PADS: A Domain-Specific Language for Processing Ad Hoc Data

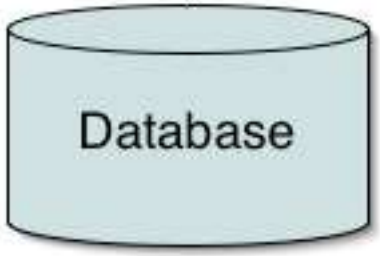
Kathleen Fisher
AT&T Labs Research

Robert Gruber
Google

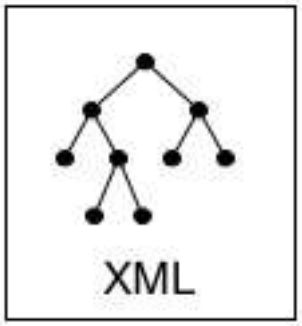
Data, data, everywhere!

Incredible amounts of data stored in well-behaved formats:

Databases:



XML:



Tools

- Schema
- Browsers
- Query languages
- Standards
- Libraries
- Books, documentation
- Conversion tools
- Vendor support
- Consultants…

… but not all data is well-behaved!

Vast amounts of chaotic *ad hoc* data:

Tools

- Perl?
- Awk?
- C?

Ad hoc data from www.geneontology.org

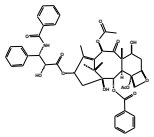
```
!date: Fri Mar 18 21:00:28 PST 2005
!version: $Revision: 3.223 $
!type: % is_a is a
!type: < part_of part of
!type: ^ inverse_of inverse of
!type: | disjoint_from disjoint from $Gene_Ontology ;
GO:0003673 <biological_process ;
GO:0008150 %behavior ;
GO:0007610 ; synonym:behaviour %adult behavior ;
GO:0030534 ; synonym:adult behaviour %adult feeding behavior ;
GO:0008343 ; synonym:adult feeding behaviour %feeding behavior ;
GO:0007631 %adult locomotory behavior ;
GO:0008344 ;
```

Ad hoc in biology: Newick format

```
((raccoon:19.19959,bear:6.80041):0.84600,
((sea_lion:11.99700, seal:12.00300):7.52973,
((monkey:100.85930,cat:47.14069):20.59201,
weasel:18.87953):2.09460):3.87382,dog:25.46154);
(Bovine:0.69395,(Gibbon:0.36079,(Orang:0.33636,
(Gorilla:0.17147,(Chimp:0.19268, Human:0.11927):
0.08386):0.06124):0.15057):0.54939,Mouse:1.21460):0.10;
(Bovine:0.69395,(Hylobates:0.36079,(Pongo:0.33636,
(G._Gorilla:0.17147,
(P._paniscus:0.19268,H._sapiens:0.11927):0.08386):
0.06124):0.15057):0.54939, Rodent:1.21460);
```

Ad hoc data in chemistry

```
O=C([C@@H]2OC(C)=O)[C@@]3(C)[C@]([C@](CO4)
(OC(C)=O)[C@H]4C[C@@H]3O)([H])[C@H]
(OC(C7=CC=CC=C7)=O)[C@@]1(O)[C@@](C)(C)C2=C(C)
[C@@H](OC([C@H](O)[C@@H](NC(C6=CC=CC=C6)=O)
C5=CC=CC=C5)=O)C1
```



Ad hoc data from www.investors.com

Date: 3/21/2005 1:00PM PACIFIC
Investor's Business Daily ®
Stock List Name: DAVE

Stock Symbol	Company Name	Price	Price Change	Price % Change	Volume % Change	EPS Rating	RS Rating
AET	Aetna Inc	**73.68**	-0.22	0%	31%	64	93
GE	General Electric Co	**36.01**	0.13	0%	-8%	59	56
HD	Home Depot Inc	**37.99**	-0.89	-2%	63%	84	38
IBM	Intl Business Machines	**89.51**	0.23	0%	-13%	66	35
INTC	Intel Corp	**23.50**	0.09	0%	-47%	39	33

