# Transforming Arbitrary Tables into F-Logic Frames with TARTAR

Aleksander Pivk [a,b,*], York Sure [b], Philipp Cimiano [b], Matjaz Gams [a], Vladislav Rajkovič [c,a], Rudi Studer [b,d]

[a] *Jozef Stefan Institute, Department of Intelligent Systems, Ljubljana, Slovenia*
[b] *Institute AIFB, University of Karlsruhe, Karlsruhe, Germany*
[c] *Faculty of Organizational Sciences, University of Maribor, Kranj, Slovenia*
[d] *Research Center for Information Technologies (FZI), Karlsruhe, Germany*

**Abstract**

The tremendous success of the World Wide Web is countervailed by efforts needed to search and find relevant information. For tabular structures embedded in HTML documents typical keyword or link-analysis based search fails. The Semantic Web relies on annotating resources such as documents by means of ontologies and aims to overcome the bottleneck of finding relevant information. Turning the current Web into a Semantic Web requires automatic approaches for annotation since manual approaches will not scale in general. Most efforts have been devoted to automatic generation of ontologies from text, but with quite limited success. However, tabular structures require additional efforts, mainly because understanding of table contents requires a table structures comprehension task and a semantic interpretation task, which exceeds in complexity the linguistic task. The focus of this paper is on automatic transformation and generation of semantic (F-Logic) frames from table-like structures. The presented work consists of a methodology, an accompanying implementation (called TARTAR) and a thorough evaluation. It is based on a grounded cognitive table model which is stepwise instantiated by the methodology. A typical application scenario is the automatic population of ontologies to enable query answering over arbitrary tables (e.g. HTML tables).

*Key words:* Table Structure, Table Modeling, Knowledge Frame, Ontology learning, Semantic Web, Query Answering

## 1 Introduction

The World Wide Web has in the years of its exponential growth become a universal repository of human knowledge and culture, thus enabling an exchange of ideas and information on a global level. The tremendous success of the Internet is based on its usage simplicity, efficiency, and enormous market potential [30,45]. The infrastructure of the Internet, as originally planned, was intended for direct user exploitation, because arbitrary information sources provide their services in a user friendly, understandable fashion.

The negative side of the Internet success is evident when one wants to conduct a non-trivial task, i.e. find or gather relevant information. The search of interesting information turned out to be a difficult, time-consuming task, especially due to the size, poor structure and lack of organization of the Internet [3,30]. These are the main reasons for the appearance of a number of approaches in the last decade with common objectives, such as searching and gathering information. One of the first solutions to cope with the information overload was search engines, where a user enters keywords and the engine returns a set of 'interesting' links with their descriptions that best suit the query. The major disadvantage of the approach is that an engine might return thousands of potentially interesting links, for a user to manually explore. The study, described in [30], showed that the number of keywords in queries is in general shorter than three, which clearly cannot sufficiently narrow down the search space. In principle, this can be seen as the problem of users of search engines themselves. Reducing the ambiguity of search requests can be achieved by adding semantics, which can help to improve search results even with a request of a very limited size. In addition, search engines favor largest information providers due to name-branding and time optimization. As a result, this largely reduces the basic advantage of the Internet.

Most information on the Web is presented in semi-structured and unstructured documents, i.e. loosely structured natural language text encoded in HTML, and only a small portion represents structured documents [1,46]. Structured documents are mostly created out of databases using various templates. A semi-structured document is a mixture of loosely structured natural language text and template structures [1,44]. The lack of metadata which would precisely annotate the structure and semantics of documents, and ambiguity of natural

* Corresponding author.
*Email addresses:* aleksander.pivk@ijs.si (Aleksander Pivk), sure@aifb.uni-karlsruhe.de (York Sure), cimiano@aifb.uni-karlsruhe.de (Philipp Cimiano), matjaz.gams@ijs.si (Matjaz Gams), vladislav.rajkovic@fov.uni-mb.si (Vladislav Rajkovič), studer@aifb.uni-karlsruhe.de (Rudi Studer).
*URL:* http://dis.ijs.si/sandi (Aleksander Pivk).

language, in which these documents are encoded, makes automatic computer processing very complex [6,23,34]. The Semantic Web [2] aims to overcome this bottleneck.

Template structures, in particular tabular structures (i.e. tables or lists), incorporated into semi-structured documents, may have many different forms and can differ substantially even if they represent the same content or data [29,25,38]. Efficient handling of such structures and automatic transformation into explicit semantics is a crucial task we deal with in this paper. We developed a method that consists of a methodology, an accompanying implementation and a thorough evaluation.

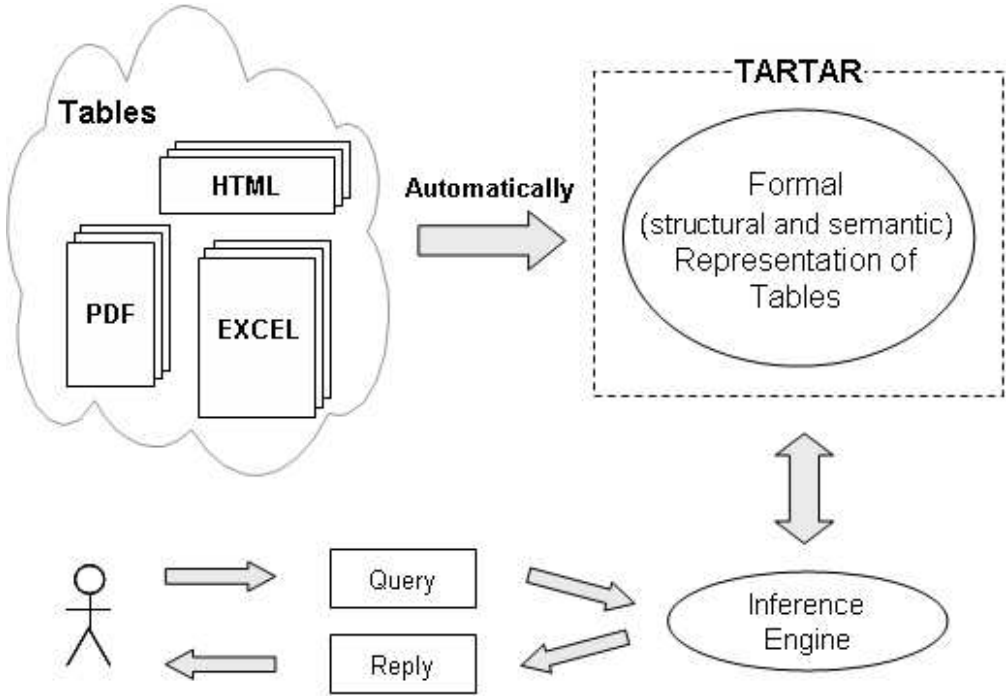



Fig. 1. The desired functionality of our approach.

The problem we are dealing with is represented in Figure 1. The input to the system is semi-structured information in the form of **arbitrary** domain-related (HTML, PDF, EXCEL, etc.) tables. No knowledge of the domain is given to the system – in particular no pre-defined ontology is required. The task of the system is to **automatically** transform semi-structured information into structured, to derive the logical description of input structures and to formalize them in a way that both structural and semantic perspective are considered and explicitly observable. The goal is also to automatically annotate input structures according to the derived/generated schemas and use them, by exploiting an inference engine, to reply to users' queries. This task is in a way similar to those of information extraction, wrapper generation, or in general to Web mining, but the essential request in our view is that

the system must achieve at least 50% success rate in terms of successful frame generation. Note that the success rate of automatic ontology generation for an arbitrary domain-related text is around 20-30% [1,3,7,23,34]. The requested performance of our system is based on the assumption that tables facilitate the extraction of their structure, and the meaning.

The main benefits for applying our method are the following ones:

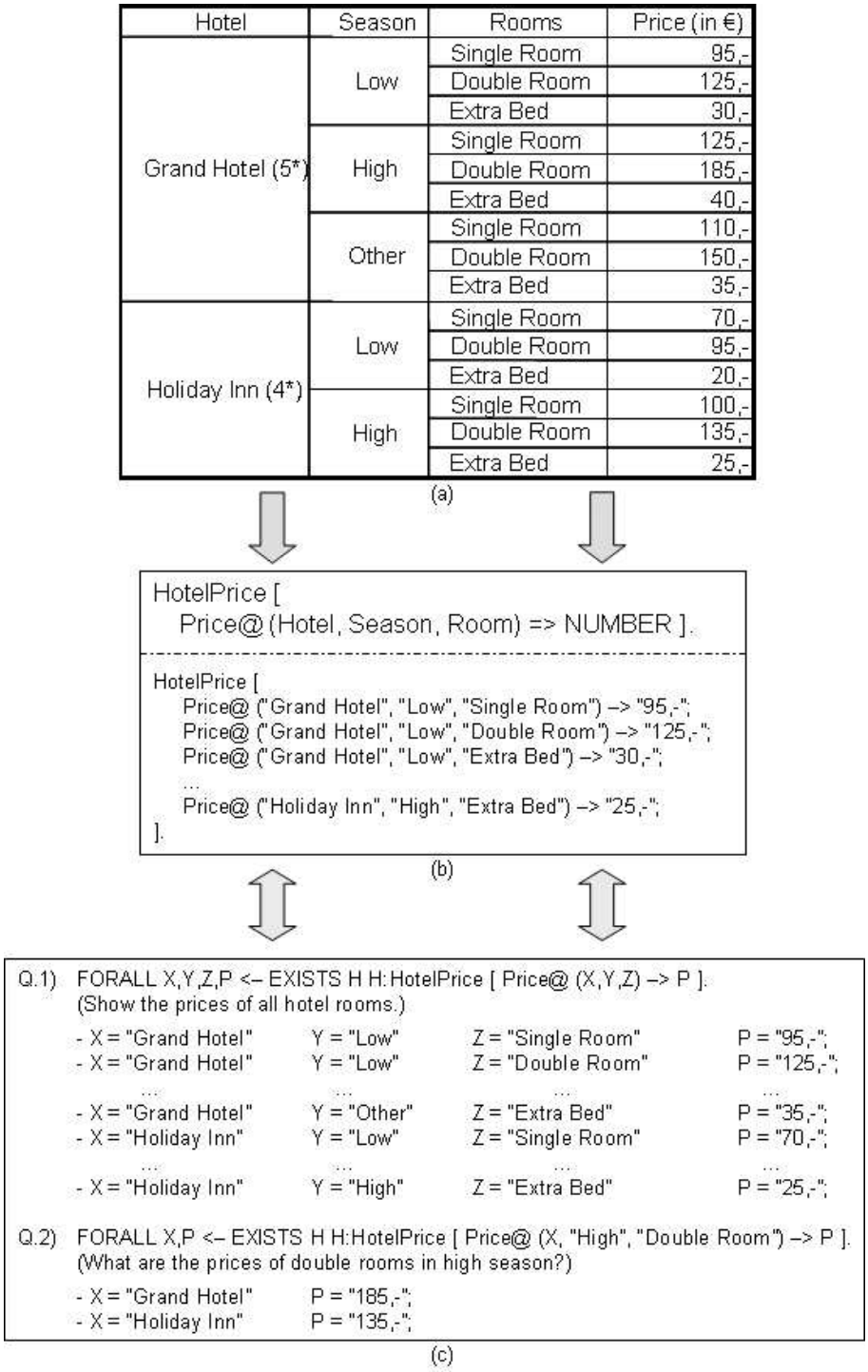
- fully automated knowledge formalization and assimilation of tables,
- applicability for arbitrary tables,
- independence of domain knowledge,
- independence of document types,
- explicit semantics of generated frames,
- enabled query answering over heterogeneous tables.

The method is based on a grounded cognitive table model [27] which is stepwise instantiated by our methodology. It is implemented in a system named TARTAR [1] (Transforming ARbitrary TAbles into fRames) which is responsible for actual transformation of arbitrary tables into explicit semantics, in particular F-Logic frames. Since there are an endless number of table layout variations we identified a couple of most relevant table types which were used in the experimental setting during the evaluation of our approach.

F-Logic is the acronym for Frame Logic [32], and integrates features from object-oriented programming, frame-based knowledge representation languages and first order logic. At first it was used for deductive and object-oriented databases, but was later adapted and used for representing ontologies. The major advantages for choosing it as a representation language are: model-theoretic semantics, sound and complete resolution-based proof theory, expressive power of logic and availability of efficient reasoning tool support. Further details as well as a brief introduction of the syntax can be found in Section 5.

Slightly more detailed example of the overall table transformation is presented in Figure 2. The figure consists of three parts where (a) represents a table describing information from a tourist domain, in particular about accommodation prices in hotels, (b) depicts the output of the system, in particular the automatically generated frame encoded in F-Logic and accordingly formalized table content, together constituting an object base, and (c) gives two possible queries that could be posted against the object base with respective results returned by an inference engine. The queries are written in F-Logic syntax (see Section 5 for details), but for a better readability also their natural language equivalents are provided. The natural language interface to an inference engine, i.e. OntoBroker, is not part of this paper, as it is described in [5].

