Towards 1-click Tool Generation with PADS[1]

David Burke and Peter White

Galois Connections

Kathleen Fisher

AT&T Labs

David Walker and Kenny Q. Zhu

Princeton University

Introduction

An *ad hoc data source* is any semistructured data source for which useful data analysis and transformation tools have not yet been defined. XML, HTML and CSV are *not* ad hoc data sources as there are numerous programming libraries, query languages, manuals and other resources dedicated to helping analysts manipulate data in these formats. However, ad hoc data does arise often in many fields ranging from computational biology to finance to networking.

The goal of the PADS project (PADS Project, 2007) is to improve the productivity of data analysts who need to deal with new and evolving ad hoc data sources on a daily basis. Our central technology is a domain-specific language in which programmers can specify the structure and expected properties of ad hoc data sources, whether they be ASCII, binary, Cobol or a mixture of formats. These specifications, which resemble extended type declarations from conventional programming languages, are compiled into a suite of programming libraries, such as parsers and printers, and end-to-end data processing tools including a query engine and several format translators.

Unfortunately, it often takes substantial time, energy and expertise to write a PADS description for a new ad hoc data source – days or even weeks for complex sources that have little or no documentation, a common scenario in the world of ad hoc data. Consequently, we have begun to study techniques for automatic inference of PADS descriptions from example data, focusing for now on ASCII data sources. The rest of this paper describes our efforts to do so.

[1] This material is based upon work supported by DARPA under grant FA8750-07-C-0014 and the NSF under grants 0612147 and 0615062. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or the NSF.

The PADS Language

Before outlining the main ideas in description inference, we explain the structure of the PADS specification language in greater detail. As mentioned in the introduction, PADS specifications are related to conventional type declarations.[2] For example, PADS has a number built-in base types for atomic items ranging from integers of various kinds (`Puint32` for a 32-bit unsigned integer; `Pint64` for a signed 64-bit integer), strings with different termination conditions (`Pstring(' ')` terminates on seeing a space character) and more complex items such as dates (`Pdate`) and times (`Ptime`). To describe complex formats, PADS contains a number of type constructors including `Pstruct` and `Parray` to describe sequences of items in a file, and `Punion`, `Pswitch`, `Penum` to describe various sorts of choices in a file. Programmers can also attach arbitrary constraints to descriptions.

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]
"GET /tk/p.txt HTTP/1.0" 200 30
```

This record contains the IP address of the requester, a pair of dashes, the date and time of the request, the request string itself, a response code and the number of bytes transmitted. A fragment of the PADS description for such a data source follows.

```
Pstruct  webRecord {
  Pip        ip;   " - - [";
  Pdate(':') date; ":";
  Ptime(']') time; "]";
  httpMeth   meth; " ";
  Puint8     code; " ";
  Puint8     size; " "; };
Parray  webLog { webRecord[] };
```

This description gives a definition for the `webLog` type, which is a sequence (a `Parray`) of arbitrarily many `webRecord`s. The definition for the `webRecord` type includes an ip address, a date, a time and some other items. Notice there are two sorts of fields in the Web record structure: named fields (such as the field named `ip` with type `Pip`) and unnamed fields (such as `" - - ["`). The for-

[2] Though there are two versions of PADS, one for C and one for O'CAML, we restrict ourselves to discussion of the C version, which should be more familiar to most readers.

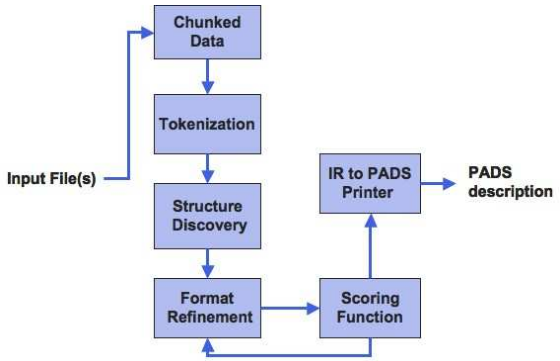

Figure 1. Architecture of the format inference engine

mer hold data that programmers or generated tools may access; the latter serve as syntactic separators read by generated parsers and printed by generated printers. The type `httpMeth` is another user-defined type not shown here.

The Inference Engine

Figure 1 gives an overview of our format inference architecture. The input data, or "training set," is first "chunked" into records where each record is a piece of recurrent data such as a line, a paragraph, or a file (if the input consists of multiple files). The user specifies the unit of repetition when invoking our learning tool. Each record is then broken down into a series of tokens where each token can be a punctuation symbol, a number, a date, a time, or a number of other basic types. Our learning system has a basic tokenization scheme skewed toward systems data, but users may specify a different scheme for their own domain through a configuration file. For example, computational biologists may want to specify new base types for DNA strings or other common recurring patterns.

In the structure discovery phase, we use a top-down, divide-and-conquer scheme inspired in part by the work of Arasu on information extraction from web pages (Arasu & Garcia-Molina, 2003). This scheme calculates frequency distributions for tokens within records, and using this information, chooses a simple type constructor such as a `Pstruct`, `Punion`, or `Parray` to describe the top-level structure of the record. When a type constructor has been chosen, the data is partitioned accordingly and the algorithm recursively analyzes subparts. This rough structure is represented in an intermediate representation (IR) that has similar expressive power to the PADS language.

The format refinement phase analyzes the IR produced by structure discovery and repeatedly applies rewrite rules. There are two sorts of rewriting rules: value-independent rules and value-dependent rules. The value-independent rules examine the inferred description structure to find ways to merge or rearrange components to improve the description. Value-dependent rules analyze both the inferred description and the underlying training data looking for fields with little or no variation in order to introduce constants and enumerations. The value-dependent rules also infer inter-field dependency information. At any given point during the refinement process, many rewriting rules may apply; our algorithm repeatedly chooses the one that optimizes an information-theoretic scoring function we have defined. In effect, this refinement phase is equivalent to a greedy, local search procedure aimed at improving the quality of the inferred format. Below is a fragment of the IR learned from web log data similar to the data illustrated above. In this case, the learning system was quite successful; the only mistake it made was over-specializing the response code to the specific integer constant 200. It did so because the records in the training data contained no variation in this field.

```
Pstruct
  [IP];              [StringConst] " - - [";
  [Date];            [StringConst] ":";
  [Time];            [StringConst] "]";
  ...
  [IntConst] [200]; [StringConst] " "
  [Pint];
End Pstruct
```

Finally, after no more refinement is possible, a pretty printer translates the IR into a working PADS specification, which can be used to generate a suite of useful tools. Here is the final result for our running example.

```
Pstruct Struct_29 {
  Pip        var_0; " - - [";
  Pdate(':') var_7; ':';
  Ptime(']') var_9; "] ";
  ...
  Puint8 var_26 : var_26 == 200; ' ';
  Pint64 var_28;
};
Parray entries_t { Struct_29[]; };
```

Conclusions

The primary conclusion of our preliminary study is that it is possible to infer useful PADS descriptions from a sufficiently large corpus of training data. Indeed, with just a push of a button, we can now automatically generate tools that convert raw data into XML and generate statistical reports. Where it once took days or weeks to access and transform ad hoc data, it now takes seconds.

References

Arasu, A., & Garcia-Molina, H. (2003). Extracting structured data from web pages. *SIGMOD 2003*.

PADS Project (2007). `http://www.padsproj.org/`.