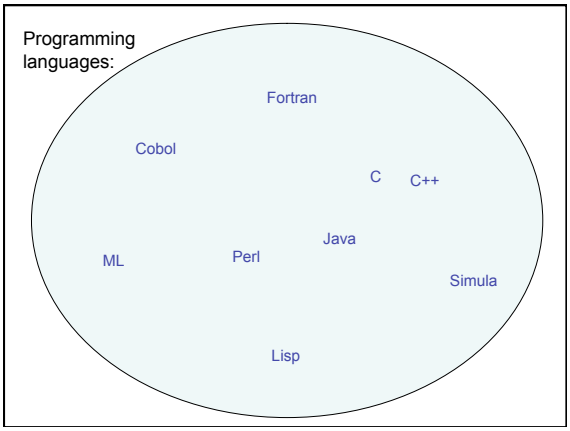
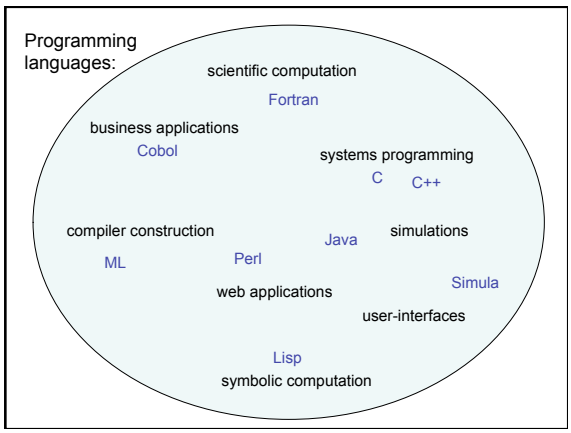
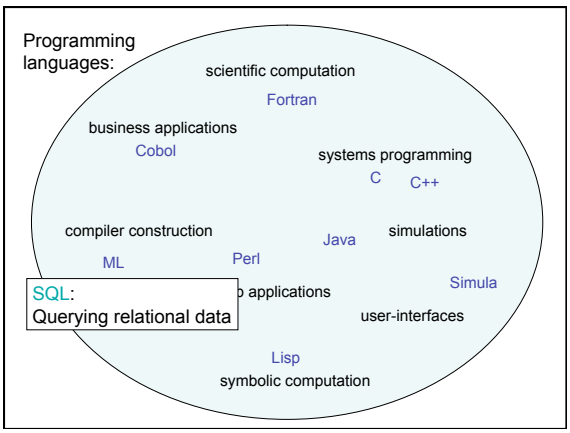
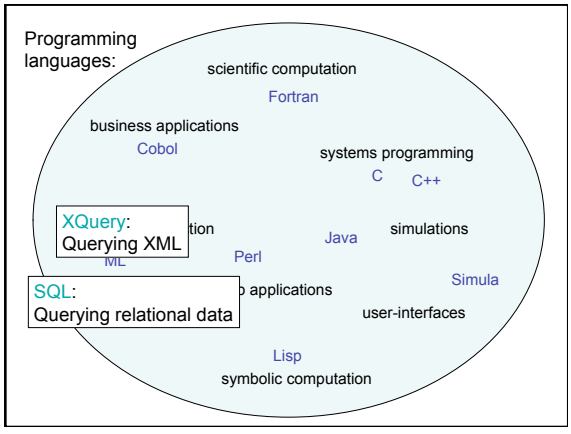
Domain-Specific Languages