[1] Source code of the system is freely available and can be downloaded from [47].



| Hotel | Season | Rooms | Price (in €) |
|---|---|---|---|
| Grand Hotel (5*) | Low | Single Room | 95,- |
| | | Double Room | 125,- |
| | | Extra Bed | 30,- |
| | High | Single Room | 125,- |
| | | Double Room | 185,- |
| | | Extra Bed | 40,- |
| | Other | Single Room | 110,- |
| | | Double Room | 150,- |
| | | Extra Bed | 35,- |
| Holiday Inn (4*) | Low | Single Room | 70,- |
| | | Double Room | 95,- |
| | | Extra Bed | 20,- |
| | High | Single Room | 100,- |
| | | Double Room | 135,- |
| | | Extra Bed | 25,- |

(a)

```
HotelPrice [
   Price@ (Hotel, Season, Room) => NUMBER ].
```

```
HotelPrice [
    Price@ ("Grand Hotel", "Low", "Single Room") –> "95,-";
    Price@ ("Grand Hotel", "Low", "Double Room") –> "125,-";
    Price@ ("Grand Hotel", "Low", "Extra Bed") –> "30,-";
    ...
    Price@ ("Holiday Inn", "High", "Extra Bed") –> "25,-";
].
```

(b)

```
Q.1)  FORALL X,Y,Z,P <– EXISTS H H:HotelPrice [ Price@ (X,Y,Z) –> P ].
      (Show the prices of all hotel rooms.)
      - X = "Grand Hotel"   Y = "Low"     Z = "Single Room"   P = "95,-";
      - X = "Grand Hotel"   Y = "Low"     Z = "Double Room"   P = "125,-";
            ...                 ...             ...                 ...
      - X = "Grand Hotel"   Y = "Other"   Z = "Extra Bed"     P = "35,-";
      - X = "Holiday Inn"   Y = "Low"     Z = "Single Room"   P = "70,-";
            ...                 ...             ...                 ...
      - X = "Holiday Inn"   Y = "High"    Z = "Extra Bed"     P = "25,-";

Q.2)  FORALL X,P <– EXISTS H H:HotelPrice [ Price@ (X, "High", "Double Room") –> P ].
      (What are the prices of double rooms in high season?)
      - X = "Grand Hotel"   P = "185,-";
      - X = "Holiday Inn"   P = "135,-";
```

(c)

Fig. 2. The objective of applying our method is to transform (a) an arbitrary input table into (b) an F-Logic frame with formalized data which (c) subsequently supports the ontology population and query answering using an inference engine OntoBroker [12].

Note that in the rest of the paper we will use table examples from two different domains. In particular, tables in Figures 2, 4(a), 4(b), and 5 belong to a tourist

domain; the rest of the tables are based on our running example shown in Figure 3 which belongs to a student courses domain. By this we want to show that our system is able to deal with information from different domains and is not bound to a single domain since there are no domain-specific operations in our approach. Also, our method can in general be applied to any table layout description (i.e. html, excel, pdf, text, etc.) but as a proof-of-concept we implemented it prototypically only for Web tables.

The paper is structured as follows. In Section 2 we first introduce the grounding table model which forms the base for our stepwise methodology to generate frames out of tables. Subsequently we explain each methodological step in detail and also show relevant substeps. In Section 4 we present a thorough two step evaluation of the accompanying implementation. Section 5 describes potential applications where the results of this work could make a significant impact, particulary as related to the Semantic Web. After presenting the related work in Section 6 we conclude in Section 7.

## 2 Tabular Structures

Tables are one of the most popular and commonly used structures when it comes to presenting, visualizing or comparing data [3,9,22,28]. More precisely, they visualize indexing schemes for relations, which may be understood as a set of $n$-tuples where $n$ is the number of dimensions in the relation [15]. A relation may be presented in a table in many different ways while its dimensions may be laid out in different row or column arrangements, repeated, or ordered in various ways. The organization of dimensions in a table affects which data are most easily accessed and compared [11,22].

Figure 3 describes a student's course results for various terms and assignments/exams and will be used throughout the paper as a running example. The presented table comprehends a typical table anatomy and corresponds to an object (a) in Figure 2. It is composed of two independent parts (different logical orientation), where the first two rows describe course and student information, while the rest of the table presents assignment/exam results. Table terminology is given only on the latter table part, since it includes all types of information, is (visually) more descriptive, and therefore adequate for our presentation purposes. A *dimension* is a grouping of cells representing similar entities. It may also contain a *dimension header* which usually semantically describes its elements. For example, the header 'Final Grade' semantically describes the dimension elements: 77.5, 67.3,...,61.3. Relation dimensions, whose content is to be searched and compared, have their elements located in the *body*; the remaining dimensions, typically located on the left-hand side, named *stubs*, usually serve as headers and are used to index elements located in the

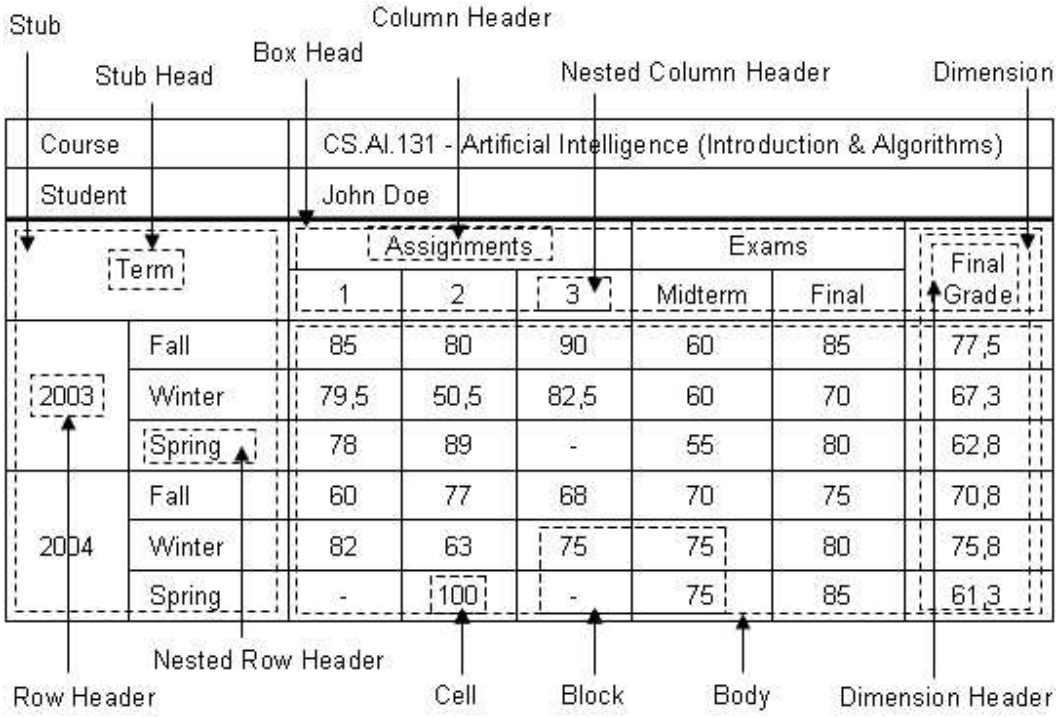



Fig. 3. The figurative example and terms shown here are taken from [53] where Chicago Manual of Style [16] terminology is used.

body. A table *cell* is an individual dimension element; a group of topologically contiguous cells in the body is referred to as a *block*. The headers, placed in a *box head* (such as a (*nested*) *column header*), or in a stub (such as a *stub head*, or a (*nested*) *row header*), are often nested - hierarchically organized - to visually bind dimension names with elements and to factor one dimension by another. The headers typically contain a conceptual naming or semantic description of the elements they are bound to. For example, the dimension 'Term' in Figure 3 is factored by 'year' in the stub of the table, which is indicated by nesting the elements (e.g. 'Winter') below the 'year' elements (e.g. '2004').

The implementation of our methodological approach so far handles only tables found in Web documents but in general it can handle tables in arbitrary types and domains. In Web tables header nesting is usually indicated by special HTML tags, in particular *rowspan* and *colspan*, which cannot be shown in Figure 3, since they serve as layout arguments. In particular, if the example was encoded in HTML, then the header cell 'Course' spanning two columns would contain an argument colspan with value two, and the cell '2004' spanning three rows would contain an argument rowspan with value three.

Linguistic models traditionally describe natural language in terms of syntax and semantics. There also exist models to describe tables in similar ways (cf. [27,28,53]) where tables are analyzed along the following aspects:

- **Graphical** – an image level description of the pixels, lines and text or other content areas,
- **Physical** – a description in terms of inter-cell relative location,
- **Structural** – the organization of cells (topology) as an indicator of their navigational relationship,
- **Functional** – the purpose of areas of the tables in terms of data access,
- **Semantic** – the meaning of text in the table and the relationship between the interpretation of cell content, the meaning of structure in the table and the meaning of its reading.

Our approach builds on the model described above with some modifications. First, we will not consider the *graphical* dimension. Regarding the *physical* dimension, we will process the tables encoded in HTML format in order to get a physical model of the table. Inspection of the physical model will tell us the relative position of the cells. The relative position is calculated by finding the minimum number of overlaid rows and columns required to capture all the cells in the table.

In order to capture the *structural* dimension of the table, further processing is necessary. By structure, we mean that aspect of the table which restricts the manner in which we may navigate the table. The structure relates firstly to the functional component of the model, and secondly to the semantic view of the table. Firstly, structure facilitates physical economy, where the organization of cells allows the author to juxtapose certain information bearing elements to imply certain concepts. The second point relates to the semantics holding between the content bearing elements of the table. Organizing the table physically helps to indicate where certain inter-cell relationships exist. Clearly, these two aspects of structure contribute to ambiguities in the interpretation of the system.

*Functional* description concerns the reading of tables (i.e. local or global search) and functional regions, where Hurst [28] distinguishes between two functional cell roles *access* and *data*. Cells of role data are the ones users are interested in when reading a table and which contain the actual information, while cells of role access determine the path (index) to follow in the table in order to find the data cell of interest. Hurst also distinguishes *local* (looking for one specific data cell) from *global* (comparing the value of different data cells) search in a table. In our approach we describe the functional dimension of a

table in order to support the global search, where a table's indexing scheme [8] needs to be discovered. Such a functional description requires to: (a) find all data and access regions in a table, and (b) discover relational information in order to support global search.

In our method we distinguish between two *functional types* of cells: A(ttribute)-cells and I(nstance)-cells. A-cells describe the conceptual nature of the instances in a table. I-cells represent the actual data cells or the instances of the concepts represented by a certain A-cell. I-cells can have two different *functional roles* described by Hurst, i.e. each cell plays a role of data or access cell. For example, the cells 'Final Grade' or 'Term' are of functional type A-cell, where the first semantically describes I-cells '77.5', '67.3', ..., '61.3', while the second one describes I-cells '2003'...'2004', and also I-cells 'Fall', 'Winter', and 'Spring'. The first group of I-cells are of functional role data, and the latter ones of role access.

A *semantic* model of linguistic communication ultimately describes what is true about the world (possible world semantics), which should also be the goal of the semantic interpretation od information presented in tables. However, it is important to recognize that this task exceeds in complexity the linguistic tasks. Consequently, we investigate the notion of a semantic interpretation of tables in a number of levels. We aim to cover the following topics, as required by Hearst [28], where each topic contributes conceptually and systematically to our semantic view of the table:

- Relation Semantics: a table is viewed as a relational information structure,
- Cell Content: the relation semantics advance a truth functional model of the table cells, however, the cells themselves are complex semantic entities and require further analysis, and
- Inter-cell Relationships: relationships hold between the interpretation of the cell content elements in different cells.

Regarding the semantic description we chose a slightly different paradigm as Hurst. Instead of only adopting the relational model [8], we describe the semantics of a table in terms of F-Logic frames [32]. F-Logic combines the intuitiveness of modeling with frames and the expressive power of logic. By resorting to F-Logic, which complies to part (b) of Figure 2, we are thus able to describe the semantics of tables in a model-theoretic way. Furthermore, as required by Hurst, the frame makes explicit: (a) the meaning of cell contents, (b)the functional dimension of the table, and (c) the meaning of the table based on its structure. Furthermore, existing F-Logic inference engines such as Ontobroker [12] allow later on query answering over multiple heterogeneous tables. Therefore it was our primary choice as representation language.

In the next subsection we present a limited vocabulary of physical arrangements of tables which enable the organization of their elements.

### *2.2 Table Classes*

In this paper we will restrict ourselves to the tables that present mostly textual data, encoded in HTML documents, although our method is in general capable of covering other document types, as already shown in Figure 1. HTML tags are used to define the table grid (contents and relative positions of the cells), and may also define the header (box head), and footer areas. On the other hand, HTML does not encode the stub location, indexing structure, or underlying relation of a table. In practice, tables encoded in HTML often do not use the header and footer tags, and use tags that are not part of the table tag set (i.e. paragraph tag). Also, the table environment is often used to layout lists and matrices of data in a grid with no indexing structure [29,57].

In practice there is an endless number of table variations and forms. For example, in Figure 3 headers or explanatory text might appear in the body of a table. In some cases, tables even contain tables within cells, or are compositions of tables, producing complicated indexing structures [22,53]. However, the majority of table subtypes studied in the literature are encapsulated in Figure 3.

We have identified three major table layout classes that appear frequently on the Web: *1-Dimensional* (1D), *2-Dimensional* (2D), and *Complex* tables as shown in Figures 4(a), 4(b), and 5, respectively. The first two classes are rather simple and appear more often compared to the last class. A similar classification into classes has also been introduced in [52].

| Trip Code | Trip Duration (in days) | Cost | Insurance |
|---|---|---|---|
| LM202 | -7 | 520 | 50 |
| LM208 | 9-15 | 725 | 70 |
| LM209 | 16-23 | 949 | 90 |
| LM311 | 23-31 | 1495 | 120 |
| LM223 | 32-45 | 2275 | 195 |
| XM001 | 46-60 | 3180 | 220 |

(a) One-dimensional (1D) table

| | Departure | Arrival |
|---|---|---|
| **City:** | Dallas/Ft Worth, TX (DFW) | Honolulu, HI (HNL) |
| **Scheduled:** | Mar 16 - 11:45am | Mar 16 - 4:20pm |
| **Actual:** | Mar 16 - Not Available | Mar 16 - Not Available |
| **Gate/Terminal:** | A29 | 18 |
| **Baggage Claim:** | - | F2 |

(b) Two-dimensional (2D) table

Fig. 4. One-dimensional (1D) and two-dimensional (2D) tables.

**1-Dimensional tables:** this class of tables has a box head and at least one-

line body. If the box head consists of nested (hierarchically organized) headers then we assume they are somehow related. The content of the body cells represents the instances of header cells above. An example of this type is given in Figure 4(a) where information about trips and corresponding costs are depicted.