Ad hoc binary data: DNS packets

```
00000000: 9192 d8fb 8480 0001 05d8 0000 0000 0872  ...............r
00000010: 6573 6561 7263 6803 6174 7403 636f 6d00  esearch.att.com.
00000020: 00fc 0001 c00c 0006 0001 0000 0e10 0027  ...............'
00000030: 036e 7331 c00c 0a68 6f73 746d 6173 7465  .ns1...hostmaste
00000040: 72c0 0c77 64e5 4900 000e 1000 0003 8400  r..wd.I.........
00000050: 36ee 8000 000e 10c0 0c00 0f00 0100 000e  6...............
00000060: 1000 0a00 0a05 6c69 6e75 78c0 0cc0 0c00  ......linux.....
00000070: 0f00 0100 000e 1000 0c00 0a07 6d61 696c  ............mail
00000080: 6d61 6ec0 0cc0 0c00 0100 0100 000e 1000  man.............
00000090: 0487 cf1a 16c0 0c00 0200 0100 000e 1000  ................
000000a0: 0603 6e73 30c0 0cc0 0c00 0200 0100 000e  ..ns0...........
000000b0: 1000 02c0 2e03 5f67 63c0 0c00 2100 0100  ......_gc...!...
000000c0: 0002 5800 1d00 0000 640c c404 7068 7973  ..X.....d...phys
000000d0: 0872 6573 6561 7263 6803 6174 7403 636f  .research.att.co
```

Ad hoc data from AT&T

Name & Use	Representation	Size
Web server logs (CLF): Measure web workloads	Fixed-column ASCII records	≤ 12 GB/week
Sirius data: Monitor service activation	Variable-width ASCII records	2.2GB/week
Call detail: Detect fraud	Fixed-width binary records	~7GB/day
Altair data: Track billing process	Various Cobol data formats	~4000 files/day
Regulus data: Monitor IP network	ASCII	≥ 15 sources, ~15 GB/day
Netflow: Monitor IP network	Data-dependent number of fixed-width binary records	>1Gigabit/second

Technical challenges

- Data arrives "as is."
- Documentation is often out-of-date or nonexistent.
  - Hijacked fields.
  - 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.

Your Turn: Alarm Data for Network

- Alarm type: `RTE` or `CPU`
- Severity: an integer between 1 and 4, inclusive
- Optional timestamp: DD/MM/YYYY:HH:MM:SS TZONE
- IP address of node exhibiting problem
- Payload
  - if `RTE` alarm:
    - number of nodes in path
    - list of IP addresses in path
  - if `CPU`, an integer identifying the relevent CPU
- Operator comment

```
RTE 2|12/Dec/2003:23:10:29 -0700|123.45.65.233__2:240.230.155.13;240.120.11.13|intermitten
CPU 4||200.42.42.110__1346575|unusual workload
```

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
  - Focus on correct data.

Our approach: PADS

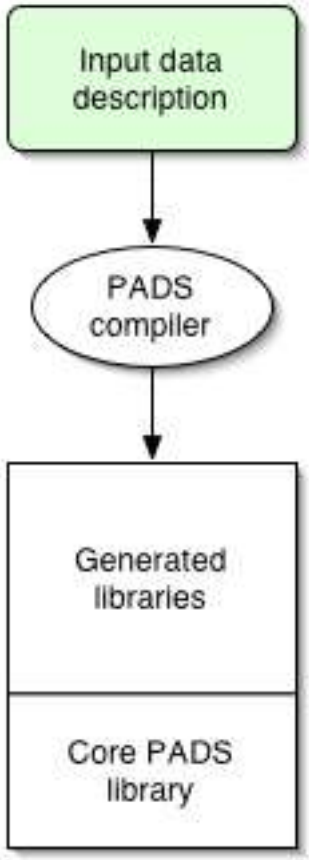
Data expert writes declarative description of data source:

