Automatic Ad Hoc Data Processing Using PADS

David Burke and Peter White {DAVIDB,PETER}@GALOIS.COM
Galois Connections, Inc, Portland, OR

Kathleen Fisher KFISHER@RESEARCH.ATT.COM
AT&T Labs

David Walker and Kenny Q. Zhu {DPW,KZHU}@CS.PRINCETON.EDU
Princeton University, Princeton, NJ

1. Introduction

Transactional data streams, such as sequences of stock-market buy/sell orders, credit-card purchase records, web server entries, and electronic fund transfer orders, can be mined very profitably. As an example, researchers at AT&T have built customer profiles from streams of call-detail records to significant financial effect (Cortes & Pregibon, 1998; Cortes & Pregibon, 1999; Cortes et al., 2000).

Often such streams are high-volume: AT&T's call-detail stream contains roughly 300 million calls per day requiring approximately 7GBs of storage space. Typically, such stream data arrives "as is" in *ad hoc* formats with poor documentation. In addition, the data frequently contains errors. The appropriate response to such errors is application-specific. Some applications can simply discard unexpected or erroneous values and continue processing. For other applications, however, errors in the data can be the most interesting part of the data.

Understanding a new data stream and producing a suitable parser are crucial first steps in any use of stream data. Unfortunately, writing parsers for such data is a difficult task, both tedious and error-prone. It is complicated by lack of documentation, convoluted encodings designed to save space, the need to handle errors robustly, and the need to produce efficient code to cope with the scale of the stream. Often, the hard-won understanding of the data ends up embedded in parsing code, making long-term maintenance difficult for the original writer and sharing the knowledge with others nearly impossible.

The goal of this project is to provide a generic framework that includes languages and tools to seamlessly automate data stream analysis. Given some samples of the data stream, our prototype system produces an intermediate representation of the structure of the data through structure discovery and refinement, and translates that representation into a declarative data-description language, PADS/C. PADS/C is expressive enough to describe a variety of data feeds including ASCII, binary, EBCDIC, Cobol, and mixed data formats. From PADS/C, a suite of tools can generated with functions for parsing, manipulating, and summarizing the data. All these can be done with a "push of a button."

2. PADS Language

Intuitively, a PADS/C description specifies complete information about the physical structure and semantic constraints for the associated data stream. Most type declarations in PADS/C are analogous to type declarations in C. PADS/C has built-in base types such as ASCII string terminated by space `Pa_string(:' ':)`, 32-bit unsigned integer `Puint32`, date `Pdate` and time `Ptime`. It also has **`Pstruct`**s, **`Punion`**s, and **`Parray`**s to describe record-like structures, as well as a **`Pswitch`** to represent

As an example, consider the common log format for Web server logs. A typical record looks like the following:

```
207.136.97.49 - - [15/Oct/2006:18:46:51 -0700]
"GET /tk/p.txt HTTP/1.0" 200 30
```

This record contains the IP address or hostname of the requester, the owner of the TCP session or dash, the login of the requester or dash, the date and time of the request, the actually request, a response code and the number of bytes transmitted. One possible PADS/C description of the actually request could be

```
Pstruct http_request_t {
  '\"'; http_method_t    meth;
  ' ';  Pa_string(:' ':) req_uri;
  ' ';  http_v_t version :
        checkVersion(version, meth);
  '\"';
};
```

The auxiliary types `http_method_t` and `http_v_t` describes the various HTTP methods such as GET/PUT/LINK, and the version formats, respectively. The `version` field is subject to a constraint predicate

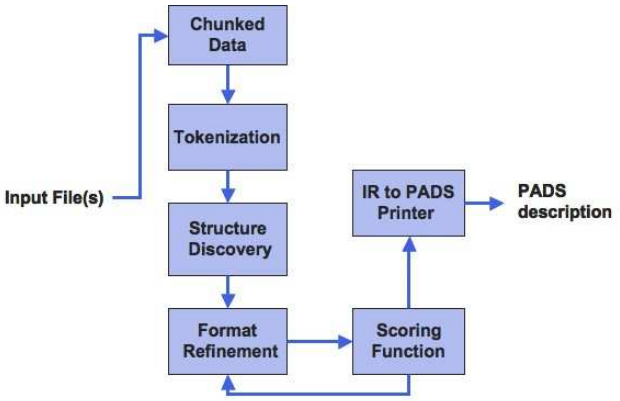

Figure 1. Architecture of the format inference engine

`checkVersion` to ensure that obsolete methods such as `LINK` and `UNLINK` are only used with version HTTP/1.0.

