Automatically Locating, Extracting and Analyzing Tabular Data

William Kornfeld
kornfeld@kornfeld.com

John Wattecamps[1]
Gaithersburg, Maryland

Abstract Information retrieval of ASCII documents generally refers to retrieval based on linear patterns found in the source documents. We have developed a method for recognizing and extracting *tabular data*, in this case financial tables. Tables are extracted using a version of the LR(k) parsing algorithm adapted for this purpose. Because of sloppiness in the construction of tables, somewhat less than 100% of the tables can be retrieved automatically; a method has been found to integrate the parsing algorithm into a module analogous to a programming language debugger that allows operators to quickly correct defects in the source document. This paper describes an application in commercial use.

1 Introduction

In every nation with a stock exchange, each company with publicly traded stock must make periodic reports to the public on its financial state. The form, composition and meaning of the *financial statements* contained in these documents are defined by centuries of tradition and an ongoing standards effort in the accounting profession. [1] Traditionally, these documents were produced by financial printers who were adept at using fonts, color and layout to make the financial statements easy to read. (They are typically very complex in their internal structure.)

In 1996, the U.S. Securities Exchange Commission's switched to an electronic means of filing these reports known as "EDGAR." [2] The EDGAR specification, in an attempt to be compatible with the least common denominator of hardware/software configurations, is ASCII-based. In fact, one requirement for the specification is that it facilitate printing on high-speed 132 column line printers. Paradoxically, while increasing the distributability of these documents, EDGAR has hampered their readability. At the present time some 24 gigabytes of EDGAR filings are made available per year. The time distribution of these filings throughout the year is highly nonuniform, with almost all appearing in just a few weeks of the year; hence a requirement for nearly-automatic extraction to achieve acceptable latency times in obtaining extraction products.

We have taken these ASCII documents and have developed a method to semi-automatically infer the structure of the financial statements by a parsing method that yields the implicit hierarchical structure. With them we are able to automatically generate several types of derivative data streams including nicely-printed financial statements and *templates*.

Templates may be most easily thought of as *normalized financial statements*, statements whose line items are pre-specified so that financial statements derived from the same template type are the same except for the numerical values. Thus, the financial statements of any pair of (for example) insurance companies are directly comparable to one another because they contain precisely the same set of line items. The mapping between line items in the original financial statements and line items in the template is, in general, *many-to-one*.

2 Structure of a Parse Tree

The key to the automated analysis we do is the production of a data structure explicitly representing the hierarchical structure inherent in a financial statement. While there may be many cues to the existence of hierarchical components in a financial statement, the two that we use are: *arithmetic relationships* and *indentation*. Because the process of determining the hierarchical structure is quite similar to the process used to parse computer languages, we will refer to the resulting hierarchical data structure as a *parse tree*.

A node of a parse tree (a *"unit"*) can be either a *primitive unit* (terminal node of the parse tree) or a *compound* (nonterminal node of the parse tree). Compounds are generally *sum compounds* consisting of a series of units followed by a line in the statement containing the sum of those units, although compounds defined by other arithmetic relations occur. We have seen actual statements with hierarchical depth as high as six. Complex statements in the EDGAR stream are often difficult or impossible for financial experts to understand because they lack the typographic aids-to-comprehension generally found in printed statements.

3 Method of Parsing

The parsing of a financial statement to determine its hierarchical structure is accomplished with a single-stack non-backtrack parser analogous to an LR(k) parser. [3] Each *token* (in the stream of tokens input to the stack-based parser) consists of a *label* and numbers, one for each column in the statement. Each token essentially corresponds to a horizontal line within the text of the financial statement. Often, however, because labels are longer than the available horizontal space, they are broken up across multiple lines; an algorithm determines when this is the case and amalgamates these into a single token data structure. An earlier (non-linear) version of this parser is reported in [4].

As with an LR parser, the basic data structures consist of an *input buffer* initialized with the tokens produced by the tokenizing phase in the order contained in the source document and a *stack* containing units (either primitive or compound) which correspond to fragments of parse tree as they are constructed. The outermost loop of the

[1] at Disclosure, Incorporated when this work was performed.

control module scans the tokens in the input buffer from top to bottom without backtracking. At each step the algorithm executes a *shift* and/or a *reduce* operation.

