From Dirt to Shovels: Inferring PADS descriptions from ASCII Data

Kathleen Fisher
David Walker
Kenny Zhu
Peter White

March 2008
Data, Data, everywhere!

Incredible amounts of data stored in well-behaved formats:

**Databases:**

- Database

**XML:**

**Tools**

- Schema
- Browsers
- Query Languages
- Standards
- Libraries
- Books, documentation
- Training courses
- Conversion tools
- Vendor support
- Consultants...
We’re not always so lucky!

Vast amounts of chaotic *ad hoc* data:

<table>
<thead>
<tr>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl</td>
</tr>
<tr>
<td>Awk</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>MSN</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>TEAJBUS</td>
</tr>
<tr>
<td>TEAJBUS</td>
</tr>
<tr>
<td>TEAJBUS</td>
</tr>
<tr>
<td>TEAJBUS</td>
</tr>
<tr>
<td>TEAJBUS</td>
</tr>
<tr>
<td>TEAJBUS</td>
</tr>
<tr>
<td>TEAJBUS</td>
</tr>
<tr>
<td>TEAJBUS</td>
</tr>
</tbody>
</table>
Train Stations

<table>
<thead>
<tr>
<th>Railroad Authority</th>
<th>City, State</th>
<th>Stations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Southern California Regional Railroad Authority</td>
<td>Los Angeles, CA</td>
<td>U,45,46,46,47,49,51,U,45,46,46,47,49,51</td>
</tr>
<tr>
<td>Connecticut Department of Transportation</td>
<td>New Haven, CT</td>
<td>U,U,U,U,U,U,8,U,U,U,U,U,U,8</td>
</tr>
<tr>
<td>Tri-County Commuter Rail Authority</td>
<td>Miami, FL</td>
<td>U,U,U,U,U,U,18,U,U,U,U,U,U,18</td>
</tr>
<tr>
<td>Northeast Illinois Regional Commuter Railroad Corporation</td>
<td>Chicago, IL</td>
<td>226,226,226,227,227,227,91,104,104,111,115,125,131</td>
</tr>
<tr>
<td>Northern Indiana Commuter Transportation District</td>
<td>Chicago, IL</td>
<td>18,18,18,18,18,18,18,20,7,7,7,7,7,11</td>
</tr>
<tr>
<td>Massachusetts Bay Transportation Authority</td>
<td>Boston, MA</td>
<td>U,U,117,119,120,121,124,U,U,67,69,74,75,78</td>
</tr>
<tr>
<td>New Jersey Transit Corporation</td>
<td>New York, NY</td>
<td>158,158,158,162,162,162,167,22,22,41,46,46,46,51</td>
</tr>
</tbody>
</table>
Web logs

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
- ...

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.
  - Pint8, Puint8, ...  // -123, 44
  - Pstring(:'|':)      // hello|
  - Pstring_FW(:3:)     // catdog
  - Pdate, Ptime, Pip, ...

- Provides type constructors to describe data source structure:
  - sequences: Pstruct, Parray,
  - choices: Punion, Penum, Pswitch
  - 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

