SCULPT: A Schema Language for Tabular Data on the Web

Wim Martens
Universität Bayreuth

Frank Neven
Hasselt University and transnational University of Limburg

Stijn Vansummeren
Université Libre de Bruxelles

ABSTRACT

Inspired by the recent working effort towards a recommendation by the World Wide Web Consortium (W3C) for tabular data and metadata on the Web, we present in this paper a concept for a schema language for tabular web data called SCULPT. The language consists of rules constraining and defining the structure of regions in the table. These regions are defined through the novel formalism of region selection expressions. We present a formal model for SCULPT and obtain a linear time combined complexity evaluation algorithm. In addition, we consider weak and strong streaming evaluation for SCULPT and present a SCULPT fragment for each of these streaming variants. Finally, we discuss several extensions of SCULPT including alternative semantics, types, complex content, and explore region selection expressions as a basis for a transformation language.

Categories and Subject Descriptors

H.2.3 [**Information Systems**]: Languages—*Data description languages*

Keywords

Tabular data, comma separated values, metadata

1. INTRODUCTION

Despite the availability of numerous standardized formats for semi-structured and semantic web data such as XML, RDF, and JSON, a very large percentage of data and open data published on the web, remains tabular in nature.[1] Tabular data is most commonly published in the form of comma separated values (CSV) files because such files are open and therefore processable by numerous tools, and tailored for all sizes of files ranging from a number of KBs to several TBs. Despite these advantages, working with CSV files is often cumbersome because they are typically not accompanied by a *schema* that describes the file's structure (i.e., "the second column is of integer datatype", "columns are delimited by tabs", ...) and captures its intended meaning. Such a description is nevertheless vital for any user trying to interpret the file and execute queries or make changes to it. In other data models, the presence of a schema is also important for query optimization (required for scalable query execution if the file is large), as well as other static analysis tasks. Finally, we strongly believe that schemas are a prerequisite for unlocking huge amounts of tabular data to the Semantic Web. Indeed, unless we have a satisfactory way of describing the structure of tabular data we cannot specify how its content should be interpreted as RDF. Drawing parallels with relational databases, observe that R2RML mappings [7] (the W3C standard for mapping relational databases to RDF) inherently need to refer to the schema (structure) of the relational database in order to specify how database tuples can be represented as RDF.

In recognition of this problem, the *CSV on the Web* Working Group of the World Wide Web Consortium [34] argues for the introduction of a schema language for tabular data to ensure higher interoperability when working with datasets using the CSV or similar formats. In particular, their charter states [34]:

> *Whether converted to other formats or not, there is a need to describe the content of the CSV file: its structure, datatypes used in a specific column, language used for text fields, access rights, provenance, etc. This means that metadata should be available for the dataset, relying on standard vocabulary terms, and giving the necessary information for applications. The metadata can also be used for the conversion of the CSV content to other formats like RDF or JSON, it can enable automated loading of the data as objects, or it can provide additional information that search engines may use to gain a better understanding of the content of the data.*

In the present paper, we introduce SCULPT as a concept for such a schema language for tabular data.[2]

The critical reader may wonder whether designing such a schema language isn't trivial. After all, doesn't it suffice to

[1] Jeni Tennison, one of the two co-chairs of the W3C CSV on the Web working group claims that "over 90% of the data published on data.gov.uk is tabular data" [30].

[2] The name SCULPT for the language is in honour of Michelangelo, who allegedly said *"Every block of stone has a statue inside it and it is the task of the sculptor to discover it."* Readers who like acronyms can read SCULPT as **SC**hema for **U**n-**L**ocking and **P**rocessing **T**abular data.

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741142.

be able to specify, for each column, the column's name and the type of data allowed in its cells—similar to how relational database schemas are defined using the SQL data definition language? The answer is no. The reason is that there is a lot of variation in the tabular data available on the web and that examples abound of tabular data whose structure cannot be described by simple rules of the form "column x has datatype y". Figures 3, 5, and 7, for example, show some tabular data sets drawn from the Use Cases and Requirements document drafted by the W3C CSV on the Web working group [29]. Notice how, in contrast to "standard" CSV files, Figure 3 has a header consisting of multiple lines. This causes the data in the first column to be non-uniform. Further notice how the `provenance` data in the Figure 5 is spread among multiple columns. Finally, notice how the shape of the rows in Figure 7 depends on the label in the first column: `TITLE` rows have different structure than `AUTHOR` rows, which have a different structure than `ATOM` rows, and so on.

Sculpt schemas use the following idea to describe the structure of these tables. At their core, Sculpt schemas consist of rules of the form $\varphi \to \rho$. Here, φ selects a *region* in the input table (i.e., a subset of the table's cells) and ρ constrains the allowed structure and content of this region. A table is valid with respect to a Sculpt schema if, for each rule in the schema, the region selected by φ satisfies the content constraints specified by ρ. It is important to note that Sculpt's expressive power goes well beyond that of classical relational database schemas since Sculpt's region selectors are not limited to selecting columns. In particular, the language that we propose for selecting regions is capable of navigating through a table's cells bears much resemblance to the way XPath [3] navigates through the nodes of an XML tree. For tokenizing the content of single cells, we draw inspiration from XML Schema simple types [8, section 2.2]. Both features combined will allow us to express the use cases of the W3C CSV on the Web Working Group.

We note that the W3C is also working on a schema language for tabular data [24]. At the moment, however, that schema language focuses on orthogonal issues like describing, for instance, *datatypes* and *parsing cells*. Also, it only provides facilities for the selection of *columns*, and is hence not able to express the schema of the more advanced use cases. Sculpt, in contrast, draws inspiration from well-established theoretical tools from logic and formal languages, which adds to the robustness of our approach. Due to the above mentioned orthogonality we expect that it is not difficult to integrate ideas from this paper in the W3C proposal.

In summary, we make the following contributions.

1. We illustrate the power of Sculpt, and its suitability as a schema language for tabular data on the web, by expressing several use cases drafted by the W3C CSV on the Web working group [29]. (Section 2)
2. We provide a formal model for the core of Sculpt. A key contribution in this respect is the introduction of the region selection language. (Section 3)
3. We show that, despite its rather attractive expressiveness, tables can be efficiently validated w.r.t. Sculpt schemas. In particular, when the table is small enough to be materialized in main memory, we show that validation can be done in linear time combined complexity (Section 4.1). For scenarios where materialization in main memory is not possible, we consider the scenario of streaming (i.e., incremental) validation, for which formally introduce two versions: *weak streamability* and *strong streamability*. (Their differences are described in detail in Section 4.2.) We show in particular that the fragment of core-Sculpt where region selectors can only look "forward" and never "backward" in the CSV file is *weakly streamable*. If we further restrict region selectors to be both forward-looking and *guarded* (a notion formalized in Section 4.2) validation becomes *strongly streamable*. All of the W3C Working group use cases considered here can be expressed using forward and guarded region selectors, hence illustrating the practical usefulness of this fragment.
4. While our focus in this paper is on introducing Sculpt as a means for specifying the structure of CSV files and related formats, we strongly believe that region selector expressions are a fundamental component in developing other features mentioned in the charter of the W3C CSV on the Web Working Group, such as a CSV transformation language (for converting tabular data into other formats such as RDF or JSON), the specification of the language used for text fields; access rights; provenance; etc. While a full specification of these features is out of this paper's scope, we illustrate by means of example how Sculpt could be extended to incorporate them. (Section 5)

Note. Due to space restrictions, proofs of formal statements are only sketched. A full version containing all proofs is available on ArXiv [21].

Related Work. The present paper fits in the line of research, historically often published in the WWW conference, that aims to formalize and study the properties of various W3C working group drafts and standards (including XML Schema [4,5], SPARQL [2,18,23], and RDF [25,26]) with the aim of providing feedback and input to the working group's activities.

