible to universally promote values to portals in all spreadsheets in practice. Even if performance is unaffected, users may not want bidirectional editing. We envisage a model where the user explicitly enables portals using a function such as UPDATABLE that enriches a value with provenance; to enable bidirectional editing, the formula in Figure 1 will become SORT(UNIQUE(UPDATABLE(B2:B6))).

VI. Where-provenance and Lenses

There is much prior work on bidirectional editing, including lenses and specifically the work by Macedo et al. [33] that implement lenses for spreadsheet formulas. Their work allows the user to designate some formulas as being the forwards

[7] https://web.archive.org/web/20210302005953/http://www.vbaexpress.com/forum/showthread.php?540-Solved-VLOOKUP-Edit-Results-Paste-Changes

[8] https://web.archive.org/web/20170607034133/https://www.excelforum.com/excel-formulas-and-functions/651204-edit-vlookup-result.html

get direction of a lens, and then synthesizes, guided by user annotations, a formula to compute *put*, the backwards update.

Lenses, in theory, support a wider range of formulas that can be bidirectionalized; in contrast, portals do not support editing of formulas like A1 + B1 because arithmetic computations do not propagate where-provenance. Whilst portals are less expressive in principle, it is unclear how often users want to bidirectionalize arithmetic calculations. The examples we provide are all motivated by real spreadsheet tasks.

Increased expressive power incurs a per-function implementation cost that our approach does not pay. Synthesizing a *put* formula requires knowledge about each function; for instance, a *put* formula for the TRANSPOSE function cannot be synthesized without knowing what TRANSPOSE does. In contrast, portals work with TRANSPOSE for free. Still, there are other lens-based approaches, such as semantic bidirectionalization [42], without this limitation.

Another important aspect of bidirectional editing is explaining the effect of an update to the user. Macedo et al. [33] synthesize put formulas, so that the update is described using the same formula language. However, showing formulas to the user may or may not make the effect of the update transparent. For instance, the put formula for VLOOKUP uses IF and MATCH, functions perhaps unfamiliar to the user. If the lookup array was constructed by SORT, then the put formula grows in complexity to accommodate the effect of sorting. In general, a put formula can be significantly more complex than the get formula. In contrast, the explanation for the effect of an update using portals is uniform, by construction. The where-provenance describes exactly what will change and how (by replacement), and PROTO illustrates via coloured foils.

Semantic bidirectionalization (or SBX for short) [34], [35], [42] is a technique that exploits relational parametricity [39], [43] to automatically synthesize a putback function for a container transformation. The technique has not been applied to spreadsheets. Fundamentally, SBX also uses where-provenance, however, we identify two differences with our application. First, we forgo the lens update property PUTGET which enforces consistency between the input and output. For example, given formula $\mathsf{SORT}(\mathsf{A1:A3})$, where $\mathsf{A1:A3} = \{3;1;2\}$, we allow the user to edit the first item in the output (pointing to A2) to have the value 10; in contrast, SBX prohibits this updated because it does not preserve the sort order. We think that in most cases the pointer-based update of our approach is more useful to spreadsheet users. The second difference is that provenance exists in our system independently of bidirectional editing, and useful on its own; in contrast, provenance in SBX remains hidden. In summary, our approach is a spreadsheet specific application of SBX techniques, emphasising where-provenance.

VII. RELATED WORK

a) Bidirectional Programming: [38] pioneered early work on *constraint maintainers* for user interfaces. [15] implement SKETCH-N-SKETCH, a system with bidirectional editing using trace-based program synthesis. Constraints are produced from a trace, and edits to the output are reconciled against the constraints to produce *small updates* to the input. [36] implement an *evaluation update* algorithm that modifies a program and input in response to output edits. Their approach addresses a limitation of lenses which is that they do not readily manipulate arbitrary programs. [27] extend SKETCH-N-SKETCH with new capabilities for output-directed programming. Their system shares ideas with trace-based provenance [3], including where-provenance, but they do not make a formal correspondence with where-provenance.