- Physical format information
- Semantic constraints

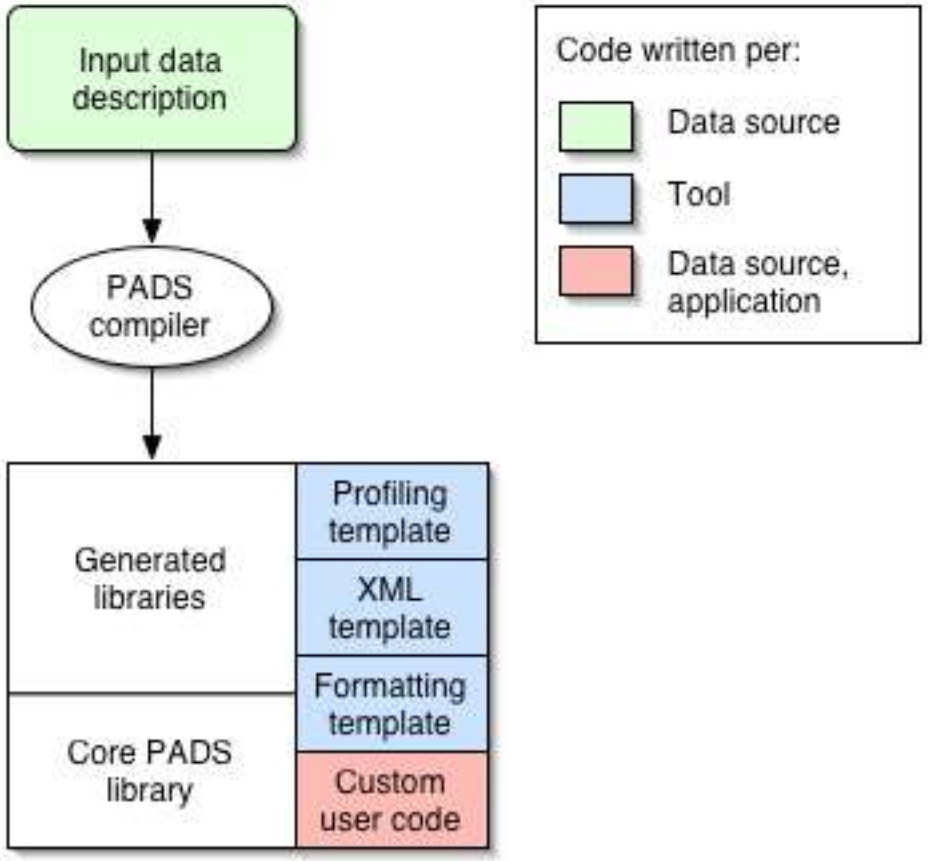
Many data consumers use description and generated parser.

- Description serves as living documentation.
- Parser exhaustively detects errors without cluttering user code.
- From declarative specification, we generate auxiliary tools.