Given the numerous benefits of schemas for data processing, there is a large body of work on the development, expressiveness, and properties of schema languages for virtually all data models, including the relational data model, XML [4, 5, 11, 19, 20], and, RDF [25, 26]. Sculpt differs from the schema languages considered for XML and RDF in that it is specifically designed for tabular data, not tree-structured or graph-structured data. Nevertheless, the rule-based nature of Sculpt draws inspiration from our prior work on rule-based and pattern-based schema languages for XML [11, 19, 20].

As already mentioned, while traditional relational database schemas (formulated in e.g., the SQL data definition language) are specifically designed for tabular data, they are strictly less expressive than Sculpt schemas in the sense that relational schemas limit region selection expressions to those that select columns only. A similar remark holds for other recent proposals of CSV schema languages, including the CSV Schema language proposed by the UK National Archives [1], and Tabular Data Package [16]. The remark also applies to the part of Google's Dataset Publishing Language (DSPL) [12] describing the content of CSV files. In contrast, DSPL also has features to relate data from multiple CSV files, which Sculpt does not yet have.

The problem of streaming schema validation has been investigated in the XML context for DTDs and XML schemas [14, 20, 27, 28]. In this work, the focus is on finding algorithms that can validate an XML document in a single pass

using constant memory or, if this is not possible, a memory that is bounded by the depth of the document. Our notion of streaming, in contrast, is one where we can use a memory that is not constant but at most logarithmic in the size of the table (for strong streaming), or at most linear in the number of columns and logarithmic in the number of rows (for weak streaming). This allows us to restrict memory when going from one row to the next and is essential to be able to navigate downwards in SCULPT region selection expressions.

While streaming validation is undoubtedly an important topic for all of the CSV schema languages mentioned above [1,12,16] (the National Archives Schema Language mentions it as an explicit design goal), no formal streaming validation algorithm has been proposed for them, to the best of our knowledge.

We briefly mention some major lines of research on tables on the web. One line of work considers HTML tables. Here, the focus is on finding (related) HTML tables (at a search engine scale) [6], extracting meaning of the tables based on its content (see, e.g. [33]), and extracting RDF from HTML tables [22]. Other work targets conversions from spreadsheets to RDF (e.g. [13]).

2. SCULPT BY EXAMPLE

In this section, we introduce SCULPT through a number of examples. The formal semantics of the examples is defined in Section 3. The syntax we use here is tuned for making the examples accessible to readers and is, of course, flexible.

SCULPT schemas operate on *tabular documents*, which are text files describing tabular data. SCULPT schemas consist of two parts (cf. Fig. 2). The first part, *parsing information*, defines the row and column delimiters and further describes how words should be tokenized. This allows to parse the text file and build a table-like structure consisting of rows and columns. In this section we allow some rows to have fewer columns than others but we require them to be aligned to the left. That is, non-empty rows always have a first column. The second part of the schema consists of *rules* that interpret the table defined by the first part as a rectangular grid and that enforce structure. In particular, rules are of the form $\varphi \to \rho$, where φ selects a *region* consisting of cells in the grid while ρ is a regular expression constraining the content of the selected region. We utilize a so-called *row-based* semantics: every row in the region selected by φ should be of a form allowed by ρ. We refer to φ as the *selector expression* and to ρ as the *content expression*.

Next, we illustrate the features of the language by means of examples. All examples are inspired by the use cases and requirements drafted by the CSV on the Web W3C working group [29].

EXAMPLE 2.1. Fig. 1 contains a slightly altered fragment (we use a comma as a column separator) of a CSV file mentioned in Use Case 3, *"Creation of consolidated global land surface temperature climate databank"* [29]. Its SCULPT schema, displayed as Fig. 2, starts by describing parsing information indicating that the column delimiter is a comma while the row delimiter is a newline. Lines starting with a %-sign are comments. Tokens are defined based on regular expressions (regex for short).[3] For instance, anything that

[3]For ease of exposition, we adopt the concise regex syntax popularized by scripting languages such as Perl, Python, and Ruby [10] in all of our examples.

```
       ,   ARUA,  BOMBO, ENTEBBE AIR
1935.04, -99.00, -99.00,       27.83
1935.12, -99.00, -99.00,       25.72
1935.21, -99.00, -99.00,       26.44
1935.29, -99.00, -99.00,       25.72
1935.37, -99.00, -99.00,       24.61
1935.46, -99.00, -99.00,       24.33
1935.54, -99.00, -99.00,       24.89
```

Figure 1: Example tabular data inspired by Use Case 3 in [29].

```
%  Parsing information
%% Delimiters
Col Delim = ,
Row Delim = \n

%% Tokens
%% left: token name, right: regex
Timestamp = [0-9]{4}"."[0-9]{2}
Temperature = (-)?[0-9]{2}"."[0-9]{2}
ARUA = ARUA
BOMBO = BOMBO
ENTEBBE AIR = ENTEBBE AIR

% Rules
row(1) -> Empty, ARUA, BOMBO, ENTEBBE AIR
col(1) -> Empty | Timestamp
col(ARUA) -> Temperature
col(BOMBO) -> Temperature
col(ENTEBBE AIR) -> Temperature
```

Figure 2: Schema for tabular data of the type in Fig. 1.

matches the regex `[0-9]{4}"."[0-9]{2}` follows the format *four digits, dot, two digits*, and is interpreted by the token `Timestamp` in the rules of the schema (similar for `Temperature`). Notice that we keep the regexes short (and sometimes imprecise) for readability, but they can of course be made arbitrarily precise if desired.

All the XML Schema primitive types such as xs:integer, xs:string, xs:date, etc. are pre-defined as tokens in a SCULPT schema. There is also a special pre-defined token `Empty` to denote that a certain cell is empty.

Notice that the schema in Fig. 2 has three token definitions in which the regex defines only one character sequence (namely: `ARUA`, `BOMBO`, `ENTEBBE AIR`). In the sequel, we will omit such rules for reasons of parsimony. For the same reason, we omit the explicit definition of column and row delimiters when they are a comma and newline character, respectively.

The rule

```
row(1) -> Empty, ARUA, BOMBO, ENTEBBE AIR
```

selects all cells in the first row and requires that the first is empty, the second contains `ARUA`, the third `BOMBO`, and the fourth `ENTEBBE AIR`. Next, `col(1)` selects the region consisting of all cells in the first column. As SCULPT assumes a row-based semantics per default,[4] the rule

```
col(1) -> Empty | Timestamp
```

requires that every row in the selected region (notice that each such row consists of a single cell) is either empty (`Empty`) or contains data that matches the `Timestamp` token. The expression `col(ARUA)` selects all cells in the column below the

[4]We discuss an extension in Section 5.

```
QS601EW
Economic activity
27/03/2011

         ,         , Count    , Count
         ,         , Person   , Person
         ,         , Activity, Activity
GeoID    , GeoArea, All      , Part-time
E92000001, England, 38881374, 27183134
W92000004, Wales  , 2245166 , 1476735
```

Figure 3: Fragment of a CSV-like-file, inspired by Use Case 2 in [29].

```
%% Tokens
%% left: token name, right: regex
name = QS[0-9]*EW
ctype = Economic Activity
geo_id = E[0-9]*

% Rules
row(1) -> name
row(2) -> ctype
row(3) -> Date
row(4) -> Empty
row(5) -> Empty, Empty, Count*
row(6) -> Empty, Empty, Person*
row(7) -> Empty, Empty, Activity*
row(8) -> GeoID, GeoArea, String*
col(GeoID) -> geo_id
col(GeoArea) -> String
down+(right+(GeoArea)) -> Number*
```

Figure 4: Schema for files of the type in Fig. 3.

cell containing `ARUA`. The rule

```
col(ARUA) -> Temperature
```

therefore requires that every row in this region matches the `Temperature` token. The two remaining rules are analogous. The fragment in Fig. 1 satisfies the schema of Fig. 2. ■

Before moving on to some more advanced examples, we discuss in more detail the semantics of selector and content expressions. Each cell in a table is identified by its *coordinate*, which is a pair (k, ℓ) where k indicates the row number ($k \geq 1$) and ℓ the column number ($\ell \geq 1$). We use the convention that the top left coordinate in tabular data bears the coordinate $(1, 1)$ — for *first row*, *first column*. In each rule $\varphi \to \rho$, the selector expression φ returns a *set* of coordinates (a region) and ρ is a regular expression defining the allowed structure of each row in the region selected by φ. It is important to note that in each such row only the cells which are selected by φ are considered. Another way to interpret the row-based semantics is that of a 'group by' on the selected region per row.