**2-Dimensional tables:** this class of tables has a box head, a stub and a rectangular table body. Discovering, handling, and transforming this class is hard as it is difficult for a system without any prior knowledge to decide if the stub content presents headers (A-cells) or instances (I-cells) Our solution here is to interpret the stub as A-cells only if a stub header is a non-spanning cell with an empty label or a label containing a character '/'. An example of this type is given in Figure 4(b) where the information on a particular flight is given.

| Rate (%) | Regular | Float |
|---|---|---|
| *Regular Fixed Deposit* | | |
| 1 Year | 5,05 | 5,05 |
| 2 Years | 5,1 | 5,1 |
| 3 Years | 5,1 | 5,1 |
| *Fixed Deposit* | | |
| 3 Months | 4,35 | 4,35 |
| 6 Months | 4,6 | 4,6 |
| 9 Months | 4,7 | 4,7 |
| 1 Year | 5 | 5 |
| 2 Years | 5,05 | 5,05 |
| 3 years | 5,05 | 5,05 |

| *Regular Fixed Deposit* | | |
|---|---|---|
| **Rate (%)** | **Regular** | **Float** |
| 1 Year | 5,05 | 5,05 |
| 2 Years | 5,1 | 5,1 |
| 3 Years | 5,1 | 5,1 |
| *Fixed Deposit* | | |
| **Rate (%)** | **Regular** | **Float** |
| 3 Months | 4,35 | 4,35 |
| 6 Months | 4,6 | 4,6 |
| 9 Months | 4,7 | 4,7 |
| 1 Year | 5 | 5 |
| 2 Years | 5,05 | 5,05 |
| 3 Years | 5,05 | 5,05 |

Fig. 5. Two variations of a complex partition table with the same content.

**Complex tables:** this class of tables shows a great variety in layout structure. The examples of such a table type can be observed in Figures 3 and 5. The first figure refers to our running example table; the latter one describes various deposit types and applicable rates. A complex table might have the following features:

- **Partition data labels:** Special over-spanning data labels between the data and/or header labels mark several partitions of the table. Each partition shares the same headers, such as in Figure 5. In this case the connection between the headers and their instance values cannot be obtained directly.
- **Over-expanded labels:** some entries might expand over multiple cells. There are two options: (a) instance values span over multiple rows in the same column or (b) a header spans over multiple columns. An example of this class is shown in Figure 5, in particular the lower six rows of the table.
- **Combination:** large tables might consist of several smaller, simpler ones.

For example, Figure 3 consists of two structurally 'independent' tables. Detecting and partitioning of complex tables into smaller parts is usually a very hard task, in some cases even not feasible for computers.

The system tries to fit, if feasible, each table (or sub-tables in case of a complex table) to one of these classes during the transformation process, and acts accordingly.

## 3 Methodological Approach

Our methodology consists of four main steps as depicted in Figure 6. For each building block of the table model there exists a corresponding methodological step to create this part of the table model. The methodology is fully implemented in the TARTAR system, where tables, encoded in HTML, serve as input to the system. These tables are cleaned and canonicalized in the first step. In each subsequent step respective actions are taken until finally reaching the semantic level of transformation process. Note that the first step returns a representation that is independent of the input document encoding type which is then used in the subsequent steps. This kind of processing makes it very straightforward for extending our approach for other input document encoding types. As output the system returns F-Logic frames, where each frame corresponds to a particular input table. The same process is presented as a flowchart in Figure 7, this time with an emphasis on the algorithm. In the following subsections we will describe all steps in details.

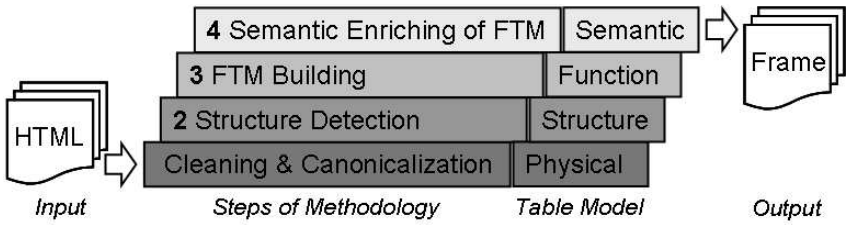



Fig. 6. Building blocks of the methodology, complying to the transformation from (a) to (b), presented in Figure 2.

### *3.1 Cleaning and Canonicalization*

It is assumed that the input HTML documents are represented with respect to the DOM (Document Object Model) [13]. A *DOM tree* is an ordered tree, where each node is either an element or a text node. An element node includes an ordered list of zero to many child nodes, and contains a string-valued tag

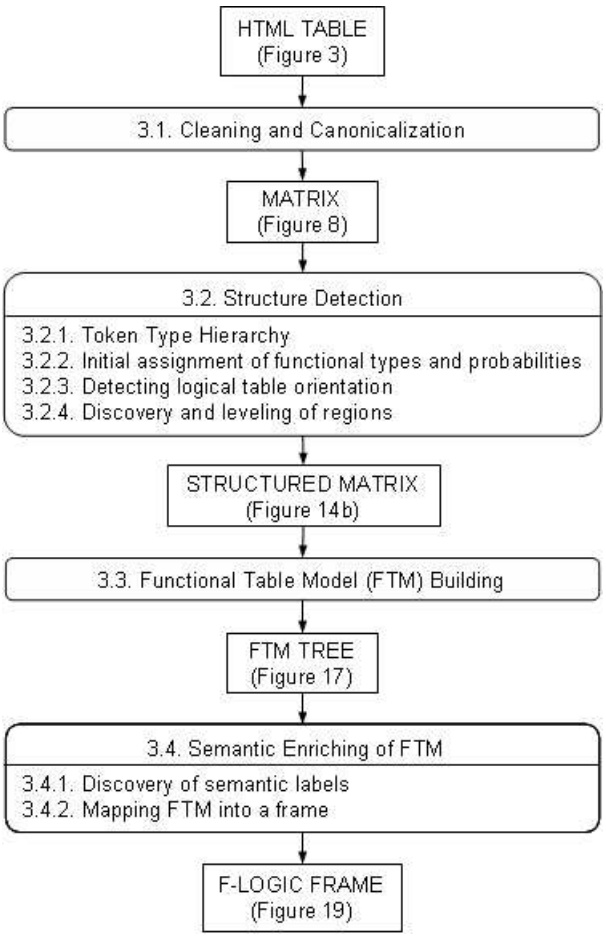



Fig. 7. Flowchart of the proposed methodology.

(i.e. 'table' or 'h1') and zero to many string-valued attributes (i.e. 'href' or 'src'). A text node normally contains a single text string and has no child nodes.

We want to construct an initial table model out of a DOM tree. This model cannot be simply generated by applying the algorithm recommended by W3C [24] on a table node, therefore some additional steps of processing and refinement are required. Web documents are often very noisy in a sense that their syntactic structure is incorrect. In order to clean the code and make it syntactically correct, we employ the CyberNeko HTML Parser [41]. The outcome is a cleaned and corrected DOM tree.

The canonicalization of the rendered table is necessary when an explicit tag attribute (rowspan or colspan) indicates multiple row or column spanning cells and the actual total number of rows or columns differs from the attribute value. In this step our system updates the corresponding DOM subtrees accordingly.

Figure 8 shows the reformulation of the table in Figure 3, where cleaning has been performed and copies of cells with rowspan and colspan tag attributes have been properly duplicated into the matrix structure.

The size of each cell in the matrix is by default one, but the logical size of the overexpanding cells, which is defined by rowspan/colspan tag attributes values, is greater. For example, the cell 'Term', which extends over two columns and two rows, as can be seen in Figure 8, is assigned the logical size $2 \times 2$.

| Course | Course | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - |
|---|---|---|---|---|---|---|---|
| Student | Student | John Doe | John Doe | John Doe | John Doe | John Doe | John Doe |
| Term | Term | Assignments | Assignments | Assignments | Exams | Exams | Final Grade |
| Term | Term | 1 | 2 | 3 | Midterm | Final | Final Grade |
| 2003 | Fall | 85 | 80 | 90 | 60 | 85 | 77,5 |
| 2003 | Winter | 79,5 | 50,5 | 82,5 | 60 | 70 | 67,3 |
| 2003 | Spring | 78 | 89 | - | 55 | 80 | 62,8 |
| 2004 | Fall | 60 | 77 | 68 | 70 | 75 | 70,8 |
| 2004 | Winter | 82 | 63 | 75 | 75 | 80 | 75,8 |
| 2004 | Spring | - | 100 | - | 75 | 85 | 61,3 |

Fig. 8. Cleaned and canonicalized matrix representation of the table shown in Figure 3.

In general, the method is not bound to a single document type nor domain since there are no domain-specific operations in the approach. Note that only this methodological step depends on the table layout description and that from this point onwards the system is unaware of the incoming document type.

### *3.2 Structure Detection*

In this section we first introduce a token type hierarchy (3.2.1) which serves as the backbone for various calculations and assignments. Later we describe three main processes of this methodological step as follows: the assignment of functional types and probabilities to cells (3.2.2), the detection of table orientation (3.2.3), which indicates the manner of navigation, and the region discovery process (3.2.4). All these processes are accompanied with running example descriptions.

#### *3.2.1 Token Type Hierarchy*

The hierarchy of data types has been developed in order to enable the assignment of types to the content tokens. The hierarchical organization is flexible and explicitly presents the distance $\delta$ among different types, which is measured in the number of edges among corresponding nodes. These properties are widely exploited in the following steps of the methodology, especially for different calculations and comparisons.

Figure 9 presents a hierarchy constituted of two main parts. On the one hand there are alpha-numeric data types that are divided into three main categories, such as the ones representing punctuation marks or other special symbols/characters, character strings, and numeric values. The latter two are further divided into four subcategories. The strings are categorized according to the case of the letters within them. The numeric type covers micro ($< 1$), small ($1 \leq n < 100$), medium ($100 \leq n < 10.000$), and large ($\geq 10.000$) values, where a negative numeric value is assigned the type according to its absolute value.

On the other hand the hierarchy is extendable to include composed data types. Composed data types usually cover more successive tokens at once and include known facts from a real world, domain specific information, etc. Figure 9 presents only two composed data types, namely date and currency, but more are actually included. Date type is responsible for recognizing different representation of dates, where currency type recognizes monetary properties of consecutive tokens.

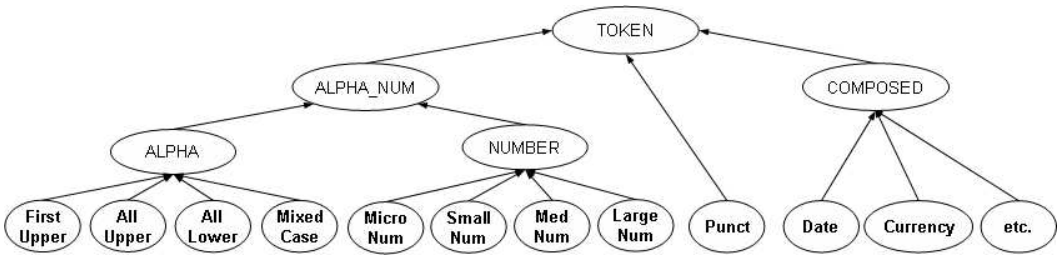



Fig. 9. Hierarchy of token types.

### *3.2.2 Assignment of functional types and probabilities to cells*

In the initial pass over the table element node of the DOM tree, we convert a sub-tree into a matrix structure, which is populated by cells according to its layout information as shown in Figure 8. During this step the text of each cell is tokenized, and each token is assigned a *token type* according to the hierarchy tree leaves presented in Figure 9. Lets define the required structures, as follows:

**Definition 1 (Cell Vector)** *The cell is represented as a vector $t$ of all the tokens in the cell. Henceforth, $t_i$ denotes the i-th component of the vector $t$, and $|t|$ the size of the vector.*

**Definition 2 (Token Type Vector)** *A cell $t$ is represented as a vector $c$ of token types, where $c_i$ denotes the i-th component of the vector $c$, corresponding to the token type of $t_i$.*

For example, the cell located in the first row and the third column of the canonicalized table, presented in Figure 8, is represented as a cell vector $t_{1,3} =$

{*'CS', '.', 'AI', '.', '131', '-', 'Artificial', 'Intelligence', '(', 'Introduction', '&', 'Algorithms', ')'*} and a corresponding token type vector $c_{1,3}$ = {*All_Upper, Punct, All_Upper, Punct, Med_Num, Punct, First_Upper, First_Upper, Punct, First_Upper, Punct, First_Upper, Punct*}.

At the same time, each cell in the rendered table is assigned a *functional type* and the probability of that type. By default, a cell is assigned either no functional type (probability value equals zero), or I-cell type, in case it includes only/mostly($> 50\%$) tokens, recognized as dates, currencies, or numerical values. Its probability is then calculated based on the portion of these relevant tokens. Finally, we assume that the cell in the lower-right corner is always an I-cell, and the cell in the upper-left corner is an A-cell. Therefore we assign those two cells the types, regardless of their content, with probability one.

In the structure detection block several assumptions, regarding languages and reading orientations, are made. These assumptions are based on the fact that the prototypical implementation can deal only with documents encoded in (western) languages, that read from left-to-right, i.e. English. Eastern languages, reading from right-to-left, are not considered here, but it is straightforward to extend our approach to properly cover them. In particular, each extracted table from a document encoded in such a language is altered into the matrix structure, which requires an additional transformation, i.e. mirroring, rotation, or transposition, depending on the language in question. These transformations enable a suitable representation that our approach can handle in the following steps.

