From Dirt to Shovels: Inferring PADS descriptions from ASCII Data
Kathleen Fisher
David Walker
Peter White
Kenny Zhu
July 2007

Data, Data, everywhere!
Incredible amounts of data stored in well-behaved formats:

<table>
<thead>
<tr>
<th>Databases:</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Database" /></td>
<td>- Schema</td>
</tr>
<tr>
<td></td>
<td>- Browsers</td>
</tr>
<tr>
<td></td>
<td>- Query Languages</td>
</tr>
<tr>
<td></td>
<td>- Standards</td>
</tr>
<tr>
<td></td>
<td>- Libraries</td>
</tr>
<tr>
<td></td>
<td>- Books, documentation</td>
</tr>
<tr>
<td></td>
<td>- Training courses</td>
</tr>
<tr>
<td></td>
<td>- Conversion tools</td>
</tr>
<tr>
<td></td>
<td>- Vendor support</td>
</tr>
<tr>
<td></td>
<td>- Consultants...</td>
</tr>
</tbody>
</table>

We’re not always so lucky!
Vast amounts of chaotic ad hoc data:

![Image of chaotic data]

Tools
- Perl
- Awk
- C
- ...

Government stats

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Stations</td>
<td>Includes statistics for train stations in various cities</td>
</tr>
<tr>
<td>Web logs</td>
<td>Includes statistics for web logs from various sites</td>
</tr>
</tbody>
</table>

Train Stations

<table>
<thead>
<tr>
<th>City</th>
<th>Number of Trains</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>12345</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>67890</td>
</tr>
<tr>
<td>Chicago</td>
<td>10111</td>
</tr>
</tbody>
</table>

Web logs

<table>
<thead>
<tr>
<th>Site Name</th>
<th>Number of Visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>99999</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>88888</td>
</tr>
<tr>
<td>Chicago</td>
<td>77777</td>
</tr>
</tbody>
</table>


And many others...

- Gene ontology data
- Cosmology data
- Financial trading data
- Telecom billing data
- Router config files
- System logs
- Call detail data
- Netflow packets
- DNS packets
- Java JAR files
- Jazz recording info
- ...

Learning: Goals & Approach

Problem: Producing useful tools for ad hoc data takes a lot of time.
Solution: A learning system to generate data descriptions and tools automatically.

PADS Reminder

Inferred data formats are described using a specialized language of types
- Provides rich base type library; many specialized for systems data.
  - `Int8`, `Point8`, ...
  - `Patring:|'\(':` // 123.44
  - `Patring_FW:|':` // hello!
  - `Pdate, Ptime, Pip`, ...
- Provides type constructors to describe data source structure:
  - `sequence`: `Patstruct, Parrey`
  - `choices`: `Punion, Penum, Pawitch`
  - `constraints`: allow arbitrary predicates to describe expected properties.

PADS compiler generates stand-alone tools including xml-conversion, Xquery support & statistical analysis directly from data descriptions.

Go to demo

Format inference overview

- Convert raw input into sequence of “chunks.”
  - 123, 24
  - 731, Harry
  - 574, Hermione
  - 9378, 56
  - 12, Hogwarts
  - 112, Ron

- Supported divisions:
  - Various forms of “newline”
  - File boundaries
- Also possible: user-defined “paragraphs”
Tokenization

- Tokens expressed as regular expressions.
- Basic tokens
  - Integer, white space, punctuation, strings
- Distinctive tokens
  - IP addresses, dates, times, MAC addresses, ...

Clustering

Group clusters with similar frequency distributions

- Rare queries by metric that rewards high, unique
- Ignore distributions. 
- Close clusters with higher
- Sorted by height.

