

# A Rectangle Mining Method for Understanding the Semantics of Financial Tables

Xilun Chen\*  
Cornell University  
Ithaca, NY, USA

xlchen@cs.cornell.edu

Laura Chiticariu, Marina Danilevsky, Alexandre Evfimievski and Prithviraj Sen  
IBM Research, Almaden  
San Jose, CA, USA

{chiti,mdanile,evfimi,sen}@us.ibm.com

**Abstract**—Financial statements report crucial information in tables with complex semantic structure, which are desirable, yet challenging, to interpret automatically. For example, in such tables a row of data cells is often explained by the headers of other rows. In a departure from prior art, we propose a rectangle mining framework for understanding complex tables, which considers rectangular regions rather than individual cells or pairs of cells in a table. We instantiate this framework with REMINE, an algorithm for extracting row header semantics of table, and show that it significantly outperforms prior pair-wise classification approaches on two datasets: (i) a set of manually labeled financial tables from multiple companies, and (ii) the ICDAR 2013 Table Competition dataset.

## I. INTRODUCTION

Financial reports provide information on enterprise activity and are crucial in applications such as investment, regulatory compliance, and market research [1]. Most, if not all, such reports employ *financial tables* to report a comprehensive set of information at a fine granularity (see Fig. 1). The current age of exploding information has resulted in a rapidly increasing demand for automatically understanding such tables.

Today, as in the past, financial tables are designed for human consumption: domain experts understand the semantics of any data cell at a glance. While guidelines exist for reporting financial content, there are no presentation standards such as markup or visual cues (e.g. indentation, font style) to communicate information relationships [2], making financial tables difficult to comprehend automatically. Furthermore, most financial tables employ complex semantic structures (sometimes referred to as disaggregation in the literature) to present denser information, adding more challenges to automatic comprehension [3], [4].

**Table Semantics Extraction** Given a data cell in a table, our goal is to identify all other semantically relevant cells, i.e. the cells that explain the meaning of this cell, and form a *semantic tuple* (see Fig. 1). In tables with a simple structure, it is enough to find the row header cells and the column header cells that overlap the data cell. Our task is to identify the non-overlapping semantically relevant cells as well.

Previous work has addressed automatic table comprehension for Web spreadsheets [5], [6]. Although in principle,

\*The research was done while the first author was an intern at IBM Research.

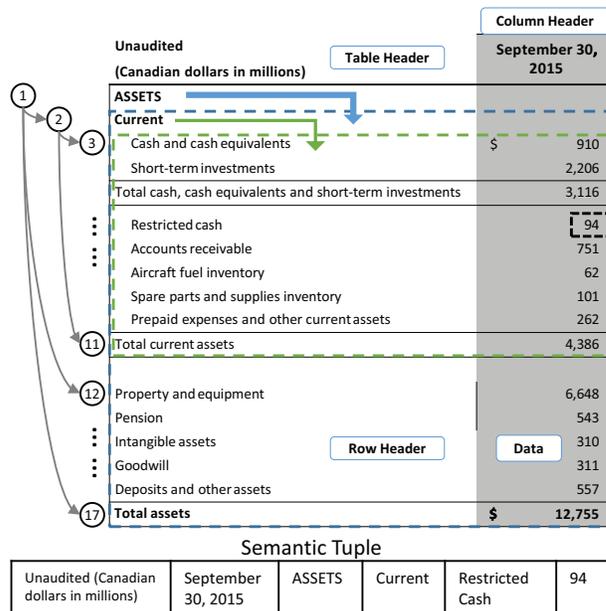


Fig. 1: A financial table (partial) and the semantic hierarchy of rows. It shows two ostensibly equivalent representations: the tree hierarchy on the left, and the rectangle hierarchy. There are two “interesting” rectangles (defined in Section II): rows 2-17 headed by row 1; and rows 3-11 headed by row 2.

Web spreadsheets can come from arbitrary domains and have diverse structures, a survey conducted by [5] showed that a substantial portion had much simpler structure than financial tables in documents. Furthermore, they show that many tables exhibit a dataframe structure consisting of a subset of four regions: *table headers*, *row headers*, *column headers*, and *data*. Thus, the table semantic extraction problem is naturally divided into: (i) identifying the 4 regions, and (ii) constructing a semantic hierarchy for the row and column header regions respectively. [5] noted (ii) is more challenging, and within (ii), row headers usually have more complicated structures than column headers. Therefore, we focus on extracting the semantic hierarchy of row header cells. In this paper, we use *row hierarchy* and *row header hierarchy* interchangeably, as each row is characterized by its row header and the remaining data cells. We call a table’s semantic row header hierarchy *complex* if there exist multiple cells in the same row header column that are semantically related to each other. For instance, the table

in Figure 1 has complex row header hierarchy since row 3 is semantically headed by row 2, which is in turn headed by row 1 in the first column.