### *3.2.3 Detecting logical table orientation*

One problem related to the interpretation of a table is that its logical orientation is a priori not clear. In fact, when performing a local search on a table, the data of interest can be either ordered in a top-to-down (vertical orientation) or left-to-right manner (horizontal orientation). For example, in Figure 3 the relationship ('Course, CS.AI.131...') reads left-to-right, but grades of an attribute 'Midterm' appear top-to-down.

When trying to determine the table orientation we rely on the similarity of cells. The intuition here is that if rows are similar to each other, then orientation is vertical and on the contrary, if columns are similar, then interpretation is horizontal. In order to calculate the differences among rows and columns of the table, we need first to define how to calculate the difference between two cells. When comparing the token type vectors of two cells, we compare the token types with same indices in case the vectors have equal length; otherwise, we calculate the distance for the left-side tokens (tokens aligned at the head) and for the right-side tokens (tokens aligned at the tail). The distance is in

both cases also averaged.

$$\delta_{cells}(c_P, c_Q) = \begin{cases} \frac{w}{u} \left( \sum_{i=1}^{u} \delta(c_{P_i}, c_{Q_i}) + \delta(c_{P_{|c_P|-i+1}}, c_{Q_{|c_Q|-i+1}}) \right) & \text{if } u \neq v \\ \\ \frac{1}{u} \sum_{i=1}^{u} \delta(c_{P_i}, c_{Q_i}) & \text{otherwise} \end{cases} \quad (1)$$

where $u = min(|c_P|, |c_Q|)$, $v = max(|c_P|, |c_Q|)$ and $w = v - u + 1$. Now, given a table with $r$ rows and $s$ columns, the total distance ($\Delta_{cols}$) between columns is calculated by summing up the distance between the last column and each of the preceding $m - 1$ columns, where $m = min(r, s)$, i.e.

$$\Delta_{cols} = \sum_{i=1}^{m-1} \delta_{cols}(col_{s-i}, col_s) \quad (2)$$

$$\delta_{cols}(col_p, col_q) = \sum_{i=r'}^{r} \delta_{cells}(c_{i,p}, c_{i,q}) \quad (3)$$

where $c_{x,y}$ is the cell in row $x$ and column $y$, and

$$r' = \begin{cases} r - m + 1 & \text{if } r > m \\ 1 & otherwise \end{cases} \quad (4)$$

The total distance ($\Delta_{rows}$) between rows is by analogy calculated by summing up the distance between the last row and each of the $m - 1$ preceding rows:

$$\Delta_{rows} = \sum_{i=1}^{m-1} \delta_{rows}(row_{r-i}, row_r) \quad (5)$$

$$\delta_{rows}(row_p, row_q) = \sum_{i=s'}^{s} \delta_{cells}(c_{p,i}, c_{q,i}) \quad (6)$$

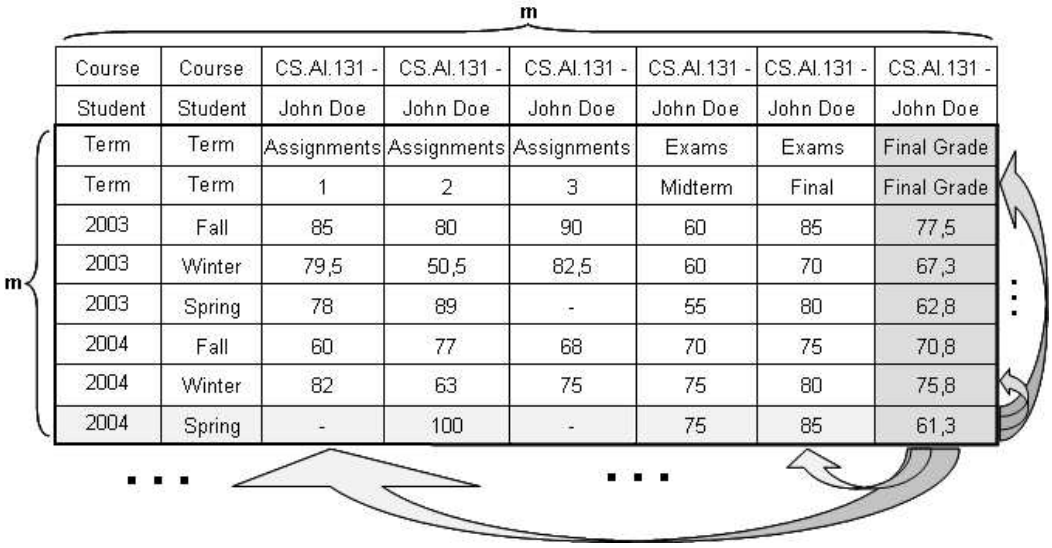
where

$$s' = \begin{cases} s - m + 1 & \text{if } s > m \\ 1 & otherwise \end{cases} \quad (7)$$

For example, if we compare rows $r_1 = \{1, 2, 3\}$ and $r_2 = \{400, 5, -\}$ then

$$\begin{aligned} \Delta_{rows}(r_1, r_2) &= \delta_{cells}(c_{r_1,1}, c_{r_2,1}) + \delta_{cells}(c_{r_1,2}, c_{r_2,2}) + \delta_{cells}(c_{r_1,3}, c_{r_2,3}) = \\ &= (2 + 0 + 4) = 6 \end{aligned} \quad (8)$$

Here we only compare an equal number of rows and columns, starting at the lower-right corner, thus optimizing the number of comparisons (not all the rows and columns to the top of the table need to be compared). Finally, to determine the orientation of the table, we compare both results. If the distance among columns is greater than among rows ($\Delta_{cols} > \Delta_{rows}$), orientation is set to *vertical* (top-to-down). On the other hand, if the distance among columns is lower than among rows ($\Delta_{cols} < \Delta_{rows}$), then orientation is set to *horizontal* (left-to-right). In the last case, where the two results are equal, orientation is assigned as default, *vertical*.




| Course | Course | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - |
|---|---|---|---|---|---|---|---|
| Student | Student | John Doe | John Doe | John Doe | John Doe | John Doe | John Doe |
| Term | Term | Assignments | Assignments | Assignments | Exams | Exams | Final Grade |
| Term | Term | 1 | 2 | 3 | Midterm | Final | Final Grade |
| 2003 | Fall | 85 | 80 | 90 | 60 | 85 | 77,5 |
| 2003 | Winter | 79,5 | 50,5 | 82,5 | 60 | 70 | 67,3 |
| 2003 | Spring | 78 | 89 | - | 55 | 80 | 62,8 |
| 2004 | Fall | 60 | 77 | 68 | 70 | 75 | 70,8 |
| 2004 | Winter | 82 | 63 | 75 | 75 | 80 | 75,8 |
| 2004 | Spring | - | 100 | - | 75 | 85 | 61,3 |

Fig. 10. An example of table orientation detection.

Figure 10 shows how the orientation is calculated on our example where the system compares $m - 1$ columns to the column $m$ and $m - 1$ rows to the row $m$. The difference among columns is greater than among rows, hence the orientation is set to vertical.

In case a table is assigned the horizontal reading orientation, the system performs transpose of the matrix (replacing the cell pairs $a_{i,j}$ with $a_{j,i}$), which returns the table matrix in the vertical orientation. Note that from this point onwards it is assumed that all tables are to be read in the vertical fashion. In this way we simplify the subsequent processing steps while only one reading direction needs to be covered.

There are also special cases (such as table in Figure 3), where the table consists of several 'independent' tables, and each one of them may have a different interpretation. In such cases, we first assign the table the orientation as described, and later (when we discover that the table actually consists of several 'independent' logical units) split the table and assign each one a respective orientation.

*3.2.4 Discovery and leveling of regions*

Here we present an algorithm for discovering regions in tables. The motivation is to determine which data represents similar instances and forms table dimensions. First we give definitions of a logical unit and a region, followed by a step-by-step description of the algorithm's pseudocode given in Figure 11.

**Definition 3 (Logical unit)** *A logical unit (LU) is a part of a table produced by a horizontal split in case of vertical reading orientation or by a vertical split in case of horizontal orientation.*

**Definition 4 (Region)** *A region is a rectangular area of a table consisting of cells with the same functional type which must appear within one logical unit.*

---

(A.1) Initialize logical units and regions
(A.2) Learn string patterns of regions
**for** all non-uniform logical units $LU_k$
  **repeat**
    (A.3) Select the best coherent region and set the sub-unit
    (A.4) Level neighboring regions within the sub-unit
  **until** logical unit $LU_k$ is not uniform

---

Fig. 11. Region discovery and leveling algorithm.

*A.1. Initialize logical units and regions.* Splitting a table into regions enables the system to form table dimensions by binding body data with respective header(s). There are two major steps in this subtask: (a) splitting of a table into logical units, and (b) discovery of initial regions within logical units.

The first step is to split a table into logical units. As we deal with vertical orientation only, the horizontal split occurs at every row that satisfies at least one of the following conditions: (a) if a row contains a cell spanning multiple columns, or (b) if a row contains an over-spanning cell of type I-cell with its probability higher that a treshold value. While splitting the table into initial logical units the system tries to merge the consecutive ones. Two consecutive logical units are merged only if their layout structure is equal and the same split condition(s) applied for a split. For example, the first and the second row in Figure 12 are merged for having the same structure and also contain an over-spanning cell of type I-cell. Note that a table itself is by definition one logical unit.

Figure 12 depicts our table example which is split into three (horizontal) logical units. The first and second row have multiple-column spanning cells

(i.e. 'CS.AI.131 - ...' and 'John Doe', respectively) of I-cell type and equal layout structure and are therefore grouped into logical unit LU1. A third and fourth row have a common cell ('Term') spanning multiple columns (and rows), which results in a common logical unit LU2. The rest of the table is grouped into logical unit LU3.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| LU1 | Course<br>Student | Course<br>Student | CS.AI.131 -<br>John Doe | CS.AI.131 -<br>John Doe | CS.AI.131 -<br>John Doe | CS.AI.131 -<br>John Doe | CS.AI.131 -<br>John Doe | CS.AI.131 -<br>John Doe | LU1 |
| LU2 | Term<br>Term | Term<br>Term | Assignments<br>1 | Assignments<br>2 | Assignments<br>3 | Exams<br>Midterm | Exams<br>Final | Final Grade<br>Final Grade | LU2 |
| LU3 | 2003 | Fall | 85 | 80 | 90 | 60 | 85 | 77,5 | LU3 |
| | 2003 | Winter | 79,5 | 50,5 | 82,5 | 60 | 70 | 67,3 | |
| | 2003 | Spring | 78 | 89 | - | 55 | 80 | 62,8 | |
| | 2004 | Fall | 60 | 77 | 68 | 70 | 75 | 70,8 | |
| | 2004 | Winter | 82 | 63 | 75 | 75 | 80 | 75,8 | |
| | 2004 | Spring | - | 100 | - | 75 | 85 | 61,3 | |

Fig. 12. An example table after initial split into logical units.

Once splitting is over, the process of determining region boundaries within logical units begins. The system starts at a lower-right corner of the lowest logical unit and proceeds upwards over all logical units. The region initialization is performed at one time within one logical unit. As the orientation is assumed vertical, the initialization proceeds from the rightmost column towards the leftmost column trying to initialize all regions within each column. The cell $c_k$ is added to a temporary region $r$ if the lower conditions apply, otherwise the temporary region is added to a set of initial regions $R$ and a new temporary region with an element $c_k$ is created. The conditions for extending a region are as follows:

(1) the cell $c_k$ is within the same logical unit as other cells of region $r$,
(2) its logical size is equal to the size of cells in $r$, and
(3) it keeps the distance among cells in $r$ within a threshold value, defined by an equation $\delta_{cells}(c_k, c_{1(r)}) \leq 2.8$, where $c_{1(r)}$ denotes the first cell added to a region $r$, and the value of 2.8 reflects a significant token type change according to the token type hierarchy depicted in Figure 9. In this way a set $R$ of all initial regions over all logical units is created.

*A.2. Learn string patterns for regions.* For each region $r \in R$ we learn a set $P_R$ of significant patterns, which are used later in the region leveling process. A pattern is a sequence of tokens and token types (see Figure 9), describing the content of a significant number of cells within the region. The positive property of a pattern is that it generalizes over data strings within regions, which, at the end of table transformation, reflects a generalization of possible concept instance values.

The patterns are of two types, depending on the reading of the cell's content: a) left-to-right (forward), and b) right-to-left (backward) reading. For example,

the generated (forward) pattern "First_Upper Room", composed of a token type 'First_Upper' and a token 'Room', has been learnt by generalization of "Single Room" and "Double Room" cells (10 out of 15) that appear in the column 'Rooms' of Figure 2a, thus leaving out all "Extra Bed" cells. Hence, the coverage of the pattern is $\frac{2}{3}$. For the purpose of pattern construction we have implemented the DATAPROG algorithm, which is described in [33] together with a detailed pattern learning process. In case there are not enough examples (less than 20) to statistically choose the most significant patterns using the algorithm, only the most specific (having their coverage over the threshold) are chosen.

Before entering the loop (see pseudocode in Figure 11), the system checks the *uniformity* of logical units. A logical unit is uniform when it consists of logical sub-units where each sub-unit includes only regions of the same size and orientation. Logical units that are not uniform are further processed in steps 3 and 4 as described in sequel, the rest remains unchanged.

The non-uniform logical units are split into one-to-many sub-units. Their boundaries are not calculated in advance, but are set along the process of selection and leveling. In particular, the system starts at the bottom row of $LU_k$ and proceeds upwards (vertical orientation). Then it finds the best coherent region (described in A.3.) and sets the interim boundaries. These boundaries are re-set and confirmed in the leveling step. The loop repeats by finding another best region above this sub-unit until all sub-units of $LU_k$ are found and leveled.

