Satisfiability for SCULPT-Schemas for CSV-Like Data

Johannes Doleschal
Universität Bayreuth, Bayreuth, Germany

Wim Martens
Universität Bayreuth, Bayreuth, Germany

Frank Neven
Hasselt University and transnational University of Limburg, Hasselt, Belgium

Adam Witkowski
University of Warsaw, Warsaw, Poland

Abstract

SCULPT is a simple schema language inspired by the recent working effort towards a recommendation by the World Wide Web Consortium (W3C) for tabular data and metadata on the Web. In its core, a SCULPT schema consists of a set of rules where left-hand sides select sets of regions in the tabular data and the right-hand sides describe the contents of these regions. A document (divided in cells by row- and column-delimiters) then satisfies a schema if it satisfies every rule. In this paper, we study the satisfiability problem for SCULPT schemas. As SCULPT describes grid-like structures, satisfiability obviously becomes undecidable rather quickly even for very restricted schemas. We define a schema language called L-SCULPT (Lego SCULPT) that restricts the walking power of SCULPT by selecting rectangular shaped areas and only considers tables for which selected regions do not intersect. Depending on the axes used by L-SCULPT, we show that satisfiability is PTIME-complete or undecidable. One of the tractable fragments is practically useful as it extends the structural core of the current W3C proposal for schemas over tabular data. We therefore see how the navigational power of the W3C proposal can be extended while still retaining tractable satisfiability tests.

2012 ACM Subject Classification Information systems → Semi-structured data, Theory of computation → Formal languages and automata theory, Theory of computation → Logic

Keywords and phrases CSV, Schema languages, Semi-structured data

Digital Object Identifier 10.4230/LIPIcs.ICDT.2018.14

Funding This work is supported by grant number MA 4938/2–1 from the Deutsche Forschungsgemeinschaft.

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

[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 [14].

	1	2	3	4	5	6	7	8	9	10
1	subject	predicate	object	provenance						
2	:e4	type	PER							
3	:e4	mention	"Bart"	D00124	283-286					
4	:e4	mention	"JoJo"	D00124	145-149	0.9				
5	:e4	per:sibling	:e7	D00124	283-286	173-179	274-281			
6	:e4	per:age	"10"	D00124	180-181	173-179	182-191	0.9		
7	:e4	per:parent	:e9	D00124	180-181	381-380	399-406	D00101	220-225	230-233

Figure 1 Fragment of a CSV-like file (added row and column numbers), inspired by use case 13 in [13].

	1	2	3	4	5	6	7	8	9	10
1	subj	pred	obj	prov						
2	iri	pred-type	ent-type							
3	iri	pred-type	literal	doc-id	position					
4	iri	pred-type	literal	doc-id	position	certainty				
5	iri	iri	iri	doc-id	position	position	position			
6	iri	iri	literal	doc-id	position	position	position	certainty		
7	iri	iri	iri	doc-id	position	position	position	doc-id	position	position

Figure 2 Tokenized version of Figure 1, with added row and column numbers.

to several TBs. Despite these advantages, working with CSV files is often cumbersome [14] since 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. In fact, without schema information, already converting CSV-like data into a relational database is a challenging engineering problem [14]. In recognition of this problem, the *CSV on the Web* Working Group of the World Wide Web Consortium (W3C) [17] 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. Inspired by the recent W3C effort towards a recommendation for tabular data and metadata on the Web, Martens et al. proposed the tabular schema language SCULPT [11]. At its core, SCULPT is a rule-based language with rules of the form $\varphi \to \rho$ where φ selects a set of regions[2] of the input table and ρ constrains the allowed structure and content of each such region. The region selection expressions φ are not limited to selecting columns but can navigate through a table, much like XPath expressions can navigate the nodes of an XML tree. This generalization beyond columns is necessary since there are natural cases in practice in which CSV-like data is not rectangular [2, 13] (see also Figure 1). In this paper, we address static optimization of SCULPT schemas, but first we present the main ideas behind the language by means of an example.

SCULPT schemas operate on *tabular documents* which are text files describing tabular data. Figure 1, coming from use case 13 in [13], shows an example CSV file, to which we added row numbers 1–7 on the left and column numbers 1–10 at the top. The original file uses tab (`\t`) as a column delimiter and newline (`\n`) as row delimiter. The rows and columns divide the document into *cells*. In this example, rows can have different numbers of cells, e.g., row two has three cells, whereas row three has five. The W3C describes the data as coming from a text extraction scenario, where "facts [are extracted] from text and [represented] as [RDF] triples along with associated metadata that include provenance and certainty values" [13]. Furthermore, "a single row in the table comprises a triple (subject-predicate-object), one or more provenance references and an optional certainty measure" [13]. In Figure 1, we see

[2] A region is a set of cells.

that the provenance information includes a document ID (e.g., the value D00124), pairs of string offsets within the document (e.g., 283–286), and an optional float representing a certainty measure (e.g., 0.9). This information can be repeated for several documents as is the case in row seven. There may not be an a priori bound on the number of columns that are needed for representing the provenance information. As we can infer from the W3C's textual description of the data, the logical organization of the data from column four to the right is in rows rather than in columns. The current W3C proposal for schemas [12, Section 5.5] does not deal with row-wise organization (and not even with different types of data in the same column) and therefore cannot adequately describe the data in their own use case 13. As we will see, SCULPT can capture the logic inherent in this example by describing the structure of the rectangle starting at cell (3,4) in the rule (†).[3]

Figure 3 shows an example SCULPT schema for CSV files of the form as depicted in Figure 1. It consists of two parts. The first part concerns *parsing information* – it defines the row and column delimiters and describes how cells should be tokenized. This allows to parse the text file and build a table-like structure consisting of rows and columns. Tokenization then proceeds as follows. The content of each cell is matched against the regexes in the schema's token rules. To each cell the first token is assigned for which the corresponding regex matches.[4] For instance, cell (4,3) in Figure 1 gets the token `literal` because it matches its regex `"[a-zA-Z0-9]*"` and none of the earlier regexes. Figure 2 depicts the tokenized CSV resulting from Figure 1, using the schema in Figure 3. The second part of the schema consists of *structural rules*. Left-hand sides select a set of regions, whereas right-hand sides are regular expressions that the tokens in each region should match. Consider the rule

```
row((1,1)) -> subj, pred, obj, prov
```

whose left-hand side selects one region (the row of the tokenized table starting at (1,1)) and requires it to match the regular expression `subj, pred, obj, prov` where the comma stands for concatenation. In our example, the right-hand side expressions always describe what *each row* in the selected region should look like. This is mostly important for the rule

```
rectangle(prov +(1,0)) -> (doc-id, position*, certainty?)*.        (†)
```

where the left-hand side selects an unbounded rectangular region for which the top left corner is the node matching the token `prov`, plus an offset $(1, 0)$, i.e., one row, zero columns. Each row in the selected rectangle should then match the expression

```
(doc-id, position*, certainty?)*
```

which it does in the example, as we can see in Figure 2. The language SCULPT is formally defined in Section 3.

In this paper, we study static optimization of SCULPT schemas. In particular, we address the satisfiability problem that asks whether for a given a SCULPT schema there is a CSV file that satisfies it. Not only is satisfiability a core problem in the foundations of database management field that has been studied in depth for a variety of formalisms, it is also particularly relevant for schema design as it allows to detect schemas that are not well-defined.

