Typing AD Hoc Data
Kathleen Fisher
AT&T Labs Research

Data, Data, everywhere!

Table amounts of data stored in well-behaved files

Databases:

Tools

XML:

We’re not always so lucky!

Vast amounts of chaotic ad hoc data:

Tools

Perl
Awk
C
...

Government stats

Train Stations

Web logs
Genetic data

Haskell HI files

Ad hoc data from AT&T

<table>
<thead>
<tr>
<th>Name &amp; Use</th>
<th>Representation</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web server logs (CLF)</td>
<td>Fixed-column ASCII records</td>
<td>12 GB/week</td>
</tr>
<tr>
<td>Sirius data:</td>
<td>Variable-width ASCII records</td>
<td>2.2 GB/week</td>
</tr>
<tr>
<td>Call detail: DDoS fraud</td>
<td>Fixed-width binary records</td>
<td>~7 GB/day</td>
</tr>
<tr>
<td>Altair data:</td>
<td>Various Cobuf data formats</td>
<td>~4000 files/day</td>
</tr>
<tr>
<td>Regular data: Monitor IP network</td>
<td>ASCII</td>
<td>2.15 sources, ~15 GB/day</td>
</tr>
<tr>
<td>Netflow: Monitor IP network</td>
<td>Data-dependent number of fixed-width binary records</td>
<td>&gt;1 Gigabit/second</td>
</tr>
</tbody>
</table>

And many others...

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

Technical challenges

- Data arrives "as is."
- Documentation is often out-of-date or nonexistent.
  - Hijacked files.
  - Undocumented "missing value" representations.
- Data is buggy.
  - Missing data, human error, malfunctioning machines, race conditions on log entries, "extra" data, ...
  - Processing must detect relevant errors and respond in application-specific ways.
  - Errors are sometimes the most interesting portion of the data.
  - Data sources often have high volume.
  - Data may not fit into main memory.

Existing approaches

- Lex/Yacc
  - No one we have encountered uses them for ad hoc data.
- Perl/C
  - Code brittle with respect to changes in input format.
  - Analysis often ends up interwoven with parsing, precluding reuse.
  - Error code, if written, swamps main-line computation. If not written, errors can corrupt "good" data.
  - Everything has to be coded by hand.
- Data description languages (PacketTypes, Datascript)
- Binary data
types to the rescue!

Relational and XML data are relatively easy to manage (partly) because schema exist to describe the data.

<table>
<thead>
<tr>
<th>Relational Data</th>
<th>Relational Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XML Schema</td>
</tr>
<tr>
<td>Ad Hoc Data</td>
<td>Physical Types</td>
</tr>
</tbody>
</table>

Thesis: Types can facilitate ad hoc data management, and the types developed for in-memory values are suited to the task.

Our approach: PADS

- Data expert writes declarative description of data source:
  - Physical format information
  - Semantic constraints
- Many data consumers use description and generated parser.
- Description serves as living documentation.
- Parser exhaustively detects errors without cluttering user code.
- From declarative specification, we generate auxiliary tools.

Typing ad hoc data

<table>
<thead>
<tr>
<th>Physical Type</th>
<th>Standard Type</th>
</tr>
</thead>
</table>

Base types

<table>
<thead>
<tr>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>String, Int, Float</td>
</tr>
</tbody>
</table>

Tuple types

| String * Int * Float * String * Int |

Singleton types

| `
``` * String * `\` * `,` |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>* Int</td>
</tr>
<tr>
<td>* Float</td>
</tr>
<tr>
<td>* String</td>
</tr>
<tr>
<td>* Int</td>
</tr>
</tbody>
</table>

Where we write `\`, for `S(\`,`).
**Simple dependent types**

```
\"" * String(\"") * \"" * \",
* Int    * \",
* Float  * \",
* String(\',\') * \",
* Int
```

**Records**

```
{  
source: String(\"")   ,  \"\" ,
date: Int,  \"\" ,
measurement: Float,  \"\" ,
units: String(\',\')  ,  \"\" ,
order: Int
}
```

**Unions**

```
Anonymous:

    \"U\" + Int

Named:

    type OptInt = unavailable of \"U\" |
                  available of Int
```

**Arrays/Lists**