Find subcontexts

Tokens in selected cluster:

| Quote Int Comma White Int Quote |
| Quote Int Comma White Int Quote |
| Quote Int Comma White Int Quote |
| Quote Int Comma White Int Quote |
| Quote Int Comma White Int Quote |
| Quote Int Comma White Int Quote |

Then Recurse...

| Int |
| Int |
| Int |
| Int |
| Int |
| Int |
| Int |

In our example, all the tokens appear in the same order in all chunks, so the union is degenerate.
Inferred type

```
"123, 45
"/a/b/c/d
"/1, \3
"\n, %
"\n, 
```

```
String * XML = VALID * (DATA + XML) = UNIL
```

Finding arrays

```
libxml2 | glib | libev
make | glib-2.48 | glib-2.50 | libxml2 | xml2 | xml
```

Single cluster with high coverage, but wide distribution.

Partitioning

Selected tokens for array cluster: String Pipe

```
termcast, string, lavender
utils, system, sqlite, sqlite3, builtins, portable, locale, ... tokenize
libxml, tokenizer, login, debug, cf, doc
Context 1,2:
String * Pipe
Context 3: String String [] sep(\'\')
```

Structure Discovery Review

- Compute frequency distribution for each token.
- Create hypothesis about data structure from cluster distributions:
  - Struct
  - Array
  - Union
  - Basic type (bottom out)
- Partition data according to hypothesis & recurse

Format inference overview

```
Raw Data

Chaining Process

Tokenization

Structure Discovery

Scoring Function

XML/Rex

XML

Accumulator

Analysis Report

PADS Compiler

IR to PADS Printer

Format Refinement
```

Format Refinement

- Rewrite format description to:
  - Optimize information-theoretic complexity
    - Simplify presentation
    - Merge adjacent structures and unions
    - Improve precision
    - Identify constant values
    - Introduce enumerations and dependencies
    - Fill in missing details
  - Find completions where structure discovery stops
  - Refine types
    - Termination conditions for strings
    - Integer sizes
    - Identify array element separators & terminators
**Scoring**

- **Goal:** A quantitative metric to evaluate the quality of inferred descriptions and drive refinement.
- **Challenges:**
  - *Underfitting.* Pstring(Proof) describes data, but is too general to be useful.
  - *Overfitting.* Type that exhaustively describes data (**H**, **e**, **r**, **m**, **i**, **o**, **n**, **e**,...) is too precise to be useful.
- **Sweet spot:** Reward compact descriptions that predict the data well.

**Minimum Description Length**

- Standard metric from machine learning.
- Cost of transmitting the syntax of a description plus the cost of transmitting the data given the description:
  \[
  \text{cost}(T,d) = \text{complexity}(T) + \text{complexity}(d|T)
  \]
- Functions defined inductively over the structure of the type T and data d respectively.
- Normalized MDL gives compression factor.
- Scoring function triggers rewriting rules.

**Testing and Evaluation**

- Evaluated overall results qualitatively
  - Compared with Excel -- a manual process with limited facilities for representation of hierarchy or variation
  - Compared with hand-written descriptions -- performance variable depending on tokenization choices & complexity
- Evaluated accuracy quantitatively
  - Implemented infrastructure to use generated accumulator programs to determine inferred description error rates
- Evaluated performance quantitatively
  - Tokenization & rough structure inference perform well: less than 1 second on 300K
  - Dependency analysis can take a long time on complex format (but can be cut down easily).