Raw Data

PADS Learning System

PADS Description
Format inference overview

Raw Data

PADS Learning System

PADS Compiler

PADS Description
Format inference overview

Raw Data

XMLifier

Accumulator

PADS Compiler

XML

Analysis Report

PADS Description

PADS Learning System

IR to PADS

Printer

XMLifier

Accumulator

PADS Compiler

XML

Analysis Report

PADS Description

PADS Learning System
Chunking Process

• Convert raw input into sequence of “chunks.”

```
"123, 24"
"731, Harry"
"574, Hermione"
"9378, 56"
"12, Hogwarts"
"112, Ron"
```

```
"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, ...

<table>
<thead>
<tr>
<th>Input</th>
<th>Tokenized</th>
<th>Tokenizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;123, 24&quot;</td>
<td>Quote Int Comma White Int Quote</td>
<td></td>
</tr>
<tr>
<td>&quot;731, Harry&quot;</td>
<td>Quote Int Comma White String Quote</td>
<td></td>
</tr>
<tr>
<td>&quot;574, Hermione&quot;</td>
<td>Quote Int Comma White String Quote</td>
<td></td>
</tr>
<tr>
<td>&quot;9378, 56&quot;</td>
<td>Quote Int Comma White Int Quote</td>
<td></td>
</tr>
<tr>
<td>&quot;12, Hogwarts&quot;</td>
<td>Quote Int Comma White String Quote</td>
<td></td>
</tr>
<tr>
<td>&quot;112, Ron&quot;</td>
<td>Quote Int Comma White String Quote</td>
<td></td>
</tr>
</tbody>
</table>
Histograms

- Quote Int Comma White Int Quote
- Quote Int Comma White String Quote
- Quote Int Comma White String Quote
- Quote Int Comma White Int Quote
- Quote Int Comma White String Quote
- Quote Int Comma White String Quote

Frequency Analysis

<table>
<thead>
<tr>
<th>Appears Once</th>
<th>Appears Twice</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two frequency distributions are *similar* if they have the same shape (within some error tolerance) when the columns are sorted by height.
Clustering

Group clusters with similar frequency distributions

Cluster 1
Cluster 2
Cluster 3

Appears Once
Appears Twice

Rank clusters by metric that rewards high coverage and narrower distributions. Choose cluster with highest score.
Group clusters with similar frequency distributions

Rank clusters by metric that rewards high coverage and narrower distributions. Chose cluster with highest score.
Partition chunks

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

Tokens in selected cluster:
Quote(2) Comma White
Then Recurse...

- Int
- Int
- Int
- Int
- Int

becomes

- Int

becomes

- String
- String
- Int
- String

String + Int
Inferred type

"123, 24"
"731, Harry"
"574, Hermione"
"9378, 56"
"12, Hogwarts"
"112, Ron"

becomes

Quote * Int * Comma * White * (String + Int) * Quote
Finding arrays

Single cluster with high coverage, but wide distribution.

- Hermione
- Ginny
- Lavender
- Malfoy
- Crabbe
- Goyle
- Parkinson
- Bulstrode
- Greengrass
- Nott
- Zabini
- Harry
- Ron
- Neville
- George
- Fred
- Trevor
- Ginger
- Hedwig
- Flitwick
- McGonagall
- Snape
- Sprout
- Quirrell
- Lockhart
- Lupin
- Moody
- Umbridge
- Snape
**Partitioning**

Selected tokens for array cluster: String Pipe

<table>
<thead>
<tr>
<th>hermione</th>
<th>ginny</th>
<th>lavender</th>
</tr>
</thead>
<tbody>
<tr>
<td>malfoy</td>
<td>crabbe</td>
<td>goyle</td>
</tr>
<tr>
<td>harry</td>
<td>ron</td>
<td>nevil</td>
</tr>
<tr>
<td>trevor</td>
<td>ginger</td>
<td>hedwig</td>
</tr>
<tr>
<td>flitwick</td>
<td>mcgonagall</td>
<td>snape</td>
</tr>
<tr>
<td>quirrell</td>
<td>lockhart</td>
<td>lupin</td>
</tr>
</tbody>
</table>
Partitioning

Selected tokens for array cluster: String Pipe

hermione | ginny | lavender
malfoy | crabbe | goyle | parkinson | bulstrode |
... | zabini
harry | ron | nevil | george | fred
trevor | ginger | hedwig
flitwick | mcgonagall | snape | sprout
quirrell | lockhart | lupin | moody | umbridge | snape

Context 1,2:
String * Pipe
Partitioning

Selected tokens for array cluster: String Pipe

hermione |  ginny |  lavender
malfoy |  crabbe |  goyle |  parkinson |  bulstrode | ... |  zabini
harry |  ron |  nevil |  george |  fred
trevor |  ginger |  hedwig
flitwick |  mcgonagall |  snape |  sprout
quirrell |  lockhart |  lupin |  moody |  umbridge |  snape

Context 1,2: String * Pipe

Context 3: String
Partitioning

Selected tokens for array cluster: String Pipe

Context 1,2: String * Pipe

Context 3: String[] sep(‘|’)
Format Refinement: Rewriting

- Optimize information-theoretic complexity
  - Simplify presentation
    - Merge adjacent structures and unions
  - Improve precision
    - Identify constant values
    - Introduce enumerations and dependencies

- Refine types
  - Termination conditions for strings
  - Integer sizes
  - Identify array element separators & terminators
“0, 24"
“foo, beg"
“bar, end"
“0, 56"
“baz, middle”
“0, 12”
“0, 33”
...
"0, 24" "foo, beg" "bar, end" "0, 56" "baz, middle" "0, 12" "0, 33"
...

structure discovery

