The Next 700 Data Description Languages

Kathleen Fisher
AT&T Labs Research

Yitzhak Mandelbaum, David Walker
Princeton

Ad hoc data from www.geneontology.org

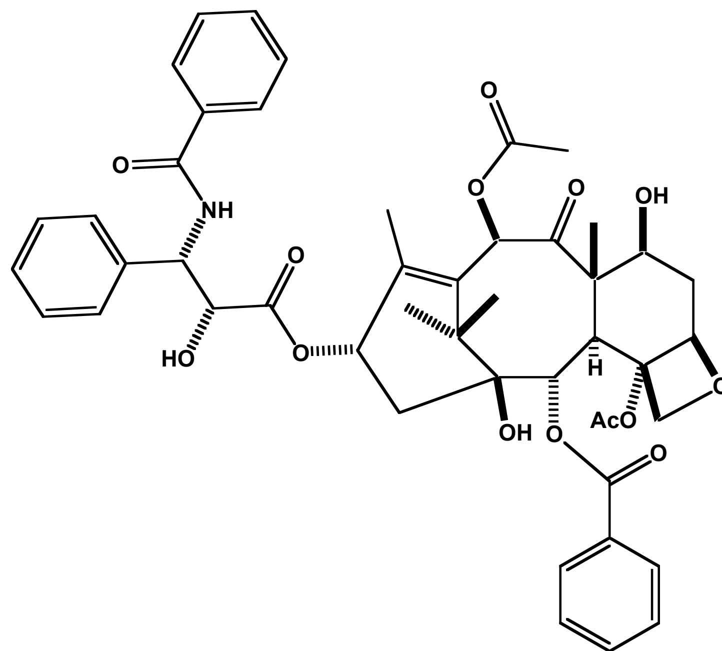
```
!date: Fri Mar 18 21:00:28 PST 2005
!version: $Revision: 3.223 $
!type: % is_a is a
!type: < part_of part of
!type: ^ inverse_of inverse of
!type: | disjoint_from disjoint from $Gene_Ontology ;
GO:0003673 <biological_process ;
GO:0008150 %behavior ;
GO:0007610 ; synonym:behaviour %adult behavior ;
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._G
orilla:0.17147,
(P._paniscus:0.19268,H._sapiens:0.11927):0.08386):0.061
24):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

Data provided by William O'Neil + Co., Inc. © 2005. All Rights Reserved.
Investor's Business Daily is a registered trademark of Investor's Business Daily, Inc.
Reproduction or redistribution other than for personal use is prohibited.
All prices are delayed at least 20 minutes.

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

Review: Technical Challenges of Ad Hoc Data

- 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.

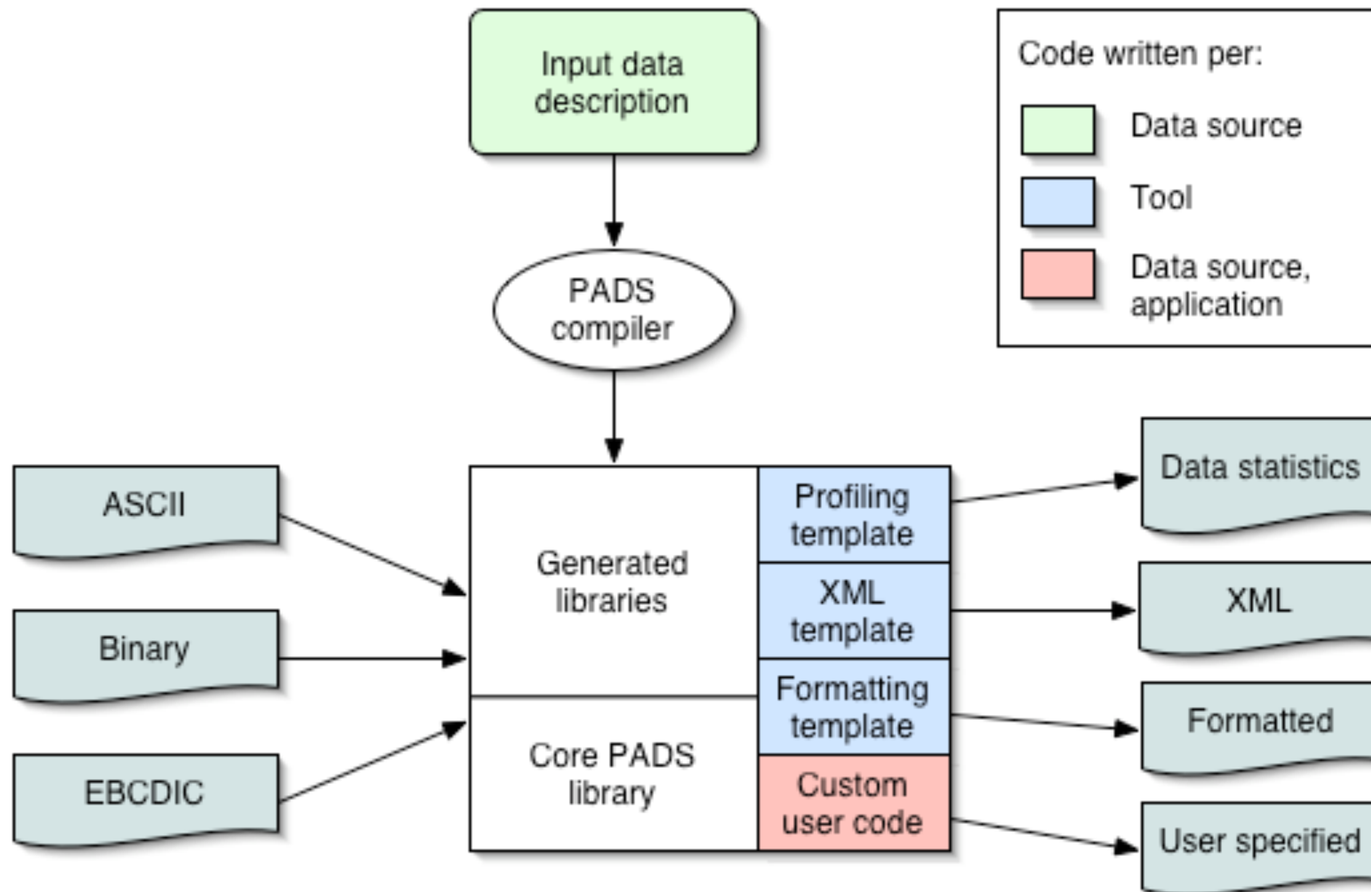
Common Log Format (CLF) in PADS