b) Lenses: Further types of lenses include Quotient lenses [24] and Edit lenses [30], providing *permutation* and *container restructuring* lenses respectively—both are related to provenance. In particular, a container restructuring lens for an array takes a permutation vector for the elements and allows edits to the permuted array to transport back to the original. If SORT or similar functions also produced a permutation then it would be possible to construct a lens that permits similar updates to our portal approach. When considering the relationship between lenses and provenance, the sort permutation used to construct the edit lens can be viewed as the *provenance* for the work performed by the sort.

c) Provenance: [11] first proposed *why* and *where* provenance in the context of databases. [14] give a review of database provenance, including a third variant *how* provenance. [1] present the Dependency Core Calculus which embeds multiple variants of program dependency analysis. [13] connect dependency analysis to provenance, and extend the nested relation calculus with *dependency provenance*.

d) Spreadsheets: [18] implement Boxer, a visual programming system based on the *spatial metaphor*. A component of the system is called a *port*: a view of another component that is bidirectionally synchronised. A port is not a value, but is like a special spreadsheet cell that holds a fixed portal. In contrast, in our system, a portal is simply a value of a formula, and its target may be determined dynamically by the formula.

There are multiple systems that allow spreadsheet interfaces to manipulate web applications, including Gneiss [12], Quilt [7], and Wildcard [32]. These systems do not define a formal semantics with provenance.

VIII. CONCLUSION

We develop a new spreadsheet concept, the portal, based on where-provenance. When a value with where-provenance is computed by a cell's formula, a user can edit the location described by the provenance from that cell. Portals are first-class spreadsheet values, hence users can now control the editing surface of a spreadsheet using formulas. We analyse a proprietary spreadsheet implementation Calc.ts to evaluate where-provenance as a basis for bidirectional transformation. We find that many commonly used spreadsheet functions support portals with little or no modification. In future work we aim to explore the connection between provenance and bidirectional transformation, in addition to further exploring the user interface questions posed by bidirectional transformations.

References

[1] M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke, "A Core Calculus of Dependency," in *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '99. New York, NY, USA: Association for Computing Machinery, 1999, pp. 147–160. [Online]. Available: https://doi.org/10.1145/292540.292555

[2] R. Abraham and M. Erwig, "GoalDebug: A Spreadsheet Debugger for End Users," in *Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE '07. USA: IEEE Computer Society, 2007, pp. 251–260. [Online]. Available: https://doi.org/10.1109/ICSE.2007.39

[3] U. A. Acar, A. Ahmed, J. Cheney, and R. Perera, "A Core Calculus for Provenance," in *Principles of Security and Trust*, P. Degano and J. D. Guttman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 410–429.

[4] H. Apfelmus, 2012. [Online]. Available: https://wiki.haskell.org/FRP_explanation_using_reactive-banana

[5] D. W. Barowy, E. D. Berger, and B. Zorn, "ExceLint: Automatically Finding Spreadsheet Formula Errors," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, Oct. 2018. [Online]. Available: https://doi.org/10.1145/3276518

[6] M. Bendre, B. Sun, D. Zhang, X. Zhou, K.-C. Chang, and A. Parameswaran, "DataSpread: Unifying Databases and Spreadsheets," in *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, vol. 8, 08 2015, pp. 2000–2003.

[7] E. Benson, A. X. Zhang, and D. R. Karger, "Spreadsheet Driven Web Applications," in *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 97–106. [Online]. Available: https://doi.org/10.1145/2642918.2647387

[8] A. F. Blackwell, M. M. Burnett, and S. L. Peyton Jones, "Champagne Prototyping: A Research Technique for Early Evaluation of Complex End-User Programming Systems," in *VL/HCC*. IEEE Computer Society, 2004, pp. 47–54.

[9] A. A. Bock, T. Bøgholm, P. Sestoft, B. Thomsen, and L. L. Thomsen, "On the semantics for spreadsheets with sheet-defined functions," *Journal of Computer Languages*, vol. 57, p. 100960, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2590118420300204

[10] J. Borghouts, A. D. Gordon, A. Sarkar, and N. Toronto, "End-User Probabilistic Programming," in *Quantitative Evaluation of Systems*, D. Parker and V. Wolf, Eds. Cham: Springer International Publishing, 2019, pp. 3–24.