### Pair-wise Classification for Row Hierarchy Extraction

Previous work [5], [7] uses a tree (or forest) structure (see Fig. 1 (left)) to encode the parent-child relationship between a pair of rows, and proposes a classifier to predict whether the parent-child relation holds for a given pair of rows. While the tree structure is an elegant representation that yields practical algorithms, this pair-wise model has two limitations:

**Fragmentation** The prediction of *direct* children of a row can lead to nonconsecutive descendants. In Figure 1, row 1 has descendants {2, 12-17}. In practice, this often results in fragmented predictions, is error-prone when deciding on the boundary of a section, and requires post-processing to ensure a valid hierarchy is generated.

**Pair-local Decisions** The pair-wise prediction approach can only take into account local features of the pair itself, severely hindering any ability to make decisions that require global reasoning. In Figure 1, rows 11 and 12 have identical or very similar features<sup>1</sup>. However, 11 is a child of 2, while 12 is not. Correct prediction involves properly inferring the boundary of section headed by 2, which is difficult in the pair-wise model.

**Proposed Approach and Contributions** **i)** We propose an alternative *rectangle* framework to look at the semantic hierarchy, which makes decisions based on contiguous blocks of cells (i.e., *rectangles*) as opposed to a single pair of cells, therefore addressing both limitations of the pair-wise model: Contiguous blocks of cells are assigned to a parent in one shot (no fragmentation), and features that are global within a rectangle become available (no pair-local decisions). **ii)** We instantiate the rectangle framework with the REMINE algorithm for extracting row semantics that is both *customizable* and *deterministic*. Hence, an added advantage is *transparency* [8]: REMINE can be easily adapted for a new domain even in the absence of training data, and its output can be automatically explained. **iii)** We validate REMINE with experiments on (1) a dataset of manually labeled financial tables and (2) the public ICDAR 2013 Table Competition dataset, and show that it significantly outperforms several existing baselines ( $\uparrow 27\%$  F1). **iv)** Our Financial Table dataset is released to the research community (See Sec. III-A).

## II. RECTANGLE MINING (REMINI) FOR SEMANTIC HIERARCHY EXTRACTION

To overcome the limitations of the pair-wise classification, we need to express features defined over multiple “descendant” cells of a header cell, or over whole “descendant” table regions. While this might look intractable, we observed that most human-readable tables satisfy the following:

**Assumption 1.** *All the cells semantically headed (explained) by a single cell form a contiguous rectangular region, defined as the support rectangle of that cell.*

We call a rectangle *interesting* if it is the support rectangle of some cell, called a *header cell* of that rectangle. Here we propose a general *rectangle framework* that represents the semantic hierarchy of a table using support rectangles and their header cells, as opposed to a tree structure. For example, in Figure 1 we find at least two interesting rectangles: the green rectangle (Row 3-11) is the support rectangle of the header of Row 2 (Current), and similarly, the blue rectangle (Row 2-17) is the support rectangle of the header of Row 1 (ASSETS).

Our framework maps each rectangle to a vector of features  $\phi(r) \in \Phi$ . Some features describe individual cells of the rectangle (e.g., font, indentation), others describe the entire rectangle (e.g., size, graphical lines, whitespace, or whether the whole rectangle is a complete section). We explain further how rectangle features can be defined in Section II-A. Such rectangle features encode richer information and alleviate the *pair-local decision* problem. These features are used to (i) identify or compare potentially interesting rectangles, and (ii) match rectangles with their potential header cells. We can also utilize rectangle features to make more informed predictions of section boundaries, greatly mitigating the *fragmentation problem*.

The rectangle framework does not prescribe a specific approach to find interesting rectangles; rather, it is a very general perspective that allows many different algorithms. Below we propose one such constructive algorithm, REMINE, and leave the exploration of more general rectangle mining algorithms for future work.

### A. The REMINI Algorithm

REMINI is a bottom-up rectangle construction algorithm that is simple yet versatile and customizable. It deals with a special case of the more general rectangle mining framework, where each row is treated as an atomic unit to extract row semantics, as described in Section I.