The last rule we discussed in Example 2.1 uses a symbolic coordinate `ARUA` in its selector expression. Its semantics is as follows: a token τ returns the set of all coordinates (k, ℓ) whose cell content matches τ. The operator `row` applied to a coordinate (k, ℓ) returns the set of coordinates $\{(k, \ell') \mid \ell' > \ell\}$. This corresponds to the row consisting of all elements to the right of (k, ℓ). Note that coordinate (k, ℓ) itself is not included. Applying `row` to a *set* S of coordinates amounts to taking the union of all `row`$((k, \ell))$ where $(k, \ell) \in S$. Similarly, the operator `col` applied to S returns the union of the regions $\{(k', \ell) \mid k' > k\}$ for each (k, ℓ) in S, corresponding to columns below elements in S. The selector expressions `row(1)` and `col(1)` that select the "first row" and "first column", respectively, use syntactic sugar to improve readability (see Remark 3.4).

The next example illustrates the use of slightly more complex expressions for navigation and content.

Example 2.2. Fig. 3 displays a (slightly altered) fragment of a CSV-like-file inspired by Use Case 2 (*"Publication of National Statistics"*) in [29]. This fragment originates from the Office for National Statistics (UK) and refers to the dataset "QS601EW Economic activity" derived from the 2011 Census. The file starts with three lines of metadata, referring to the name of the file and the census date, continues with a blank line, before listing the actual data separated by commas. Notice that this file is, strictly speaking, not a comma-separated-value file because not all rows have an equal number of columns.[5] Indeed, the first four rows have only (at most) one column and the later rows have four columns. Fig. 4 depicts the Sculpt schema describing the structure of such tables.

The schema starts by describing parsing information, analogous to Example 2.1. The first four rules are very basic and are similar to those of Example 2.1. We first describe the fifth rule of the schema:

```
row(5) -> Empty, Empty, Count*
```

selects all cells in the fifth row, requiring the first two to be `Empty` and the remaining non-empty cells to contain `Count`. We note that the original data fragment from [29] contains 16 such columns.

The rule `col(GeoID) -> geo_id` selects all cells below cells containing the word `GeoID`. The content expression says that this column contains values that match the `geo_id` token. The last rule is the most interesting one:

```
down+(right+(GeoArea)) -> Number*.
```

This rule selects all cells appearing strictly downward and to the right of `GeoArea` and requires them to be of type `Number`. More precisely, `GeoArea` is a symbolic coordinate selecting all cells containing the word `GeoArea`. The navigational operators `right` and `down` select cells one step to the right and one step down, respectively, from a given coordinate. The operator `+` indicates an arbitrary strictly positive number of applications of the navigational operator to which it is applied. In particular, as on the table given in Fig. 3, `GeoArea` is the singleton cell with coordinate $(8, 2)$, `right(GeoArea)` returns $\{(8, 3)\}$, while `right+(GeoArea)` is the region $\{(8, \ell) \mid \ell > 2\}$. Likewise, `down(right+(GeoArea))` is the region $\{(9, \ell) \mid \ell > 2\}$ and, finally, `down+(right+(GeoArea))` is the region downward and to the right of the `GeoArea` coordinate, that is, $\{(k, \ell) \mid k > 8 \text{ and } \ell > 2\}$. ■

Example 2.2 uses more refined navigation than just selecting a row or a column. Sculpt has four navigational axes: `up, down, left, right` which navigate one cell upward, downward, leftward, or rightward. These axes can be applied to a set S of coordinates and add a vector v to it. More formally, an axis A, when applied to a set S of coordinates, returns $A(S) := \{c + v_A \mid c \in S\}$. Here,

- $v_A = (-1, 0)$ when $A =$ `up`,
- $v_A = (1, 0)$ when $A =$ `down`.

[5] Actually CSV does not have a standard, but the informative memo RFC4180 (http://tools.ietf.org/html/rfc4180) states rectangularity in paragraph 2.4.

```
 subject  predicate     object   provenance
:e4      type          PER
:e4      mention       "Bart"   D00124 283-286
:e4      mention       "JoJo"   D00124 145-149 0.9
:e4      per:siblings  :e7      D00124 283-286 173-179 274-281
:e4      per:age       "10"     D00124 180-181 173-179 182-191 0.9
:e4      per:parent    :e9      D00124 180-181 381-380 399-406 D00101 220-225 230-233 201-210
```

Figure 5: Fragment of a CSV-like file, inspired by Use Case 13 in [29].

```
% Tokens
%% left: token name, right: regex
rdf-id      = [a-zA-Z0-9]*:[a-zA-Z0-9]*
rdf-lit     = "[a-zA-Z0-9]*"
prov-book   = D[0-9]{5}
prov-pos    = [0-9]{3}-[0-9]{3}
prov-node   = [0-9].[0.9]
word        = [a-z]*
entity-type = PER | ORG | GPE

% Rules
row(1) -> subject, predicate, object, provenance
col(subject)   -> rdf-id
col(predicate) -> word | rdf-id
col(object)    -> rdf-lit | rdf-id | entity-type
down+(right*(provenance))
         -> (prov-book, prov-pos*, prov-node?)*
```

Figure 6: Schema for files of the type in Fig. 5.

- $v_A = (0, 1)$ when $A =$ `right`, and
- $v_A = (0, -1)$ when $A =$ `left`.

Furthermore, there is also an axis `cell` that does not navigate away from the current cells, i.e., $\texttt{cell}(S) = S$. When applying an axis to a set of coordinates, we always return only the coordinates that are valid coordinates in the table. For example, `left({1,1})` returns the empty set because $(1, 0)$ is not a cell in the table.

While the just discussed features of Sculpt are sufficient to describe the structure of almost all CSV-like data on the Web Working group use cases [29], we extend in Section 3 Sculpt to include XPath-like navigation. These features will be useful for annotations and transformations, see Section 5. We now showcase Sculpt by illustrating it on the most challenging of the W3C use cases.

Example 2.3. Fig. 5 contains a fragment of a CSV-like file, inspired by Use Case 13 in [29] (*"Representing Entities and Facts Extracted From Text"*). Fig. 6 depicts the Sculpt schema. Compared to the previous examples, the most interesting rule is

```
down+(right*(provenance))
         -> (prov-book, prov-pos*, prov-node?)*
```

which, since `provenance` only occurs in column 4, states that every row that starts with a coordinate of the form $(k, 4)$ with $k > 1$ should match `(prov-book, prov-pos*, prov-node?)*`. Notice that the empty row starting at $(2, 4)$ also matches this expression. Here, `right*` denotes an arbitrary number (including zero) of applications of the navigational operator `right`. ■

3. FORMAL MODEL

In this section, we present a formal model for the logical core of Sculpt. We refer to this core as core-Sculpt and discuss extensions in Section 5. We first define the data model.