[3] We use coordinate (x, y) to refer to the cell in row x, column y.
[4] There are other options to assign tokens, e.g., as in [11], but tokenization is not our present focus.

```
%  Parsing information; Delimiters
Col Delim  = \t                                        Row Delim  = \n

% Token rules of the form <token name> = <regex>
subj       = subject                                   pred       = predicate
obj        = object                                    prov       = provenance
iri        = [a-zA-Z0-9]*:[a-zA-Z0-9]*                 pred-type  = type + mention
doc-id     = D[0-9]{5}                                 position   = [0-9]{3}-[0-9]{3}
certainty  = 0.[0-9] + 1.0                             literal    = "[a-zA-Z0-9]*"
ent-type   = PER + ORG + GPE

% Structural rules
row((1,1)) -> subj, pred, obj, prov                    col(subj)  -> iri*
col(obj)   -> (literal + iri + entity-type)*           col(pred)  -> (pred-type + iri)*
rectangle(prov +(1,0)) -> (doc-id, position*, certainty?)*
```

Figure 3 Schema for files of the type in Figure 1. (Syntax uses two columns to save space.)

Unsurprisingly, satisfiability of SCULPT quickly turns out to be undecidable, which we show by an easy reduction from the domino tiling problem [16]. Indeed, using only one rule, a region selection expression can be used to 'walk' over a grid testing all horizontal and vertical constraints, or alternatively many much simpler rules can be used to test all horizontal and vertical constraints in parallel for every domino type (cf. Section 4 for more details). Even though these observations are valid to demonstrate undecidability they use rather artificial constructions.

For this reason, we introduce a restricted variant of SCULPT called Lego SCULPT (L-SCULPT) that not only suffices to express the W3C use cases but also admits tractable satisfiability. In brief, L-SCULPT restricts region selection expressions to only select rectangular shaped areas, that is (parts of) rows, columns, and rectangles, thereby constraining the structural power of the language. A second restriction is that L-SCULPT only considers tables on which no two selected regions intersect. Specifically, in this paper, we make the following contributions:

1. We show that the safisfiability problem for the structural core of SCULPT is undecidable.
2. We define a fragment of SCULPT called L-SCULPT that is powerful enough to capture the structural core of the schemas for tabular data in the current W3C recommendation [12, Section 5.5]. Intuitively, L-SCULPT allows selections of rows, columns, and rectangles and bounded-size regions in the directions up, left, down, and right, whereas the W3C's recommendation only allows column selection.[5] As Figure 1 shows, column selection alone is too limited to describe schemas for the data fragments in the W3C's use cases, because the number of columns that the schema can describe is bounded by the number of rules in the schema. In the example, the number of columns can grow arbitrarily large. L-SCULPT strictly extends the structural core of the W3C recommendation.
3. Depending on which axes are used, we show that satisfiability of L-SCULPT is PTIME-complete or undecidable. Our main technical result shows that for L-SCULPT using only row, column, right, and rectangle selections satisfiability is in PTIME. The proof is an intricate reduction to the emptiness problem of nondeterministic tree automata where tables are encoded as trees.

[5] We only focus on the structural core of the languages. The W3C's proposal also supports key and foreign key constraints, which are out of scope here but easy to add to the language. (In fact, we implemented them in [3].)

4. Even the tractable fragments of L-SCULPT extend the structural core of the current W3C schema recommendation and are expressive enough to define a natural schema for Figure 1, one example is the schema in Figure 3. As such, our result shows how the W3C recommendation can be extended without making satisfiability intractable.

Related Work

Tabular or CSV-like data is one of many popular models for semi-structured data [1]. The schema language SCULPT was introduced in [11]. This work provides an initial formal model and considers efficient evaluation. In addition, several extensions like region semantics, token types, and transformations are considered. We implemented the system CHISEL for specifying, validating, analyzing, and debugging of SCULPT schemas and data transformations based on schema information [3]. Arenas et al. [2] propose a simple and expressive framework for adding metadata to CSV documents. They focus in particular on noisy CSV-like documents and consider the problem of annotating different elements of CSV-like files such as, for instance, cells, rows and columns. Documents are viewed as strings and regular expressions are used to select intervals. Navigation is restricted to moving to the next delimiter (any delimiter or one of a specific kind). They consider satisfiability as well as efficient evaluation. As the setting is considerably different from the one considered in this paper (grids versus strings), their and our results do not imply each other.

Labeled grids have been studied in the context of two-dimensional languages, also referred to as picture languages (cf., e.g., [5]). The bulk of the research in this area has focused on formalisms that could capture natural counterparts of string language theory, like regular languages, context-free languages, closure properties, etc. For instance, the equivalence of existential monadic-second order logic, complementation-free regular expressions, tiling systems (as projections of local languages) and two-dimensional online tessallation automata, provided enough motivation to refer to the latter class as the recognizable two-dimensional languages. Satisfiability for this class is shown to be undecidable through a direct simulation of Turing Machines [5]. Proposition 4 uses essentially the same idea but employs tiling systems. Although the proofs of Proposition 4 and Proposition 5 are rather straightforward and their novelty is limited, they do provide the necessary motivation for the introduction of L-SCULPT.

In Section 2, we introduce the necessary preliminaries. In Section 3, we formalize the structural core of SCULPT and show that the satisfiability problem is undecidable. In Section 4, we define L-SCULPT. In Section 5, we present our main technical result showing that satisfiability for L-SCULPT using only row, col, right and rectangle selections is PTIME-complete. In addition, we show that various extensions are undecidable. We conclude in Section 6.

2 Preliminaries

For numbers $n, m \in \mathbb{N}$, with $n < m$, we denote the sets $\{1, \ldots, n\}$ and $\{n, \ldots, m\}$ by $[n]$ and $[n, m]$, respectively. By Δ we denote an *alphabet*, that is, a finite, non-empty set of *symbols*. A *language* is a set of words over Δ. We assume familiarity with regular expressions but briefly describe their notation. The *regular expressions* over Δ are inductively defined as follows. Every symbol $a \in \Delta$ is a regular expression, and so is the special symbol ε, which denotes the empty word and which we assume not to be in Δ. If e_1 and e_2 are regular expressions, then so are $e_1 \cdot e_2$, $e_1 + e_2$, and e_1^*. We assume the usual precedence of operators.

The language $L(e)$ of e is defined as usual. We sometimes omit the concatenation symbol "$\cdot$", write e^+ to abbreviate ee^*, and write $e?$ to abbreviate $e + \varepsilon$.

Tables

CSV-like data consists of a text file with row and column delimiters (often newline and comma, respectively). These delimiters uniquely determine a tabular structure that can be given to the data, as we describe next. The main idea is very simple: if the file has n row delimiters, the table has $n + 1$ rows (a row delimiter is a separator between two consecutive rows). Likewise, if the file has m column delimiters in row i, then row i in the table has $m + 1$ cells. The main idea is visualised in Figure 4, which we revisit later.

We use a set $e = \{\sqcup, \triangleleft\}$ of special symbols that do not appear in any other set unless explicitly mentioned otherwise. We use $\sqcup$ to denote empty cells in the CSV file and $\triangleleft$ to denote cells that do not exist in the CSV file.[6]

If we denote a set by a single symbol (say, $\mathcal{V}$), we always assume that it does not contain any symbol from e. We use the following notation: $\mathcal{V}^e = \mathcal{V} \cup \{\sqcup, \triangleleft\}$ and $\mathcal{V}_x = \mathcal{V} \cup \{x\}$ for every symbol $x \in e$.

Let $\mathcal{V}$ be a set and $n, m \in \mathbb{N}$. A *matrix* M over $\mathcal{V}^e$ is a mapping from $[n] \times [m]$ to $\mathcal{V}^e$. We say that M has n *rows* and m *columns*. A *cell* is determined by its coordinate $(k, \ell) \in [n] \times [m]$ and its *content* is the value $M(k, \ell)$. We usually denote the later value as $M_{k,\ell}$. We denote the set $[n] \times [m]$ of all *coordinates* of M by $\mathrm{Coords}(M)$. A *region* in M is a set of coordinates, that is, a subset of $[n] \times [m]$.