REMINI works with many forms of rectangle features as long as a *feature partial ordering* can be defined as follows:

**Definition 1.** *A partial order  $\preceq$  defined on the feature space  $\Phi$  is called a **feature partial ordering**.*

Intuitively, given two rectangles  $r_1, r_2$  with feature vectors  $\phi_1, \phi_2$ ,  $\preceq$  is a comparator that returns one of the following: ( $\phi_1 \prec \phi_2$ ,  $\phi_1 \succ \phi_2$ ,  $\phi_1 \succ \phi_2$ , *non-comparable*). In this paper,  $\phi_1 \succ \phi_2$  means the header row of  $r_1$  tends to semantically head  $r_2$  rather than the opposite way. For simplicity, we also write  $r_1 \succ r_2$  henceforth.

In practice, there are multiple ways to define  $\preceq$ . For a single dimension in the feature vector, there usually exists some natural ordering. For instance, if one rectangle feature is the *minimum indentation of the rectangle*, the natural order is that smaller indentation is “greater than” larger indentation. For other features, such as font style and font color, we can define the most frequent style/color (plain text) as “less than” other styles/colors (emphasized text). Therefore, the most basic idea to define  $\preceq$  for two feature vectors  $\phi_1$  and  $\phi_2$  is that  $\phi_1 \succ \phi_2$  iff.  $\forall i : \phi_1^{(i)} \succ \phi_2^{(i)}$  and  $\exists i : \phi_1^{(i)} \succ \phi_2^{(i)}$  where  $\phi^{(i)}$  is the  $i$ -th

<sup>1</sup>Ignoring lines and spaces, often unavailable in the extracted table format.

Single-row Features (local)		Rectangle Features (global)	
Feature Name	Order	Feature Name	Order
Boldness	T $\succ$ F	isEndedSection	T $\prec$ F
Indentation	G $\prec$ L	isEmptySection	T $\succ$ F
Blankness	T $\prec$ F		
Capitalization	T $\succ$ F		
isSectionHeader	T $\succ$ F		
isTotalRow	See text		

TABLE I: Features and Feature Partial Order used in REMINE. For the partial order, “T” means the feature is true while “F” indicates false. For numeric features such as indentation, “G” means greater value while “L” means less. See below for the explanation of `isTotalRow` and more discussion on the global features.

dimension of  $\phi$ .  $\prec$ ,  $\succ$  can be similarly defined, and  $\preceq$  returns *non-comparable* otherwise.

One can also look beyond independent feature dimensions and use more complex partial ordering to encode more global features. An example complex heuristic is: a rectangle headed by a section header and terminated by a section total row is unlikely to be greater than other following rectangles, since a section completed by a pairing total row is less likely to grow further (the green rectangle in Figure 1). Finally, since REMINE takes an abstract partial order as input, a classifier could also be trained to define  $\preceq$  if data is abundant and transparency less important.

**REMINES Features and Partial Order** We now present the specific feature set and feature partial ordering used in this work, which is very simple and mostly intuitive.

REMINES uses 8 features, as shown in Table I. Local features pertain to a single row (the header row of the rectangle), and usually describe some basic format information such as boldness and indentation. In addition, REMINE uses two global rectangle features: `isEndedSection` (rectangles headed by a section header and terminated by a paired total row, defined by the row-pair feature `isTotalRowPair`) and `isEmptySection` (rectangles of only one row, a section header defined by `isSectionHeader`). These features are computed using simple heuristics. The first 4 features are straightforward style features of the row header cell. `isSectionHeader` is active if the row has a non-empty row header cell and empty data cells. For `isTotalRow`, we detect if the row header cell starts with certain keywords such as “total”. For `isEndedSection`, to tell whether a section header and a total row are “paired” or not, we calculate the longest common subsequence (LCS) between the two, and adopt a threshold to decide.

Given the 8 features shown in Table I for REMINE, the feature partial ordering  $\preceq$  is defined as follows: For single-row features, the partial order is: bold  $\succ$  non-bold, smaller-indentation  $\succ$  larger-indentation, not-blank  $\succ$  blank, capitalized  $\succ$  not-capitalized, `isSectionHeader`  $\succ$  not-`isSectionHeader`. For `isTotalRow`, the partial order is defined differently, based on the heuristics that total rows are primarily used to summarize and end a section. In particular, when a total row is considered as a parent candidate,  $\preceq$  will

**Algorithm 1** REMINE Algorithm for Extracting Row Header Semantic Hierarchy. Functions such as `findMinimalRects` are explained later.