```
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;

    '['; Pdate(:']':)   date;
   "] "; request_t      request;
    ' '; response_t     response;
    ' '; length_t       length;
};

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

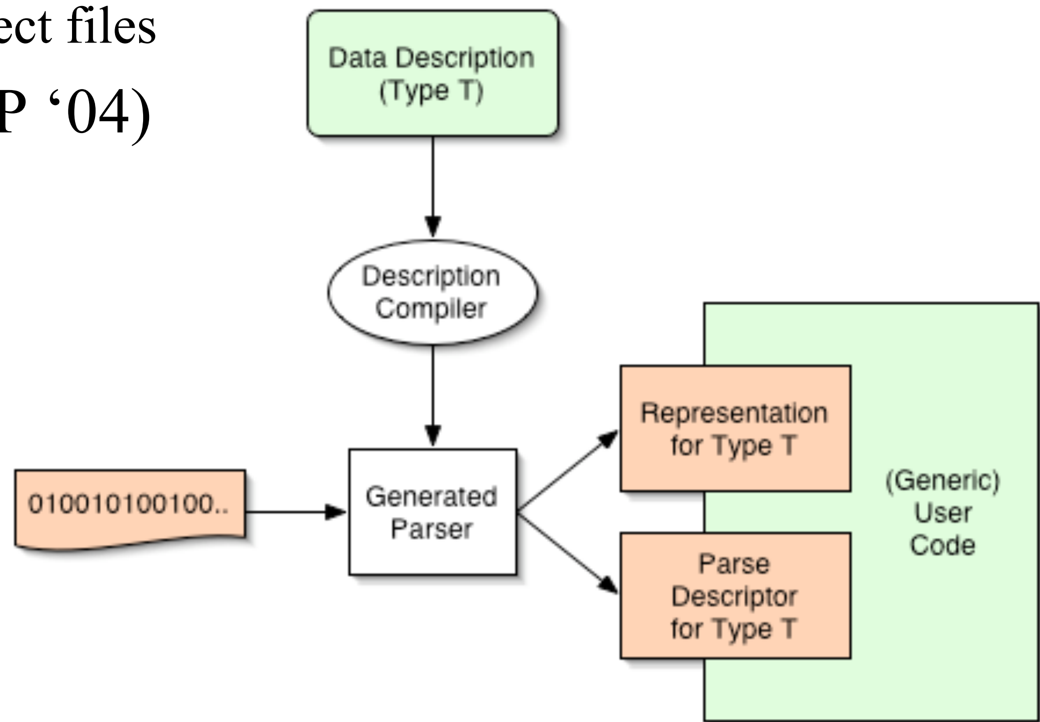
207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