Tables. For a number $n \in \mathbb{N}$, we denote the set $\{1, \ldots, n\}$ by $[n]$. By $\perp$ we denote a special distinguished null value. For any set $\mathcal{V}$, we denote the set $\mathcal{V} \cup \{\perp\}$ by $\mathcal{V}_\perp$. The W3C formalizes tabular documents through *tables*, which can be defined as follows.

Definition 3.1 (Core Tabular Data Model, [31]). Let $\mathcal{V}$ be a set. A *table* over $\mathcal{V}$ is an $n \times m$ matrix T (for some $n, m \in \mathbb{N}$) in which each cell carries a value from $\mathcal{V}_\perp$. We say that T has n *rows* and m *columns*. A *(table) coordinate* is an element of $[n] \times [m]$. A *cell* is determined by coordinate $(k, \ell) \in [n] \times [m]$ and its *content* is the value $T_{k,\ell} \in \mathcal{V}_\perp$ at the intersection of row k and column ℓ. We denote the set $[n] \times [m]$ of all *coordinates* of T by $\text{coords}(T)$.

Tabular documents. Notice that tables are always rectangular[6] whereas, in Section 2, this was not the case for some of the use cases. We model this by padding shorter rows by $\perp$. More precisely, we see the correspondence between *tabular documents*, i.e., text files that describe tabular data (like CSV files), and tables as follows. Let Σ be a finite set of symbols and let D be a finite set of *delimiters*, disjoint from Σ. We assume that D contains two designated elements which we call *row delimiter* and *column delimiter*, which, as the name indicates, separate cells vertically or horizontally. (We discuss other delimiters in Section 5.) Therefore, a sequence of symbols in $(D \cup \Sigma)$ can be seen as a table over Σ^*: every row delimiter induces a new row in the table, every column delimiter a new column, and the sequences of Σ-symbols between delimiters define a cell's content. In the case that some rows have fewer columns than others, missing columns are expanded to the right and filled with $\perp$. Notice that, hence, we take a $\perp$-cell to be distinct from a cell that has empty content (i.e., a cell that contains the empty Σ-string). Conversely, a table over Σ^* can also be seen as a string over $(D \cup \Sigma)$ by concatenating all its cell values in top-down left-to-right order and inserting cell delimiters and row delimiters in the correct places; we do not insert column delimiters next to $\perp$-cells. As such, when we convert a tabular document into a table and back; we obtain the original tabular document.

We consider both representations in the remainder of the paper. In particular we view the table representation as a structure that allows efficient navigation in all directions and the string representation as structure for streaming validation.

Core-Sculpt schemas. Abstractly speaking, a core-Sculpt schema S is a tuple (D, Δ, Θ, R) where D is the finite set of

[6]Tables are required to be rectangular by Section 2.1 of [31]; as by paragraph 2.4 of the memo RFC4180 on CSV (http://tools.ietf.org/html/rfc4180).

delimiters; Δ is a finite set of tokens; Θ is a mapping that associates a regular expression over Σ to each token $\tau \in \Delta$; and R is a *tabular schema*, a set of rules that constrain the admissible table content (further defined below).

Checking whether a tabular document σ in $(D \cup \Sigma)^*$ satisfies S proceeds conceptually in three phases. In the first phase, the delimiters are used to parse σ into a table T^{raw} over Σ^*, as described above. In the second phase, the token definitions Θ are used to transform T^{raw} into a *tokenized* table T, which is a table where each cell contains a set of tokens (i.e., each cell contains a subset of Δ, namely those tokens that match the cell). Formally, T is the table of the same dimension as T^{raw} such that

$$T_{k,\ell} = \{\tau \in \Delta \mid T^{\text{raw}}_{k,\ell} \in \mathcal{L}(\Theta(\tau))\}.$$

Here $\mathcal{L}(\cdot)$ denotes the language of a regular expression. Finally, the rules in R check validity of the tokenized table T (and not of the raw table T^{raw}), as explained next.

Tabular schema. The *tabular schema* R describes the structure of the tokenized table. Intuitively, a tabular schema is a set of rules $s \to c$ in which s *selects* a region in the table and c describes what the *content* of the selected region should be. More formally, a *region* z of a table T is a subset of coords(T). A *region selection language* $\mathcal{S}$ is a set of expressions such that every $s \in \mathcal{S}$ defines a region in every table T. More precisely, $s[T]$ is always a (possibly empty) region of T. A *content language* $\mathcal{C}$ is a set of expressions such that every $c \in \mathcal{C}$ maps each region z of T to true or false. We denote by $T, z \models c$ that c maps z to true in T and say that z *satisfies* c in T.

Definition 3.2 (Tabular Schema). A *(tabular) schema* (over $\mathcal{S}$ and $\mathcal{C}$) is a finite set R of rules $s \to c$ for which $s \in \mathcal{S}$ and $c \in \mathcal{C}$. A table T *satisfies* R, denoted $T \models R$, when for every rule $s \to c \in R$ we have that $T, s[T] \models c$.

The above definition is very general as it allows arbitrary languages for selecting regions and defining content. We now propose concrete languages for these purposes.

Region selection expressions. Our region selection language is divided into two sorts of expressions: *coordinate expressions* (ranged over by φ, ψ) and *navigational expressions* (ranged over by α, β), defined by the following syntax:

$$\varphi, \psi := a \mid \mathsf{root} \mid \mathsf{true} \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \neg\varphi \mid \langle\alpha\rangle \mid \alpha(\varphi)$$
$$\alpha, \beta := \varepsilon \mid \mathsf{up} \mid \mathsf{down} \mid \mathsf{left} \mid \mathsf{right} \mid [\varphi] \mid (\alpha \cdot \beta) \mid (\alpha|\beta) \mid (\alpha^*)$$

Here, a ranges over tokens in Δ and root is a constant referring to coordinate $(1,1)$. When evaluated over a table T over 2^Δ, a coordinate expression φ defines a region $[\![\varphi]\!]_T \subseteq \text{coords}(T)$, whereas a navigational expression α defines a function $[\![\alpha]\!] : 2^{\text{coords}(T)} \to 2^{\text{coords}(T)}$, as follows.

$$\begin{aligned}
[\![a]\!]_T &:= \{(i,j) \in \text{coords}(T) \mid a \in T_{i,j}\} \\
[\![\mathsf{root}]\!]_T &:= \{(1,1)\} \\
[\![\mathsf{true}]\!]_T &:= \text{coords}(T) \\
[\![(\varphi \vee \psi)]\!]_T &:= [\![\varphi]\!]_T \cup [\![\psi]\!]_T \\
[\![(\varphi \wedge \psi)]\!]_T &:= [\![\varphi]\!]_T \cap [\![\psi]\!]_T \\
[\![(\neg\varphi)]\!]_T &:= \text{coords}(T) \setminus [\![\varphi]\!]_T \\
[\![\langle\alpha\rangle]\!]_T &:= \{c \in \text{coords}(T) \mid [\![\alpha(\{c\})]\!]_T \neq \emptyset\} \\
[\![\alpha(\varphi)]\!]_T &:= [\![\alpha([\![\varphi]\!]_T)]\!]_T
\end{aligned}$$