*A.3. Select the best coherent region and set the sub-unit.* The best region is used to propagate and level neighboring regions and consequently the logical unit itself. The best region $r_{max}$ is selected according to the formula $\Phi_{r_{max}} = max_{r \in l}\phi_{r,l}$, which is calculated by the following equation:

$$\phi_{r,l} := \left( \frac{|r|}{|l|} + P(r) + \frac{1}{|r|*|P_r|} \sum_{p \in P_r} covers(p, r) \right) \tag{9}$$

where $l$ denotes a *logical unit*, $r$ denotes a *region* in the unit, and $P_r$ is the set of *significant string (forward and backward) patterns* for the region $r$ as described above. The function $covers(p, r)$ returns a coverage value of a pattern $p$ in region $r$. According to the above formula, the selected region maximizes the sum of averaged region size (1st operand of the sum), region probability (2nd operand) and averaged pattern coverage over a particular region $r$ (3rd operand).

In our example, if we consider the non-uniform logical unit $LU_3$, depicted in Figure 14a, the system selects a region in column eight ($r_8$) as the best region, according to the formula $\Phi_{r_{max}} = max(\Phi_{r_i,l_{LU_3}}) = \Phi_{r_8,l_{LU_3}}$. Note that several

regions (i.e. $r_1$, $r_4$, $r_6$, and $r_7$) reach the same result as the selected one, but the selected one reached the highest score first.

$$\Phi_{r_8,l_{LU_3}} = \frac{6}{6} + 1 + \frac{1}{6*2}(6+6)_{p_1}^{p_2} = 3 \tag{10}$$

*A.4. Level neighboring regions of the best region.* The intuition here is to use the best region as a propagator for other regions in their leveling process. First, the system selects all neighboring regions of the best region, i.e. those that appear in the same rows (left/right columns) for vertical orientation. Now, two possibilities exist: a) neighboring regions do not extend, or (b) they do extend over the boundaries of the best region. In the first case, the solution is straightforward, because the 'new' neighboring region is extended in a way to combine all common column regions (see LC in Figure 13). In the second case the best region is extended according to the oversized regions (see RC in Figure 13), and the whole leveling step repeated. More details of combining regions are given in Figure 13.

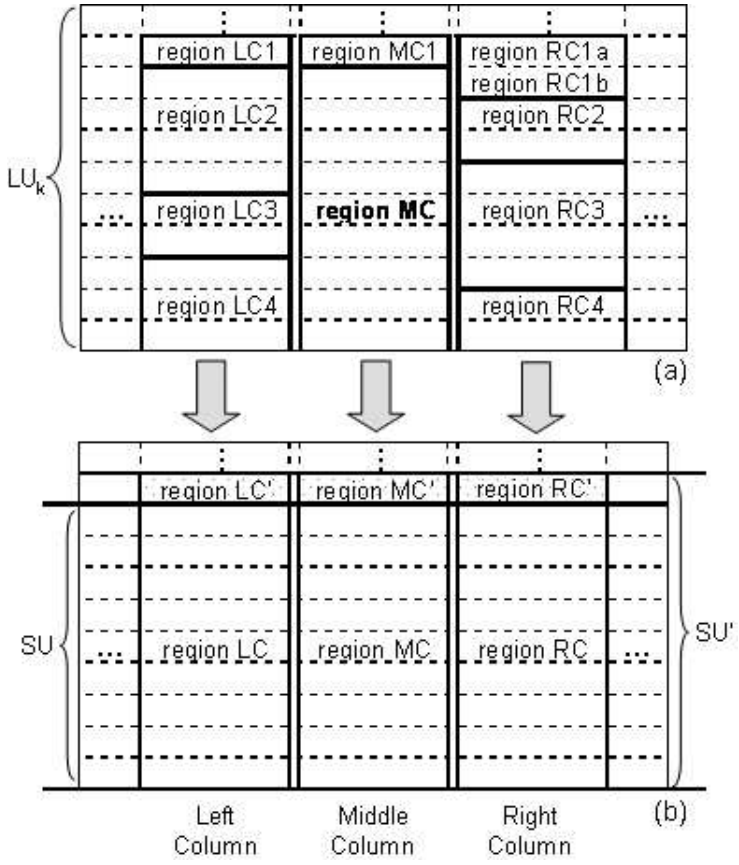



Fig. 13. An example of a non-uniform logical unit $LU_k$ before region leveling process (a) and after it (b).

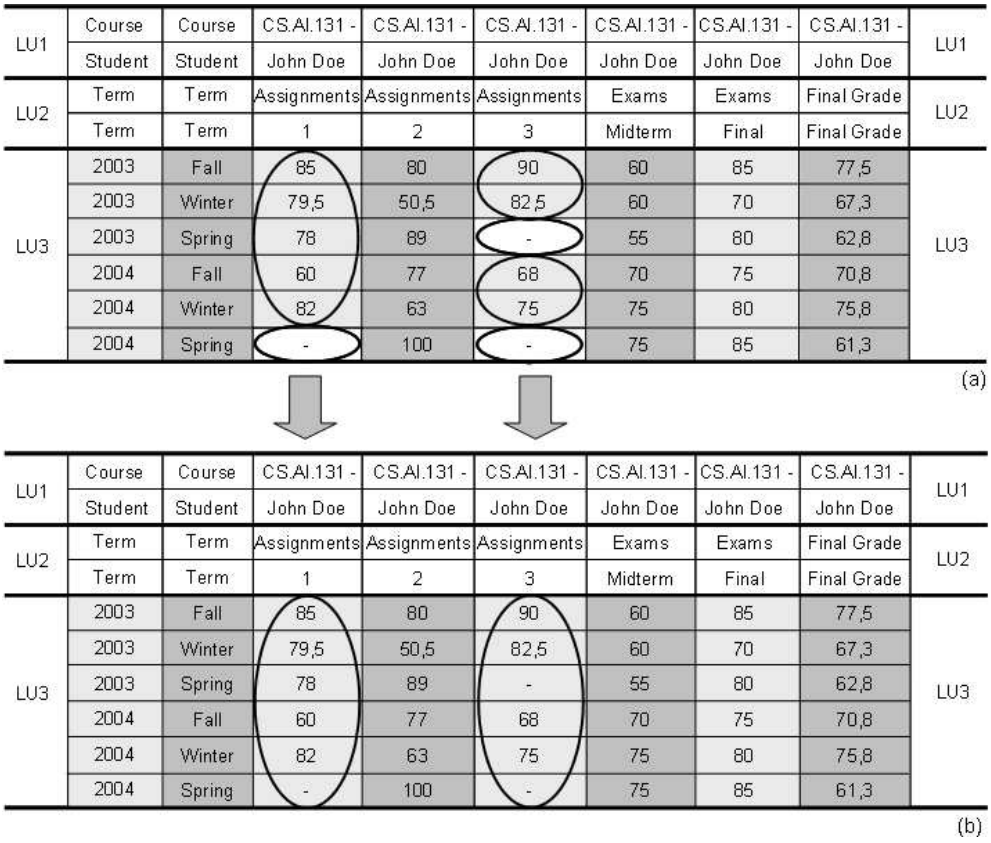
Figure 13 presents an example where the middle column (MC) region is selected as the best region. The role of the best region is to set the boundaries for the neighboring regions, where as many small regions as possible are being merged into one bigger region. Merging left column regions ($LC = LC_2 + LC_3 + LC_4$) results in a perfect match to the best (MC) region (case a). Merging right column regions ($RC' = RC_1 + RC_2 + RC_3 + RC_4$) would create a region that would extend over the boundaries of the best (MC) region, which leads to two options: (b) extend the best (MC) region to fit the RC' ($MC' = MC + MC_1$), or (c) split $RC_1$ into two regions ($RC_{1a}$ and $RC_{1b}$) and exclude $RC_{1a}$ ($RC = RC' \backslash RC_{1a}$). Note that the latter option is not possible, because region splitting whilst merging is not permitted. As shown in Figure 13, there could be two solutions after finalizing the leveling process: leveled regions (LC, MC, and RC) forming sub-unit SU for the cases (a) and (c), and leveled regions (LC', MC', and RC') forming sub-unit SU' for the case (b). Importantly, if the best (MC) region is enlarged to MC' this affects also other neighboring regions (i.e. LC must also be enlarged to $LC' = LC + LC_1$). Hence, case (b) is the only valid option and solution for the system.

The non-uniform logical unit is processed within the loop as long as the system is not able to divide it into logical sub-units, where each sub-unit includes only regions of the same size and orientation (uniformity condition). Note that string patterns, probabilities and functional types of leveled regions are also updated in every iteration. Figure 14 depicts the discovery and leveling process on our table example.

### *3.3 Functional Table Model (FTM) Building*

The key step of translating a table into a frame is building a model of the functional dimension of the table. This model is called *Functional Table Model* (FTM) and essentially arranges regions of the table in a tree, whereby the leaves of the tree are all the regions consisting exclusively of I-cells. Most importantly, in the FTM these leaves are assigned their functional role, i.e. *access* or *data*, and semantic labels as described in Section 3.4.1.

The construction of the FTM proceeds bottom up: we start with the lowest logical unit in the table and proceed with further logical units towards the top. For each logical unit in question we first determine its type. There are three possibilities: (a) the logical unit consists only of A-cells, in which case all its regions will be turned into inner nodes of the tree and thus connected to some other nodes in the tree, (b) the logical unit consists only of I-cells, in which case they will constitute leaves and will be connected to appropriate inner nodes, and (c) the logical unit consists of I-cells and A-cells, in which case we determine the logical separation between them by taking the uniformity



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| LU1 | Course | Course | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | LU1 |
| | Student | Student | John Doe | John Doe | John Doe | John Doe | John Doe | John Doe | |
| LU2 | Term | Term | Assignments | Assignments | Assignments | Exams | Exams | Final Grade | LU2 |
| | Term | Term | 1 | 2 | 3 | Midterm | Final | Final Grade | |
| LU3 | 2003 | Fall | 85 | 80 | 90 | 60 | 85 | 77,5 | LU3 |
| | 2003 | Winter | 79,5 | 50,5 | 82,5 | 60 | 70 | 67,3 | |
| | 2003 | Spring | 78 | 89 | - | 55 | 80 | 62,8 | |
| | 2004 | Fall | 60 | 77 | 68 | 70 | 75 | 70,8 | |
| | 2004 | Winter | 82 | 63 | 75 | 75 | 80 | 75,8 | |
| | 2004 | Spring | - | 100 | - | 75 | 85 | 61,3 | |

(a)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| LU1 | Course | Course | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | CS.AI.131 - | LU1 |
| | Student | Student | John Doe | John Doe | John Doe | John Doe | John Doe | John Doe | |
| LU2 | Term | Term | Assignments | Assignments | Assignments | Exams | Exams | Final Grade | LU2 |
| | Term | Term | 1 | 2 | 3 | Midterm | Final | Final Grade | |
| LU3 | 2003 | Fall | 85 | 80 | 90 | 60 | 85 | 77,5 | LU3 |
| | 2003 | Winter | 79,5 | 50,5 | 82,5 | 60 | 70 | 67,3 | |
| | 2003 | Spring | 78 | 89 | - | 55 | 80 | 62,8 | |
| | 2004 | Fall | 60 | 77 | 68 | 70 | 75 | 70,8 | |
| | 2004 | Winter | 82 | 63 | 75 | 75 | 80 | 75,8 | |
| | 2004 | Spring | - | 100 | - | 75 | 85 | 61,3 | |

(b)

Fig. 14. The table example before region discovery and leveling process (a) and after it (b).

condition into account.

In some cases a special *connection node* (see Figure 15) needs to be inserted into the tree. This occurs when we encounter a logical unit that reflects a split in the table, in particular when a lower logical unit contained only A-cells, but the upper logical unit again contains I-cells. In such cases, we check if reading orientation of the upper logical unit differs from the lower one. In particular, the system recalculates the orientation of the upper logical unit as described in Section 3.2.3, and if the orientation directions differ from each other, the upper logical unit is restructured as described in Section 3.2.4. For example, the logical unit $LU_1$ (first two rows) in Figure 13a has four regions and there is no logical unit on top of it. So, if the orientation was vertical (i.e. like in logical units $LU_2$ and $LU_3$), there would be no inner node (consisting of A-cells) to connect the I-cells to. Thus orientation has to be changed from vertical to horizontal for this logical unit.

As already mentioned above, each region in a leaf position is assigned its corresponding functional role. The role *access* is assigned to all consecutive regions (starting at the left subnodes of a subtree) together forming a unique

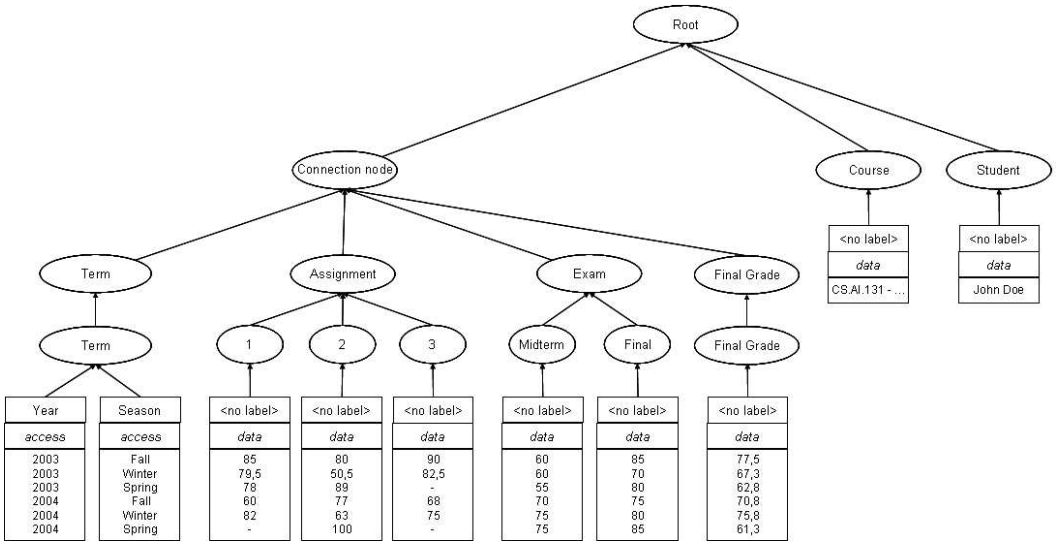



Fig. 15. An initial functional table model (FTM) of the running example, shown in Figure 13b, with square components representing I-cells and rounded components representing A-cells.

identifier or key in the database terminology. The rest of the leaf nodes in the subtree get assigned the role *data*. When all logical units have been processed, we connect the remaining unconnected nodes to a root node. Figure 15 depicts the intial FTM constructed on our example.