▶ **Definition 1** (Core Tabular Data Model, [11, 15])**.** A *table* T *over* $\mathcal{V}^e$ is a matrix over $\mathcal{V}^e$ that satisfies the requirement that whenever $T_{i,j} = \triangleleft$ then $T_{i,j'} = \triangleleft$ for all $j' > j$ (i.e., $\triangleleft$ is only used for padding to the right).

The purpose of the $\triangleleft$ symbol is just to indicate how many cells there are in a row in the underlying CSV-like text file. The text file in Figure 4 has two row separators ($\hookleftarrow$), so the corresponding table has three rows. The number of columns in the table is the maximal number of column separators (comma) in a row of the text file, plus one. The first row of the CSV-like file has three cells, for which the first two are empty (do not even contain whitespace). We use $\sqcup$ to denote this in the table. Furthermore, the last cell in the first row in the table is labeled $\triangleleft$ to indicate that this cell does not exist in the underlying text file on the left. (Since rows are left-aligned in CSV-like files, there can only be $\triangleleft$ to the right of $\triangleleft$.) The second row in the text file on the left has one cell, which is empty. The third and final row has four cells. The last cell contains $\sqcup$ since the column separator is the last symbol of the CSV-like text file.

We assume the natural *table order* on coordinates. That is, we say that coordinate (k, ℓ) precedes coordinate (k', ℓ') (denoted $(k, \ell) < (k', \ell')$) if (k, ℓ) precedes (k', ℓ') lexicographically, that is: either (1) $k < k'$ or (2) $k = k'$ and $\ell < \ell'$. When depicting tables, we always put cell (1,1) on the top left.

[6] These symbols play a similar role as the "end of tape" marker and the blank symbol in some definitions of Turing Machines. Here we only focus on $\sqcup$ and $\triangleleft$ and their correspondence to the underlying CSV-like text files.

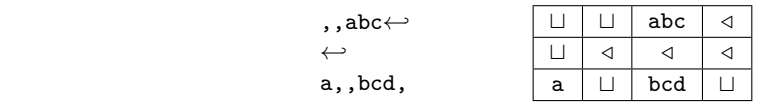


Figure 4 A CSV-like text file (left) and its corresponding tabular representation (right).

3 A Structural Core of SCULPT

We present a formal model for the structural core of SCULPT. In our formalization, we will work with tables corresponding to *tokenized* CSV files as exemplified in Figure 2. Formally, a SCULPT schema is a tuple $S = (\Delta^e, R)$ where

- Δ^e is a finite set of elements which we call *tokens*, and
- R is a set of *structural rules* of the form $s \to \rho$ that constrain the admissible table content:
  - s is a *region selection expression* that maps every table over Δ^e to a set of regions; and,
  - ρ is a *content expression* that defines the permitted content of regions selected by s.

The schema in Figure 3 has, in addition to the structural rules R, a set of *token rules* that associate tokens to cell contents. We omitted these here because we focus on the structural core of the language. In what follows we just use the term *rules* to refer to *structural rules*.

In brief, a SCULPT schema defines a set of tables over Δ^e. Intuitively, such tables should satisfy all rules $s \to \rho$ in the schema. Let T be a table over Δ^e and z be a region of T. In the different versions of SCULPT that we consider, we will define when z satisfies ρ, which we denote by $z \models \rho$. The region selection expressions s, when applied to a table T, returns a *set of regions*, i.e., $[\![s]\!]_T$ is a subset of $2^{\mathrm{Coords}(T)}$.

▶ **Definition 2.** A table T over Δ^e *satisfies* a SCULPT schema $S = (\Delta^e, R)$, denoted $T \models S$, if $z \models \rho$ for every rule $s \to \rho$ in R and $z \in [\![s]\!]_T$.

We now define the region selection and content expressions for SCULPT. The full language SCULPT will only be used in this section, where the main goal is to understand which properties of SCULPT make satisfiability undecidable. In Section 4, we introduce L-SCULPT which avoids these properties and for which satisfiability becomes tractable for some fragments.

Region Selection Expressions

We now define *region selection expressions* for the most general schemas we consider. Intuitively, a region selection expression is of the form $f(\varphi)$ where φ is a formula that returns a region z and f is an operator in $\{\mathsf{region}, \mathsf{rows}\}$ that maps regions to sets of regions.[7] The formulas φ in SCULPT are formulas in propositional dynamic logic (PDL for short) [4], tweaked for navigation over grids. We refer to Appendix A for details. Then, for a region z, we define $\mathsf{region}(z)$ as $\{z\}$ and $\mathsf{rows}(z)$ as the set of rows in z, more specifically $\mathsf{rows}(z) = \{\{(i,j) \in z \mid j \in \mathbb{N}\} \mid i \in \mathbb{N}\}$.[8]

[7] In [11, 3], this operator is encoded in rules by using different arrows: rules with => use $f = \mathsf{region}$ and rules with -> use $f = \mathsf{rows}$. We use the same convention in Figure 3, where $f = \mathsf{row}$ for every rule.

[8] SCULPT as defined in [11] only uses regions and rows. We also implemented cols [3], that cuts z into its set of columns (and can be defined analogously), but do not use it in the present paper.

Content Expressions

As content expressions, we simply use regular expressions over Δ^e. Let T be a table, let z be a region of T, and let ρ be a content expression. Then z *satisfies* ρ *(denoted* $z \models \rho$*)* if there exist tokens $a_1, \ldots, a_n \in \Delta^e$ such that $a_1 \cdots a_n \in L(\rho)$ and $a_1 \cdots a_n$ is the enumeration of all tokens in z in table order. Recall that Definition 2 now implies that $T \models (\Delta^e, R)$ if, for every rule $f(\varphi) \to \rho$ in R, we have that $z \models \rho$ for every $z \in f(Z)$, where Z is the region selected by φ in T.

Decision Problems

We recall that validation of SCULPT schemas is in linear time:

▶ **Theorem 3** ([11]). *Given a table T and* SCULPT *schema S, testing if $T \models S$ can be done in time $O(|T| \cdot |S|)$.*

In this paper, we study satisfiability problems for SCULPT. The most straightforward variant is defined as follows:

Problem:	SAT
Input:	A SCULPT schema S.
Question:	Is there a table T such that $T \models S$?

In its full generality, SAT is easily seen to be undecidable. The proof is a simple reduction from Domino Tiling where only *one* region selection expression is used to 'walk' over the grid checking all horizontal and vertical constraints.

▶ **Proposition 4.** SAT *is undecidable for* SCULPT*, even if schemas use only one rule that selects only one cell.*

We note that a similar result was obtained by Göller et al. [7, Theorem 4.11], where satisfiability of PDL with restricted negation was shown to be undecidable. The main difference is that Göller et al. consider infinite satisfiability whereas we consider finite satisfiability. Göller et al. encode the grid structure in their formula but, from there on, their proof and ours use a similar main idea.

Restricting the 'walking' power of region selection expression does not suffice for decidability as the next proposition shows. Again, a reduction from Domino Tiling is used but now the schema needs to select 'intersecting' regions: every cell containing a certain domino is in a selected region that checks the horizontal constraints and another region that tests the vertical constraints for this domino.

▶ **Proposition 5.** SAT *is undecidable for* SCULPT*, even if rules only use left-hand-sides that select "the row of cell (1,1)", "the column of cell (1,1)", "the cell(s) to the immediate right of a given token", and "the cell(s) immediately below a given token".*

4 Lego SCULPT

The use cases put forward by W3C [13] do not use powerful 'walking' expressions or select 'intersecting' regions as is done in the proofs of Propositions 4 and 5. As in addition the schemas used in the proofs of the mentioned propositions are rather artificial, we introduce a restricted variant of SCULPT called Lego SCULPT (L-SCULPT). From a structural perspective, this language is still more powerful than the W3C's proposal for a schema

language, can more accurately describe the data in the W3C use cases (see Figures 1–3), and admits tractable satisfiability. In brief, L-SCULPT restricts region selection expressions to select rows, columns, and rectangles. A second restriction is that L-SCULPT only considers tables on which no two selected regions intersect.[9] Formally, an L-SCULPT schema $S = (\Delta^e, R)$ is a pair as before but with some restrictions that we explain next.