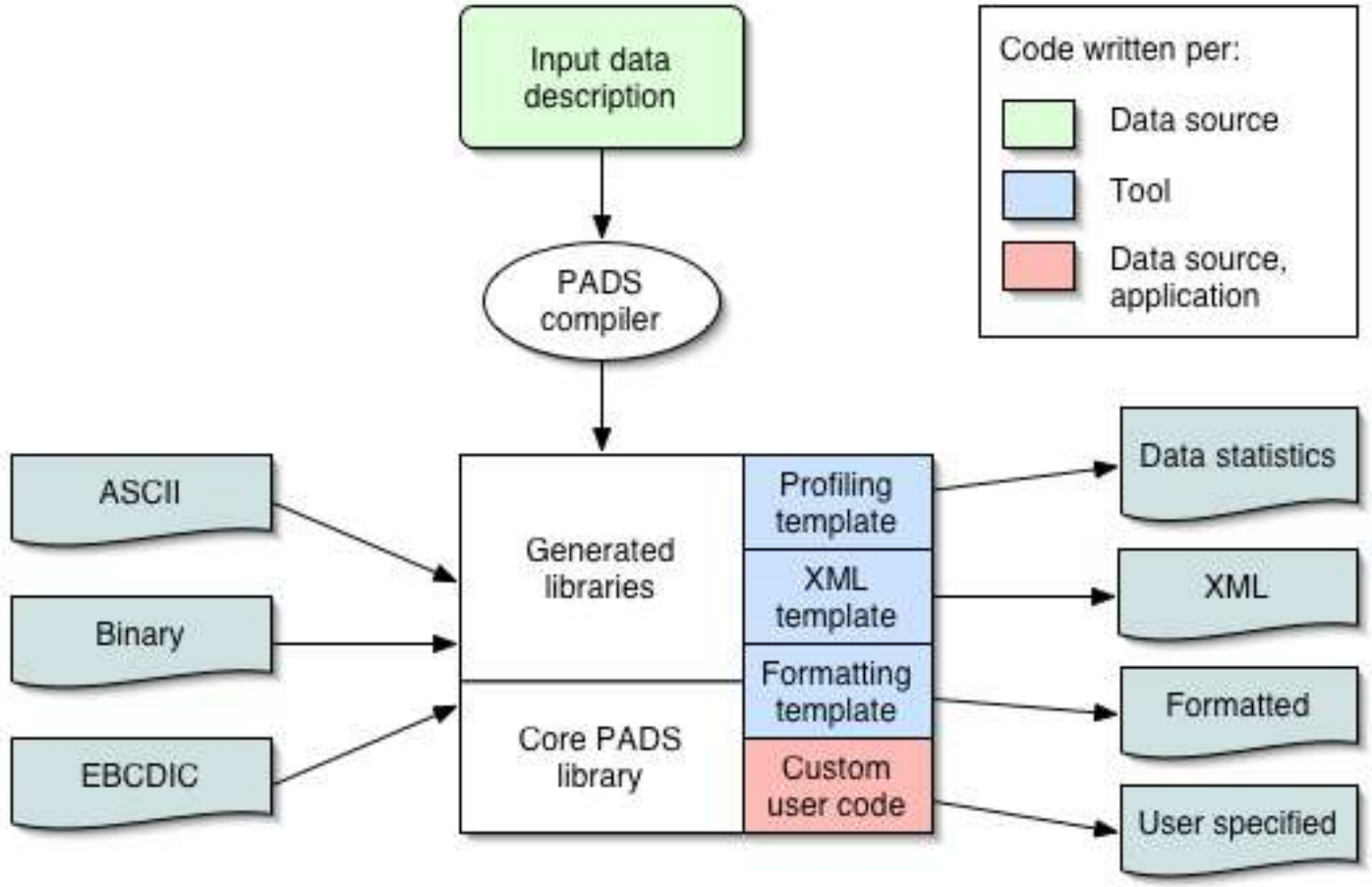
PADS architecture



PADS architecture



PADS architecture



PADS language

Type-based model: types indicate how to process associated data.

- Provides rich and *extensible* set of base types.
  - `Pint8, Puint8, …` `// -123, 44`
  - `Pstring(:'|':)` `// hello|`
    `Pstring_FW(:3:)` `// catdog`
    `Pstring_ME(:"/a*/":)` `// aaaaaab`
  - `Pdate, Ptime, Pip, …`
- Provides type constructors to describe data source structure:

  `Pstruct, Parray, Punion, Ptypedef, Penum`
- Allows arbitrary predicates to describe expected properties.

Running example: CLF web log

- Common Log Format from *Web Protocols and Practice.*

```
207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013
```

- Fields:
  - IP address of remote host
  - Remote identity (usually '-' to indicate name not collected)
  - Authenticated user (usually '-' to indicate name not collected)
  - Time associated with request
  - Request (request method, request-uri, and protocol version)
  - Response code
  - Content length

Example: Pstruct

```
Precord Pstruct http_weblog {
        host client;              /- Client requesting service
  ' ';  auth_id remoteID;         /- Remote identity
  ' ';  auth_id auth;             /- Name of authenticated user
  " [";  Pdate(:']':) date;       /- Timestamp of request
  "] ";  http_request request;    /- Request
  ' ';  Puint16_FW(:3:) response; /- 3-digit response code
  ' ';  Puint32 contentLength;    /- Bytes in response
};
```