---

**Input:** List of rows  $\mathcal{C}$  starting from the top; Partially ordered set  $(\Phi, \preceq)$   
**Output:** Set of parent-child relations  $\mathcal{P}$   
**State:** Maintains *ordered list*  $\mathcal{R}$  of active top-level rectangles

- 1:  $\mathcal{P} = \emptyset$
- 2:  $\mathcal{R} = \mathcal{C}$  ▷ Initially each row is a rectangle
- 3: **repeat**
- 4:    $combineR = new List()$  ▷ Init with empty list
- 5:    $\mathcal{M} = findMinimalRects(\mathcal{R}, \Phi)$
- 6:   ▷ Combination Stage
- 7:   **for all**  $r_i \in \mathcal{R}$  **do** ▷  $r_i$  is the  $i$ -th rectangle in  $\mathcal{R}$
- 8:     **if**  $r_i \in \mathcal{M}$  and  $r_i \preceq r_{i+1} \preceq \dots \preceq r_j$  **then**
- 9:        $combineR.append(Combine(r_i, \dots, r_j))$
- 10:     **else**  $combineR.append(r_i)$
- 11:    $attachR = new List()$
- 12:    $\mathcal{M} = findMinimalRects(combineR, \Phi)$
- 13:   ▷ Attaching Stage
- 14:   **for all**  $r_i \in combineR$  **do**
- 15:     **if**  $r_{i+1} \in \mathcal{M}$  and  $r_i \succ r_{i+1}$  **then**
- 16:        $attachR.append(Attach(r_i, r_{i+1}))$
- 17:        $\mathcal{P} = \mathcal{P} \cup addRelations(r_i, r_{i+1})$
- 18:     **else**  $attachR.append(r_i)$
- 19:    $\mathcal{R} = attachR$
- 20: **until**  $\mathcal{R}$  not changing
- 21: **return**  $\mathcal{P}$

---

always return non-comparable so that a total row cannot take children. Otherwise, when considered as child candidate, the feature `isTotalRow` is ignored.

For global rectangle features, the order is: `isEndedSection`  $\prec$  not-`isEndedSection`, `isEmptySection`  $\succ$  not-`isEmptySection`. The intuition for the rectangle features are that i) for `isEndedSection`, a section ended by a pairing total row is less likely to grow further (the green rectangle in Figure 1); and ii) for `isEmptySection`, it is rare that a section only has a header row but no other contents. Therefore, an empty section (header) is likely to take children below.

**REMINES Algorithm** REMINE is a two-stage iterative algorithm (Algorithm 1). Larger rectangles are built by iteratively combining smaller rectangles until convergence. A list of active top-level rectangles  $\mathcal{R}$  is maintained. Note that rectangle order in  $\mathcal{R}$  is **not** dictated by  $\preceq$ , but rather follows natural top-down order in the table, allowing us to work on  $\mathcal{R}$  sequentially. Each iteration consists of a combination stage, where consecutive *minimal* rectangles with *equal* features (as defined by  $\preceq$ ) are combined, followed by an attaching stage, where all *minimal* rectangles  $r_i$  are attached to  $r_{i-1}$  if  $r_{i-1} \succ r_i$ . Minimal rectangles are defined as:

**Definition 2.** A rectangle  $r$  is minimal within a list of rectangles  $\mathcal{R}$  under partial order  $\preceq$ , if  $\forall r'$  after  $r$  in  $\mathcal{R}$ , it satisfies  $r \not\preceq r'$ .

In Algorithm 1, `findMinimalRects( $\mathcal{R}, \Phi$ )` is used to return all minimal rectangles within  $\mathcal{R}$ . We only work on minimal rectangles in each iteration because they are “safe” to be

①	ASSETS
②	Current
③	Cash and cash equivalents
④	Short-term investments
⑤	Total cash, cash equivalents and short-term investments
⑥	Restricted cash
⑦	Accounts receivable
⑧	Aircraft fuel inventory
⑨	Spare parts and supplies inventory
⑩	Prepaid expenses and other current assets
⑪	Total current assets
⑫	Property and equipment
⑬	Pension
⑭	Intangible assets
⑮	Goodwill
⑯	Deposits and other assets
⑰	Total assets

Fig. 2: REMINE Example

combined or attached with no potential children remaining in  $\mathcal{R}$ . The method `Combine()` combines consecutive rectangles with *equal* features, and all the header rows of the combined rectangles become headers of the new rectangle. In addition, the method `Attach( $r_1, r_2$ )` attaches  $r_2$  to  $r_1$  as a child, and the header row of  $r_1$  remains as the header of the new rectangle. When  $r_2$  is attached to  $r_1$ , we also generate parent-child relations between the header row of  $r_1$  and those of  $r_2$  using `addRelations`. This way, the semantic hierarchy is reconstructed by iteratively building larger rectangles.