**Benchmark Formats**

<table>
<thead>
<tr>
<th>Data source</th>
<th>Chunks</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T967/Transactions_short</td>
<td>0.20</td>
<td>3.32</td>
<td>Transaction records</td>
</tr>
<tr>
<td>NER_TOI_01.csv</td>
<td>0.11</td>
<td>2.82</td>
<td>Comma-separated records</td>
</tr>
<tr>
<td>Ai_3000</td>
<td>1.97</td>
<td>26.35</td>
<td>Log file of MAC AOL</td>
</tr>
<tr>
<td>Ai.log</td>
<td>2.90</td>
<td>52.07</td>
<td>Mac OS boot log</td>
</tr>
<tr>
<td>Crashreporter.log</td>
<td>0.11</td>
<td>3.40</td>
<td>Original crashreporter daemon log</td>
</tr>
<tr>
<td>Crashreporter.log.mod</td>
<td>0.13</td>
<td>3.58</td>
<td>Modified crashreporter daemon log</td>
</tr>
<tr>
<td>Sirrus_1000</td>
<td>0.15</td>
<td>3.83</td>
<td>AT&amp;T phone provision data</td>
</tr>
<tr>
<td>Lat_tst</td>
<td>0.07</td>
<td>0.74</td>
<td>Command line output</td>
</tr>
<tr>
<td>Netstat-an</td>
<td>0.07</td>
<td>0.74</td>
<td>Output from netstat-an</td>
</tr>
<tr>
<td>Page_log</td>
<td>0.07</td>
<td>0.74</td>
<td>Printer log from CUPS</td>
</tr>
<tr>
<td>quarter/personalincome</td>
<td>0.07</td>
<td>0.74</td>
<td>Spreadsheet</td>
</tr>
<tr>
<td>Railroad_tst</td>
<td>0.07</td>
<td>0.74</td>
<td>railroad info</td>
</tr>
<tr>
<td>Scrollkeeper.log</td>
<td>0.07</td>
<td>0.74</td>
<td>Application log</td>
</tr>
<tr>
<td>Windowserror_last.log</td>
<td>0.07</td>
<td>0.74</td>
<td>Log from Mac Log/Windows server</td>
</tr>
<tr>
<td>Tmm.txt</td>
<td>0.11</td>
<td>1.91</td>
<td>Log from package installer Tmm</td>
</tr>
</tbody>
</table>

**Execution Times**

<table>
<thead>
<tr>
<th>Data sources</th>
<th>SD (s)</th>
<th>Ref (s)</th>
<th>Tst (s)</th>
<th>HM (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T967/Transactions_short</td>
<td>0.20</td>
<td>3.32</td>
<td>2.36</td>
<td>4.0</td>
</tr>
<tr>
<td>NER_TOI_01.csv</td>
<td>0.11</td>
<td>2.82</td>
<td>2.92</td>
<td>0.5</td>
</tr>
<tr>
<td>Ai_3000</td>
<td>1.97</td>
<td>26.35</td>
<td>28.64</td>
<td>1.0</td>
</tr>
<tr>
<td>Ai.log</td>
<td>2.90</td>
<td>52.07</td>
<td>55.26</td>
<td>1.0</td>
</tr>
<tr>
<td>Crashreporter.log</td>
<td>0.11</td>
<td>3.40</td>
<td>3.53</td>
<td>1.0</td>
</tr>
<tr>
<td>Crashreporter.log.mod</td>
<td>0.13</td>
<td>3.58</td>
<td>3.73</td>
<td>2.0</td>
</tr>
<tr>
<td>Sirrus_1000</td>
<td>0.15</td>
<td>3.83</td>
<td>4.00</td>
<td>2.0</td>
</tr>
<tr>
<td>Lat_tst</td>
<td>0.07</td>
<td>0.74</td>
<td>0.82</td>
<td>1.0</td>
</tr>
<tr>
<td>Netstat_an</td>
<td>0.07</td>
<td>0.74</td>
<td>0.82</td>
<td>1.0</td>
</tr>
<tr>
<td>Page_log</td>
<td>0.06</td>
<td>0.55</td>
<td>0.65</td>
<td>0.5</td>
</tr>
<tr>
<td>quarter/personalincome</td>
<td>0.07</td>
<td>5.11</td>
<td>5.18</td>
<td>48</td>
</tr>
<tr>
<td>Railroad_tst</td>
<td>0.06</td>
<td>2.69</td>
<td>2.76</td>
<td>2.0</td>
</tr>
<tr>
<td>Scrollkeeper.log</td>
<td>0.13</td>
<td>3.24</td>
<td>3.36</td>
<td>1.0</td>
</tr>
<tr>
<td>Windowserror_last.log</td>
<td>0.37</td>
<td>9.65</td>
<td>10.07</td>
<td>1.5</td>
</tr>
<tr>
<td>Tmm.txt</td>
<td>0.11</td>
<td>1.91</td>
<td>2.03</td>
<td>5.0</td>
</tr>
</tbody>
</table>