A shift operation transfers a token (corresponding, then, to a line of the financial statement) to the top of the stack. A reduce operation takes some number of units at the top of the stack and reduces to a compound. A sum-type compound is reduced if the unit at the top of the stack is the *vector sum* of two or more units directly beneath it. The algorithm terminates successfully iff there is only one unit at the top of the stack (corresponding to the desired parse tree) *and* the label of that unit is consistent with the kind of statement being parsed. If the parser reaches the final token in the input buffer and has not terminated successfully, then it terminates with failure.

4 Templates

Although there are a number of information products that can be derived from parse trees, the most important ones to date are the templates—financial statements normalized to a particular industry that support useful comparison of the financial state of two different companies. At present, we maintain thirteen different template types for different types of companies.

Templates, as a target information tool, have been in use for years. In prior years they were largely done by hand by accountants and were highly labor-intensive. Earlier experiments with automated template filling involving "flat" data structures—essentially equivalent to the token-level input to the parser—were failures. Interestingly, the addition of the hierarchical description has made the problem not only tractable, but relatively straightforward.

We developed a simple match language for classifying units (both primitive and compound). The match language is applied to a string that has been tokenized and normalized for case.

Match string in the match language consists of disjunctions of token sequences. The token sequences are matched token for token against the tokenized, normalized label. There is one wild card token ('*') that can match *zero or more* tokens in the label. One such match string for identifying "Long Term Investments" (one line item in a template) is:

[<land, *, resale, *>| <investments, *>| investment, *>
| <securities, *>| <total, investments, *>
| <long, term, investments, *>| <long, term, investment, *>
|<other, equity, *>| <other, securities, *>
| <other, investments, *>| <long, term, *, securities, *>
|<notes, receivable, *>| <joint, venture, *>
| <joint, ventures, *>| <net, investment, *>
| <net, investments, *>| <marketable, securities, *>]

The language of financial statement labels is sufficiently stylized that this style of matching works virtually all the time to correctly categorize line items. The match language is embedded in a very simple algorithm that classifies those units at top level and then proceeds to classify those units within the body using a set of choices determined by the classification of the containing compound. Many times precisely the same label can appear in different places within the same statement, and yet contribute to different fields of the template; hence the necessity to conditionalize based on the hierarchical decomposition.

Using this simple approach applied to the parse trees we were able to very rapidly achieve accuracies similar to those obtained by the fully manual process.

5 The Need for Manual Intervention

For approximately 85% of the financial statements, the parser is able to automatically construct the parse tree and from this derive the template and other data structures. Various formatting irregularities (typos, arithmetic errors, etc.) can cause the parser to fail. Interestingly, about two percent of the documents appearing on EDGAR contain actual arithmetic errors which are detected by us.

The easiest way to understand the manual intervention process is by analogy to *source-linked debuggers* found in modern program development systems. In such a system, a program is stepped while the user observes the working state of the program's environment until an unexpected state variable is observed. By noting the behavior of the program temporally proximal (and, presumably, causally related) to the anomalous state variable, the bug can be located and cured. Recall that the outermost loop of the parser is a scan of the lines of the table from top to bottom with no backtracking. As the scanning is performed, compound units (understandable to the user as hierarchical groupings) are produced in a bottom-up manner and stored on the stack. By displaying the contents of the stack in a human-readable way and connecting it with the source code by color overlays, the editor can quickly discover the problem, correct it, and proceed to a correct parse.

6 Final Remarks

The EDGAR project has been widely criticized as a step backwards in financial reporting. We have discovered, however, that electronic reporting, even if containing less explicit structure than alternatives, allow the possibility for enhancement of this data by automatic means. Ironically, the SEC has responded to the criticism by proposing future reporting using Adobe's portable document format ("PDF") which allows the richness of printed paper in electronic format. While PDF would perhaps accomplish the goal the SEC had originally embarked upon EDGAR to solve, it would make the kinds of automated data enhancement we can now accomplish with EDGAR far more difficult to perform, and probably lower the ultimate utility of the EDGAR database.

7 References

[1] Financial Accounting Standards Board (professional organization), http://www.rutgers.edu/Accounting/raw/fasb/

[2] Securities Exchange Commission, *Edgar Filer Manual*, U.S. Federal Register [62 FR 8877], February 27, 1997.

[3] Aho, A. V., R. Sethi and J. D. Ullman, *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1986.

[4] Ferguson, D., *Parsing Financial Statements Efficiently and Accurately Using C & Prolog*, Practical Applications of Prolog Conference '97, London, UK.