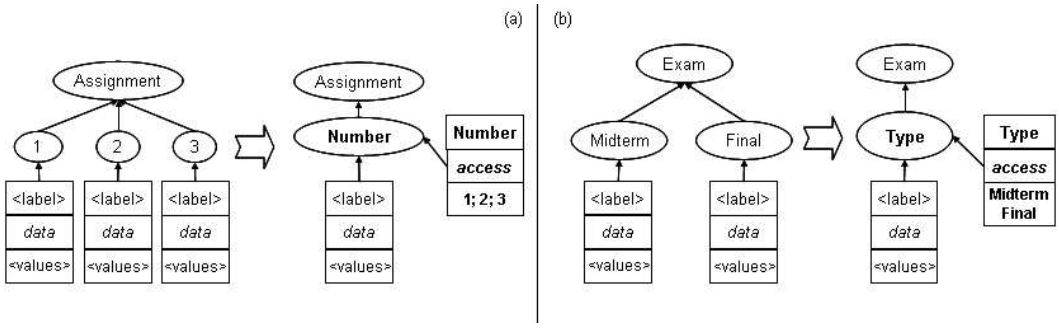



Fig. 16. Recapitulation process on the two subtrees (a) and (b) of the running example, where discovered semantic labels for the new nodes are also included.

After the initial FTM is constructed, we proceed with a *recapitulation* process, where we examine the possibilities of merging its subtrees. This process starts at the root node and proceeds downwards, and is completed after the following two steps:

(a) identification of compatible subtrees: The candidate subtrees must be multi-level (at least two levels of inner A-cell nodes) subtrees positioned at the same level in the tree, must have the same structure (number of levels and nodes) and at least one level of (content) matching A-cells. If there are any candidates that fulfill these requirements, then the system combines the subtrees by merging same position nodes. As we only require one level

of matching A-cells, there might be some nodes that do not match. For every such case, the following steps are taken: (i) find a semantic label of a new merged A-cell node (see Section 3.4.1), (ii) connect the new merged A-cell to a new leaf node, which is populated by the contents of merged nodes, and (iii) assign the functional role of the new leaf node to access.

(b) identification of compatible last-level A-cell nodes: The candidate nodes must have the same A-cell parent node, must all be of functional role data, and must share a common semantic label, which is discovered based on their contents, as described in Section 3.4.1. If there are any such nodes, then the system merges them by following the steps (i), (ii), and (iii) as described in the previous paragraph. The example of recapitulation is depicted in Figure 16.

In this way we identify and merge all matching multi-level subtrees and last-level A-cell nodes of the FTM and thus finalize the FTM construction process. The final FTM of our example is depicted in Figure 17.

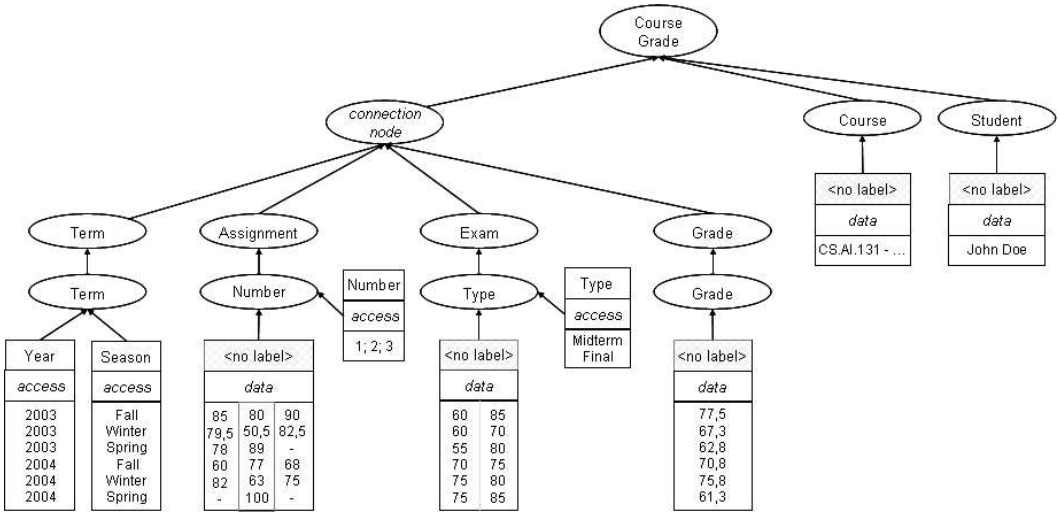



Fig. 17. A final version of the FTM after the recapitulation process.

### *3.4 Semantic Enriching of FTM*

In the last methodological step we deal with the discovery of semantics. In the first subsection the discovery of semantic labels is shown, where content analysis of cell regions is performed. The latter subsection describes the finalization of the whole process, in particular how a mapping of a semantically enriched FTM into a final frame is accomplished.

### *3.4.1 Discovery of Semantic Labels*

In order to find semantic labels for each table region (node), we resort to the WordNet lexical ontology [19] to find an appropriate hypernym covering all data strings contained in the region. Furthermore, we also make use of the GoogleSets [21] service to find synonyms for certain tokens. For example, the leaf node in Figure 14 describing weather seasons consists of data strings *Fall*, *Winter*, and *Spring*, for which WordNet suggests the hypernym *Season*. However, the data strings are not always so 'pure', therefore we start with uncleaned data strings and then stepwise remove words from them as shown in Figure 18. The figure depicts the algorithm of discovering semantic labels (i.e. hypernyms) and is performed on each region of a table.

---

**Consequently** process **steps** (**1**) to (**5**) on region's data strings
**until** a suitable region **hypernym** [2] is found or **no steps** to continue:
(B.1) punctuation removal,
(B.2) stopword removal,
(B.3) compute IDF (Inverse Document Frequency) [3] value for each word, and filter out all words with IDF value lower than the threshold value,
(B.4) select only words that appear at the end of the each cell string as these are more significant [4],
(B.5) query GoogleSets [21] with the remaining words in order to filter words which are, according to Google, not mutually similar.

---

Fig. 18. Algorithm for discovery of semantic labels.

The process of semantic label discovery prototypically covers English language only, but the extensions for other languages can be easily incorporated. Each language extension should include its formalized lexical knowledge, and should modify the existing approach according to the language properties.

### *3.4.2 Mapping Functional Table Model (FTM) into a Frame*

In order to define how to transform an FTM into a frame, we first give a formal definition of a method and a frame:

**Definition 5 (Method)** *A method is a tuple* $M := (name_M, P_M, range_M)$*, where* $name_M$ *is the name of the method,* $range_M$ *is a string describing the*

[2] After each step (B.1 thru B.5) the system consults WordNet to yield a suitable hypernym and quits when it finds one.
[3] To calculate an IDF value we assume that a cell represents a document.
[4] The intuition here is that for nominal compounds the nominal head is at the end.

*range of the method, and $P_M$ is a set of strings describing the parameters of the method.*

For example, the method $Exam(TermYear, TermSeason, ExamType) \Rightarrow NUMBER$ would be formally represented as a tuple (Exam, {TermYear, TermSeason, ExamType}, NUMBER).

**Definition 6 (Frame)** *A frame F is a pair F:=($name_F, M_F$) where $name_F$ is the name of the frame and $M_F$ is a set of methods as described above.*

When generating a frame, we create one method $m$ for every region with functional role *data* with all the regions of type *access* as parameters of this method. These parameters must either be located on the same level within the same subtree or on a parent path to the root node. Here it is crucial to find appropriate names for the method ($name_M$) and parameter identifiers $p \in P_M$, which was described in Section 3.4.1. The semantic label for each identifier is a combination of a region label (described in Figure 18) and parent A-cell node labels. The range $range_M$ of the method $m$ is set according to the syntactic token type of the region with functional role *data* for which the method was generated. For better understanding, compare the FTM tree depicted in Figure 17 and the final generated frame given in Figure 19.

```
CourseGrade [
   Course => STRING;
   Student => STRING;
      Assignment (TermYear, TermSeason, AssignmentNumber) => NUMBER;
      Exam (TermYear, TermSeason, ExamType) => NUMBER;
      Grade (TermYear, TermSeason) => NUMBER;
   ].
```

Fig. 19. Final frame of the running example, as generated by the system.

## 4 Evaluation

In order to evaluate our system we performed two types of experiments:

(a) the crawling, extraction, and filtering of the proper Web tables from a particular Web source, and
(b) comparison of a subset of automatically generated frames against manually created frames.

In the first experiment we implemented a crawler system that crawled a particular Web source [39] from a tourist domain and downloaded all pages that contained tables (HTML table element denotes a table). After collecting the pages, the crawler automatically extracted the tables out of them, and then we manually filtered out the tables that were not proper data tables (i.e. those abused for a better graphical page layout purposes). In this way we gathered 158 real Web data tables, which were fed to our transformation system in order to transform them into F-Logic frames. The system successfully constructed frames for 135 tables thus resulting in 85,44% success rate, which is certainly a very promising result. The system was able to successfully transform all tables of class 1D and 2D (see Section 2.2), but had problems with complex tables, in particular with a third, combined type of complex tables. Note that all successfully transformed frames enabled 100% correct answers to proper queries.

In order to test the similarity to humans, we compared the automatically generated frames with frames manually created by 14 different persons. Each person manually annotated only 3 different tables in order to obtain an appropriate distribution. Namely, we obtained 21 tables that were successfully transformed in the first experiment: 3 tables for each of a bit less demanding table classes 1D and 2D, and 5 for each complex subclass, most of them describing different tourist paradigms (i.e. flight/hotel/tour information). In this way, each table in our dataset got annotated by 2 different persons with an appropriate F-Logic frame. In summary, each person (14) annotated three different tables in a way that each table (21) got annotated twice, resulting in total of 42 annotations ($14 \times 3 = 21 \times 2 = 42$). This experiment also enabled to measure the inter-annotator agreement.

The input that was given to the annotators was the same as for the system - the tables only. Beside that, they were acquainted with the definition of the task as well as with the instructions, which can be found online [50]. In what follows we first present the evaluation methodology and then the actual results of the experiment.

### *4.1 Evaluation Methodology*

The approach was evaluated by the well-known information retrieval measures Precision, Recall and F-Measure in the way that for each table we evaluated the automatically generated frame with respect to the two frames manually created by two different persons along the following perspectives: *Syntactic Correctness, Strict Comparison, Soft Comparison*, and *Conceptual Comparison.*

In order to assess how similar two strings are, the introduced string comparison operator $\sigma_{SOFT}$ is

$$\sigma_{SOFT} : String \times String \rightarrow [0..1] \tag{11}$$

In particular, in our evaluation we use a string comparison operator based on a combination of a TFIDF weighting scheme with the Jaro-Wrinkler string-distance scheme [10], based on the survey by Cohen et al. [10], who showed that it gives best results for a similar task to ours.

The *Syntactic Correctness* measures how well the frame captures the syntactic structure of the table, i.e. to what extent the number of arguments matches the number of parameters as specified by the human annotator for a given method. In what follows we define three functions giving the syntactic correctness between two methods as well as a method and a frame, respectively.

$$Syntactic_{M\times M}(m_1, m_2) = \begin{cases} \frac{|P_{m_1}|}{|P_{m_2}|} & \text{if } |P_{m_2}| > 0 \\ 1 & \text{if } |P_{m_1}| = |P_{m_2}| = 0 \\ 0 & otherwise \end{cases} \tag{12}$$

$$Syntactic_{M\times F}(m, f) = Syntactic_{M\times M}(m, m'), \tag{13}$$

where $m' \in f_M$ maximizes the equation $\sigma_{CONCEPTUAL}(name_m, name_{m'}) * Syntactic_{M\times M}(m, m')$.

Note that the above measures are directed; they will be used in one direction to obtain the Precision and in the other direction (achieved by replacing function parameters) to obtain the Recall of the system.

Each of the following three semantic perspectives measures the agreement among the identifiers for the method name, the range and the parameters of involved frames. The calculation of the agreement among two methods is based on the following function:

$$\begin{aligned} X_{M\times M}(m_1, m_2) = & \frac{1}{2 + |p_{m_1}|}(\sigma_X(name_{m_1}, name_{m_2}) + \\ & + \sigma_X(range_{m_1}, range_{m_2}) + \\ & + \sum_{i=1}^{|p_{m_1}|} \sigma_X(p_{m_{1_i}}, p')) \end{aligned} \tag{14}$$

where p' is chosen according to the criteria $max_{p'\in m_{2_P}} \sigma_{CONCEPTUAL}(p_{m_{1_i}}, p')$, and X stands either for *Strict, Soft* or *Conceptual*.