Region Selection Expressions

We first discuss the region selection expressions occurring as left-hand sides of rules in R:

$$s \quad := \quad c \mid \mathsf{up}(d) \mid \mathsf{down}(d) \mid \mathsf{left}(d) \mid \mathsf{right}(d) \mid \mathsf{row}^\bullet(d) \mid \mathsf{col}^\bullet(d) \mid \mathsf{rect}(d+o)$$

Here, $\bullet \in \{*, +\}$, c is a coordinate, d is a coordinate or a token (different from $\triangleleft$ or $\sqcup$), and $o \in \{0,1\} \times \{0,1\}$ is an offset. We allow offsets in rectangles for flexibility. In Figure 3, it is convenient to use the offset (1,0), for example. For the definition of $[\![s]\!]_T$, we use the following shorthand: If c is a coordinate and a is a token, we write $c \models a$ if $T_c = a$. Additionally, we define $c \models c$ for each coordinate c of T. Furthermore, we denote by R_T the region consisting of all cells of T. Then, $[\![s]\!]_T$ defines a set of regions as follows:[10]

$$\begin{aligned}
[\![c]\!]_T &:= \{\{c\}\} \\
[\![\mathsf{rect}(d+o)]\!]_T &:= \{\{c + o + (k,\ell) \mid k,\ell \in \mathbb{N}\} \cap R_T \mid c \models d\} \\
\\
[\![\mathsf{up}(d)]\!]_T &:= \{\{(i-1,j)\} \cap R_T \mid (i,j) \models d\} \\
[\![\mathsf{down}(d)]\!]_T &:= \{\{(i+1,j)\} \cap R_T \mid (i,j) \models d\} \\
\\
[\![\mathsf{left}(d)]\!]_T &:= \{\{(i,j-1)\} \cap R_T \mid (i,j) \models d\} \\
[\![\mathsf{right}(d)]\!]_T &:= \{\{(i,j+1)\} \cap R_T \mid (i,j) \models d\} \\
\\
[\![\mathsf{row}^*(d)]\!]_T &:= \{\{c + (0,k) \mid k \in \mathbb{N}\} \cap R_T \mid c \models d\} \\
[\![\mathsf{row}^+(d)]\!]_T &:= \{\{c + (0,k+1) \mid k \in \mathbb{N}\} \cap R_T \mid c \models d\} \\
\\
[\![\mathsf{col}^*(d)]\!]_T &:= \{\{c + (k,0) \mid k \in \mathbb{N}\} \cap R_T \mid c \models d\} \\
[\![\mathsf{col}^+(d)]\!]_T &:= \{\{c + (k+1,0) \mid k \in \mathbb{N}\} \cap R_T \mid c \models d\}
\end{aligned}$$

Again, c is a coordinate, d is a coordinate or token, and o an offset. We give some intuition. Rule $\mathsf{col}^*(c)$ selects a singleton region consisting of c and all cells below c. Rule $\mathsf{row}^+(a)$ selects a set of regions, namely, for each cell c with token a, the region having all cells to the right of c. The rule $\mathsf{right}(a)$ contains, for each cell c with token a, the region $\{c + (0,1)\}$. Finally, $\mathsf{rect}(c + (1,0))$ contains the set of rows starting in coordinates below c.

We refer to rules with a coordinate in their left-hand side as *coordinate rules* and to the other rules as *token rules*. We say that the expressions of the form rect are *rectangular*. Although rect selects sets of rows, the terminological intuition is the following: since the rows we select are consecutive and all start in the same column, their union in T is always rectangular.

Observe that, in the case of coordinate rules, $\mathsf{row}^+((x,y))$ is syntactic sugar for $\mathsf{row}^*((x, y+1))$, whereas $\mathsf{row}^*((1,1))$ ("select the first row") cannot be expressed with row^+. In the

[9] The restriction to brick-like regions together with the disjointness requirement explains why we refer to this fragment as Lego SCULPT.

[10] For simplicity, we do not make use of a 'slice' operator f as in Section 3, but rather defined the set of regions directly.

case of token rules, although one can use $\mathsf{row}^*(a)$, it introduces redundancy: its content expression will have to repeat that the first cell in the region contains a. In the remainder of the paper, we therefore do not consider $\mathsf{row}^+(c)$ or $\mathsf{row}^*(a)$. Furthermore, we just write $\mathsf{row}(c)$ for $\mathsf{row}^*(c)$ and $\mathsf{row}(a)$ for $\mathsf{row}^+(a)$ and follow the same conventions for columns. The rules in Figure 3 are all in L-SCULPT, with the semantics as define here.

We assume in the sequel that the coordinates c are encoded in unary. We feel that this assumption is reasonable because we have not yet encountered data for which large coordinates are needed. In W3C schemas for tabular data, such numbers (the number of columns of the data) are encoded in unary as well.

Content Expressions

Content expressions define the allowed content of cells but can no longer force cells in the underlying CSV-file to be missing. We therefore restrict content expressions in L-SCULPT, by disallowing the explicit use of $\triangleleft$, i.e., content expressions are now regular expressions over $\Delta_{\sqcup}$. In addition, when matching a region against a content expression, we allow arbitrarily long padding at the end. Formally, for a content expression ρ, denote by $L^e(\rho)$ the language $L(\rho \cdot \sqcup^* \cdot \triangleleft^*)$. We say that a region z *satisfies* ρ (denoted $z \models \rho$) if $a_1 \cdots a_n \in L^e(\rho)$, where $a_1, \ldots, a_n$ are the tokens in z, in table order.

Padding with $\triangleleft$ allows us to take non-rectangular CSV-like files into account. Padding with $\sqcup$ ensures that content expressions cannot "force" a row in a CSV-file to be short. Indeed, consider the following contrived example with a rule $\mathsf{right}(a) \to \varepsilon$. If we would define $L^e(\rho) = L(\rho \cdot \triangleleft^*)$ then the content of the cell to the right of a would need to be $\triangleleft$, therefore forcing the cell in the underlying CSV data to be missing. In our definition, we allow the cell to be missing or empty.

Region Disjointness

Let $S = (\Delta^e, R)$ be an L-SCULPT schema and T a table. Intuitively, we say that S is *region-disjoint* on T if all regions selected by S are pairwise disjoint. Formally, S is *region-disjoint* on T if, for every pair of rules $r_1 = s_1 \to \rho_1$ and $r_2 = s_2 \to \rho_2$ and every pair of regions $z_1 \in [\![s_1]\!]_T$ and $z_2 \in [\![s_2]\!]_T$, if $z_1 \cap z_2 \neq \emptyset$, then $r_1 = r_2$ and $z_1 = z_2$. Finally, we say that $T \models S$ if S is region-disjoint on T and $r \models \rho$ for every rule $s \to \rho$ in R and for *every* region $r \in [\![s]\!]_T$.

Notice that, given a table T and schema S, it is easy to check whether T is region-disjoint on S. This can be checked during evaluation, which is in in time $O(|T| \cdot |S|)$, even for full SCULPT (Theorem 3).