207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

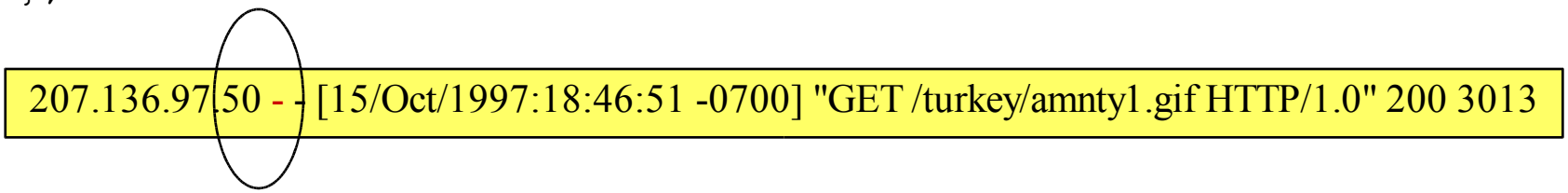
User constraints

```
int chkVersion(http_v version, method_t meth) { …  };

Pstruct http_request {
   '\"'; method_t        meth;
   ' ';  Pstring(:' ':) req_uri;
   ' ';  http_v         version : chkVersion(version, meth);
   '\"';
};
```

207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

Example: Punion

```
Punion id {
  Pchar unavailable : unavailable == '-';
  Pstring(:' ':) id;
};
```



- Union declarations allow the user to describe variations.
- Implementation tries branches in order.
- Stops when it finds a branch whose constraints are all true.

Example: Switched `Punion`

```
Punion info_t (:code_t code :) {
  Pswitch (code) {
    Pcase RTE : rte_t rte;
    Pcase CPU : Puint32 cpu;
  }
};
```

```
RTE 2|12/Dec/2003:23:10:29 -0700|123.45.65.233__2:240.230.155.13;240.120.11.13|intermitten
CPU 4||200.42.42.110__1346575|unusual workload
```

- Switched union uses expression to choose correct branch.

Example: Popt

```
Popt Puint32_opt Puint32;

Pstruct alarm_t {
    …
  Puint32_opt  length;
  Popt Puint32 length2;
  …
};
```

- **Popt** expresses optional data
- Can be used in-line as `length2` field shows
- Encodable as a **Punion**

Example: `Parray`

```
Parray host {
  Puint8[4]: Psep('.');
};
```

207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

Array declarations allow the user to specify:

- Size (fixed, lower-bounded, upper-bounded, unbounded)
- **`Psep`**, **`Pterm`**, and termination predicates
- Constraints over sequence of array elements

Array terminates upon exhausting EOF, reaching terminator, reaching maximum size, or satisfying termination predicate.

Example: Penum

```
Penum method_t {
  GET,
  PUT,
  POST,
  HEAD,
  DELETE,
  LINK,      /- Unused after HTTP 1.0
  UNLINK     /- Unused after HTTP 1.0
};
```

207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

Enumerations are strings on disk, 32-bit integers in memory.

Example: Ptypedef

```
Ptypedef Puint16_FW(:3:) response_t :
         response_t x => { 100 <= x && x < 600};
```

207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

Typedefs allow the user to add constraints to existing types.

Example: Pcompute and Pomit

```
Pstruct log_t {
        Puint32 pn;
  ' ';  Pcompute Puint8 isIntl = isIntl(pn);
  ' ';  Pomit Pstring(:'|':);
   …
};
```

- **Pcompute** allows the user to set part of the internal representation from a computed value, which can depend upon "earlier" data.
- **Pomit** excludes the annotated data from the representation.