### B. REMINE Example

We provide a running example of REMINE, illustrated in Figure 2, which is the row header region excerpted from the table in Figure 1. The features used in the example are the same as those shown in Table I. As shown in Algorithm 1, REMINE starts with each row as a rectangle. At the beginning of Iteration 1, the minimal rectangles recognized by Line 5 are: row 3-17 according to Definition 2, since none of them has “smaller” rectangles (based on  $\preceq$ ) after them in the active rectangle list  $\mathcal{R}$ . In the Combination stage, row 3,4 will be combined into a single rectangle  $Rect(3, 4)$ , since they have identical feature vectors. Similarly, row 6-10 will be combined into  $Rect(6, 10)$ . Furthermore, row 12-16 will be combined into  $Rect(12, 16)$ . Row 5, 11 and 17 were not combined since they have the feature `isTotalRow`. Before the Attaching stage, the minimal rectangles are the newly combined rectangles  $Rect(3, 4)$ ,  $Rect(6, 10)$  and  $Rect(12, 16)$ , as well as the total rows  $Rect(5)$ ,  $Rect(11)$  and  $Rect(17)$ . In the Attaching stage, since  $Rect(3, 4) \prec Rect(2)$ ,  $Rect(3, 4)$  is attached to  $Rect(2)$ , becoming a new rectangle  $Rect(2, 4)$ , headed by row 2. Meanwhile, a parent-child relation is generated between rows 2 and 3, and rows 2 and 4. On the other hand,  $Rect(6, 10)$  is not comparable with  $Rect(5)$  under  $\preceq$ , since `isTotalRow` is true for  $Rect(5)$ . Similar argument applies to other minimal rectangles, hence no further attachment is done in Iteration 1.

In Iteration 2, no adjacent minimal rectangles have identical features, so nothing is done in the Combination stage. In the

Attaching stage,  $Rect(2, 4)$  will continue to consume  $Rect(5)$  since row 2 has `isBold` and `isSectionHeader`. Since row 2 and row 5 are *not* pairing total rows, `isEndedSection` is *not* activated for the resulting  $Rect(2, 5)$ . Similar to Iteration 2, in Iteration 3 and 4, the only thing happens is that  $Rect(2, 5)$  further consumes  $Rect(6, 10)$  and  $Rect(11)$ . At the end of Iteration 4,  $Rect(2, 11)$  will now have feature `isEndedSection` since row 11 is a pairing total row for section header row 2. Therefore,  $Rect(2, 11)$  stops taking any further descendants. In Iteration 5,  $Rect(2, 11)$  will be attached to  $Rect(1)$ . Similarly in Iteration 6,  $Rect(1, 11)$  will consume  $Rect(12, 16)$ , and eventually in Iteration 7,  $Rect(17)$  is attached to  $Rect(1, 16)$ , finalizing the REMINE algorithm. REMINE perfectly reconstructed the semantic hierarchy for this complex table, whereas pair-wise classification approaches perform poorly due to the fragmentation and the pair-local decision problems, as previously discussed.

## III. EXPERIMENTS

### A. Experimental Setup

**Financial Table Dataset** We collected a set of financial statements from public companies, available online in PDF format. We label the semantics of the resulting set of 72 tables<sup>2</sup>, comprising the 2015 Q3 financial statements of 6 companies: Air Canada (6 tables), Cogeco (9 tables), Rogers (20 tables), Telus (21 tables), Transat (8 tables) and Westjet(8 tables).

**ICDAR 2013 Table Competition Dataset** In addition, we consider the ICDAR 2013 Table Extraction competition dataset, which has a table semantics sub-task [9]. We use the same model we developed for financial tables directly to test on the ICDAR dataset, as an additional out-of-domain evaluation. Of the 70 tables available in the dataset, 7 tables exhibit a complex row hierarchy structure and are therefore relevant for evaluating REMINE.