[11] P. Buneman, S. Khanna, and W. C. Tan, "Why and Where: A Characterization of Data Provenance," in *Proceedings of the 8th International Conference on Database Theory*, ser. ICDT '01. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 316–330.

[12] K. S.-P. Chang and B. A. Myers, "Creating Interactive Web Data Applications with Spreadsheets," in *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 87–96. [Online]. Available: https://doi.org/10.1145/2642918.2647371

[13] J. Cheney, A. Ahmed, and U. A. Acar, "Provenance as Dependency Analysis," in *Database Programming Languages*, M. Arenas and M. I. Schwartzbach, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 138–152.

[14] J. Cheney, L. Chiticariu, and W.-C. Tan, "Provenance in Databases: Why, How, and Where," *Found. Trends Databases*, vol. 1, no. 4, Apr. 2009. [Online]. Available: https://doi.org/10.1561/1900000006

[15] R. Chugh, B. Hempel, M. Spradlin, and J. Albers, "Programmatic and Direct Manipulation, Together at Last," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 341–354. [Online]. Available: https://doi.org/10.1145/2908080.2908103

[16] E. Czaplicki and S. Chong, "Asynchronous Functional Reactive Programming for GUIs," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 411–422. [Online]. Available: https://doi.org/10.1145/2491956.2462161

[17] U. Dayal and P. A. Bernstein, "On the correct translation of update operations on relational views," *ACM Trans. Database Syst.*, vol. 7, no. 3, pp. 381–416, 1982.

[18] A. A. diSessa and H. Abelson, "Boxer: A Reconstructible Computational Medium," *Commun. ACM*, vol. 29, no. 9, pp. 859–868, Sep. 1986. [Online]. Available: https://doi.org/10.1145/6592.6595

[19] C. Elliott and P. Hudak, "Functional reactive animation," in *International Conference on Functional Programming*, 1997. [Online]. Available: http://conal.net/papers/icfp97/

[20] R. Fagin, J. D. Ullman, and M. Y. Vardi, "On the semantics of updates in databases," in *PODS*. ACM, 1983, pp. 352–365.

[21] S. Fehrenbach and J. Cheney, "Language-Integrated Provenance," in *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming*, ser. PPDP '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 214–227. [Online]. Available: https://doi.org/10.1145/2967973.2968604

[22] M. Fisher and G. Rothermel, "The euses spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms," in *Proceedings of the first workshop on End-user software engineering*, 2005, pp. 1–5.

[23] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt, "Combinators for Bidirectional Tree Transformations: A Linguistic Approach to the View-Update Problem," *ACM Trans. Program. Lang. Syst.*, vol. 29, no. 3, May 2007. [Online]. Available: https://doi.org/10.1145/1232420.1232424

[24] J. N. Foster, A. Pilkiewicz, and B. C. Pierce, "Quotient lenses," in *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '08. New York, NY, USA: Association for Computing Machinery, 2008, pp. 383–396. [Online]. Available: https://doi.org/10.1145/1411204.1411257

[25] S. Gulwani, "Automating String Processing in Spreadsheets Using Input-Output Examples," in *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 317–330. [Online]. Available: https://doi.org/10.1145/1926385.1926423

[26] S. Gulwani, W. R. Harris, and R. Singh, "Spreadsheet Data Manipulation Using Examples," *Commun. ACM*, vol. 55, no. 8, pp. 97–105, Aug. 2012. [Online]. Available: https://doi.org/10.1145/2240236.2240260

[27] B. Hempel, J. Lubin, and R. Chugh, "Sketch-n-Sketch: Output-Directed Programming for SVG," in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 281–292. [Online]. Available: https://doi.org/10.1145/3332165.3347925

[28] F. Hermans, B. Jansen, S. Roy, E. Aivaloglou, A. Swidan, and D. Hoepelman, "Spreadsheets are Code: An Overview of Software Engineering Approaches Applied to Spreadsheets," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5, 2016, pp. 56–65.