Recall that we introduced region-disjointness to avoid the undecidability problems in Proposition 5. An alternative, more restrictive way of introducing region-disjointness would be to require that a schema is in L-SCULPT only if it is region-disjoint on *every* table for which the rules match. But as we discuss next, this severely limits the power of L-SCULPT and makes schemas more difficult to design. For instance, already the schema consisting of the two rules $\mathsf{row}(a_1) \to b^*$ and $\mathsf{col}(a_2) \to b^*$ would be disallowed, because there exist tables (say, with a_1 in cell (2,1) and a_2 in cell (1,2)) in which the row and column intersect. (In our semantics, the schema for instance is satisfied by putting a_1 in (1,1), a_2 in (2,1), and no b at all.) Furthermore, the alternative semantics would introduce strange behavior. While the schema with the rule $\mathsf{row}(a) \to b^*$ is clearly satisfiable, the rule $\mathsf{row}(a) \to b^* + c^*(ac + d)^*c^*$ would not be allowed due to the occurrence of a in the content expression. The burden then lies with the user to rewrite the rule to $\mathsf{row}(a) \to b^* + c^*d^*c^*$ to become allowed again. In the semantics we propose, the last two rules are both allowed and define the same set of tables.

Finally, notice that the problem of testing whether there is a table T such that S is region-disjoint on T and all rules of S match is precisely SAT for L-SCULPT, which we study in Section 5.

Comparison With W3C Schemas for Tabular Data

The W3C proposes a schema language for tabular data in [12, Section 5.5]. From a structural perspective, this language is a strict subset of L-SCULPT, since it can be expressed as L-SCULPT using only rules of the form $\mathsf{col}(c) \to a^*$, where c is a coordinate and a is a token. Furthermore, the W3C schemas also do not admit selection of intersecting regions. Concerning our example in the introduction, although it is possible to define a schema using only rules of the form $\mathsf{col}(c) \to a^*$ for the data in Figure 1, we feel that such an approach leads to a much less accurate description of the data than our example in Figure 3.

▶ **Example 6.** The set of rules in Figure 3 are written in L-SCULPT. Furthermore, the schema is region-disjoint on the table T corresponding to the CSV file of Figure 2. (Therefore, T witnesses that the schema is satisfiable.) Recall that it is impossible to describe the data in such CSV files using column navigation only, as is currently the case in the W3C recommendation.

5 Satisfiability for L-SCULPT

In this section, we discuss the complexity of the satisfiability problem for L-SCULPT schemas. We distinguish between L-SCULPT fragments by explicitly listing the allowed operators in region selection expressions. For instance, L-SCULPT(row, col, right) denotes the fragment of L-SCULPT that only uses the operators row, col and right as region selection expressions.

In Section 5.1, we obtain the main technical result of the paper by delineating a relevant L-SCULPT fragment for which SAT is tractable, namely L-SCULPT(row, col, rect, right). In Section 5.2, we show that various extensions of this fragment become undecidable.

5.1 Polynomial-Time Fragments

We first show that satisfiability is in polynomial time for L-SCULPT(row, col, rect, right):

▶ **Theorem 7.** SAT *for* L-SCULPT(*row*, *col*, *rect*, *right*) *is PTIME-complete.*

The lower bound is a straightforward reduction from the emptiness problem of context-free grammars. The upper bound is significantly more challenging and is a reduction to the emptiness problem of nondeterministic tree automata where we represent tables T satisfying L-SCULPT schemas S as trees and use tree automata to match the rules of S on T and to test for region-disjointness.

We start by introducing the necessary terminology concerning trees, tree automata and the embedding of trees into tables in Section 5.1.1. When a schema has no coordinate rule, it is trivally satisfiable as it is satisfied by any table containing only $\triangleleft$-entries. So, we prove in Section 5.1.2 the upper bound for Theorem 7 for the special case where schemas consist of *exactly* one coordinate rule and do not make use of rectangular regions. Hereafter, we extend the proof to multiple coordinate rules and rectangles in Section 5.1.3 thereby completing the proof of Theorem 7. Finally, we discuss at the end of Section 5.1 how to obtain PTIME satisfiability for other L-SCULPT fragments.

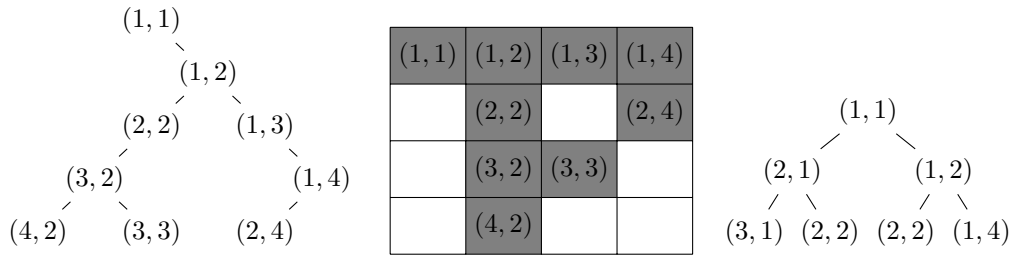


Figure 5 A binary tree (left), a table embedding for the tree on the left (middle), and a tree that does not have a table injection (right).

5.1.1 Preliminaries regarding trees

A (rooted, ordered, finite, labeled) *binary tree* is a finite tree where every node has at most two children (*left child* and *right child*). We allow nodes to have a left (resp., right) child only. We denote the empty tree by ε. Non-empty trees are denoted as $a(t_1, t_2)$. Here, the root carries the label a, has left subtree t_1, and right subtree t_2. Notice that t_1 or t_2 can be empty. For instance, $a(\varepsilon, \varepsilon)$ is a tree that consists of a single node, labeled a. We denote by $\mathrm{Nodes}(t)$ the set of nodes in the tree t. Every node u in the tree has a single *label* from Δ. For a formal introduction into tree automata we refer to Appendix B.1.

Table Embeddings and Table Trees

As we want to use tree automata to reason about tables, we define a correspondence between tables and trees. A *table embedding* of a binary tree t in a table T is a mapping $\mu : \mathrm{Nodes}(t) \to \mathrm{Coords}(T)$ such that, for each node v of t,

- the left child v_1 is mapped directly below its parent, that is, $\mu(v_1) = \mu(v) + (1, 0)$, and
- the right child v_2 is mapped directly to the right of its parent, that is, $\mu(v_2) = \mu(v) + (0, 1)$.

A *table injection* is an injective table embedding.

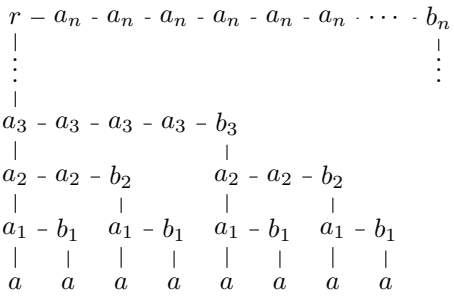
Notice that a table embedding is always completely determined by the cell on which the root of t is mapped. We illustrate table embeddings in Figure 5. The tree on the left has a table injection which is depicted in the middle. The tree on the right does not have a table injection: its canonical embedding is not injective on the nodes labeled $(2, 2)$.

In the remainder, we use the term *table tree* for a tree that has a table injection. Since we use trees to reason about tables, it will be more natural to speak of the *downward* and the *right* child of nodes in trees (as opposed to the *left* and the *right* child). Similarly, a *right path* (resp., *downward path*) is a path consisting only of right (resp., downward) children.

We note that other variants of table (or grid) embeddings have been studied in the literature (see, e.g., [8, 9]). These works allow children of nodes to be mapped to neighboring cells in the table, but they leave freedom as to which neighboring cells can be chosen. Under this alternative definition of embeddings, it is NP-complete to decide if an injective embedding exists for a given tree [8]. For table injections, however, this question is trivially in PTIME because it only amounts to testing if the canonical table embedding is injective.

$$
\begin{array}{lcl}
(1,1) & \to & r \\
\mathsf{col}(r) & \to & a_{n-1}\cdots a_1 a \\
\mathsf{right}(r) & \to & a_n \\
\mathsf{right}(a_i) & \to & a_i + b_i \\
 & & \quad (\text{for all } i = 1,\ldots,n) \\
\mathsf{col}(b_i) & \to & a_{i-1}a_{i-2}\cdots a_1 a \\
 & & \quad (\text{for all } i = 1,\ldots,n)
\end{array}
$$

(a) SCULPT schema consisting of $2n+1$ rules, each of size $O(n)$.