CLF in PADS

```
Parray Phostname{
  Pstring_SE(:"/[. ]/":) [] : Psep('.')
                            && Pterm(Pnosep);
};

Punion client_t {
  Pip       ip;       /- 135.207.23.32
  Phostname host;     /- www.research.att.com
};

Punion auth_id_t {
  Pchar unauthorized : unauthorized == '-';
  Pstring(:' ':) id;
};

Penum method_t {
    GET,    PUT,  POST,  HEAD,
    DELETE, LINK, UNLINK
};

Pstruct version_t {
  "HTTP/";
  Puint8 major; '.';
  Puint8 minor;
};

int chkVersion(version_t v, method_t m) {
  if ((v.major == 1) && (v.minor == 1)) return 1;
  if ((m == LINK) || (m == UNLINK)) return 0;
  return 1;
};
```

```
Pstruct request_t {
  '\"';   method_t       meth;
  ' ';    Pstring(:' ':) req_uri;
  ' ';    version_t      version :
                  chkVersion(version, meth);
  '\"';
};

Ptypedef Puint16_FW(:3:) response_t :
         response_t x => { 100 <= x && x < 600};

Punion length_t {
  Pchar unavailable : unavailable == '-';
  Puint32 len;
};

Precord Pstruct entry_t {
         client_t       client;
   ' ';  auth_id_t      remoteID;
   ' ';  auth_id_t      auth;
   " ["; Pdate(:']':)   date;
   "] "; request_t      request;
   ' ';  response_t     response;
   ' ';  length_t       length;
};

Psource Parray clt_t {
  entry_t [];
}
```

Your Turn (Part 2)

- Alarm type: `RTE` or `CPU`
- Severity: an integer between 1 and 4, inclusive
- Optional timestamp: DD/MM/YYYY:HH:MM:SS TZONE
- IP address of node exhibiting problem
- Payload
  - if `RTE` alarm:
    - number of nodes in path
    - list of IP addresses in path
  - if `CPU`, an integer identifying the relevent CPU
- Operator comment

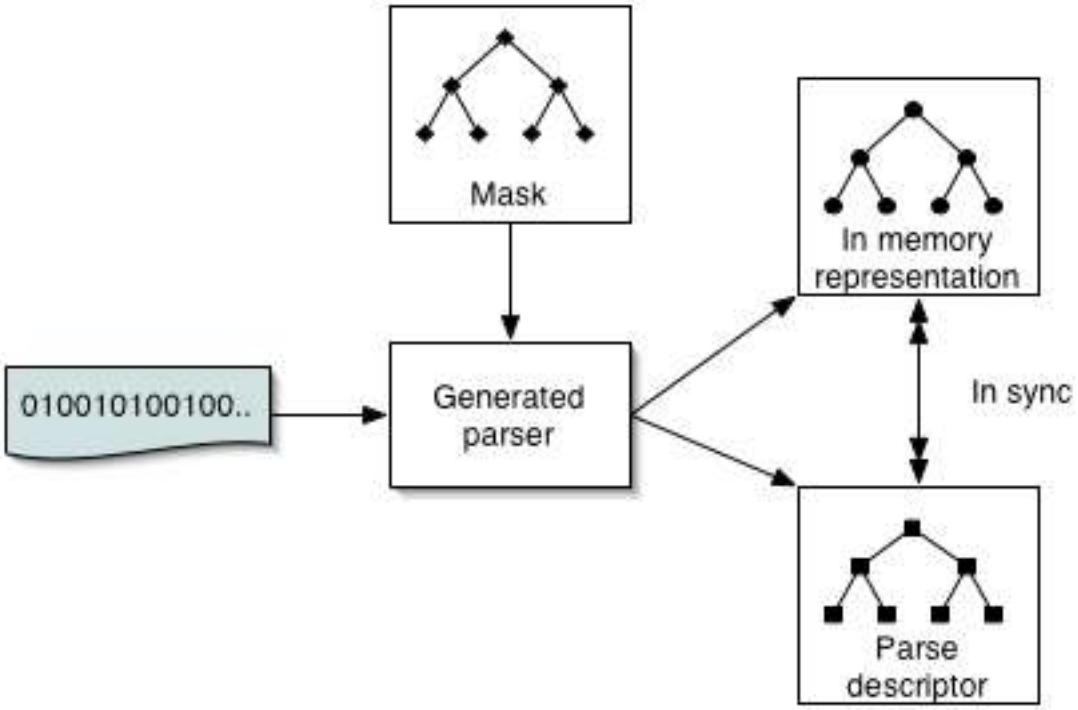
```
RTE 2|12/Dec/2003:23:10:29 -0700|123.45.65.233__2:240.230.155.13;240.120.11.13|intermitten
CPU 4||200.42.42.110__1346575|unusual workload
```