```
struct
  union
    int alpha
  union
    int alpha
  
struct
  union (id1)
    int (id3) alpha (id4)
  union (id2)
    int (id5) alpha (id6)
```

<table>
<thead>
<tr>
<th>id1</th>
<th>id2</th>
<th>id3</th>
<th>id4</th>
<th>id5</th>
<th>id6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>--</td>
<td>24</td>
<td>--</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>--</td>
<td>foo</td>
<td>--</td>
<td>beg</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
structure discovery

id1 = id2
(first union is “int” whenever second union is “int”)

id3 = 0

<table>
<thead>
<tr>
<th>id1</th>
<th>id2</th>
<th>id3</th>
<th>id4</th>
<th>id5</th>
<th>id6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>--</td>
<td>24</td>
<td>--</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>--</td>
<td>foo</td>
<td>--</td>
<td>beg</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
more accurate:
-- first int = 0
-- rules out \text{"int, alpha-string"} records
Format inference overview

- Raw Data
- Chunking Process
- Tokenization
- Structure Discovery
- Scoring Function
- Format Refinement
- IR to PADS Printer
- PADS Description
- PADS Compiler
- Accumulator
- XMLifier
- XML
- Analysis Report
Scoring

- **Goal:** A quantitative metric to evaluate the quality of inferred descriptions and drive refinement.

- **Challenges:**
  - *Underfitting.* $P_{\text{string}}(P_{\text{eof}})$ 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>1967Transactions.short</td>
<td>999</td>
<td>70929</td>
<td>Transaction records</td>
</tr>
<tr>
<td>MER_T01_01.cvs</td>
<td>491</td>
<td>21731</td>
<td>Comma-separated records</td>
</tr>
<tr>
<td>Ai.3000</td>
<td>3000</td>
<td>293460</td>
<td>Web server log</td>
</tr>
<tr>
<td>Asl.log</td>
<td>1500</td>
<td>279600</td>
<td>Log file of MAC ASL</td>
</tr>
<tr>
<td>Boot.log</td>
<td>262</td>
<td>16241</td>
<td>Mac OS boot log</td>
</tr>
<tr>
<td>Crashreporter.log</td>
<td>441</td>
<td>50152</td>
<td>Original crashreporter daemon log</td>
</tr>
<tr>
<td>Crashreporter.log.mod</td>
<td>441</td>
<td>49255</td>
<td>Modified crashreporter daemon log</td>
</tr>
<tr>
<td>Sirius.1000</td>
<td>999</td>
<td>142607</td>
<td>AT&amp;T phone provision data</td>
</tr>
<tr>
<td>Ls-l.txt</td>
<td>35</td>
<td>1979</td>
<td>Command ls -l output</td>
</tr>
<tr>
<td>Netstat-an</td>
<td>202</td>
<td>14355</td>
<td>Output from netstat -an</td>
</tr>
<tr>
<td>Page_log</td>
<td>354</td>
<td>28170</td>
<td>Printer log from CUPS</td>
</tr>
<tr>
<td>quarterlypersonalincome</td>
<td>62</td>
<td>10177</td>
<td>Spread sheet</td>
</tr>
<tr>
<td>Railroad.txt</td>
<td>67</td>
<td>6218</td>
<td>US Rail road info</td>
</tr>
<tr>
<td>Scrollkeeper.log</td>
<td>671</td>
<td>66288</td>
<td>Application log</td>
</tr>
<tr>
<td>Windowserver_last.log</td>
<td>680</td>
<td>52394</td>
<td>Log from Mac LoginWindow server</td>
</tr>
</tbody>
</table>
| Yum.txt                  | 328    | 18221     | Log from package installer Yum
## Execution Times

<table>
<thead>
<tr>
<th>Data source</th>
<th>SD (s)</th>
<th>Ref (s)</th>
<th>Tot (s)</th>
<th>HW (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1967Transactions.short</td>
<td>0.20</td>
<td>2.32</td>
<td>2.56</td>
<td>4.0</td>
</tr>
<tr>
<td>MER_T01_01.cvs</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>Asl.log</td>
<td>2.90</td>
<td>52.07</td>
<td>55.26</td>
<td>1.0</td>
</tr>
<tr>
<td>Boot.log</td>
<td>0.11</td>
<td>2.40</td>
<td>2.53</td>
<td>1.0</td>
</tr>
<tr>
<td>Crashreporter.log</td>
<td>0.12</td>
<td>3.58</td>
<td>3.73</td>
<td>2.0</td>
</tr>
<tr>
<td>Crashreporter.log.mod</td>
<td>0.15</td>
<td>3.83</td>
<td>4.00</td>
<td>2.0</td>
</tr>
<tr>
<td>Sirius.1000</td>
<td>2.24</td>
<td>5.69</td>
<td>8.00</td>
<td>1.5</td>
</tr>
<tr>
<td>Ls-l.txt</td>
<td>0.01</td>
<td>0.10</td>
<td>0.11</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>0.08</td>
<td>0.55</td>
<td>0.65</td>
<td>0.5</td>
</tr>
<tr>
<td>quarterlypersonalincome</td>
<td>0.07</td>
<td>5.11</td>
<td>5.18</td>
<td>48</td>
</tr>
<tr>
<td>Railroad.txt</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>3.40</td>
<td>1.0</td>
</tr>
<tr>
<td>Windowserver_last.log</td>
<td>0.37</td>
<td>9.65</td>
<td>10.07</td>
<td>1.5</td>
</tr>
<tr>
<td>Yum.txt</td>
<td>0.11</td>
<td>1.91</td>
<td>2.03</td>
<td>5.0</td>
</tr>
</tbody>