Furthermore, for every $n \times m$ table T and every set of coordinates $C \subseteq \text{coords}(T)$,

$$\begin{aligned}
[\![\varepsilon(C)]\!]_T &:= C \\
[\![\mathsf{up}(C)]\!]_T &:= \{(i-1,j) \mid (i,j) \in C, i > 1\} \\
[\![\mathsf{down}(C)]\!]_T &:= \{(i+1,j) \mid (i,j) \in C, i < n\} \\
[\![\mathsf{left}(C)]\!]_T &:= \{(i,j-1) \mid (i,j) \in C, j > 1\} \\
[\![\mathsf{right}(C)]\!]_T &:= \{(i,j+1) \mid (i,j) \in C, j < m\} \\
[\![[\varphi](C)]\!]_T &:= C \cap [\![\varphi]\!]_T \\
[\![(\alpha \cdot \beta)(C)]\!]_T &:= [\![\beta([\![\alpha(C)]\!]_T)]\!]_T \\
[\![(\alpha|\beta)(C)]\!]_T &:= [\![\alpha(C)]\!]_T \cup [\![\beta(C)]\!]_T \\
[\![(\alpha^*)(C)]\!]_T &:= \bigcup_{i \geq 0} [\![\alpha^i(C)]\!]_T
\end{aligned}$$

Here, $\alpha^i(C)$ abbreviates the i-fold composition $\alpha \cdots \alpha(C)$. We also use this abbreviation in the remainder. Notice that every coordinate (k,ℓ) of T can be expressed as $\mathsf{down}^{k-1} \cdot \mathsf{right}^{\ell-1}(\mathsf{root})$. For navigational expressions α, we abbreviate $\alpha \cdot \alpha^*$ by α^+ and $\alpha|\varepsilon$ by $\alpha?$. One can read $\alpha(\varphi)$ as "apply the regular expression α to φ". The definition of the semantics of $\alpha \cdot \beta$ conforms with this view.

Example 3.3. Region selection expressions navigate in tables, similar to how XPath expressions navigate on trees. For example, assuming dummy to be a token for `-99.00` in Figure 1, the expression

$$\mathsf{right}^+[\neg\langle\mathsf{down}^*[\mathsf{dummy}]\rangle](\mathsf{root})$$

selects top cells of columns (other than the first) that do not contain a dummy value anywhere. Starting from the root, it first navigates to the right and, from those cells, selects the cells c for which $\mathsf{down}^*[\mathsf{dummy}](c)$ is empty. In the excerpt of Figure 1, this expression hence selects the cell containing `ENTEBBE AIR`. Equivalently, one could also write

$$(\mathsf{right}^+(\mathsf{root}) \wedge \neg(\mathsf{up}^*(\mathsf{dummy})))$$

for the above expression. This expression selects cells right from the root that are not above a dummy-cell.

We present a second example. Assuming the token literal for cells with quotation marks (regex `\"[a-zA-Z0-9]\"`) in Figure 5, the expression

$$\mathsf{down}^+ \cdot [\mathsf{literal}] \cdot \mathsf{right}^+(\mathsf{object})$$

selects all provenance information for rows in which the `object` is between quotes, like `"Bart", "JoJo", and "10"`. Notice in particular that the semantics of the operator "[ ]" in navigational expressions is the same as filter-expressions in XPath. ■

Readers familiar with propositional dynamic logic (PDL for short) [9] will recognize that the above language is nothing more than propositional dynamic logic, tweaked to navigate in tables. As such, the language is also very close to some fragments of Graph XPath [17].

There are some differences between the syntax of core-Sculpt and the region selection expressions used in the examples of Section 2:

Remark 3.4. (i) As already observed, absolute coordinates in Section 2 are syntactic sugar for navigations that start at the root. For example, the coordinate $(2,2)$ would be unfolded to $\mathsf{down} \cdot \mathsf{right}(\mathsf{root})$ in core-Sculpt.

(ii) The keywords `row` and `col` in Section 2 are syntactic sugar for right^+ and down^+ in core-Sculpt, respectively. So, `col((2,2))`, which denotes *the column below the cell* $(2,2)$ in Section 2, is syntactic sugar for $\mathsf{down}^+(\mathsf{down} \cdot \mathsf{right}(\mathsf{root}))$.

(iii) The only exception to rule (ii) above are row and column expressions of the form $\mathsf{row}(k)$ and $\mathsf{col}(\ell)$. These abbreviate $\mathsf{right}^*(k,1)$ and $\mathsf{down}^*(1,\ell)$, respectively. (Where $(k,1)$ and $(1,\ell)$ need to be further unfolded themselves.)

As an example, the selection expression `row(1)` of Figure 6 can be written as $\mathsf{right}^*(\mathsf{root})$ or, equivalently, right^* and the expression `col(subject)` as $\mathsf{down}^+(\texttt{subject})$.

(iv) It is also easy to add syntactic sugar in the form $\mathsf{rectangle}((k_1,\ell_1),(k_2,\ell_2))$ for selecting an area with (k_1,ℓ_1) and (k_2,ℓ_2) as upper left and lower right corner. ■

Content expressions. A *content expression* is simply a regular expression ρ over the set of tokens Δ. To define when a region in a tokenized table T is valid with respect to content expression ρ, let us first introduce the following order on coordinates. We say that coordinate (k,ℓ) precedes coordinate (k',ℓ') if we visit (k,ℓ) earlier than (k',ℓ') in a left-to-right top-down traversal of the cells of T, i.e., it precedes it in lexicographic order. Formally, $(k,\ell) < (k',\ell')$ if $k < k'$ or if $k = k'$ but $\ell < \ell'$.

Now, let T be a tokenized table, let z be a region of T, and let ρ be a content expression. Then (T,z) satisfies the content expression ρ *under the region-based semantics*, denoted $T, z \models_{\text{region}} \rho$ if there exist tokens $a_1,\ldots,a_n \in \Delta$ such that $a_1 \cdots a_n \in \mathcal{L}(\rho)$ and $a_i \in T_{c_i}$, where $c_1,\ldots,c_n$ is the enumeration in table order of all coordinates in z.

To define the row-based semantics we used in Section 2, we require the following notions. Let z be a region of T. We say that subregion $z' \subseteq z$ is a *row of* z if there exists some k such that $z' = \{(k,\ell) \mid (k,\ell) \in z\}$. Now, (T,z) satisfies the content expression ρ *under the row-based semantics*, denoted $T, z \models \rho$, if for every row z' of z, we have $T, z' \models_{\text{region}} \rho$.

Remark 3.5. Recall that, for ease of exposition, we allowed tables to be non-rectangular in Section 2 whereas in our formal model, tables are always rectangular. In particular, shorter rows are padded with $\perp$ to obtain rectangularity. This implies that, some content expressions of Section 2 need to be adapted in our formal model. For example, the rule `row(1) -> name` of Figure 4 needs to be adapted to $\mathsf{row}(1) \to \texttt{name}, \perp, \perp, \perp$ to take the padding into account. ■

4. EFFICIENT VALIDATION

In this section we consider the *validation* (or *evaluation*) problem for tabular schemas. This problem asks, given a tokenized table or tabular document T and a tabular schema R, whether T satisfies R. We consider the problem in a *main-memory* and *streaming* variant. Intuitively, T is given as a table in the former and as a tabular document in the latter setting.

4.1 Validation in Linear Time

When T is given as a tokenized table, we can essentially assume that we can navigate from a cell (i,j) to any of its four neighbours $\mathsf{up}(\{(i,j)\})$, $\mathsf{down}(\{(i,j)\})$, $\mathsf{left}(\{(i,j)\})$, and $\mathsf{right}(\{(i,j)\})$ in constant time. Under these assumptions we show that T can be validated against a tabular schema in linear time combined complexity.[7] The proof strongly relies on the known linear time combined complexity of propositional dynamic logic.

[7]Combined complexity is a standard complexity measure introduced by Vardi; see [32].

Theorem 4.1. *The evaluation problem for a tabular document T and a tabular schema R is in linear time combined complexity, that is, time $O(|T||R|)$.*

4.2 Streaming Validation

Even though Theorem 4.1 implies that Sculpt schemas can be efficiently validated, this only holds when the tabular document can be fully loaded in memory and multiple passes can be made through the document. However, when the input data is large it is sometimes desirable to have a *streaming validation algorithm* that makes only a single pass over the input tabular document and uses only limited memory. In this section we identify several fragments of core-Sculpt that admits such streaming validation algorithms.