PADS architecture



Many Data Description Languages

- PacketTypes (SIGCOMM ‘00)
  - Packet processing
- DataScript (GPCE ‘02)
  - Java jar files, ELF object files
- Erlang Binaries (ESOP ‘04)
  - Packet processing
- PADS (PLDI ‘05)
  - General ad hoc data



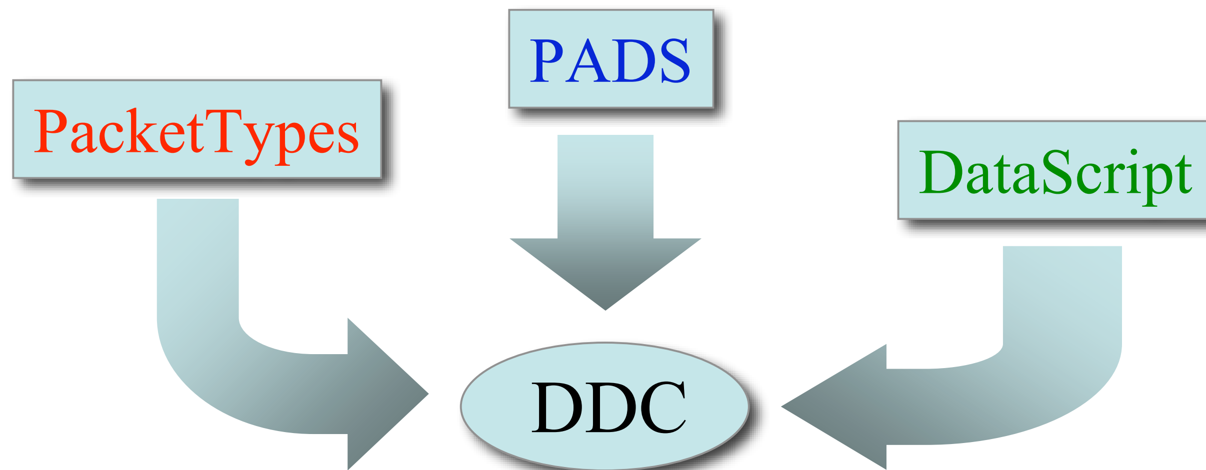
The Next 700 Data Description Languages

- What is the family of data description languages?
- How do existing languages relate to each other?
- What differences are crucial, which “accidents of history”?
- What do the existing languages mean, precisely?
- How would we design the next one?

To answer these questions, we introduce a semantic framework for understanding data description languages.

Contributions

- Key idea: use types for physical & logical representation
- A core data description calculus (DDC)
  - Based on dependent type theory
  - Simple, orthogonal, composable types
  - Types transduce external data source to internal representation.
- Encodings of high-level DDLs in low-level DDC



Outline

- Introduction
- A Data Description “Calculus” (DDC)
  - Well-kinding judgment
  - Representation, parse descriptor, and parser generation
- Data description languages (DDLs)
  - Idealized PADS (IPADS)
  - Features from other DDLs.
- Applications of the semantics

Overview: DDC Primitives

- Base types parameterized by expressions (Pstring(:`|`:))
  - Type constructor constants
- Pair of fields with cascading scope (Pstruct)
  - Dependent sums
- Additional constraints (Ptypedef, Pwhere, field constraints).
  - Set types
- Alternatives (Punion, Popt)
  - Sums
- Open-ended sequences (Parray)
  - Some kind of list?
- User-defined parameterized types
  - Abstraction and application
- “Active types”: compute, absorb, and scanning
  - Built-in functions

Base Types and Sequences

- C(e): base type parameterized by expression e.
- $\Sigma x{:}\ \tau.\ \tau'$: dependent sum describes sequence of values.
  - Variable x gives name to first value in sequence.
  - Note syntactic sugar: $\tau * \tau'$ if x not in τ' .
- Examples:

"123hello|"	int * string('|') * char	(123, "hello", '|')
"3513"	Σ*width*:int_fw(1). int_fw(*width*)	(3,513)
":hello:"	Σ*term*:char.string(*term*) * char	(':',"hello",':')

