Incremental Learning of System Log Formats

Kenny Zhu  Shanghai Jiao Tong University
Kathleen Fisher  AT&T Labs Research
David Walker  Princeton University

October 2009
A complex system emits a variety of log files describing its behavior. Declarative data descriptions may describe the format of log files.

An **incremental learning system** analyzes logs and infers or refines descriptions to cover observed data.

Description compiler converts inferred description into parsers and other tools for use in backend analysis systems.
Motivation: Enterprise Web Hosting Service

Many computers and programs working together:

Monitoring system:
- Pulls log files from every machine and every application every 5 minutes.
- Triggers alarms on boundary conditions.
- Builds signatures to track normal conditions.
- Triggers alarms when deviate from normal.
- Loads into datastore for further analysis.
More monitored systems at AT&T...

Wireless Network

Backbone Network

Common Characteristics:
- Need to guarantee availability/performance.
- Many machines.
- Many applications.
- Many vendors.
- Many versions of software.
- Format of log files determined by vendor.
- Log file formats evolve over time.
- May only have access to log files, not generating programs.

Phone Service Provisioning System

Billing System

Corporate Web Sites

Corporate Network

Wednesday, October 14, 2009
Ingesting such log data is difficult

- Data arrives “as is” in a wide variety of formats.
- Documentation is out of data or non-existent.
- Data is buggy and potentially malicious.
- Processing must detect errors and respond in application-specific ways.
- Data sources often have high volume.
- Data evolves over time.
- Existing solutions are insufficient
  - Lex/Yacc-like technologies are both over- and underkill.
  - Hand-coded parsers are time-consuming to write, brittle with respect to changes, and don’t handle errors well.
Data Description Languages

- Data description languages (DDLs) address these issues.
  - Data expert writes **declarative** description rather than a parser.
  - Description serves as **living** documentation.
  - Parser exhaustively detects errors **without cluttering** user code.
  - From declarative specification, we can **generate auxiliary tools**.

"PADS: A Data Description Language for Processing Ad hoc Data" (PLDI 2005)
Inferred data formats are described using specialized types:

- Base type library; specialized types for systems data.
  
  \[
  \begin{align*}
  \text{Pint8, Puint8, ...} & \quad \text{// -123, 44} \\
  \text{Pstring(:'|':)} & \quad \text{// hello|} \\
  \text{Pstring_FW(:3:)} & \quad \text{// catdog} \\
  \text{Pdate, Ptime, Pip, ...}
  \end{align*}
  \]

- Type constructors to describe data source structure:
  
  **Sequences:** \text{Pstruct, Parray,}
  
  **Choices:** \text{Punion, Penum, Pswitch, Popt}
  
  **Constraints:** Arbitrary predicates to describe expected properties.
Example Data Description: Simple CLF

**Punion** machine_t {
  Pip ip;
  Phostname host;
};

**Punion** id_t {
  Pchar unk : unk == '-';
  Pstring(:' ':' ) id;
};

**Pstruct** request_t {
  "\"GET \";
  Ppath resource;
  " HTTP/";
  Pfloat version;
  '"';
};

**Precord Pstruct** entry_t {
  machine_t client;
  '; id_t identdID;
  '; id_t userID;
  " ["; Pdate date;
  ':'; Ptime time;
  "] " ; request_t request;
  ';' Pint response;
  ';' Pint length;
};

207.136.97.49 — — [05/May/2009:16:37:20 -0400] "GET /README.txt HTTP/1.1" 404 216
Format Inference

Original Learning System

Raw Data → XMLifier → XML
XMLifier → Accumulator → Analysis Report
Accumulator → PADS Compiler
Accumulator → PADS Description

"From Dirt to Shovels: Fully Automatic Tool Generation from Ad Hoc Data" (POPL 2008)
Making Inference Incremental

• Original inference algorithm converts sequence of records into a description.
  - Cannot start with an existing description.
  - Does not scale to large data sets because it keeps all records in memory.
  - Cannot respond to changes in streams of data over time.

• Incremental version addresses these problems:
  - Can take initial description as input.
  - Scales better because it processes records in batches.
  - Can refine description to respond to changes.
Incremental Learning Architecture

Log data

Filter Program (Generated)

Bad Data

Incremental Learning

Current Data Description

Records that fail to parse with the current description are used to refine the description.
Incremental Algorithm

- **Input**: Description $T$ and new data $rs$.
- **Output**: Revised description $T_R$ that extends $T$ and parses the new data $rs$.