Streaming model. Let us begin by defining when an algorithm validates in a streaming fashion. In this respect, we draw inspiration from the SAX Streaming API for XML: we can view a tokenized table T as a sequence of events generated by visiting the cells of T in table order. Here, whenever we visit a new cell, an event $\langle \mathsf{cell}\ \Gamma\rangle$ is emitted, with Γ the set of tokens in the visited cell. Whenever we move to a new row, an event of type $\langle \mathsf{new\ row}\rangle$ is emitted.

Note that the tokenized event stream can easily be generated "on the fly" when parsing a tabular document: we start reading the tabular document, one character at a time, until we reach a delimiter. All non-delimiter characters are used as input to, e.g., a finite state automaton that allows us to check which tokens match the current cell's content. When we reach a delimiter, a $\langle \mathsf{cell}\ \Gamma\rangle$ event is emitted with the corresponding set of matching tokens. If the delimiter is a row delimiter, then also a $\langle \mathsf{new\ row}\rangle$ is emitted. We repeat this until the end of the file.

Example 4.2. Consider the tabular document from Figure 1 together with the corresponding Sculpt schema S in Figure 2. The tokenized table of this document according to S yields the event stream

$$\langle \mathsf{cell}\ \emptyset\rangle\langle \mathsf{cell}\ \{\texttt{ARUA}\}\rangle\langle \mathsf{cell}\ \{\texttt{BOMBO}\}\rangle\langle \mathsf{cell}\ \{\texttt{ENTEBBE AIR}\}\rangle$$
$$\langle \mathsf{new\ row}\rangle\langle \mathsf{cell}\ \{\texttt{Timestamp}\}\rangle\langle \mathsf{cell}\ \{\texttt{Temperature}\}\rangle$$
$$\langle \mathsf{cell}\ \{\texttt{Temperature}\}\rangle\langle \mathsf{cell}\ \{\texttt{Temperature}\}\rangle\langle \mathsf{new\ row}\rangle\ldots$$

Definition 4.3 (Streamability). A tabular schema R is said to be *weakly streamable*, if there exists a Turing Machine M that
- can only read its input tape once, from left to right;
- for every tokenized table T, when started with the event stream of T on its input tape, accepts iff $T \models R$; and
- has an auxiliary work tape that can be used during processing, but it cannot use more than $O(m\log(n))$ of space on this work tape, where n is the total number of cells in T, and m the number of columns.

We say that R is *strongly streamable* if the Turing Machine M only requires $O(\log(n))$ space on its work tape.

Here, strong streamability corresponds to the commonly studied notions of streaming evaluation. We consider weak streamability to be very relevant as well because, based on the W3C use cases, tabular data often seems to be similar in spirit to relational tables and, in these cases, is very narrow and deep. In particular, $m = O(\log n)$ in these cases.

Weak streamability. To enable streaming validation, we restrict our attention to so-called *forward* coordinate and navigational expressions which are expressions where $\langle\alpha\rangle$ is

not allowed, and we never look up or left. That is, a coordinate or navigational expression is *forward* if it is generated by the following syntax.

$$\varphi, \psi := a \mid \mathsf{root} \mid \mathsf{true} \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \neg\varphi \mid \alpha(\varphi)$$
$$\alpha, \beta := \varepsilon \mid \mathsf{down} \mid \mathsf{right} \mid [\varphi] \mid (\alpha \cdot \beta) \mid (\alpha + \beta) \mid (\alpha^*)$$

We do not consider the operator $\langle\alpha\rangle$ in the forward fragment because it can be seen as a backward operator: $\langle \mathsf{right} \cdot [a]\rangle$ is equivalent to $\mathsf{left}(a)$.

A core-SCULPT schema is *forward* if it mentions only forward coordinate expressions.

THEOREM 4.4. *Forward core-*SCULPT *is weakly streamable.*

PROOF SKETCH. Consider a rule $\varphi \to \rho$ with φ a forward coordinate expression and ρ a content expression. We can show that coordinate expressions φ can be evaluated in a streaming fashion by constructing a special kind of finite state automaton (called *coordinate automaton*) that allows us to decide, at each position in the event stream, if the currently visited cell is in $[\![\varphi]\!]_T$. Whenever we find that this is the case, we apply the current cell contents to ρ (which we also evaluate by means of a finite state automaton). Now observe that $T \models \varphi \to \rho$ iff (1) under the row based semantics, whenever we see $\langle$new row$\rangle$, the automaton for ρ is in a final state and (2) under the region-based semantics, when we reach the end of the event stream, the automaton for ρ is in a final state. We then obtain weak streamability by showing that coordinate automata for φ can be simulated in space $O(m \log(n))$, whereas it is known that the finite state automaton for ρ can be simulated in constant space. □

Strong streamability. Forward core-SCULPT is unfortunately not strongly streamable: no schema with a rule that contains subexpressions of the form $\mathsf{col}(a)$ (which are prevalent in Section 2) can be strongly streamable. This can be seen using a simple argument from communication complexity. Indeed, assume that the first row has k cells, some of which have the token a and some of which do not. If we want to evaluate $\mathsf{col}(a)$ in a streaming fashion, we need to identify the cells in the second row that are in the same columns as the a-tokens in the first row. But, this is precisely the equality of two k-bit strings problem, which requires $\Omega(k)$ bits in deterministic communication complexity (Example 1.21 in [15]). These $\Omega(k)$ bits are what we need to store when going from the first to the second row. Since k can be $\Theta(n)$, this amount of space is more than we allow for strongly streamable tabular schemas, and hence no schema containing $\mathsf{col}(a)$ is strongly streamable.

The underlying reason why $\mathsf{col}(a)$ is not strongly streamable is because, in general, the token a can occur arbitrarily often. However, in all such cases in Section 2 and in the W3C use cases, the occurrences of a are very restricted. We could therefore obtain strong streamability for such expressions by adding constructs in the language that restrict how certain tokens can appear:

`unique(a)` `unique-per-row(a)`

The former asserts that token a should occur only once in the whole table and the latter that a occurs at most once in each row. More formally, the former predicate holds in a table T if $[\![a]\!]_T$ contains at most one element and the latter holds in table T if $[\![a]\!]_T$ contains at most one element of the form (r, c) for each row number r. Notice that a strong streaming algorithm can easily check whether these predicates hold.

We use the above predicates to define two notions of *guardedness* for region selection expressions. Guarded formulas will be strongly streamable. We say that token a is *row-guarded* if `unique-per-row(a)` appears in the schema. If `unique(a)` appears in the schema it is, in addition, also *guarded*. The two notions of guardedness capture the following intuition: if φ is row-guarded, then $\mathsf{down}(\varphi)$ is strongly streamable and if it is guarded, then $\mathsf{down}^*(\varphi)$ is strongly streamable. The main idea is that, in both cases, the number of cells we need to remember when going from one row to the next does not depend on the width of the table. We now define (row)-guardedness inductively on the forward language:

- root and true are guarded and row-guarded;
- $\mathsf{right}^*(\varphi)$ is guarded and row-guarded for every φ that does not contain a navigational subexpression;
- if φ, ψ, $\alpha(\varphi)$, $\beta(\varphi)$ are guarded (resp., row-guarded), then
  - $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varepsilon(\varphi)$, $\mathsf{down}(\varphi)$, $\mathsf{right}(\varphi)$,
  - $(\alpha \cdot \mathsf{down})(\varphi)$, $(\alpha \cdot \mathsf{right})(\varphi)$, and $(\alpha + \beta)(\varphi)$

  are guarded (resp., row-guarded);
- if φ and $\alpha(\psi)$ are guarded then $\mathsf{down}^*(\varphi)$ and $(\alpha \cdot \mathsf{down}^*)(\psi)$ are guarded; and
- if φ and $\alpha(\psi)$ are row-guarded then $\mathsf{right}^*(\varphi)$ and $(\alpha \cdot \mathsf{right}^*)(\psi)$ are guarded.