Constraints

- $\{x{:}\ \tau \mid e\}$: set types add constraints to the type τ and express relationships between elements of the data.
- Examples:

‘a’	$\{c : \text{char} \mid c = \text{‘a’}\}$ *(abbrev: S(‘a’))*	inl ‘a’
“101”, “82”	$\{x : \text{int} \mid x > 100\}$	inl 101, inr 82
“43|105|67”	$\Sigma min{:}\text{int}.S(\text{‘\|’}) * \Sigma max{:}\{m{:}\text{int} \mid min \leq m\}.S(\text{‘\|’}) * \{mid{:}\text{int} \mid min \leq mid \,\&\, mid \leq max\}$	(43, inl ‘|’, inl 105, inl ‘|’, inl 67)

Unions and the Empty String

- $\tau + \tau'$: deterministic, exclusive or
  - try τ; on failure, try τ'.
- unit: matches the empty string.
- Examples:

"54", "n/a"	int + S_s("n/a")	inl 54, inr "n/a"
"2341", ""	int + unit	inl 2341, inr ()

Abstraction and Application

- Can parameterize types over values: $\lambda x.\ \tau$
- Correspondingly, can apply types to values: $\tau\ e$
- Example: IP address with terminator

none	λ*term*.int seq(S_c('.'); len 4, S_c(*term*))	*none*
"1.2.3.4|"	IP_addr '|' * S_c('|')	([1,2,3,4], inl '|')

Absorb, Compute and Scan

- Absorb, Compute and Scan are *active* types.
  - absorb(τ) : consume data from source; produce nothing.
  - compute(e:σ) : consume nothing; output result of computation e.
  - scan(τ) : scan data source for type τ.
- Examples:

"|"	absorb(S_c('|'))	()
"10|12"	Σ*width*:int.S_c('|') * Σ*length*:int. *area*:compute(*width* · *length*:int)	(10,12,120)
"^%$!&_|"	scan(S_c('|'))	(6,inl '|')

A data description calculus

$C(e)$	Atomic type parameterized by expression e
$\Sigma x{:}\ \tau.\ \tau'$	Field sequence with cascading scope
$\{x{:}\ \tau \mid e\}$	Adding constraints to existing descriptions
$\tau + \tau'$	Alternatives
$\tau\ \mathrm{seq}(\tau_s, e, \tau_t)$	Open ended sequences
$\lambda x.\ \tau$	Types parameterized by an expressions.
$\tau\ e$	Applications of parameterized types to expressions.
unit/bottom	Empty strings: ok/error
absorb, compute, scan	"Active types"

DDC Example: Idealized Web Server Log

S = $\lambda ch.\{c : \text{char} \mid c = ch\}$

authid_t = S(‘-’) + string(‘ ’)

response_t = $\lambda x.\{y : \text{int16_FW}(x) \mid 100 <= y \text{ and } y < 600\ \}$

entry_t =

Σ*client* : ip. S(‘ ’) *

Σ*remoteid* : authid_t. S(‘ ’) *

Σ*response* : response_t 3.

compute(getdomain *client* = “edu” : bool)

entry_t seq(S(‘\n’), λx.false, bottom)

```
124.207.15.27 - 234
12.24.20.8 kfisher 208
```

Semantics Overview

- Well formed DDC type: Γ |- τ : κ
- Representation for type τ : $[\tau]_{rep}$
- Parse descriptor for type τ : $[\tau]_{pd}$
- Parsing function for type τ : $[\tau]$
  - $[\tau]$: bits * offset → offset * $[\tau]_{rep}$ * $[\tau]_{pd}$

Type Kinding

- Kinding ensures types are well formed.

$$\frac{\Gamma \mid\!- \tau : \sigma \rightarrow \kappa \qquad \Gamma \mid\!- e : \sigma}{\Gamma \mid\!- \tau\ e : \kappa}$$

$$\frac{\Gamma \mid\!- \tau : \text{type} \qquad \Gamma \mid\!- \tau' : \text{type}}{\Gamma \mid\!- \tau + \tau' : \text{type}}$$

$$\frac{\Gamma \mid\!- \tau : \text{type} \qquad \Gamma, x{:}[\tau]_{rep} * [\tau]_{pd} \mid\!- e : \text{bool}}{\Gamma \mid\!- \{x{:}\ \tau \mid e\} : \text{type}}$$