A number of tools can be generated in the form of a C library from the PADS/C description. Some examples are: a parser that allows the user to parse the data stream into structured or semi-structure data such as comma-separated formats or XMLs; an accumulator tool that presents the user with statistics of input data such as the frequency of each type appearing in the data, the accumulated error rate of each type declaration in the PADS/C description; or a filter program that allows the users to specify some filtering condition and process the data to return on the information of interest to the users.

Now we would like to automatically generate PADS/C descriptions base on sample data with minimum user intervention. To this end, we propose an architecture to *infer* or "learn" formats from ad hoc data samples and convert the formats to PADS/C.

3. Architecture

Figure 1 gives the big picture of the format inferencing architecture. The input data, or the "training set", is first "chunked" into records where each record is often a piece of recurrent data such as a line, a paragraph or even a file (if the input consists of multiple files). Each record is then broken down into a series of tokens where each token can be a number, a date, a time using a regular expression based tokenization scheme. In the structure discovery phase, frequency distribution of each type of tokens is examined, and a rough structure is discovered in a recursive manner. This rough structure is represented by an intermediate representation or IR, which has similar expressive power as the PADS language.

The format refinement step refines the IR by applying a number of rewrite rules sequentially. This is equivalent to a local search procedure aimming at improving the "quality" of the inferred formmat. The quality is measured by a scoring function which is based on the information theory. The objective of this function is to minimize the cost of transmitting the inferred format plus the training set. The refinement step loops until there is no possible improvement in the score. Below is an example IR for the web log data above.

```
Pstruct(Id = BTy_29 6)
        [IP] (Id = BTy_0 6);
        [StringConst] " - - [" (Id = BTy_1 6);
        [Date] (Id = BTy_7 6);
        [StringConst] ":" (Id = BTy_8 6);
        [Time] (Id = BTy_9 6);
        [StringConst] "] "" (Id = BTy_10 6);
        [Enum] {[StringConst] "GET",
[StringConst] "POST"}
        [StringConst] " " (Id = BTy_15 6);
        [Path] (Id = BTy_16 6);
        [StringConst] " HTTP/" (Id = BTy_17 6);
        [Pfloat] (Id = BTy_20 6);
        [StringConst] "" " (Id = BTy_23 6);
        [IntConst] [200] (Id = BTy_26 6);
        [StringConst] " " (Id = BTy_27 6);
        [Pint] (Id = BTy_28 6);
End Pstruct
```

The IR is a tree structure where each if its node is a type. We include auxilary information such as labels of the nodes and the number of records covered by a node in the IR to assist the rewriting. Finally, the refined IR is translated by a "pretty printer" into a PADS program like this:

```
#include "vanilla.p"
Penum Enum_14
        GET14 Pfrom("GET"),
        POST14 Pfrom("POST")
;
Precord Pstruct Struct_29
        PPip var_0;
        " - - [";
        PPdate var_7;
        ':';
        PPtime var_9;
        "] ";
        Enum_14 var_14;
        ' ';
        PPpath var_16;
        " HTTP/";
        Pfloat64 var_20;
        "" ";
        Puint8 var_26 : var_26 == 200;
        ' ';
        Pint64 var_28;
;
Psource Parray entries_t
        Struct_29[];
;
```

References

Cortes, C., Fisher, K., Pregibon, D., Rogers, A., & Smith, F. (2000). Hancock: A language for extracting signatures from data streams. *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining* (pp. 9–17).

Cortes, C., & Pregibon, D. (1998). Giga mining. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*.

Cortes, C., & Pregibon, D. (1999). Information mining platform: An infrastructure for KDD rapid deployment. *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*.