Strict evaluation checks if the identifier for the method name, the range and the parameters are identical. We define the string comparison function from a strict perspective and the corresponding function *Strict* on a method and a frame, respectively:

$$\sigma_{STRICT}(s_1, s_2) = \begin{cases} 1 \text{ if } s_1 \text{ a}nd\ s_2 \text{ are identical} \\ 0 \qquad\quad \text{otherwise} \end{cases} \tag{15}$$

$$Strict_{M\times F}(m, f) = max_{m' \in f_M} Strict_{M\times M}(m, m') \tag{16}$$

The Soft Evaluation also measures how far the identifiers for the method name, the range and the parameters match using the string comparison operator ($\sigma_{SOFT}$) defined above:

$$Soft_{M\times F}(m, f) = max_{m' \in f_M} Soft_{M\times M}(m, m') \tag{17}$$

Further, we have modified a string comparison $\sigma_{SOFT}$ which returns 1 if the strings to compare are equivalent from a conceptual point of view and $\sigma_{SOFT}$ otherwise, i.e.

$$\sigma_{CONCEPTUAL}(s_1, s_2) = \begin{cases} 1 & \text{if } s_1 \text{ a}nd\ s_2 \text{ are conceptually equivalent} \\ \sigma_{SOFT}(s_1, s_2) & \text{otherwise} \end{cases} \tag{18}$$

The *Conceptual* measure was introduced to check how far the system is able to learn the frame from a conceptual point of view. In order to assess this, two of the authors compared the frames produced by the system and the ones given by the humans and determined which identifiers can be regarded as conceptually equivalent. In this line *RegionType*, *Region* and *Location* can be regarded as conceptually equivalent. Here is the formal definition of the corresponding function:

$$Conceptual_{M\times F}(m, f) = max_{m' \in f_M} Conceptual_{M\times M}(m, m') \tag{19}$$

For all the above measures two frames are compared as follows:

$$X_{F\times F}(f, f') = \frac{\sum_{m \in f_M} X_{M\times F}(m, f')}{|f_M|}, \tag{20}$$

where X stands either for *Syntactic, Strict, Soft* or *Conceptual*.

In our evaluation study we give results for Precision, Recall and F-Measure between the frame $F_S$ produced by the system and the frames $F_1, ...F_n$ (in our case $n = 2$) produced by the human annotators. In particular, we will consider the above evaluation functions *Syntactic, Strict, Soft* and *Conceptual* in order to calculate the Precision, Recall and F-Measure of the system. Thus, in the following formulas, X stands either for *Syntactic, Strict, Soft* or *Conceptual*:

$$Prec_{Avg,X}(F_S, \{F_1, ...F_n\}) = \frac{\sum_{1 \leq i \leq n} X(F_S, F_i)}{n} \tag{21}$$

Recall is defined inversely, i.e.

$$Rec_{Avg,X}(F_S, \{F_1, ...F_n\}) = \frac{\sum_{1 \leq i \leq n} X(F_i, F_S)}{n} \tag{22}$$

Obviously, according to the definitions of the measures, the following relations hold:

$$\begin{aligned} &Prec_{Strict} \leq Prec_{Soft} \leq Prec_{Conceptual} \quad \text{and} \\ &Rec_{Strict} \leq Rec_{Soft} \leq Rec_{Conceptual} \end{aligned} \tag{23}$$

The value of the Precision and Recall for the frame which maximizes these measures is

$$Prec_{max,X}(F_S, \{F_1, ...F_n\}) = max_i \ X(F_S, F_i) \tag{24}$$

Recall is defined inversely, i.e.

$$Rec_{max,X}(F_S, \{F_1, ...F_n\}) = max_i \ X(F_i, F_S) \tag{25}$$

Obviously, the following relations hold:

$$Prec_X \leq Prec_{max,X} \quad \text{and} \quad Rec_X \leq Rec_{max,X}. \tag{26}$$

The reason for calculating Precision and Recall against the frame given by both annotators, which maximizes the measures, is that some frames given by the annotators were not modeled correctly according to the intuitions of the authors. By this we avoid penalizing the system for an answer which is actually correct. By calculating $Recall_X$ and $Recall_{max,X}$ we also measure the agreement between humans. Actually, we calculated the agreement measures among the annotators in terms of Precision, Recall and F-Measure.

Finally, Recall and Precision are balanced against each other by the F-Measure given by the formula:

$$F_X(P_X, R_X) = \frac{2P_X\ R_X}{P_X + R_X} \tag{27}$$

The system was evaluated by calculating the above measures for each automatically generated frame and the corresponding frames given by the human annotators.

### *4.2 Discussion of Results*

Table 1 gives the results of the agreements among annotators (inter-annotator agreements) in terms of Precision, Recall and F-Measure as described in Section 4.1. The first and most important observation is that the frames among annotators differentiate significantly, because the highest agreement value in the table does not even reach 60% ($F_{Conceptual} = 58.58\%$). The results of the *Syntactic* comparison ($F_{Syntactic} = 45.62\%$) show a great disagreement. Regarding the naming of the methods, their range and their parameters, the results vary considerable depending on the measure in question. A significant improvement is obvious in a step from *Strict* ($F_{Strict} = 40.45\%$) to *Soft* ($F_{Soft} = 51.88\%$) feature, which clearly shows that people were to some extent using similar names for methods and parameters. A slight improvement appears also in a step towards a conceptual measure ($F_{Conceptual} = 58.58\%$), where conceptually equivalent identifiers are allowed, but the value is lower than we expected. Among possible reasons for low agreements we identified the following: (a) the annotation guidelines were not clear/detailed enough, (b) the annotators did not follow the guidelines precisely, and (c) the task itself is very hard. We agree that all these aspects contributed their share to the resulting human generated frames and hence to high disagreement among annotators, but the reasons (c) and (b) definitely stand out.

Table 1
Results of the inter-annotators agreements (in percent).

| | Syntactic | Strict | Soft | Conceptual |
|---|---|---|---|---|
| Precision | 43.60 | 42.30 | 54.09 | 59.82 |
| Recall | 49.23 | 39.22 | 50.78 | 57.38 |
| F-Measure | 46.25 | 40.70 | 52.38 | 58.58 |

When we examined the manually generated frames in details, we identified only 2 identical frame pairs ($2/21 = 9.52\%$). The tables, out of which these frames were generated, belong to the simplest (1D) table class. After we loosen

the constraints to a conceptual measure, then the number of identical frame pairs rose to 5 (5/21 = 23.81%), covering all 1D and 2 (out of 3) 2D tables. This indicates that a complicated table has a wide variety of possible solutions, some better than others.

Table 2 gives the results of the agreement among system-generated frames and frames created by annotators. Results are given in terms of Precision, Recall and F-Measure as described in Section 4.1. The first interesting observation is that the values for the *maximum* evaluation are quite higher than the ones of the *average* evaluation, which again clearly shows that there was a considerable disagreement between annotators, as shown in Table 1, and thus the task, we are considering, is far from trivial. Obviously, people do interpret the same tables (same content) in various ways.

The results of the *Syntactic* comparison are $F_{avg,Syntactic} = 49.60\%$ for the *average* evaluation and $F_{max,Syntactic} = 65.11\%$. The values show that the system is interpreting the table to a satisfactory extent from a syntactic point of view, i.e. it is determining the number of parameters correctly in most of the cases. Regarding the naming of the methods, their range and their parameters the results vary considerably depending on the measure in question. For the *average* evaluation the results are: $F_{avg,Strict} = 37.77\%$, $F_{avg,Soft} = 46.27\%$ and $F_{avg,Conceptual} = 57.22\%$. These results show that the system has problems to find the appropriate name for methods, their ranges and their parameters. However, as the conceptual evaluation shows, most of the names given by the system are from a conceptual point of view equivalent to the ones given by the human annotator. For the *maximum* evaluation we have: $F_{max,Strict} = 50.29\%$, $F_{max,Soft} = 60.05\%$ and $F_{max,Conceptual} = 74.18\%$.

Thus, we can conclude that from a conceptual point of view the system is getting an appropriate name in almost 75% of the cases and it is getting the totally identical name in more than 50% of the cases.

Table 2
Comparison results among system and annotators generated frames (in percent).

| | Average | | | | Maximum | | | |
|---|---|---|---|---|---|---|---|---|
| | Syntactic | Strict | Soft | Conceptual | Syntactic | Strict | Soft | Conceptual |
| Precision | 48.71 | 36.78 | 44.88 | 56.01 | 62.85 | 48.84 | 58.26 | 71.02 |
| Recall | 50.53 | 38.81 | 47.75 | 58.50 | 67.54 | 51.83 | 61.95 | 77.65 |
| F-Measure | 49.60 | 37.77 | 46.27 | 57.22 | 65.11 | 50.29 | 60.05 | 74.18 |

## 5 Application: Automated Query Answering

Our approach relies on F-Logic as a representation language for the mapping model (*cf.* [32], 'F' stands for "Frames") and Ontobroker as the inference engine to process F-Logic (*cf.* [12]). F-Logic combines deductive and object-oriented aspects: *"F-logic ... is a deductive, object-oriented database language which combines the declarative semantics of deductive databases with the rich data modelling capabilities supported by the object oriented data model"* (*cf.* [20]).

F-Logic allows for concise definitions with object oriented-like primitives (classes, attributes, object-oriented-style relations, instances). Furthermore, it also has Predicate Logic (PL-1) like primitives (predicates, function symbols). F-Logic allows for axioms that further constrain the interpretation of a model. Axioms may either be used to describe constraints or they may define rules, *e.g.* in order to define a relation $R$ by the composition of two other relations $S$ and $Q$.

F-Logic rules have the expressive power of Horn-Logic with negation and may be transformed into Horn-Logic rules. The semantics for a set of F-Logic statements is defined by the well-founded semantics [51]. This semantics is close to First-Order semantics. In contrast to First-Order semantics not all possible models are considered but one "most obvious" model is selected as the semantics of a set of rules and facts. F-Logic has been chosen as representation language for a number of reasons, one of the most important ones being the availability of efficient tool support for query answering. However, as shown in [37] query answering is feasible for OWL-DL ontologies [40]. This conceptual approach is (at the time of writing) being implemented in the KAON2 [31] system. Thus we envision support for OWL as representation language for future versions of TARTAR.

The most relevant syntactical elements are: ':' is used to represent "instance of" and '::' to denote the "subconcept of" relationship. Arbitrary relationships are specified by the following syntax: $concept[relation \Longrightarrow concept]$, $-\!\!\gg$ is used for the instantiation of such a relationship. # splits an identifier into namespace and local name. Please refer to [18,32] for further details.

After the table transformation process, the table data are converted into F-Logic facts which together with the generated frame constitute an object base. Note that such a representation enables querying and reasoning in a way that answers to a query (encoded in F-Logic) consist of all variable bindings such that the corresponding ground instances of the query body are true in the object base.

Here we will not describe how the conversion of table data into an object base

is performed, nor the translation of the natural language query into its F-Logic equivalent, which is dealt with in [5], but will rather give two query examples with respective results. For better intuitiveness we will present each query by the following items:

(a) introduction of a query in natural language,
(b) translation into its F-Logic equivalent, and
(c) presentation of the results returned by the inference engine.

Figure 20 depicts two possible queries. The first query (1) is about the final grade of John Doe for AI course in Spring 2003. By posting a query presented as item (1.b) the inference engine OntoBroker [12] returns a correct answer '62.8'. The second query (2) tries to find the John Doe's best exam grade. Translation into F-Logic (2.b) indicates that this example is more complicated than the first one since it demands value grouping and exploiting of Ontobroker built-in functions. Nevertheless the result '77.5', presented as item (2.c) in Figure 20, is correct if we compare it against our table example.

Automatic transformation of tables into frames, automatic conversion of table data into F-Logic object base and available querying features enhance our methodology to a great degree and make it potentially useful for a variety of applications (e.g. [35]).

(1) (a) What is the final grade for AI course of John Doe in Spring 2003?

(b) FORALL G ← EXISTS C C:CourseGrade[Course → "CS.AI.131";
Student → "John Doe";
Grade@(2003,_Spring) → G].

(c) G = "62.8"

(2) (a) What is the best exam grade of each student?

(b) FORALL P,M ← EXISTS C,Y,S,G C:CourseGrade[Grade@(Y,S) → G;
Student → P] AND maximum(P,G,M).

(c) P = "John Doe", M = "77.5"

Fig. 20. Two possible queries for the generated frame, shown in Figure 19.

## 6 Related Work

A very recent systematic overview of related work on table recognition, transformation, and inferences can be found in [60]. Several conclusions can be drawn from this survey. Firstly, only few table models have been described explicitly. Apart from the table model of Hurst which we applied in our approach [27,28] the most prominent other model is Wang's [53]. However, the model of Hurst is better suited for our purpose since it is targeted towards table recognition and transformation whereas Wang is targeted towards table generation. A table model for recognition must support two tasks: the detection of tables, and the decomposition of table regions into logical structure representation. These models tend to be more complex than generative models, since they must define and relate additional structure for recovering the components of generative models [60].

Secondly, research in table recognition, transformation, and inferences so far addressed several types of document encodings. The most work was done on plain text files, images, and HTML documents [60]. Work performed on textual tables and images was mainly oriented towards table recognition [14,26,38,54,57], row labeling [26,36,43], and cell classification [26,36,43], where the work on Web tables was extended to indexing relation detection [4,48,59] and cell/row/table merging or splitting [58]. Other approaches aim at the deep understanding of table structure, applying different techniques such as cell cohesion measures [29,55], deriving regular expressions [38], edit distance [38], graphs [26] as well as shallow parsing of the content. Knowledge engineering techniques employing certain heuristics based on the formatting cues and machine learning techniques like decision trees [38,54], Expectation Maximization [58], Hidden Markov Models [36,54], and conditional random fields [43] have been previously explored. Table extraction methods have also been applied in the context of question answering [42,52], and ontology learning [17,49].

The work done on the task of table detection was performed by [4,26,55,56]. As also evident from this work, heuristics and machine learning based approaches have been generally used to perform table detection. The documents containing both real tables and tables used for layout formatting serve as an input to a table detection system. The table detection task involves separating tables that contain logical and relational information from those that are used only for layout purposes. As the output the system returns tables classified in two categories: real table and non-real table. Usually this is a pre-step in table extraction process but it could also be combined with the extraction algorithm. In contrast, we assume that tables are already harvested, and we provide a methodology and implementation which completely instantiates a table model and additionally closes the gap to formal semantics provided by ontologies.