(b) Smallest table tree satisfying the schema in Figure 6a in a region disjoint manner. The path at the top contains 2^n nodes labeled a_n.

Figure 6 Schema for which the smallest region-disjoint table has an exponentially long path.

5.1.2 One Coordinate and No Rectangles

We assume that an L-SCULPT schema has at least one coordinate rule. As already mentioned above, when a schema has no coordinate rule, it is trivially satisfiable. We first prove that SAT is in PTIME for schemas with a single coordinate rule and that uses no rectangular regions.

Given an L-SCULPT(row, col, right) schema S, we construct a tree automaton $\mathcal{A}_S$ of size polynomial in S such that $L(\mathcal{A}_S) \neq \emptyset \Leftrightarrow S \in \text{SAT}$. Ideally, $\mathcal{A}_S$ would accept precisely those trees whose table injection satisfies S. But as the following example shows the latter condition could require $\mathcal{A}_S$ to be of size exponential in S. Indeed, consider the class of schemas in Figure 6a (which depends on a number n). Figure 6b shows that the smallest trees corresponding to tables satisfying S in a region-disjoint manner have an exponentially long path. Therefore, we construct $\mathcal{A}_S$ to accept precisely those trees that *can be pumped* to a tree that has a table injection satisfying S.

Pumping trees

We describe how trees are pumped, but first we need some terminology. We say that a token a is a col token if $\mathsf{col}(a)$ is a left-hand side of a rule in S. We define row tokens and right tokens analogously. A *horizontal token* is a row token or a right token. We sometimes also call col tokens *vertical tokens* for consistency in terminology.

Consider a tree t with a long right path containing node u_1 and, further right, u_2. Moreover, u_1 and u_2 have col tokens and no node between them has a col token. (One can think of u_1 being the root node in Figure 6b and u_2 the b_n-labeled node.) Furthermore, assume that t has a long downward path rooted at u_2. In any table embedding e, nodes u_1 and u_2 would be mapped to cells on the same row, i.e., with coordinates (i, j_1) and (i, j_2) for some i and $j_1 < j_2$. So, for e to be injective, the entire subtree rooted at u_1's downward child must be mapped into the region $\{(x, y) \mid x > i \text{ and } j_1 \leq y < j_2\}$, in order to avoid intersection with downward path below u_2.

A crucial observation is that we do not need to check this for very long paths. Since content expressions are regular, if $j_2 - j_1$ exceeds the size of the largest content expression in S, then the gap between u_1 and u_2 can be made arbitrarily large by pumping, avoiding the use of any column tokens. This means that, in this case, the size of the subtree at u_1's downward child does not matter for testing if the embedding e can be made injective. This

will later help us to ensure region-disjointness. The tree automaton will therefore count the distance between such branches up to a certain length, exceeding the size of the largest content expression in S. Beyond that, it classifies the distance as 'arbitrarily large'.

We also need to reason about the *width* of a tree t, which intuitively corresponds to the number of columns that will be used in its table embedding. Formally, if $t = \varepsilon$ then $\text{width}(t) = 0$. If $t = a(t^{\downarrow}, t^{\rightarrow})$, then $\text{width}(t) = \max\{\text{width}(t^{\downarrow}), 1 + \text{width}(t^{\rightarrow})\}$.

As we only consider row, col, and right axes, testing if a schema is region-disjoint on a table that corresponds to a table tree t amounts to the following. We need to test if, for every subtree $a(t^{\downarrow}, t^{\rightarrow})$ with $\text{width}(t^{\downarrow}) = k$, the tree $t^{\rightarrow}$ starts with a right path of length at least k before we see the first node with a col token or a downward child. We formalize this as follows. For a tree $t = a(t^{\downarrow}, t^{\rightarrow})$, we define path-length$(t)$ to be the number of nodes on the right path starting at the root, before we reach a node with a col token or a downward child. We set the value to ∞ if such a node does not exist. Formally, taking $\infty + 1 = \infty$, we define path-length as follows:

- if $t^{\downarrow} \neq \varepsilon$ or a is a col token, then path-length$(t) = 0$,
- else, if $t^{\rightarrow} = \varepsilon$, then path-length$(t) = \infty$, and
- otherwise, path-length$(t) = 1 +$ path-length$(t^{\rightarrow})$.

Main Construction

We construct a tree automaton $\mathcal{A}_S$ from a given L-SCULPT(row, col, right) schema S, such that $L(\mathcal{A}_S) \neq \emptyset \Leftrightarrow S \in \text{SAT}$. We need one more technicality. We call a token a *forbidden* if there are either two rules $\mathsf{right}(a) \to \rho_1$ and $\mathsf{row}(a) \to \rho_2$, two distinct rules $\mathsf{right}(a) \to \rho_1$ and $\mathsf{right}(a) \to \rho_2$, or two distinct rules $s_1(a) \to \rho_1$ and $s_2(a) \to \rho_2$ where $s_1 = s_2$. Notice that forbidden tokens can never appear in a table satisfying S. Observe that the set of forbidden tokens can be trivially computed in PTIME.

The tree automaton $\mathcal{A}_S$ is the intersection of the following automata:

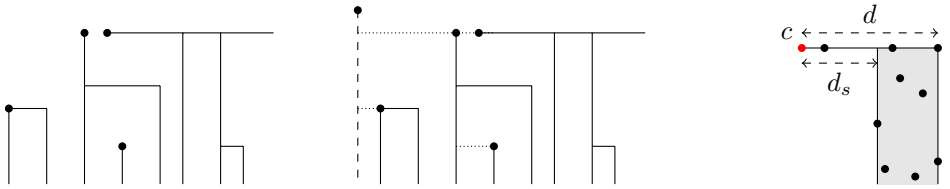
- $\mathcal{A}_{forbidden}$, which accepts the trees without forbidden tokens;
- $\mathcal{A}_{content}$, which checks conformity w.r.t. the content expressions and for region disjointness in the same dimension. More precisely, $\mathcal{A}_{content}$ checks
  1. whether the table induced by the tree matches the rules in the schema, and
  2. whether two row rules, a row and a right rule or two col rules select a common cell;[11]
- $\mathcal{A}_{\mathsf{row/col}}$, which checks that there are no row and col rules that select a common cell; and
- $\mathcal{A}_{\mathsf{right/col}}$, which checks that the input tree can be pumped to a tree where no right and col rule select a common cell.

Formally, the invariants of the automata are captured by the four Lemmas 12, 13, 14, 15, which can be found in Appendix B.2. Using these lemmas, it can be proved that the product automaton $\mathcal{A}_S$ accepts a tree if and only if it can be pumped to a table tree that satisfies S. By construction, in $L(\mathcal{A}_S)$, we will only be able to pump paths that are longer than $|\mathcal{A}_{content}|$.

▶ **Theorem 8.** *Let S be an* L-SCULPT(*row*, *col*, *right*) *schema that contains exactly one coordinate rule and no rectangular regions. Then we can construct in time polynomial in $|S|$ a tree automaton $\mathcal{A}_S$ such that $L(\mathcal{A}_S) \neq \emptyset \Leftrightarrow S \in \text{SAT}$.*

As testing for non-emptiness of tree automata is in PTIME, SAT for L-SCULPT(row, col, right) is in PTIME for schemas that contain at most one coordinate rule.

[11] A col rule is a rule with col(d) as a left-hand side. Similar for row and right rule.



(a) Example structure of a model for a schema with multiple coordinates.

(b) How the tree automaton sees it similar to one coordinate case.

(c) Area of responsibility of a state (c, d_s, d) in the automaton.

Figure 7 Dealing with the multiple coordinate case.