For both datasets, the tables are available in PDF format, which is not readily machine readable. We use an internal tool to automatically convert PDF to HTML documents, manually correcting mistakes<sup>3</sup>. For our baseline methods which require training, we conduct *leave-one-company-out* cross validation on the Financial Table Dataset, using 5 companies for training and the remaining one for testing. The results are aggregated and the overall performance is reported. For the ICDAR Dataset, due to its small size, for all systems including REMINE and baseline systems, we directly use the models trained or developed for the Financial Table Dataset for evaluating on the ICDAR Dataset.

**Comparison Baselines** We compare REMINE against two pair-wise classification baselines: (1) an SVM pair-wise classifier trained to predict parent-child relations similar to [5]<sup>4</sup>, and (2) RIPPER [11], which learns rules of logical expressions of

<sup>2</sup>The dataset can be downloaded at <http://www.cs.cornell.edu/~xlchen/resources/FinancialTableDataset.zip>

<sup>3</sup>PDF conversion to HTML is a challenging problem [10] outside the scope of this work.

<sup>4</sup>The authors were unable to share the dataset of 100 web tables used for evaluation in [5].

Systems	Financial Tables Dataset						ICDAR 2013 Dataset					
	Direct			Transitive			Direct			Transitive		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
SVM	76.27	62.03	68.42	76.27	50.68	60.89	54.69	47.30	50.72	41.86	43.37	42.60
RIPPER	78.40	48.27	59.74	81.31	47.15	59.69	92.31	36.36	52.17	87.61	36.40	51.43
REMINE	82.45	85.84	<b>84.11</b>	86.30	89.56	<b>87.90</b>	91.43	86.49	<b>88.89</b>	85.88	87.95	<b>86.90</b>

TABLE II: Leave-one-company-out cross validation results. Section III defines *Direct* and *Transitive* evaluation schemes.

Feature Name	Pair-wise	REMINE	Ablation
<b>Single-row Features</b> (local)			
Boldness	✓	✓	85.36
Indentation	✓	✓	83.73
Blankness	✓	✓	87.90
Capitalization	✓	✓	88.94
isSectionHeader	✓	✓	87.50
isTotalRow	✓	✓	85.87
<b>Row-pair Features</b> (local)			
isTotalRowPair	✓	N/A	N/A
<b>Rectangle Features</b> (global)			
isEndedSection	N/A	✓	86.21
isEmptySection	N/A	✓	84.33

TABLE III: Feature set comparison and features ablation results on the Financial Table dataset. A feature is used (✓) or not applicable to the system (N/A). Feature ablation is conducted on REMINE and indicates *Transitive* F1.

features, and is more transparent than SVM. To generate valid trees, only the closest predicted parent is kept for each child row in both baselines as a post-processing. All three systems utilize the same features whenever possible and all features are reported in Table III. In addition, REMINE also uses two global features as explained in Section II: *isEndedSection* and *isEmptySection*. Although these global features are not compatible with the baselines, for fairness we include the relevant single-row and pairwise features into the baselines (*isSectionHeader*, *isTotalRow*, *isTotalRowPair*).

**Metrics** We evaluate performance - the classification of whether or not each pair of rows has a parent-child relationship - using the classic *F1* measure, with two schemes: *Direct* and *Transitive*. The *Direct* scheme is used in literature and evaluates only direct parent-child relations; e.g., in Figure 1, row 1 has 8 direct children {2, 11-17}. In a departure from prior work, we propose the *Transitive* scheme, where all *descendants* are counted during evaluation; in Figure 1, row 1 has 16 descendants {2-17} under this scheme. The intuition is that mistakes on certain relations are much more costly in terms of understanding the entire table semantics, e.g., failing to recognize row 1 as a parent of 2 in Figure 1 means half the table (rows 2-11) misses the semantic relation to *ASSETS*, and may be confused with the very similar subsequent *Liabilities* section (not shown in Figure). The *Transitive* scheme penalizes such important mistakes with a higher weight.

### B. Experimental Results

As shown in Table II, on the Financial Table dataset, the rule-learning RIPPER achieves higher precision and lower recall compared to SVM, but performs similarly under the

more realistic *Transitive* metric, with better transparency. On the ICDAR dataset, where training only happens out of domain (on the financial tables), RIPPER even achieves higher F1 compared to SVM, displaying better generalization than SVM. On the other hand, REMINE substantially outperforms both approaches on both datasets. (↑27% and ↑35% respectively). In particular, the rectangle perspective and usage of global features significantly alleviate the *fragmentation* and the *pair-local decision* problems.