DEFINITION 4.5. A forward core-SCULPT schema is called *guarded*, if all region selection expressions that use the down-operator are row-guarded and all region selection expressions that use down^* are guarded.

Notice that guardedness of a SCULPT schema can be tested in linear time. Furthermore notice that every SCULPT schema in this paper becomes strongly streamable if we add the predicates `unique(a)` for tokens a that we use in expressions using col, down, or down^*.

THEOREM 4.6. *Guarded forward core-*SCULPT *is strongly streamable.*

5. SCULPT EXTENSIONS

Next, we describe a number of extensions to SCULPT. These include alternative grouping semantics, types, complex content cells, and a concept for a transformation language.

5.1 Region semantics

The examples in Section 2 all use a row-based semantics of SCULPT where the content expression is matched over every row in the selected region. That is, the cells in the selected region are 'grouped by' the row they occur in. There are of course other ways to group cells, by column, for instance, or by not grouping them at all. The latter case is already defined in Section 3 as *region-based semantics*. In SCULPT, we indicate rules using this semantics with a double arrow `=>` rather than a single arrow. Notice the difference between the rules `col(2) -> Null | Number` and `col(2) => (Null | Number)*`. Both require each cell in the second column to be empty or a number but express this differently. (The former way is closer to how one defines the schema of a table in SQL, which is why we chose it as a default.) Example 5.1 below describes a more realistic application of `=>`-rules. This

example corresponds to use case 12 in [29], is called *"Chemical Structures"* and aims to interpret Protein Data Bank (PDB) files as tabular data. This particular use case is interesting because it illustrates that the view of W3C on tabular data is not restricted to traditional comma-separated values files. We note that Theorems 4.1, 4.4, and 4.6 still hold if SCULPT schemas contain both rules under row-based and region-based semantics.

EXAMPLE 5.1. Figure 7 displays a slightly shortened version of the PDB file mentioned in use case 12 in [29]. The corresponding SCULPT schema could contain the following rules:

```
row(1) -> HEADER, Type, Date, ID
col(1) => HEADER, TITLE*, dots, EXPDATA, AUTHOR*,
          dots, REMARK*, dots, SEQRES*, dots, ATOM*
```

The last rule employs the region semantics and specifies the order in which tokens in the first column should appear. ■

5.2 Token types

The PDB fragment in Figure 7 contains cells that have the same content but seem to have a different meaning. It can be convenient to differentiate between cells by using *token types*. In the following fragment, `REMARK-Header` is the topmost cell containing `REMARK` in Figure 7, `REMARK-Comment` is the one immediately below, and `REMARK-Rest` is the rest:

```
%% Token types
%% left: name of the token type
%% right: region selection expression for token type
REMARK-Header <= down*[dots]/down[REMARK]
REMARK-Comment <= down*[dots]/down[REMARK]/down
REMARK-Rest <=
  down*[dots]/down[REMARK]/down/(down[REMARK])*
```

Note that we abbreviated rules of the form α(root) by α. We denoted the concatenation operator of navigational expressions by "/". We can now use token types to write rules such as

```
row(REMARK-Header) -> ...
row(REMARK-Comment) -> ...
row(REMARK-Rest) -> ...
```

Token types do not add additional expressiveness to the language since one can simply replace `REMARK-Header` by `down*[dots]/down[REMARK](root)` in the rule. But the ability to use different names for fields with the same content may be useful for writing more readable schemas. In this case, the names suggest that the block of remarks is divided into a header, some comment, and the rest.

5.3 Transformations and Annotations

While it is beyond the scope of this document to develop a transformation language for tables, we argue that region selection expressions can be easily employed as basic building blocks for a transformation language aimed at transforming tables into a variety of formats like, for instance, RDF, JSON, or XML (one of the scopes expressed in [34]). Region selection expressions are then used to identify relevant parts of a table.

Basic Transformations. Consider Figure 1 (of Example 2.1) again, where we see that several columns have the value −99.00. Since winter does not get this extreme in Uganda, this value is simply a dummy which should not be considered when computing, e.g., the average temperature in Uganda in 1935. Instead, for the fragment of Figure 1, it would be desirable to only select the columns that do not contain −99.00. To do this, we can simply define a new token and a new token type for the region of the table we are interested in.

```
Useless-Temp = -99.00

%% Token type
Useful <= col(1) or
          (Temperature and not Useless-Temp) or
          (row(1) and not up*(Useless-Temp))
```

The region defined by `Useful` contains

```
       , ENTEBBE AIR
1935.04,       27.83
1935.12,       25.72
1935.21,       26.44
[...]
```

which could then be exported. Using simple for-loops we can iterate over rows, columns, or cells, and compute aggregates. For example,

```
Useful-values <= (Temperature and not Useless-Temp)

For each column c in Useful-values {
  print Average(c)
}
```

would output `25.65`, the average of the values below `ENTEBBE AIR` in Figure 1. The region defined by `Useful-values` is a set of table cells, with coordinates. These coordinates can be used to handle information column-wise in the for-loop: It simply iterates over all column coordinates that are present in the region. Iteration over rows or single cells would work analogously.

Namespaces, Annotations and RDF. Assume that we want to say that certain cells in Figure 3 are geographical regions. To this end, the SCULPT schema could contain a definition of a default namespace:

```
namespace default =http://foo.org/nationalstats.csv
namespace x = [...]
```

Region selection expressions can then be used to specify which cells should be treated as objects in which namespace. For example, the code fragment

```
For each cell c in col(GeoArea) {
  c.namespace = default
}
```

could express that each cell below `GeoArea` is an entity in namespace `http://foo.org/nationalstats.csv`. So, the cell containing `England` represents the entity

`http://foo.org/nationalstats.csv:England,`

similar for the cell containing `Wales`, etc. (Here we assume that `.namespace` is a predefined operation on cells.)

We can also annotate cells with meta-information (as is currently being considered in Section 2.2 of [31]). The code fragment

```
For each cell c in col(GeoArea) {
  annotate c with "rdf:type dbpedia-owl:Place"
  annotate c with "owl:sameAs fbase:" + c.content
}
```

(assuming appropriate namespace definitions for rdf, owl, etc.) could express that each cell below `GeoArea` should be annotated with `rdf:type dbpedia-owl:Place` and, in addition, the `England` cell with `owl:sameAs fbase:England`, the

```
HEADER    EXTRACELLULAR MATRIX                    22-JAN-98   1A3I
TITLE     X-RAY CRYSTALLOGRAPHIC DETERMINATION OF A COLLAGEN-LIKE
TITLE    2 PEPTIDE WITH THE REPEATING SEQUENCE (PRO-PRO-GLY)
...
EXPDTA    X-RAY DIFFRACTION
AUTHOR    R.Z.KRAMER,L.VITAGLIANO,J.BELLA,R.BERISIO,L.MAZZARELLA,
AUTHOR   2 B.BRODSKY,A.ZAGARI,H.M.BERMAN
...
REMARK 350 BIOMOLECULE: 1
REMARK 350 APPLY THE FOLLOWING TO CHAINS: A, B, C
REMARK 350   BIOMT1   1  1.000000  0.000000  0.000000        0.00000
REMARK 350   BIOMT2   1  0.000000  1.000000  0.000000        0.00000
...
SEQRES   1 A    9  PRO PRO GLY PRO PRO GLY PRO PRO GLY
SEQRES   1 B    6  PRO PRO GLY PRO PRO GLY
SEQRES   1 C    6  PRO PRO GLY PRO PRO GLY
...
ATOM      1  N   PRO A   1       8.316  21.206  21.530  1.00 17.44           N
ATOM      2  CA  PRO A   1       7.608  20.729  20.336  1.00 17.44           C
ATOM      3  C   PRO A   1       8.487  20.707  19.092  1.00 17.44           C
ATOM      4  O   PRO A   1       9.466  21.457  19.005  1.00 17.44           O
ATOM      5  CB  PRO A   1       6.460  21.723  20.211  1.00 22.26           C
```

Figure 7: Fragment of a PDB file.