Selected Representation Types

DDC	*Host Language*
$[C(e)]_{rep}$	$\mathfrak{I}(C)$ + noval
$[\Sigma x{:}\ \tau.\ \tau']_{rep}$	$[\tau]_{rep}$ * $[\tau']_{rep}$
$[\{x{:}\ \tau \mid e\}]_{rep}$	$[\tau]_{rep}$ + ($[\tau]_{rep}$ error)
$[\tau + \tau']_{rep}$	$[\tau]_{rep}$ + $[\tau']_{rep}$ + noval
$[\ \tau\ \mathrm{seq}(\tau_s, e, \tau_t)\ ]_{rep}$	int * ($[\tau]_{rep}$ seq)
$[\lambda x.\ \tau]_{rep}$, $[\tau\ e]_{rep}$	$[\tau]_{rep}$
$[\mathrm{unit}]_{rep}$	unit

no meaningful value found in data

"semantic" error

Note that we erase all dependencies.

Selected Parse Descriptor Types

DDC	*Host Language*
$[C(e)]_{pd}$	pd_hdr
$[\Sigma x{:}\ \tau.\ \tau']_{pd}$	pd_hdr * $[\tau]_{pd}$ * $[\tau']_{pd}$
$[\{x{:}\ \tau \mid e\}]_{pd}$	pd_hdr * $[\tau]_{pd}$
$[\tau + \tau']_{pd}$	pd_hdr * $([\tau]_{pd} + [\tau']_{pd})$
$[\ \tau\ \mathrm{seq}(\tau_s, e, \tau_t)\ ]_{pd}$	pd_hdr * int * int * ($[\tau]_{pd}$ seq)
$[\lambda x.\ \tau]_{pd}$, $[\tau\ e]_{pd}$	$[\tau]_{pd}$
$[\mathrm{unit}]_{pd}$	pd_hdr

pd_hdr =
int * errcode * span

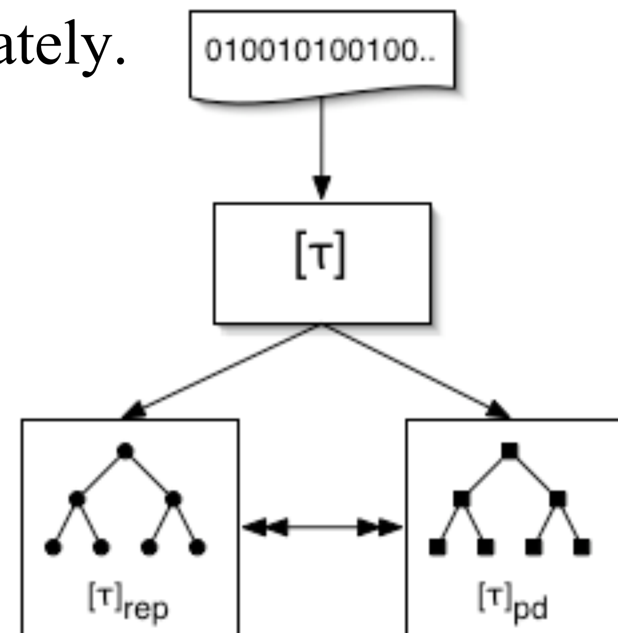
Parsing Semantics of Types

- Semantics expressed as parsing functions written in the polymorphic λ-calculus.
  - $[\tau]$: bits * offset $\rightarrow$ offset * $[\tau]_{rep}$ * $[\tau]_{PD}$
- Dependent sum case:

$$
\begin{aligned}
&[\Sigma x\!: \tau_1 .\, \tau_2] = \\
&\quad \lambda(B, \omega). \\
&\qquad \text{let } (\omega_1, r_1, p_1) = [\tau_1]\,(B, \omega) \text{ in} \\
&\qquad \text{let } x = (r_1, p_1) \text{ in} \\
&\qquad \text{let } (\omega_2, r_2, p_2) = [\tau_2]\,(B, \omega_1) \text{ in} \\
&\qquad\quad (\omega_2,\, R_\Sigma(r_1, r_2),\, P_\Sigma(p_1, p_2))
\end{aligned}
$$

Properties of the Calculus

- Theorem: If $\Gamma \vdash \tau : \kappa$ then
  - $\Gamma \vdash [\tau]$: bits * offset → offset * $[\tau]_{rep}$ * $[\tau]_{pd}$
    "*Well-formed type* τ *yields a parser that returns values with types corresponding to* τ."
- Theorem: Parsers report errors accurately.
  - Errors in parse descriptor correspond to errors in representation.
  - Parsers check all semantic constraints.