Your turn! (One answer)

```
Penum code_t {RTE, CPU };
Pstruct rte_t {Puint32 num;
          ';'; Pip[num]route : Psep(';');};

Punion info_t (:code_t code :) {
  Pswitch (code) {
    Pcase RTE : rte_t rte;
    Pcase CPU : Puint32 cpu;   } };

Precord Pstruct alarms_t {
      code_t              code;
' ';  Puint32             severity : 1 <= severity && severity <=
  4;
' ';  Popt Pdate(:'|':) timestamp;
'|';  Pip                addr;
"__"; into_t(:code:)    info;
'|';  Pstring(:'\n':)    comment;
};
```

PADS parsing

```
Perror_t entry_t_read(P_t *pdc, entry_t_m* mask,
                                                  ep);
```



Invariant: If mask is "check and set" and parse descriptor reports no errors, then the in-memory representation is correct.

Leverage!

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

Accumulators

- Statistical profile of "leaves" in a data source:

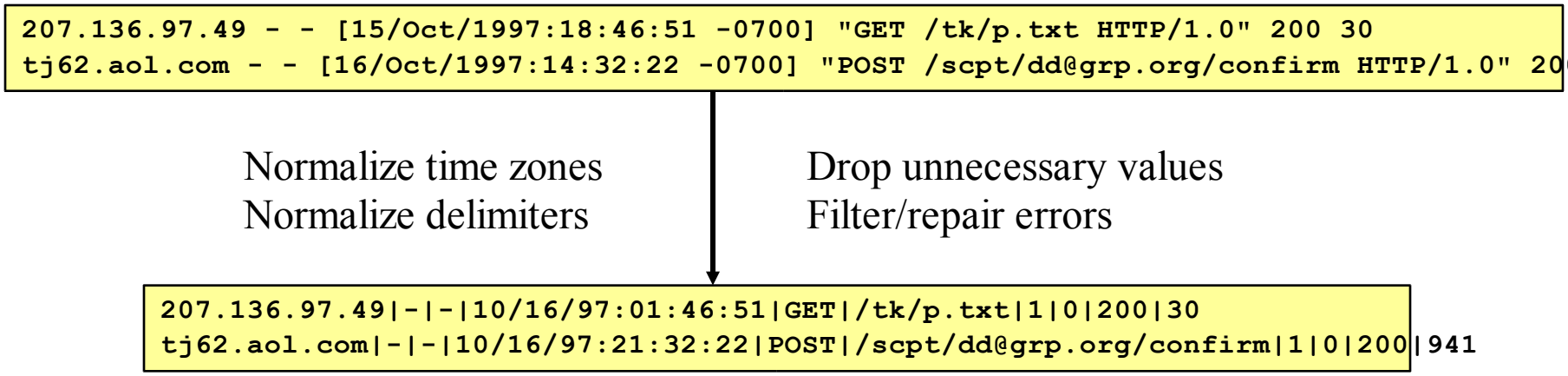
```
<top>.length : uint32
good: 53544   bad: 3824    pcnt-bad: 6.666
min: 35  max: 248591  avg: 4090.234
top 10 values out of 1000 distinct values:
tracked 99.552% of values
 val:  3082 count:  1254  %-of-good:  2.342
 val:   170 count:  1148  %-of-good:  2.144
                . . .


. . . . . . . . . . . . . . . . . . . . .
 SUMMING    count:  9655  %-of-good: 18.032
```

Not all lengths were legal!

- Suggested by AT&T user to get "bird's eye" view of their 4000 daily feeds.
- Used at AT&T for vetting data (and for debugging PADS descriptions).