</table>

**SD:** structure discovery  
**Ref:** refinement  
**Tot:** total  
**HW:** hand-written
Training Time
## Normalized MDL Scores

<table>
<thead>
<tr>
<th>Data source</th>
<th>SD</th>
<th>Ref</th>
<th>HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1967Transactions.short</td>
<td>0.295</td>
<td>0.218</td>
<td>0.268</td>
</tr>
<tr>
<td>MER_T01_01.cvs</td>
<td>0.648</td>
<td>0.112</td>
<td>0.138</td>
</tr>
<tr>
<td>Ai.3000</td>
<td>0.503</td>
<td>0.332</td>
<td>0.338</td>
</tr>
<tr>
<td>Asl.log</td>
<td>0.630</td>
<td>0.267</td>
<td>0.361</td>
</tr>
<tr>
<td>Boot.log</td>
<td>0.620</td>
<td>0.481</td>
<td>0.703</td>
</tr>
<tr>
<td>Crashreporter.log</td>
<td>0.607</td>
<td>0.328</td>
<td>0.348</td>
</tr>
<tr>
<td>Crashreporter.log.mod</td>
<td>0.612</td>
<td>0.329</td>
<td>0.347</td>
</tr>
<tr>
<td>Sirius.1000</td>
<td>0.602</td>
<td>0.470</td>
<td>0.438</td>
</tr>
<tr>
<td>Ls-l.txt</td>
<td>0.559</td>
<td>0.333</td>
<td>0.401</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>quarterlypersonalincome</td>
<td>0.544</td>
<td>0.367</td>
<td>0.354</td>
</tr>
<tr>
<td>Railroad.txt</td>
<td>0.715</td>
<td>0.506</td>
<td>0.522</td>
</tr>
<tr>
<td>Scrollkeeper.log</td>
<td>0.625</td>
<td>0.354</td>
<td>0.352</td>
</tr>
<tr>
<td>Windowserver_last.log</td>
<td>0.618</td>
<td>0.241</td>
<td>0.267</td>
</tr>
<tr>
<td>Yum.txt</td>
<td>0.827</td>
<td>0.305</td>
<td>0.474</td>
</tr>
</tbody>
</table>

**SD:** structure discovery  
**Ref:** refinement  
**HW:** hand-written
<table>
<thead>
<tr>
<th>Data source</th>
<th>Norm. Ty Complexity</th>
<th>90%</th>
<th>95%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sirius.1000</td>
<td>0.0001</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>1967Transaction.short</td>
<td>0.0003</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>Asl.log</td>
<td>0.0012</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Scrollkeeper.log</td>
<td>0.0020</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Page_log</td>
<td>0.0032</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>MER_T01_01.csv</td>
<td>0.0037</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Crashreporter.log</td>
<td>0.0052</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Crashreporter.log.mod</td>
<td>0.0053</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Windowserver_last.log</td>
<td>0.0084</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Netstat-an</td>
<td>0.0118</td>
<td>25</td>
<td>35</td>
</tr>
<tr>
<td>Yum.txt</td>
<td>0.0124</td>
<td>30</td>
<td>45</td>
</tr>
<tr>
<td>quarterlypersonalincome</td>
<td>0.0170</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Boot.log</td>
<td>0.0213</td>
<td>45</td>
<td>60</td>
</tr>
<tr>
<td>Ls-l.txt</td>
<td>0.0461</td>
<td>50</td>
<td>65</td>
</tr>
<tr>
<td>Railroad.txt</td>
<td>0.0485</td>
<td>60</td>
<td>75</td>
</tr>
</tbody>
</table>
Related Work

- Grammar Induction
  - Generally impossible with only positive examples (Gold, 1967).
  - Focus has been on theoretical problems and natural language.

- Information Extraction
  - User labels training data, trained system then extracts similar fragments (eg, rentals in Craig’s list).
  - Soderland’s Whisk system (1999), Kushmerick (1997)

- XML Inference
  - Leverages known tokenization for XML and tree-structure of XML data to infer DTDs and XSchema

And much more; see paper for more details.
How do language ideas help?

- PADS declarative data description language serves as expressive target for inference system.
  - Specify only what data format is, not how it should be parsed or what data structures to build while parsing.

- Rewriting rules allow us to improve description in semantic-preserving way.

- Core type theoretic semantics allows us to generate PADS/ML or PADS/C specifications.

- Type-directed programming techniques will enable generic tool construction.
Technical Summary

• Format inference and automatic tool generation is feasible for many ASCII data formats.
• Current work: learning tokenizations.
• More information:
  - PADS web site: [www.padsproj.org](http://www.padsproj.org)
  - POPL 2008 Paper: “From Dirt to Shovels”
Thanks & Acknowledgements

- **Collaborators**
  - David Walker (Princeton)
  - 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)
  - Qian Xi (Princeton)