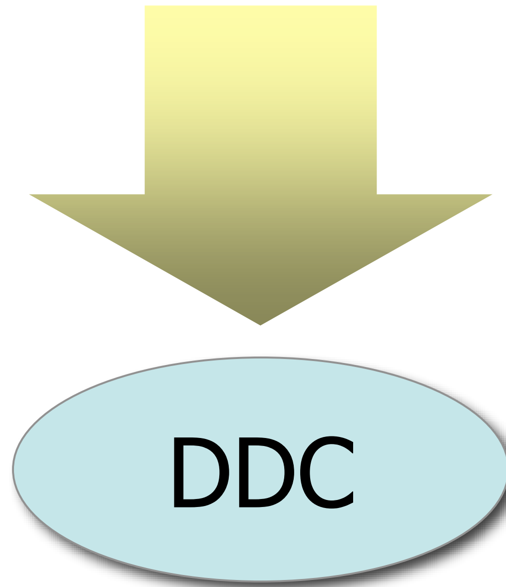


Making Use of the Calculus

IPADS

t ::= **C**(e) | **Pfun**(x:s) = t | t e
  | **Pstruct**{fields} | **Punion**{fields}
  | **Pswitch** e **of** {alts t_{def};} | **Popt** t
  | t **Pwhere** x.e | **Palt**{fields}
  | t **Parray** **[**t, t**]** | **Pcompute** e | **Plit** c

fields ::= | fields x : t;
alts ::= | alts e => t;



IPADS Example

```
authid_t = Punion { unauth : '-';   id  : Pstring(' ')};

response_t = Pfun(x:int)
                 Puint16_FW(x) Pwhere y.100 <= y and y < 600;

entry_t = Pstruct {
              client : Pip;                ' ';
              remoteid : authid_t;         ' ';
              response : response_t 3;
              academic :  Pcompute(getdomain client = "edu" : bool);
          };

entry_t Parray(Peor, Peof)
```

```
124.207.15.27 - 234
12.24.20.8 kfisher 208
```

Example: **Popt** and **Plit**

unit
$\tau_1 + \tau_2$

$$\frac{t \Rightarrow \tau}{\textbf{Popt}\ t \Rightarrow \tau + \text{unit}}$$

C(e)
$\{x{:}\tau \mid e\}$
absorb(τ)
scan(τ)

$$\frac{c : \text{char}}{\textbf{Plit}\ c \Rightarrow \text{scan}(\text{absorb}(\{x{:}\text{char} \mid x = c\}))}$$

Other Features

- **IPADS**: all features in grammar.
- **PacketTypes**: arrays, where clauses, structures, overlays, and alternation.
- **DataScript**: set types (enumerations and bitmask sets), arrays, constraints, value-parameterized types, and (monotonically increasing labels).

Other Uses of the Semantics

- Bug hunting!
  - Non-termination of array parsing if no progress made.
  - Inconsistent parse descriptor construction.
- Principled extensions
  - Adding recursion (done)
  - Adding polymorphism?
- Distinguishing the essential from the accidental
  - Highlights places where PADS sacrifices safety.
  - Pomit and Pcompute : much more useful than originally thought
  - Punion : what if correct branch has an error?

Future work

- What are the set of languages recognized by the DDC?
- How does the expressive power of the DDC relate to CFGs?
  - DDC can express some things that CFGs cannot: $a^n b^n c^n$
  - But CFGs support non-determinism, so can they say more?
- Add polymorphism to DDC and PADS.
- Relative termination of parsing.
- Basis for PADS bindings for other languages?

Summary

- Data description languages are well-suited to describing ad hoc data.
- No one DDL will ever be right - different domains and applications will demand different languages with differing levels of expressiveness and abstraction.
- Our work defines the first semantics for data description languages.
- For more information, visit www.padsproj.org.

Array Features

- What features do we need to handle data sequences?
  - Elements
  - Separator between elements
  - Termination condition ("Are we done yet?")
  - Terminator *after* sequence
- Examples:

  "192.168.1.1"

  "Harry|Ron|Hermione|Ginny;"

Bottom and Arrays

- τ seq(τ_s, e, τ_t) specifies:
  - Element type τ
  - Separator types τ_s.
  - Termination condition e.
  - Terminator type τ_t.
- bottom: reads nothing, flagging an error.
- Example: IP address.

"192.168.1.1"	int seq(S('.'), len? 4, bottom)	[192,168,1,1]