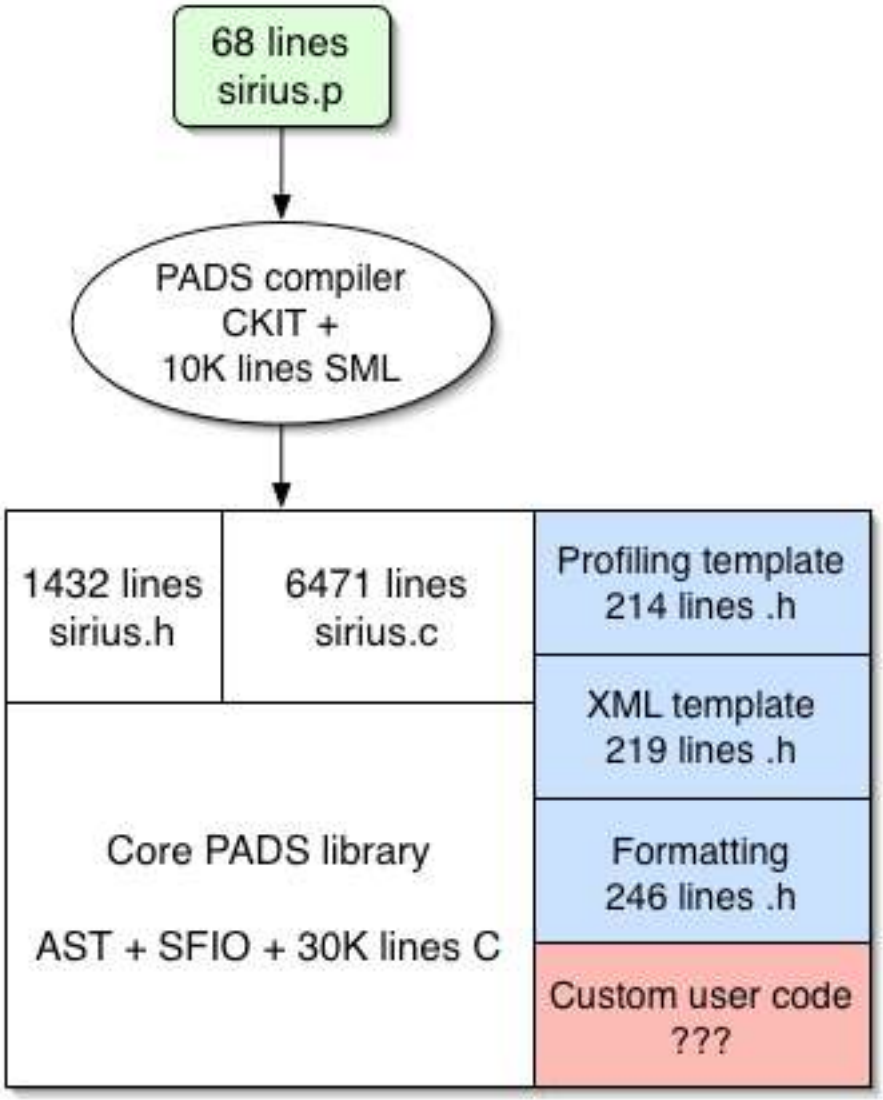
Pretty printer

- Customizable program to reformat data:



- Users can override pretty printing on a per type basis.
- Used by AT&T's Regulus project to normalize monitoring data before loading into a relational database.

Implementation



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.
- **Vision**: on-line repositories of data use PADS to describe data. Users can download data and generate tool suite as desired.

Observations on the language

- Types are a very useful way of thinking about semi-structured data.
  - Recursive types, pointers, and type functions are all in the works.
- Data description needs dependencies rather than non-determinism:
  - Length of an array, branch of a union, semantic checks.
- Record boundaries and literals provide good error recovery.

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 source?
- Implementation
  - How can we specialize generated libraries to incorporate application-specific information?
  - How can we optimize streaming XQueries?
- Theory
  - How do we precisely specify the semantics of PADS? (…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?

The (growing!) PADS team

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

Try it!

- Available for download with a non-commercial use source code license.
- Demo of accumulators, format program, and XML conversion.
- Send us feedback!

`www.padsproj.org`