- **Steps**:
  - Parse records with $T$ to produce extended parse trees:
    - **Missing**: expected data did not appear.
    - **Extra**: unexpected data did appear.
  - Collect errors in an accumulator $A$.
  - Convert accumulator $A$ to new description $T_R$.
    - **Missing**: introduce an option type
    - **Extra**: apply original inference algorithm.
  - Apply rewriting rules to simplify description $T_R$. 
Parsing New Records

Data Description:

Records

<table>
<thead>
<tr>
<th>PairR</th>
<th>Int 5</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc*</td>
<td>Error</td>
<td>Recovered “abc”</td>
</tr>
</tbody>
</table>

Parse Trees

<table>
<thead>
<tr>
<th>PairR</th>
<th>Int 8</th>
<th>Fail</th>
</tr>
</thead>
</table>

Wednesday, October 14, 2009
Collect Variants in Accumulator

Parse Tree of Record 1

Accumulator 0

PairA

IntA

SyncA *

PairA

Int 5

Good

Accumulator 1

PairA

IntA

SyncA *

Data Description:

Pair

Int

Sync *
Collect Variants in Accumulator

LearnA nodes are implicitly also OptA nodes.

Parse Tree of Record 2

Accumulator 1

Accumulator 2

PairR

Error

Recovered “abc”

PairR

PairA

IntA

SyncA *

OptA

IntA

PairA

LearnA

abc

SyncA *
Collect Variants in Accumulator

Parse Tree of Record 3

Accumulator 2

PairA
  └── OptA
      ├── IntA
      │    └── LearnA abc
      │        └── SyncA *
      └── PairA

Accumulator 3

PairA
  └── OptA
      └── IntA
      └── LearnA abc
          └── SyncA *
Apply original inference algorithm to learn description for data in LearnA nodes.
A rewriting rule $R$ applies if the **Minimum Description Length** (MDL) of the current description $T_1$ is greater than the MDL of the revision $T_2$.

$$MDL(T,rs) = TC(T) + ADC(T,rs)$$

$TC(T)$ = number of bits to encode $T$

$ADC(T,rs)$ = average number of bits required to encode each $r$ in $rs$, given $T$
The incremental algorithm often produces sequences of correlated nested options. A rewriting rule re-factors such patterns:

```
Example Rewriting Rule

• The incremental algorithm often produces sequences of correlated nested options. A rewriting rule re-factors such patterns:

```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Complications

• Many ways to parse variant records.
  - **Solution**: Define metric that rewards *correctly parsed characters* while penalizing *skipping characters* and the *number of distinct errors*. Select only *top k parses* for each record.

• Many ways to aggregate candidate parses.
  - **Solution**: Define metric that penalizes *number of OptA* and *number of LearnA Nodes*. Maintain only *top j aggregates*.

Clearly heuristic, but works well in practice so far.
### Experimental Evaluation

<table>
<thead>
<tr>
<th>Formats</th>
<th>K Lines/KB</th>
<th>original</th>
<th></th>
<th>incremental</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>TC</td>
<td>Time</td>
<td>TC</td>
</tr>
<tr>
<td>interface</td>
<td>1.2/185</td>
<td>48.5</td>
<td>0.7</td>
<td>2.9</td>
<td>1.1</td>
</tr>
<tr>
<td>asl.log</td>
<td>1.5/552</td>
<td>31.9</td>
<td>0.9</td>
<td>13.5</td>
<td>1.5</td>
</tr>
<tr>
<td>error_log</td>
<td>4.5/409</td>
<td>93.1</td>
<td>0.1</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td>access_log</td>
<td>8.2/551</td>
<td>130.5</td>
<td>0.3</td>
<td>2.8</td>
<td>0.3</td>
</tr>
<tr>
<td>coblitz</td>
<td>9.4/2561</td>
<td>-</td>
<td>-</td>
<td>31.9</td>
<td>2.9</td>
</tr>
<tr>
<td>pws</td>
<td>17.4/3432</td>
<td>-</td>
<td>-</td>
<td>133</td>
<td>5.7</td>
</tr>
<tr>
<td>ai.big</td>
<td>57.4/5608</td>
<td>-</td>
<td>-</td>
<td>26.2</td>
<td>0.5</td>
</tr>
<tr>
<td>exlog</td>
<td>260.8/76720</td>
<td>-</td>
<td>-</td>
<td>610</td>
<td>3.0</td>
</tr>
<tr>
<td>redirect</td>
<td>302.6/102404</td>
<td>-</td>
<td>-</td>
<td>1852</td>
<td>17.1</td>
</tr>
<tr>
<td>getbig</td>
<td>550.4/92192</td>
<td>-</td>
<td>-</td>
<td>668</td>
<td>8.8</td>
</tr>
</tbody>
</table>

Execution times are in seconds. Type complexity (TC) is in KBs. Platform: PowerBook G4 with 1.67 Ghz PowerPC CPU, 2GB memory, OS X 10.4

No parse errors except pws, which hits PADS greedy parsing of unions.
Preliminary Scaling Experiment

![Graph showing the relationship between the total execution time (seconds) and the number of lines (in thousands) for large web logs.](image)

**Platform:** 1.60GHz Intel Xeon CPU, 8GB memory, running GNU/Linux

---

*References*


Future Work

• Continue experimental evaluation
  - More and larger log files.
  - Investigate effects of batch size on learning.

• Explore inferring descriptions of batches in parallel and then merging results.

• Replace PADS greedy parser with Earley-based parsing algorithm.

• Improve non-incremental learning system because description quality depends on quality of initial description.
Questions?