5.1.3 Multiple Coordinate Rules and Rectangles

We now show how to extend the result from Theorem 8 to the case where the schema may contain multiple coordinate rules. The main idea is summarized in Figure 7. Figure 7a depicts an instance of a region-disjoint forest with four roots. To turn it into a tree, we compute an equisatisfiable L-SCULPT schema, where every coordinate is shifted one cell to the right, add a new coordinate at (1,1) and select its entire column; see the dashed line in Figure 7b. Since all coordinates in the new L-SCULPT schema are at column two or higher, every coordinate in Figure 7a now has at least one selected cell somewhere to its left. We then allow the tree automaton to read a branch to the right at every node, provided this branch ends in a coordinate. (These are the dotted lines in Figure 7b.) The tree automaton then tests if it finds every coordinate in the tree.

Naïvely, the latter test requires an exponential number of states (see Example 11 in Appendix B.1), since it seems that the tree automaton needs a state for every subset of C, the set of coordinates occurring in the schema. However, let $n \in \mathbb{N}$ be the smallest number such that $C \subseteq [n]^2$. We can do the test with a polynomial number of states if we exploit that the schema is in L-SCULPT(row, col, right). We prove that an automaton for finding all coordinates in C can use states (c, d_s, d), where c is a coordinate in $[n]^2$, and d_s and d are in $[n] \cup \{0\}$. Formally, if a state (c, d_s, d) is assigned to a node u in an accepting run, it means that the automaton finds in the subtree at u all coordinates in $(a, b) \in C$ of the form (1) $x = a$ and $y \leq b \leq y + d$ and (2) $a \geq x$ and $y + d_s \leq b \leq y + d$. See Figure 7c for an illustration. Furthermore, we can also deal with rectangular regions:

▶ **Theorem 9.** *Let S be an* L-SCULPT(*row*, *col*, *rect*, *right*) *schema. Then we can construct in time polynomial in $|S|$ a tree automaton $\mathcal{A}_S$ such that $L(\mathcal{A}_S) \neq \emptyset \Leftrightarrow S \in$ SAT.*

5.2 Undecidable Fragments

L-SCULPT(row, col, rect, right) is asymmetric in that it only allows bounded navigation in one direction (i.e., the right operator). Next, we show that bounded navigation in two directions makes SAT undecidable. The first cases in the following theorem are the most natural ones. Here, we need at most one direction for unbounded navigation. The last case needs two unbounded directions, but we only need them once in the proof, for navigating to the bottom right corner of the data and start the tiling encoding from there. The undecidability results follow by reductions from Domino Tiling.

▶ **Theorem 10.** SAT *is undecidable for* L-SCULPT(*row*, *up*, *down*), L-SCULPT(*right*, *down*), L-SCULPT(*col*, *left*, *right*), *and* L-SCULPT(*row*, *col*, *up*, *left*).

6 Discussion

In this paper, we considered static optimization of SCULPT. As the satisfiability problem for SCULPT becomes undecidable rather quickly, we defined the restriction L-SCULPT and show that it admits a PTIME satisfiability test for the fragment L-SCULPT(row, col, rect, right). This fragment contains selection of rows, columns, and rectangular regions, starting from a given cell, plus bounded regions to the right (i.e., it can simulate "select three cells to the right of coordinate (5,2)" or "select the five cells to the right of each a"). Although SAT is still in PTIME if we replace right with down (by symmetry), we show that it becomes undecidable when we extend this fragment with bounded navigation in two directions. Interestingly, the just mentioned PTIME fragments contain the structural core of the W3C recommendation [12, Section 5.5] and, in addition, extend it with features allowing to deal with cases like, for instance, Use Case 13 in [13] that can not be addressed with the current recommendation. To summarize, even though L-SCULPT(row, col, rect, right) seems to be very restrictive when one starts from the full language SCULPT, it still extends the current W3C recommendation and seems powerful enough to describe many data sets in practice.

Our result is still useful if one would omit the region disjointness condition in the semantics of L-SCULPT. The PTIME algorithm would still be sound but incomplete. The only case where the algorithm would return 'no', although the schema would be satisfiable is when it can only be satisfied using tables in which selected regions are not disjoint.

References

1 Serge Abiteboul, Marcelo Arenas, Pablo Barceló, Meghyn Bienvenu, Diego Calvanese, Claire David, Richard Hull, Eyke Hüllermeier, Benny Kimelfeld, Leonid Libkin, Wim Martens, Tova Milo, Filip Murlak, Frank Neven, Magdalena Ortiz, Thomas Schwentick, Julia Stoyanovich, Jianwen Su, Dan Suciu, Victor Vianu, and Ke Yi. Research directions for principles of data management (abridged). *SIGMOD Record*, 45(4):5–17, 2016.

2 Marcelo Arenas, Francisco Maturana, Cristian Riveros, and Domagoj Vrgoč. A framework for annotating csv-like data. *Proceedings of the VLDB Endownment (PVLDB)*, 9, 2016.

3 Johannes Doleschal, Nico Höllerich, Martens, and Frank Neven. CHISEL: Sculpting tabular and non-tabular data on the Web. In *World Wide Web Conference (WWW)*, 2018. To appear.

4 Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

5 Dora Giammarresi and Antonio Restivo. Two-dimensional languages. In *Handbook of Formal Languages: Volume 3 Beyond Words*, chapter 4. Springer, 1997.

6 Ian Glaister and Jeffrey Shallit. A lower bound technique for the size of nondeterministic finite automata. *Information Processing Letters*, 59(2):75–77, 1996.

7 Stefan Göller, Markus Lohrey, and Carsten Lutz. PDL with intersection and converse: satisfiability and infinite-state model checking. *The Journal of Symbolic Logic*, 74(1):279–314, 2009.

8 Angelo Gregori. Unit-length embedding of binary trees on a square grid. *Information Processing Letters*, 31:167–173, 1989.

9 Ralf Heckmann, Ralf Klasing, Burkhard Monien, and Walter Unger. Optimal embedding of complete binary trees into lines and grid. *Journal of Parallel and Distributed Computing*, 49(1):40–56, 1998.

10 Leonid Libkin, Wim Martens, and Domagoj Vrgoč. Querying graphs with data. *Journal of the ACM*, 63(2):14:1–14:53, 2016.

11 Wim Martens, Frank Neven, and Stijn Vansummeren. SCULPT: A schema language for tabular data on the web. In *World Wide Web Conference (WWW)*, pages 702–712, 2015.

12 Rufus Pollock, Jeni Tennison, Gregg Kellogg, and Ivan Herman. Metadata vocabulary for tabular data. Technical report, World Wide Web Consortium (W3C), December 2015. `https://www.w3.org/TR/2015/REC-tabular-metadata-20151217/`.

13 Jeremy Tandy, Davide Ceolin, and Eric Stephan. CSV on the Web: Use cases and requirements. Technical report, World Wide Web Consortium (W3C), Februrary 2016. `http://www.w3.org/TR/2016/NOTE-csvw-ucr-20160225/`.

14 Jeni Tennison. 2014: The year of CSV. `http://theodi.org/blog/2014-the-year-of-csv`. Visited on Sept. 18, 2017.

15 Jeni Tennison and Gregg Kellogg. Model for tabular data and metadata on the web. Technical report, World Wide Web Consortium (W3C), July 2014. `https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/`.

16 Peter van Emde Boas. The convenience of tilings. In *Complexity, Logic and Recursion Theory*, volume 187 of *Lecture Notes in Pure and Applied Mathematics*, pages 331–363. Marcel Dekker Inc., 1997.

17 World Wide Web Consortium (W3C). CSV on the web working group charter. `https://www.w3.org/2013/05/lcsv-charter.html`. Visited on Sept. 18, 2017.