Ablation results on the Financial Table Dataset in Table III show the impact of global (and other) features. Figure 1 illustrates the importance of the global features, without which the transitive F1 decreases to 82% (↓5%). The pair-wise approach marks rows 3-16 as descendants of row 2 (the correct ones are 3-11) because local features are not sufficient to indicate that the section headed by 2 should end before 12. In contrast, REMINE perfectly classifies Figure 1 via its global reasoning of section boundaries, stopping the rectangle at 11 since it is a pairing total row of 2, which triggers the rectangle feature *isEndedSection*.

**Error Analysis** We find that the most common source of mistakes for REMINE is the hand-crafted feature partial order. Despite correctly modeling most cases,  $\preceq$  is defined manually and lacks the flexibility to account for outliers. For instance, capitalized acronyms (“EBITDA”) trigger the *Capitalization* feature and  $\preceq$  may make mistakes when comparing with it. As seen in Table III, removing the *Capitalization* feature actually increased the performance. Machine learning methods for deriving  $\preceq$  while maintaining transparency can be explored in the future.

Another source of error is the extraction of certain complex features such as *isTotalRow* and *isTotalRowPair*. Currently, we are using simple heuristics to detect if such features are present in a certain sample. For instance, we detect if the row header cell starts with the word “total” to see whether it is a total row. This caused some errors in the ICDAR dataset, where total rows are also reported in the pattern “Net ...”. Similarly, *isTotalRowPair* is detected by checking if the longest common sub-sequence between a section header and a total row is longer than a threshold, and these heuristics could potentially be improved by using more advanced NLP techniques.

## IV. RELATED WORK

**Table Semantics Extraction** has been previously studied [12], and most relevant to our work is [5], [13], which extracts semantic hierarchy from web spreadsheets. They first surveyed the common types of structures of web spreadsheets,

and pointed out that the *data-frame* tables are both common (50%) and have more complicated structures compared to others. They also propose to divide the data-frame tables into four parts: table headers, row headers, column headers and data, which is also adopted in our paper. Finally, they they propose a pair-wise classification method for row header hierarchy extraction, similar to the SVM baseline in our paper. The method and limitations have been discussed in Section I. [7] proposed a semiautomatic method that also leverages the user feedback for semantic hierarchy extraction. A user can manually make corrections to the system output while the system takes such signals to improve its performance while minimizing user interference. Other prior work uses Conditional Random Fields to learn table schema by classifying each row into a number of classes such as header row, data row, title row, etc. [14].

**Question Answering using Tabular Data** is another related field [15], [16], [17]. These papers aim to use tables as a semi-structured data source to help question answering. One note is that they typically focus on tables with less complex semantic hierarchies and simple heuristics usually suffice to extract semantic tuples in the tables they use. Our work is complementary to these papers and can be combined with them to enable their methods to use more complicated tables as data source.

**Table Recognition and Parsing** concerns the recognition and extraction of tables from a human-readable format (usually PDF) into a machine-comprehensible format (e.g. HTML, XML). Identifying the table boundary and the table structure including the rows of the table, the columns and the cells from PDF documents is addressed among others by [18], [19], [20], please see surveys [21], [22], [23]. In addition, given a table, with boundary and cells identified, multiple works address the problem of classifying cells into “data cells” versus “header cells” [10], [24], [25]. These works don’t seek to understand the detailed semantic relationships of cells inside the row header. Our work starts once a table structure has been identified: including the cells classified into data cells and header cells. Specifically, our work is concerned with understanding the detailed semantic relationships between cells inside the row header.

## V. CONCLUSION

We tackle a difficult task of semantic comprehension of financial tables by proposing a general framework based on the *rectangle assumption*. We instantiate the framework for extracting row semantic hierarchy by implementing the REMINE algorithm, and show it performs well on a manually labeled dataset of financial tables from multiple companies as well as the public ICDAR 2013 Table Competition dataset. Compared to prior pair-wise approaches, REMINE not only enables global reasoning, but is also transparent and customizable. In the future, we plan to extend REMINE to multi-column row headers and explore example-driven learning to derive the feature partial order.