```

    type OptInt = unavailable of \"U\"  |
                  available of Int

    type counts = OptInt[\{4@PFE\,\}]
                  term(eor)
```

**Dependent Types**

```

    type responseCode = (x : Int | 99 < x < 600)

    type method = GET | POST | LINK | UNLINK | ...

    fun check(method, major, minor) = ...

    type request =
        { method : method, \"\",
          url   : String(\"\") ," HTTP/",
          major : Int, \"\",
          minor : Int
        | where check(method, major, minor)
```
**Value Abstraction**

Two representations for missing integer: ‘U’ and ‘-’. We can use value abstraction to reduce redundancy:

- `type OptInt = x.unavailable of x | available of Int`
- `type OptIntU = OptInt ‘U’`
- `type OptIntD = OptInt ‘-’`

**Type Abstraction**

Two different types can be missing: Int and String. We can use type abstraction to reduce redundancy:

- `type Opt = At.lx.unavailable of x | available of t`
- `type OptIntU = Opt Int ‘U’`
- `type OptIntD = Opt Int ‘-’`
- `type OptStringD = Ay.Opt (String y) ‘-’`

**Recursive types**

```haskell
-- Recursive types

type Entry = {name : String(':'), ':', dist : Float}
type Tree = Leaf of Entry |
          Interior of
           '(' * Tree[] (';', ',') * ')') * ':' * Float
```

**Pointers?**

```haskell
-- Pointers?

type Dictionary = {count : SBR_uint32(4),
                  ids : HString[count]}

... type Hi = {id : B uint32(4) where checkId(id),
              dict : Dictionary Pointer(4), ...}
```

**Type summary**

- Base types
- Tuples
- Singleton types
- Records
- Unions
- Lists/Arrays
- Value abstraction
- Type abstraction
- Dependent types
- Recursive types
- Pointers
- ??

**Differences**

- Data layout is not under the control of the type system.
- Physical types need some extra information: separators, terminators.
- Many physical types map to the same internal type: String(‘ ’), String(‘:’), SBR_uint32, B_uint32 ...
- Dependent types much more important for physical types:
- Missing value representations, value-level constraints, embedded array lengths, union tags.
- We should not assume data conforms 100% to description.
**Leverage**

- Convert PADS description into a collection of tools:
  - Accumulators
  - Histograms
  - Clustering tool
  - Formatters
  - Translator into XML, with corresponding XML Schema.
  - XQueries using Galan’s data interface
  - ...
- Long term goal: Provide a compelling suite of tools to overcome inertia of a new language and system.

**Pretty printer**

- Customizable program to reformat data:

<table>
<thead>
<tr>
<th>Normalizes time zones</th>
<th>Drop unnecessary values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalizes delimiters</td>
<td>Filter/repair errors</td>
</tr>
</tbody>
</table>

- Users can override printing on a per type basis.
- Used at AT&T to normalize monitoring data before loading into a relational database.

**Why a domain specific language?**

- Dramatically shorter code (68 lines versus ~7.9K lines).
- Description is short enough to serve as documentation.
- Safer: error code inserted automatically and completely (we just have to get the compiler right...).
- We can leverage the declarative specification to produce value-added tools.
**Future research directions**

- Design
  - How can we specify error-aware data transformations?
  - Can we infer a data transformation between two descriptions?
  - How can we express application-specific information?
  - How can we generate template programs for arbitrary data sources?

- Implementation
  - How can we specialize generated libraries to incorporate application-specific information?
  - How can we optimize streaming queries?

- Theory
  - How do we precisely specify the semantics of PADS? [not tomorrow.]
  - What is the expressiveness of PADS vs. context-free grammars?

- Engineering
  - How do we build the system to make it easy to add new base types?
  - New libraries and tools? New language bindings?

---

**Contributors**

- Kathleen Fisher (AT&T)
- Robert Gruber (Google)
- Mary Fernandez (AT&T)
- Joel Gottlieb (AT&T)
- Yitzhak Mandelbaum (Princeton → AT&T)
- Martin Strauss (University of Michigan)
- David Walker (Princeton)
- Xuan Zheng (University of Michigan)

---

**Try it!**

- Available for download with an open source license (CPL 1.0).
- Demo of accumulators, format program, and XML conversion.
- Send us feedback!

[www.padsproj.org](http://www.padsproj.org)