[29] F. Hermans and E. Murphy-Hill, "Enron's spreadsheets and related emails: A dataset and analysis," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 7–16.

[30] M. Hofmann, B. Pierce, and D. Wagner, "Edit Lenses," in *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 495–508. [Online]. Available: https://doi.org/10.1145/2103656.2103715

[31] B. Jansen, "Enron versus euses: A comparison of two spreadsheet corpora," in *Proceedings of the 2nd Ceur Workshop 1355: Software Engineering Methods in Spreadsheets, SEMS 2015, Florence, Italy, May 18, 2015. Eds.: Hermans, F., Paige, RF, Sestoft, P.* CEUR-WS, 2015.

[32] G. Litt and D. Jackson, "Wildcard: Spreadsheet-Driven Customization of Web Applications," in *Conference Companion of the 4th International Conference on Art, Science, and Engineering of Programming*, ser. ¡programming¿ '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 126–135. [Online]. Available: https://doi.org/10.1145/3397537.3397541

[33] N. Macedo, H. Pacheco, N. R. Sousa, and A. Cunha, "Bidirectional spreadsheet formulas," in *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2014, pp. 161–168.

[34] K. Matsuda and M. Wang, "Bidirectionalization for free with runtime recording: Or, a light-weight approach to the view-update problem," in *Proceedings of the 15th Symposium on Principles and Practice*

of Declarative Programming, ser. PPDP '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 297–308. [Online]. Available: https://doi.org/10.1145/2505879.2505888

[35] ——, ""Bidirectionalization for free" for monomorphic transformations," *Science of Computer Programming*, vol. 111, pp. 79–109, 2015.

[36] M. Mayer, V. Kuncak, and R. Chugh, "Bidirectional Evaluation with Direct Manipulation," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, Oct. 2018. [Online]. Available: https://doi.org/10.1145/3276497

[37] M. McCutchen, J. Borghouts, A. D. Gordon, S. Peyton Jones, and A. Sarkar, "Elastic sheet-defined functions: Generalising spreadsheet functions to variable-size input arrays," *Journal of Functional Programming*, vol. 30, p. e26, 2020.

[38] L. Meertens, "Designing Constraint Maintainers for User Interaction," Tech. Rep., 1998.

[39] J. Reynolds, "Types, abstraction, and parametric polymorphism," in *Information Processing*, 1983.

[40] P. Sestoft, "Implementing function spreadsheets," in *Proceedings of the 4th international workshop on End-user software engineering*. ACM, 2008, pp. 91–94.

[41] W.-C. Tan, "Containment of Relational Queries with Annotation Propagation," in *Database Programming Languages*, G. Lausen and D. Suciu, Eds. Springer Berlin Heidelberg, 2004, pp. 37–53.

[42] J. Voigtländer, "Bidirectionalization for Free! (Pearl)," in *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 165–176. [Online]. Available: https://doi.org/10.1145/1480881.1480904

[43] P. Wadler, "Theorems for free!" in *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*, ser. FPCA '89. New York, NY, USA: ACM, 1989, pp. 347–359. [Online]. Available: http://doi.acm.org/10.1145/99370.99404

[44] J. Williams, C. Negreanu, A. D. Gordon, and A. Sarkar, "Understanding and Inferring Units in Spreadsheets," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2020, pp. 1–9.

[45] J. Williams, N. Joharizadeh, A. D. Gordon, and A. Sarkar, "Higher-Order Spreadsheets with Spilled Arrays," in *Programming Languages and Systems. ESOP 2020. LNCS vol 12075*, P. Müller, Ed. Springer, 2020.

[46] J. Zhang, S. Han, D. Hao, L. Zhang, and D. Zhang, "Automated Refactoring of Nested-IF Formulae in Spreadsheets," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 833–838. [Online]. Available: https://doi.org/10.1145/3236024.3275532