`Wales` cell with `owl:sameAs fbase:Wales`, etc. We assume that `annotate`, `with`, and `.content` are reserved words or operators in the language.

These ingredients also seem useful for exporting to RDF. We could write, e.g.,

```
print "@prefix : <http://foo.org/nationalstats.csv>"

For each cell c in col(GeoArea) {
 print ":"+c.content+"owl:sameAs fbase:"+c.content
}
```

to produce an RDF file that says that `:England` in the default namespace is the same as `fbase:England`. Looking at Figure 5, one can also imagine constructs like

```
RDF <= col(subject) or col(predicate) or col(object)

For each row r in RDF {
  print r.cells[1] +" "+ r.cells[2] +" "+ r.cells[3]
}
```

to facilitate the construction of RDF triples taking content from several cells.

5.4 Complex content

The CSV on the Web WG is considering allowing complex content (such as lists) in cells (Section 3.8 in [24]). Sculpt can be easily extended to reason about complex content. Our formal definition of tabular documents already considers (Section 3) a finite set of delimiters, which goes beyond the two delimiters (row- and column-) that we used until now.

In a spirit similar to region-based semantics, one can also imagine a subcell-based semantics, for example, a rule of the form

```
col(1) .> (String)*
```

could express that each cell in the first column contains a list of Strings. Notice the use of `.>` instead of `->` to denote that we specify the content of each individual cell in the region, instead of each row. The statement `List Delim = ;` in the beginning of the schema could say that the semicolon is the delimiter for lists within a cell.

6. CONCLUSIONS

We presented the schema language Sculpt for tabular data on the Web and showcased its flexibility and usability through a wide range of examples and use cases. While region selection expressions are at the very center of Sculpt, we think they can be more broadly applied. Region selection expressions can be used, for instance, as a cornerstone for annotation- and transformation languages for tabular data and thus for a principled approach for integrating such data into the Semantic Web. The whole approach of Sculpt is strongly rooted in theoretical foundations and, at the same time, in well established technology such as XPath. For these reasons, we expect the language to be very robust and, at the same time, highly accessible for users. The accessibility for users may greatly benefit from a XPath-like syntax for full-fledged region selection expressions, such as `right+::*[not(down*::dummy)]` for the first expression in Example 3.3. We leave the precise definition for such a syntax as future work. Two further prominent directions for future work are the following: (1) expand the usefulness of Sculpt by further exploring the extensions in Section 5; and, (2) study static analysis problems related to Sculpt and region selector expressions leveraging on the diverse box of tools from formal language theory and logic.

Acknowledgments

We are very grateful to Marcelo Arenas for bringing [31] to our attention.

7. REFERENCES

[1] R. W. Adam Retter, David Underdown. CSV schema 1.0: A language for defining and validating CSV data.

http://digital-preservation.github.io/csv-schema/csv-schema-1.0.html.

[2] M. Arenas, S. Conca, and J. Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *International World Wide Web Conference (WWW)*, pages 629–638, 2012.

[3] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0, 2007. W3C Recommendation, January 2007.

[4] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. In *International World Wide Web Conference (WWW)*, pages 825–834, 2008.

[5] G. J. Bex, W. Martens, F. Neven, and T. Schwentick. Expressiveness of XSDs: from practice to theory, there and back again. In *International World Wide Web Conference (WWW)*, pages 712–721, 2005.

[6] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[7] S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF mapping language. W3C Recommendation, September 2012.

[8] D. Fallside and P. Walmsley. XML Schema Part 0: Primer (second edition). W3C Recommendation, October 2004.

[9] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.

[10] J. E. F. Friedl. *Mastering Regular Expressions*. O'Reilly Media, 3rd edition edition, 2006.

[11] W. Gelade and F. Neven. Succinctness of pattern-based schema languages for XML. *J. Comput. Syst. Sci.*, 77(3):505–519, 2011.

[12] Google. DSPL: Dataset publishing language. https://developers.google.com/public-data/. Last accessed 04/11/2014.

[13] L. Han, T. Finin, C. S. Parr, J. Sachs, and A. Joshi. RDF123: from spreadsheets to RDF. In *The Semantic Web - ISWC 2008*, volume 5318 of *LNCS*, pages 451–466. Springer, 2008.

[14] V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly pushdown automata for streaming XML. In *International World Wide Web Conference (WWW)*, pages 1053–1062, 2007.

[15] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[16] O. K. F. Labs. Tabular data package. http://dataprotocols.org/tabular-data-package/. Version 1.0-beta-2. Last accessed 04/11/2014.

[17] L. Libkin, W. Martens, and D. Vrgoc. Querying graph databases with XPath. In *International Conference on Database Theory (ICDT)*, pages 129–140, 2013.

[18] K. Losemann and W. Martens. The complexity of evaluating path expressions in SPARQL. In *International Symposium on Principles of Database Systems (PODS)*, pages 101–112, 2012.

[19] W. Martens, F. Neven, M. Niewerth, and T. Schwentick. Developing and analyzing XSDs through BonXai. *PVLDB*, 5(12):1994–1997, 2012.

[20] W. Martens, F. Neven, T. Schwentick, and G. Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3):770–813, 2006.

[21] W. Martens, F. Neven, and S. Vansummeren. SCULPT: A schema language for tabular data on the web. http://arxiv.org/abs/1411.2351.

[22] V. Mulwad, T. Finin, and A. Joshi. Semantic message passing for generating linked data from tables. In *The Semantic Web (ISWC 2013)*, volume 8218 of *LNCS*, pages 363–378. Springer, 2013.

[23] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3), 2009.

[24] R. Pollock and J. Tennison. Metadata vocabulary for tabular data. Technical report, World Wide Web Consortium (W3C), July 2014. www.w3.org/TR/2014/WD-tabular-metadata-20140710/.

[25] E. Prud'hommeaux, J. E. L. Gayo, and H. Solbrig. Shape expressions: An RDF validation and transformation language. In *International Conference on Semantic Systems*, 2014.

[26] A. G. Ryman, A. L. Hors, and S. Speicher. OSLC resource shape: A language for defining constraints on linked data. In *WWW Workshop on Linked Data on the Web*, 2013.

[27] L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *International Conference on Database Theory (ICDT)*, pages 299–313, 2007.

[28] L. Segoufin and V. Vianu. Validating streaming XML documents. In *International Symposium on Principles of Database Systems (PODS)*, pages 53–64, 2002.

[29] J. Tandy, D. Ceolin, and E. Stephan. CSV on the Web: Use cases and requirements. Technical report, World Wide Web Consortium (W3C), October 2014. http://w3c.github.io/csvw/use-cases-and-requirements/.

[30] J. Tennison. 2014: The year of CSV. http://theodi.org/blog/2014-the-year-of-csv. last accessed 04/11/2014.

[31] J. Tennison and G. Kellogg. Model for tabular data and metadata on the web. Technical report, World Wide Web Consortium (W3C), July 2014. www.w3.org/TR/2014/WD-tabular-data-model-20140710/.

[32] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.

[33] P. Venetis, A. Y. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *PVLDB*, 4(9):528–538, 2011.

[34] W3C. CSV on the web working group charter. http://www.w3.org/2013/05/lcsv-charter.html.