## REFERENCES

- [1] S. Cascino, M. A. Clatworthy, B. G. Osma, J. Gassen, S. Imam, and T. Jeanjean, “Who uses financial reports and for what purpose? Evidence from capital providers,” in *Accounting in Europe*, 11 (2), 2014.
- [2] PwC, “Pricewaterhouse Coopers Financial statement presentation - 2016 edition,” Available at <http://www.pwc.com/us/en/cfodirect/publications/accounting-guides/financial-statement-presentation-accounting-guide.html>, 2016.
- [3] R. Libby and S. A. Emmett, “Earnings presentation effects on manager reporting choices and investor decisions,” *Accounting and Business Research*, 2014.
- [4] R. Libby and T. Brown, “Financial statement disaggregation decisions and auditors’ tolerance for misstatement,” *The Accounting Review*, 2012.
- [5] Z. Chen and M. Cafarella, “Automatic web spreadsheet data extraction,” in *Proceedings of the 3rd International Workshop on Semantic Search over the Web*, 2013.
- [6] D. Barowy, S. Gulwani, T. Hart, and B. Zorn, “Flashrelate: extracting relational data from semi-structured spreadsheets using examples,” in *ACM SIGPLAN Notices*, 2015.
- [7] Z. Chen and M. Cafarella, “Integrating spreadsheet data via accurate and low-effort extraction,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014.
- [8] L. Chiticariu, Y. Li, and F. Reiss, “Transparent machine learning for information extraction,” in *EMNLP (Tutorial)*, 2015.
- [9] M. Göbel, T. Hassan, E. Oro, and G. Orsi, “A methodology for evaluating algorithms for table understanding in pdf documents,” in *Proceedings of the 2012 ACM symposium on Document engineering*, 2012.
- [10] B. Yildiz, K. Kaiser, and S. Miksch, “pdf2table: A method to extract table information from PDF files,” in *Proceedings of the 2nd Indian International Conference on Artificial Intelligence*, 2005.
- [11] W. Cohen, “Fast effective rule induction,” in *In Proceedings of the 12th International Conference on Machine Learning*, 1995.
- [12] R. Rastan, H.-Y. Paik, and J. Shepherd, “Texus: A task-based approach for table extraction and understanding,” in *Proceedings of the 2015 ACM Symposium on Document Engineering*, 2015.
- [13] Z. Chen, M. Cafarella, J. Chen, D. Prevo, and J. Zhuang, “Senbazuru: A prototype spreadsheet database management system,” *Proceedings of the VLDB Endowment*, 2013.
- [14] M. D. Adelfio and H. Samet, “Schema extraction for tabular data on the web,” *Proc. VLDB Endow.*, 2013.
- [15] S. K. Jauhar, P. D. Turney, and E. Hovy, “Tables as semi-structured knowledge for question answering,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.
- [16] P. Pasupat and P. Liang, “Inferring logical forms from denotations,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016, pp. 23–32.
- [17] P. Yin, Z. Lu, H. Li, and B. Kao, “Neural enquirer: Learning to query tables with natural language,” *arXiv preprint arXiv:1512.00965*, 2015.
- [18] Y. Liu, P. Mitra, and C. L. Giles, “Identifying table boundaries in digital documents via sparse line detection,” in *Proceedings of the 17th ACM conference on Information and knowledge management*, 2008.
- [19] Y. Liu, K. Bai, P. Mitra, and C. L. Giles, “Improving the table boundary detection in pdfs by fixing the sequence error of the sparse lines,” in *10th International Conference on Document Analysis and Recognition*, 2009.
- [20] S. Seth and G. Nagy, “Segmenting tables via indexing of value cells by table headers,” in *Proceedings of the 2013 12th International Conference on Document Analysis and Recognition*, 2013.
- [21] R. Zanibbi, D. Blostein, and J. R. Cordy, “A survey of table recognition: Models, observations, transformations, and inferences,” *International Journal of Document Analysis and Recognition (IJ DAR)*, 2004.
- [22] D. W. Embley, M. Hurst, D. Lopresti, and G. Nagy, “Table-processing paradigms: A research survey,” *International Journal of Document Analysis and Recognition (IJ DAR)*, 2006.
- [23] B. Couasnon and A. Lemaitre, “Recognition of tables and forms,” in *Handbook of Document Image Processing and Recognition*, 2014.
- [24] Y. Liu, “Tableseer: automatic table extraction, search, and understanding,” Ph.D. dissertation, The Pennsylvania State University, 2009.
- [25] J. Fang, P. Mitra, Z. Tang, and C. L. Giles, “Table header detection and classification,” in *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2012.