Chen et al. [4] present work on table detection and extraction on HTML tables. The table detection algorithm uses string, named entity and number category similarity to decide if it is a real or non-real table. Based on cell similarity the table extraction algorithm identifies whether the table is to be read row-wise or column-wise. They split the cells which span over multiple cells into individual cells. The table extraction algorithm presented in this work is simple and works only if spanning cells are used for nested labels. The paper did not provide evaluation results for their table-extraction algorithm.

The problem of merging different tables, which are about the same type of information, has been addressed in [58]. The merging task as defined in this paper considers combining different tables into one large table. They define different structures for tables based on the arrangement of the labels and use Expectation Maximization to classify the tables to one of the defined structures. The structure recognition task is similar to the classification task. However, structure recognition or merging does not solve the table extraction problem.

Table extraction by wrapper learning has been explored in [9]. Wrappers learn rules based on examples. The rules are composed of tokens made of HTML tags or the content itself. The rules tend to be specific and can be applied only to those documents whose structure is similar to the training documents. The use of tokens to compose rules makes it difficult to generalize across distributed websites. Hence wrappers learned for one website cannot be used on another website. No clear evaluation for table extraction has been described in this work.

Conditional random fields for table extraction from text tables were described in [43]. However, the system described does not perform a complete table extraction task; it only classifies the rows of the table into a type such as 'datarow', 'sectionheader' or 'superheader'. They used a set of 12 table types (classes) and achieve a precision of 93.5% for the classification task. Work presented in [26,36] also focused on the task of classifying table rows.

Tijerino et al. [49] presented a vision for a system that would be able to generate ontologies from arbitrary tables or table-equivalents. Their approach consists of a four step methodology which includes table recognition and decomposition, construction of mini ontologies, discovery of inter-ontology mappings, and merging of mini-ontologies. For the purpose of semantics discovery the approach is multifaceted, meaning they use all evidence at their disposal (i.e. Wordnet, data frames, named entities, etc.). Since the paper presents only a vision, no evaluation is provided.

We conclude that our approach is indeed novel in the sense that it is the first approach addressing the whole process of transforming a table into a

form reflecting its inherent meaning at a structural, functional and semantic level. Further, as far as we know our method is also original in being the first complete instantiation of a formal table model such as the one described by [27,28].

## 7 Conclusion and Discussion

We have presented a methodology that generates semantic frames from arbitrary domain-related Web tables. Our methodology stepwise instantiates the underlying table model which consists of *Physical*, *Structural*, *Functional* and *Semantic* components. The core steps of the methodology are (a) Cleaning and Canonicalization, (b) Structure Detection, (c) Building of the Functional Table Model (FTM), and (d) Semantic Enriching of the FTM. We demonstrated and evaluated the successful automatic generation of frames from HTML tables in two experiments. Such a methodology is indeed novel and original as most work in the field of table processing has merely considered single steps such as table detection and recognition, row labeling, cell classification, as well as merging of table units.

In a first experiment we have shown that our system successfully transformed 135 out of 158 Web tables into frames thus resulting in the 85,44% success rate. This is a very promising result. The system was able to successfully transform all tables of class 1D and 2D, but had problems with complex tables, in particular with the combined type of complex tables. The CPU time for the experiment was nearly 9 minutes.

In order to test the usability of the resulting frames generated by our system, we performed the second experiment consisting in comparing each of the frames generated by the system with two handcrafted by different humans (annotators). The inter-annotator agreement reached nearly 60% from the best (conceptual) point of view, which was a bit lower than expected. An analysis of the agreement among the system-generated frames and the annotator-generated frames showed that from the conceptual point of view the system is getting appropriate names for frames in almost 75% of the cases and the totally identical name in more than 50% of the cases, which shows that our method indeed provides qualitative results and thus that it is also applicable in practice.

Finally, we have also presented a concrete application of our method, i.e. accessing the extracted information through an inference engine allowing to issue arbitrarily complex queries, up to the expressivity of F-Logic. The benefit of translating the tables into a logic with a model-theoretic semantics thus at the end pays off. In fact, when adding the extracted facts to an appropriately

axiomatized logical theory we would be able to infer information beyond of what is actually stated in the table.

**Acknowledgments** This work has been supported by the EU IST-projects Dot.Kom (Designing adaptive infOrmation exTraction from text for KnOwledge Management, IST-2001-34038) and SEKT (Semantically Enabled Knowledge Technologies, IST-2004-506826), sponsored by the European Commission as part of the frameworks V and VI, respectively. During his stay at the AIFB, Aleksander Pivk has been supported by a Marie Curie Fellowship of the European Community program 'Host Training Sites' and by the Slovenian Ministry of Education, Science and Sport. Thanks to all our colleagues for participating in the evaluation of the system as well as to the reviewers for useful comments on the paper.

## References

[1] A. Antonacopoulos and J. Hu. *Web Document Analysis: Challenges and Opportunities*. World Scientific, 2004.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 2001(5), 2001.

[3] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002.

[4] H. Chen, S. Tsai, and J. Tsai. Mining tables from large scale HTML texts. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 166–172, 2000.

[5] P. Cimiano. Translating wh-questions into f-logic queries. In R. Bernardi and M. Moortgat, editors, *Proceedings of the CoLogNET-ElsNET Workshop on Questions and Answers*, pages 130–137, 2003.

[6] P. Cimiano, S. Handschuh, and S. Staab. Towards the self-annotating web. In *Proceedings of the 13th International Conference on World Wide Web*, pages 462–471. ACM Press, 2004.

[7] P. Cimiano, L. Schmidt-Thieme, A. Pivk, and S. Staab. Learning taxonomic relations from heterogeneous evidence. In P. Buitelaar, P. Cimiano, and B. Magnini, editors, *Ontology Learning from Text: Methods, Applications and Evaluation*, page to appear. IOS Press, 2005.

[8] E.A. Codd. A relational model for large shared databanks. *Communications of the ACM*, 13(6):377–387, 1970.

[9] W.W. Cohen, M. Hurst, and L.S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Proceedings of the 11th World Wide Web Conference*, pages 232–241, Honolulu, Hawaii, May 2002.

[10] W.W. Cohen, P. Ravikumar, and S.E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IIWeb Workshop at the IJCAI 2003 Conference*, pages 73–78, 2003.

[11] R.A. Coll, J.H. Coll, and G. Thakur. Graphs and tables: a four-factor experiment. *Communication of the ACM*, 37(4):76–86, 1994.

[12] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al., editors, *Database Semantics: Semantic Issues in Multimedia Systems*, pages 351–369. Kluwer, 1999.

[13] Document Object Model. http://www.w3.org/DOM/.

[14] S. Douglas and M. Hurst. Layout and language: List and tables in technical documents. In *Proceedings of ACL SIGPARSE Workshop on Punctuation in Computational Linguistics*, pages 19–24, 1996.

[15] S. Douglas, M. Hurst, and D. Quinn. Using natural language processing for identifying and interpreting tables in texts. In *Proceedings of the 4th Symposium on Document Analysis and Information Retrieval*, pages 535–546, 1995.

[16] J. Grossman (editor). *Chicago Manual Of Style, chapter 12 (Tables)*. University of Chicago Press, 14th edition, 1993.

[17] D.W. Embley, C. Tao, and S.W. Liddle. Automatically extracting ontologically specified data from html tables with unknown structure. In *Proceedings of the 21th International Conference on Conceptual Modeling*, pages 322–337, Tampere, Finland, October 2002.

[18] M. Erdmann. *Ontologien zur konzeptuellen Modellierung der Semantik von XML*. Books on Demand, 2001. PhD Thesis.

[19] C. Fellbaum. *WordNet, an electronic lexical database*. MIT Press, 1998.

[20] J. Frohn, R. Himmeröder, P. Kandzia, and C. Schlepphorst. How to write F–Logic programs in FLORID. Technical report, Institut für Informatik der Universität Freiburg, 1996. Version 1.0.

[21] GoogleSets. http://labs.google.com/sets.

[22] R. Hall. *Handbook of Tabular Presentation*. The Ronald Press Company, New York City, NY, 1943.

[23] S. Handschuh and S. Staab, editors. *Annotation in the Semantic Web*. IOS Press, 2003.

[24] HTML 4.01 Specification. http://www.w3.org/TR/html4/, 1999.

[25] J. Hu, R. Kashi, D. Lopresti, G. Nagy, and G. Wilfong. Why table ground-truthing is hard? In *Proceedings of the 6th International Conference on Document Analysis and Recognition*, pages 129–133, Seattle, Washington, September 2001.

[26] J. Hu, R.S. Kashi, D. Lopresti, and G.T. Wilfong. Evaluating the performance of table processing algorithms. *International Journal on Document Analysis and Recognition*, 4(3):140–153, March 2002.

[27] M. Hurst. Layout and language: Beyond simple text for information interaction - modelling the table. In *Proceedings of the 2nd International Conference on Multimodal Interfaces, Hong Kong*, 1999.

[28] M. Hurst. *The Interpretation of Tables in Texts*. PhD thesis, University of Edinburgh, 2000.

[29] M. Hurst. Layout and language: Challenges for table understanding on the web. In *Proceedings of the International Workshop on Web Document Analysis*, pages 27–30, 2001.

[30] B. Jansen, A. Spink, and J. Bateman. Searchers, the subjects they search, and sufficiency: A study of a large sample of excite searchers. In *Proceedings of the World Conference on the WWW, Internet and Intranet (WebNet-98)*, pages 913–928. AACE Press, 1998.

[31] KAON2: OWL-DL and SWRL infrastructure. http://kaon2.semanticweb.org/.

[32] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.

[33] K. Lerman, S. Minton, and C. Knoblock. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research*, 18:149–181, 2003.

[34] A. Maedche. *Ontology Learning for the Semantic Web*. Kluwer Academic Publishers, 2002.

[35] A. Maier, H.-P. Schnurr, and Y. Sure. Ontology-based information integration in the automotive industry. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proceedings of the 2nd Int. Semantic Web Conference (ISWC 2003)*, volume 2870 of *LNCS*, pages 897–912, Sanibel Island, FL, USA, 2003. Springer.

[36] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the ICML 2000*, pages 591–598, 2000.

[37] B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. In *Proceeding of the 3rd International Semantic Web Conference (ISWC 2004)*, LNCS, Volume 3298, pages 549–563, Hiroshima, Japan, 2004.

[38] H. T. Ng, C. Y. Kim, and J. L. T. Koo. Learning to recognize tables in free text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 443–450, Maryland, USA, 1999.

[39] Lonely Planet Online. http://www.lonelyplanet.com/.

[40] OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref.

[41] CyberNeko HTML Parser. http://www.apache.org/∼andyc/neko/doc/html/.

[42] D. Pinto, W. Croft, M. Branstein, R. Coleman, M. King, W. Li, and X. Wei. Quasm: A system for question answering using semi-structured data. In *Proceedings of the Joint Conference on Digital Libraries (JCDL) 2002*, pages 46–55, 2002.

[43] D. Pinto, A. McCallum, X. Wei, and W.B. Croft. Table extraction using conditional random fields. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, pages 235–242. ACM Press, 2003.

[44] A. Pivk, P. Cimiano, and Y. Sure. From tables to frames. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, LNCS, Volume 3298, pages 166–181, Hiroshima, Japan, 2004. Springer.

[45] A. Pivk and M. Gams. Domain-dependant information gathering agent. *Expert Systems with Applications*, 23:207–218, 2002.

[46] A. Pivk and M. Gams. A semi-universal e-commerce agent: domain-dependant information gathering. In M. Piattini, J. Filipe, and J. Braz, editors, *Enterprise Information Systems IV*, pages 260–267. Kluwer Academic Publishers, 2003.

[47] System TARTAR. http://dis.ijs.si/sandi/work/tartar/.

[48] A. Tengli, Y. Yang, and N. Li Ma. Learning table extraction from examples. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages –, Geneva, Switzerland, May 23–27 2004.

[49] Y.A. Tijerino, D.W. Embley, D.W. Lonsdale, and G. Nagy. Ontology generation from tables. In *Proceedings of 4th International Conference on Web Information Systems Engineering (WISE'03)*, pages 242–249, Rome, Italy, December 2003.

[50] Tables to Frames Experiment Instructions. http://www.aifb.uni-karlsruhe.de/wbs/pci/fromtables2frames.ps.

[51] A. van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.

[52] H. L. Wang, S. H. Wu, I. C. Wang, C. L. Sung, W. L. Hsu, and W. K. Shih. Semantic Search on Internet Tabular Information Extraction for Answering Queries. In *Proceedings of the 9th International Conference on Information and Knowledge Management*, pages 243–249, Washington DC, 2000.

[53] X. Wang. *Tabular Abstraction, Editing and Formatting.* PhD thesis, U. of Waterloo, 1996.

[54] Y. Wang, R. Haralick, and I. Phillips. Zone content classification and its performance evaluation. In *Proceedings of the 6th International Conference on Document Analysis and Recognition (ICDAR01)*, pages 540–544, Seattle, Washington, September 2001.

[55] Y. Wang and J. Hu. Detecting tables in HTML documents. In *Proceedings of the 5th International Workshop on Document Analysis Systems*, LNCS 2423, pages 249–260. Springer-Verlag, 2002.

[56] Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th International Conference on World Wide Web*, pages 242–250. ACM Press, 2002.

[57] Y. Wang, I.T. Phillips, R.M.Robert, and M. Haralick. Table structure understanding and its performance evaluation. *Pattern Recognition*, 37(7):1479–1497, July 2004.

[58] M. Yoshida, K. Torisawa, and J. Tsujii. A method to integrate tables of the world wide web. In *Proceedings of the International Workshop on Web Document Analysis (WDA 2001)*, pages 31–34, 2001.

[59] M. Yoshida, K. Torisawa, and J. Tsujii. Extracting attributes and their values from web pages. In A. Antonacopoulos and J. Hu, editors, *Web Document Analysis: Challenges and Opportunities*, Series in Machine Perception and Artificial Intelligence, pages 179–200. World Scientific, 2003.

[60] 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*, 7(1):1–16, March 2004.