A SCULPT Region Selection Expressions

SCULPT *node expressions* are formulas φ in propositional dynamic logic (PDL for short) [4], tweaked for navigation over grids:

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

Here, φ and ψ are node expressions and α and β are navigational expressions. Furthermore, a ranges over tokens in Δ^e and root is a constant referring to the cell at coordinate $(1,1)$. Node expressions map a table to a region, whereas navigational expressions are mappings from regions to regions.

When evaluated over a table T, a node expression φ defines a region $(\!|\varphi|\!)_T \subseteq \mathrm{Coords}(T)$, as follows:

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

We define $(\!|\alpha(\varphi)|\!)_T$ as $(\!|\alpha(z)|\!)_T$, where $z = (\!|\varphi|\!)_T$ and $(\!|\alpha(z)|\!)_T$ is recursively defined as follows (for an arbitrary region $z \subseteq \mathrm{Coords}(T)$):

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

Here, α^i abbreviates the i-fold composition $\alpha \cdots \alpha$. We also use this abbreviation in the remainder. Notice that every coordinate (k, ℓ) of T can be expressed as $\mathsf{down}^{k-1}\mathsf{right}^{\ell-1}(\mathsf{root})$. Henceforth, we therefore use coordinates (k, ℓ) as syntactic sugar. Due to the close connection to PDL, region selection expressions are also very close to some fragments of Graph XPath [10].

To sum up, $f(\varphi)$ on a table T defines the set of regions $\mathsf{region}(\langle\!|\varphi|\!\rangle_T)$ and $\mathsf{rows}(\langle\!|\varphi|\!\rangle_T)$ when f is region or rows, respectively.

B Polynomial-Time Fragments

B.1 Trees and Tree Automata

A *nondeterministic tree automaton (over alphabet* Δ*)* or *NTA* is a tuple $N = (Q, \Delta, \delta, F, e)$ where Q is a finite set of states, $F \subseteq Q$ is the set of accepting states, e is a bit indicating if N accepts the empty tree or not, and δ is a set of transition rules of the form $[q_1, q_2] \xrightarrow{a} q$, where $a \in \Delta$ and $q_1, q_2 \in Q \uplus \{\varepsilon\}$. A *run* of N on a labeled binary tree t is an assignment of nodes to states $\lambda : \mathrm{Nodes}(t) \to Q$ such that for every $v \in \mathrm{Nodes}(t)$ the following holds. Let ℓ_v be the label of v. If v is a leaf, then $[\varepsilon, \varepsilon] \xrightarrow{\ell_v} \lambda(v) \in \delta$; if v only has a left child v_1 then $[\lambda(v_1), \varepsilon] \xrightarrow{\ell_v} \lambda(v) \in \delta$; if v only has a right child v_2 then $[\varepsilon, \lambda(v_2)] \xrightarrow{\ell_v} \lambda(v) \in \delta$ and, finally, if v has left child v_1 and right child v_2 then $[\lambda(v_1), \lambda(v_2)] \xrightarrow{\ell_v} \lambda(v) \in \delta$. A run is *accepting* if $\lambda(r) \in F$ for the root r of t. A non-empty tree t is *accepted* if there exists an accepting run on t. The empty tree is accepted if $e = \text{true}$. The set of all accepted trees is denoted by $L(N)$.

Transition rules suggest that NTAs read trees in a *bottom-up* manner. For this reason, NTAs are usually referred to as *bottom-up* nondeterministic tree automata. However, the semantics of NTAs does not depend on whether we write rules "bottom-up" as $[q_1, q_2] \xrightarrow{a} q$ or "top-down" as $q \xrightarrow{a} [q_1, q_2]$. In our proofs, we mix notation depending on the situation at hand.

▶ **Example 11.** We give an example of a tree automaton (that is useful in Section 5.1.3). Let $C \subseteq \Delta$. The tree automaton $N_C = (Q, \Delta, \delta, C, \text{false})$ accepts precisely those trees in which every symbol from C occurs. Define $Q = 2^C$. We define δ as follows:

$$[\varepsilon, \varepsilon] \xrightarrow{a} \{a\} \cap C \qquad [C_1, \varepsilon] \xrightarrow{a} (C_1 \cup \{a\}) \cap C$$
$$[C_1, C_2] \xrightarrow{a} (C_1 \cup C_2 \cup \{a\}) \cap C \qquad [\varepsilon, C_2] \xrightarrow{a} (C_2 \cup \{a\}) \cap C$$

In an accepting run, N_C visits a node u in a state $C' \subseteq C$ iff C' is the largest subset from C such that the subtree rooted at u contains every symbol from C'. We note that N_C is exponentially larger than $|C|$. This exponential size in C cannot be avoided as even on words, the smallest NFA recognizing the words in which all symbols from C occur has exponential size. The latter can be easily proved using Glaister and Shallit's lower bound technique for the size of NFAs [6]. Intuitively, the blow-up is due to the fact that the symbols from C may occur in any order and therefore the automaton needs to remember which symbols have been already encountered.

B.2 Invariants for the Main Construction

The invariants of the four tree automata used in Theorem 8 are captured by the following four Lemmas.

▶ **Lemma 12.** *An NTA* $\mathcal{A}_{forbidden}$ *can be constructed in time polynomial in* $|S|$ *such that* $L(\mathcal{A}_{forbidden})$ *is the set of trees containing no forbidden token.*

For a node u in t we define its set of *triggering rules in* S as follows:

- if $\mathrm{lab}(u) = a$, then its triggering rules are all rules of the form $\mathsf{row}(a) \to \rho$, $\mathsf{col}(a) \to \rho$, or $\mathsf{right}(a) \to \rho$ in S and
- if u is the root of t, then, additionally, the unique coordinate rule of S is also triggering.

▶ **Lemma 13.** *An NTA $\mathcal{A}_{content}$ can be constructed in time polynomial in $|S|$ such that $L(\mathcal{A}_{content}) \cap L(\mathcal{A}_{forbidden})$ is the set of trees t such that for every node u all the following hold:*

1. *If u has a triggering rule $\mathsf{row}(d) \to \rho$ then the right path of t rooted at u forms a word in $L(\rho)$ that does not contain any horizontal tokens.*
2. *If u has a triggering rule $\mathsf{col}(d) \to \rho$ then the downward path of t rooted at u forms a word in $L(\rho)$ that does not contain any vertical tokens.*
3. *If u has a triggering rule $\mathsf{right}(d) \to \rho$ then the label of the right child of u either is $\sqcup$, or a word in $L(\rho)$.*
4. *u lies on a path described in cases 1-3.*

In the above, d can be a coordinate or a token. For the unique rule where d is a coordinate, node u is included in the respective path. In all other cases it is excluded.

We note that property 4 ensures that a tree accepted by $\mathcal{A}_{content}$ is minimal in the sense that it only contains nodes required to match the schema.

▶ **Lemma 14.** *An NTA $\mathcal{A}_{\mathsf{row}/\mathsf{col}}$ can be constructed in time polynomial in $|S|$ such that $t \in L(\mathcal{A}_{\mathsf{row}/\mathsf{col}})$ if and only if, for every subtree $a(t^{\downarrow}, t^{\rightarrow})$ of t, there is no row token in $t^{\downarrow}$ or there is no col token in $t^{\rightarrow}$.*

▶ **Lemma 15.** *An NTA $\mathcal{A}_{\mathsf{right}/\mathsf{col}}$ can be constructed in time polynomial in $|S|$ such that $t \in L(\mathcal{A}_{\mathsf{right}/\mathsf{col}})$ if and only if, for every node u of t, path-length$(t^{\rightarrow}) \leq |\mathcal{A}_{content}|$ implies that*

$$\mathit{width}(t^{\downarrow}) \leq \mathit{path\text{-}length}(t^{\rightarrow}) + 1,$$

where $a(t^{\downarrow}, t^{\rightarrow})$ is the subtree of t rooted in u.