**Training Time**

- SD: Structure discovery that refinement Tst: total HM: hand-written

![Graph showing training time](image)
**Normalized MDL Scores**

<table>
<thead>
<tr>
<th>Data source</th>
<th>SD</th>
<th>Ref</th>
<th>HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>9M71/Transaction.sh</td>
<td>0.295</td>
<td>0.219</td>
<td>0.248</td>
</tr>
<tr>
<td>MBL_Ver_01.csv</td>
<td>0.668</td>
<td>0.112</td>
<td>0.138</td>
</tr>
<tr>
<td>AI_3000</td>
<td>0.505</td>
<td>0.312</td>
<td>0.338</td>
</tr>
<tr>
<td>AI.log</td>
<td>0.630</td>
<td>0.267</td>
<td>0.361</td>
</tr>
<tr>
<td>Bsd.log</td>
<td>0.420</td>
<td>0.481</td>
<td>0.703</td>
</tr>
<tr>
<td>Crashreporter.log</td>
<td>0.607</td>
<td>0.128</td>
<td>0.148</td>
</tr>
<tr>
<td>Crashreporter.log.mod</td>
<td>0.612</td>
<td>0.329</td>
<td>0.347</td>
</tr>
<tr>
<td>Sirrus.1000</td>
<td>0.602</td>
<td>0.470</td>
<td>0.438</td>
</tr>
<tr>
<td>Sql.stat</td>
<td>0.510</td>
<td>0.353</td>
<td>0.491</td>
</tr>
<tr>
<td>Netstat.an</td>
<td>0.413</td>
<td>0.394</td>
<td>0.319</td>
</tr>
<tr>
<td>Page.log</td>
<td>0.540</td>
<td>0.107</td>
<td>0.353</td>
</tr>
<tr>
<td>quarterly/personal_income</td>
<td>0.544</td>
<td>0.367</td>
<td>0.354</td>
</tr>
<tr>
<td>Railload.txt</td>
<td>0.755</td>
<td>0.586</td>
<td>0.522</td>
</tr>
<tr>
<td>Scrollkeeper.log</td>
<td>0.629</td>
<td>0.354</td>
<td>0.352</td>
</tr>
<tr>
<td>Sreenrunner_last.log</td>
<td>0.628</td>
<td>0.241</td>
<td>0.267</td>
</tr>
<tr>
<td>Turn.txt</td>
<td>0.827</td>
<td>0.305</td>
<td>0.474</td>
</tr>
</tbody>
</table>

**Training Accuracy**

![Training Accuracy Graph](image)

**Type Complexity and Min. Training Size**

<table>
<thead>
<tr>
<th>Data source</th>
<th>Norm. Ty Complexity</th>
<th>90%</th>
<th>95%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sirrus.1000</td>
<td>0.0001</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>9M71/Transaction.sh</td>
<td>0.0023</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>AI_3000</td>
<td>0.0004</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>AI.log</td>
<td>0.0012</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Scrollkeeper.log</td>
<td>0.0040</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Page.log</td>
<td>0.0012</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>MBL_Ver_01.csv</td>
<td>0.0037</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Crashreporter.log</td>
<td>0.0052</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Crashreporter.log.mod</td>
<td>0.0053</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Window runner_last.log</td>
<td>0.0084</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Netstat.an</td>
<td>0.0118</td>
<td>23</td>
<td>35</td>
</tr>
<tr>
<td>Turn.log</td>
<td>0.0214</td>
<td>35</td>
<td>45</td>
</tr>
<tr>
<td>quarterly/personal income</td>
<td>0.0170</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Boost.log</td>
<td>0.0213</td>
<td>45</td>
<td>60</td>
</tr>
<tr>
<td>Sql.stat</td>
<td>0.0461</td>
<td>50</td>
<td>65</td>
</tr>
<tr>
<td>Railload.txt</td>
<td>0.0485</td>
<td>60</td>
<td>75</td>
</tr>
</tbody>
</table>

**Problem: Tokenization**

- **Technical problem:**
  - Different data sources assume different tokenization strategies
  - Useful token definitions sometimes overlap, can be ambiguous, aren’t always easily expressed using regular expressions
  - Matching tokenization of underlying data source can make a big difference in structure discovery.
- **Current solution:**
  - Parameterize learning system with customizable configuration files
  - Automatically generate lexer file & basic token types
- **Future solutions:**
  - Use existing PADS descriptions and data sources to learn probabilistic tokenizers
  - Incorporate probabilities into back-end rewriting system
    - Back end has more context for making final decisions than the tokenizer, which reads 1 character at a time without look ahead

**Structure Discovery Analysis**

- Usually identifies top-level structure sufficiently well to be of some use
- When tokenization is accurate, this phase performs well
- When tokenization is inaccurate, this phase performs less well
  - Descriptions are more complex than hand-coded ones
  - Intuitively: one or two well-chosen tokens in a hand-coded description is represented by complex combination of unions, options, arrays and structures
- **Technical problems:**
  - When to give up & bottom out
  - Choosing between unions and arrays
- **Current Solutions:**
  - User-specified recursion depth
  - Structures prioritized over arrays, which are prioritized over unions
- **Future Solutions:**
  - Information-theory-driven bottoming out
  - Expand infrastructure to enable “search” and evaluation of several options

**Format Refinement Analysis**

- Overall, refinement substantially improves precision of data format & sometimes improves compactness
- **Technical problem 1:**
  - Sometimes refinement is overly aggressive, unnecessarily expanding data descriptions without providing added value in terms of precision
- **Current solution 1:**
  - Do not refine all possible base types – limit refinements to simplest types (int, string, white space)
  - Refinement of complex types such as dates & URLs is not usually needed by tools or programmers (even when they really are constant) and often leads to overfitting.
- **Future solution 1:**
  - Tune complexity analysts more finely and use it as a guide for rewriting
  - Identify refinement opportunities for which insufficient data is available
Format Refinement Analysis

- Technical problem 2:
  - Value-space analysis is $O(R \times T)$ where $R$ is the number of records and $T$ is the number of abstract syntax tree nodes in the description. In some descriptions, $T$ is sufficiently large that value-space analysis grinds to a halt.

- Current solution 2:
  - Bound the size of the table generated from the abstract syntax tree, discarding the chance to find dependencies in some portions of the description.

- Future solution 2:
  - Optimize value-space algorithms intelligently
    - Perform left-to-right sweep, ignoring backward dependencies
    - Detect candidate dependencies on small data sets, discard non-candidates & verify candidate feasibility on larger data sets

Scoring Analysis

- Technical Problem: It is unclear how to weigh type complexity vs data complexity to predict human preference in description structure.
- Current Solution:
  - Final type complexity and final data complexity are weighted equally in the total cost function.
  - However, final data complexity grows linearly with the amount of data used in the experiment.
- Future Solutions:
  - Observation: some of our experiments suggest that humans weight type complexity more heavily than data complexity.
    - Introduce a hyper parameter $h$ and perform experiments, varying $h$ until cost of inferred results and expert descriptions match expectations:
      - cost = $n$-type-complexity + data-complexity
  - Bottom Line: Information theory is a powerful and general tool, but more research is needed to tune it to our application domain.

Related work

- Grammar Induction
  - Extracting Structure from Web Pages [Arasu & Hector-Holena, SigMod, 2003].
  - Grammatical Inference for Information Extraction and Visualization on the Web [Hong, PhD Thesis, Imperial College, 2003].
  - Current Trends in Grammatical Inference [Higuera, LNCS, 2001].
- Functional dependencies
- Information Theory
  - Advances in Minimum Description Length [Grünwald, MIT Press, 2004].

Technical Summary

- Format inference is feasible for many ASCII data formats.
- Our current tools infer sufficient structure that descriptions may be piped into the PADS compiler and used to generate tools for XML conversion and simple statistical analysis.

Thanks & Acknowledgements

- Collaborators
  - Kenny Zhu (Princeton)
  - Peter White (Galois)
- Other contributors
  - Alex Aiken (Stanford)
  - David Blei (Princeton)
  - David Burke (Galois)
  - Vikas Kedia (Stanford)
  - John Launchbury (Galois)
  - Rob Shapire (Princeton)

www.padsproi.org