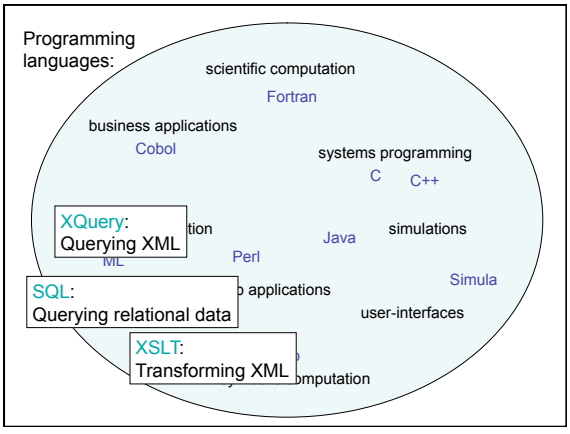
Kathleen Fisher

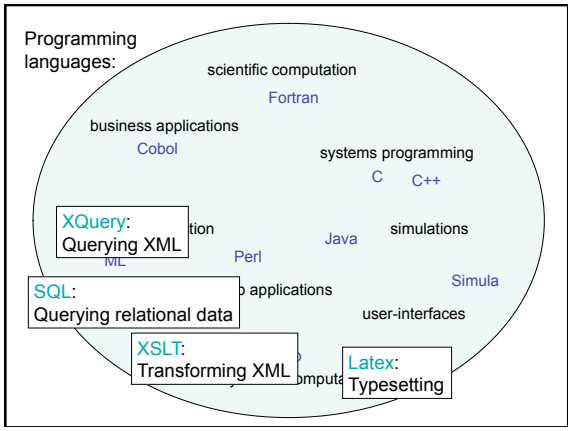


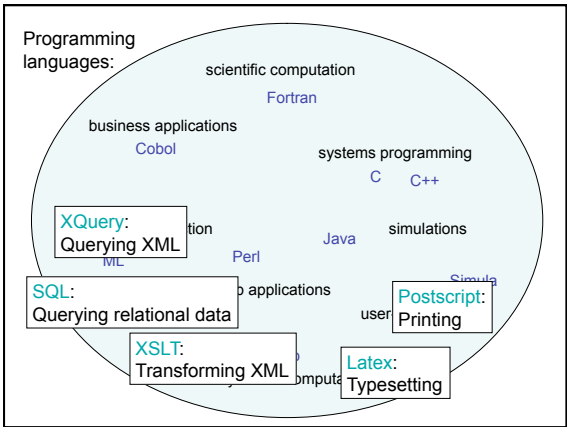


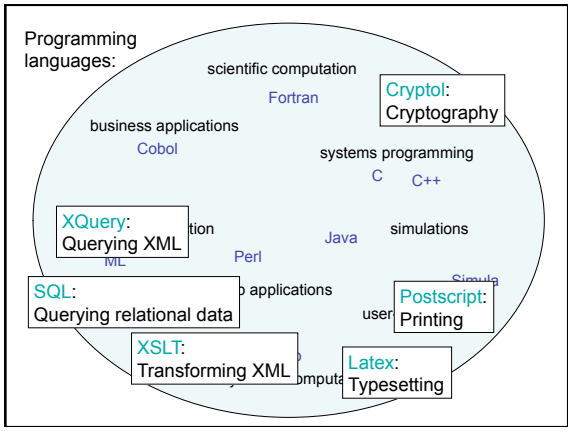


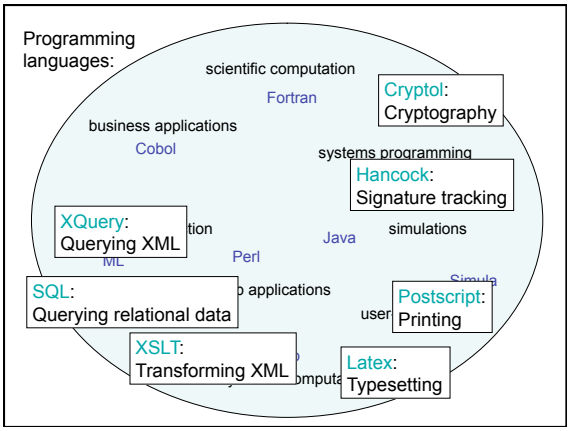


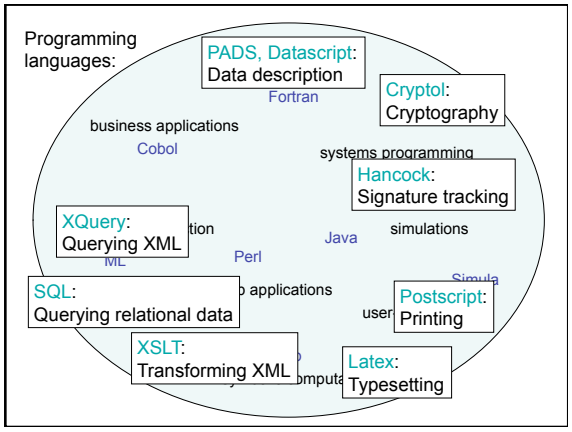


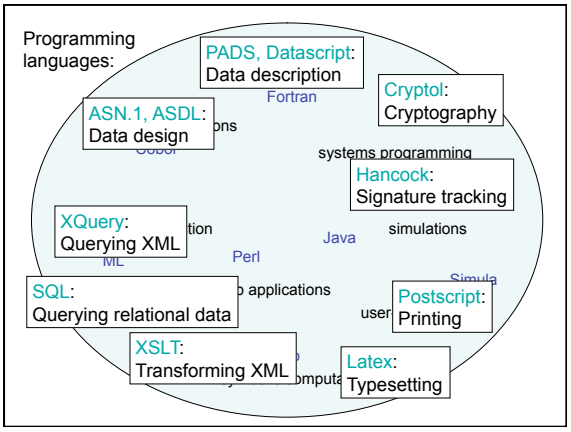

Programming languages:
scientific computation
Fortran
business applications
Cobol
systems programming
C
C++
XQuery: Querying XML
Java
simulations
Perl
SQL: Querying relational data
user-interfaces
XSLT: Transforming XML
Latex: Typesetting

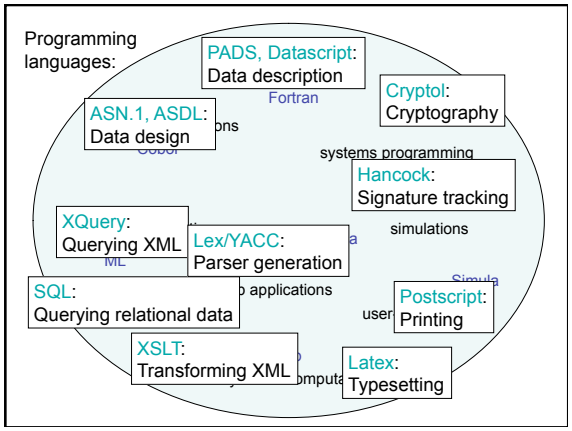

Programming languages:
scientific computation
Fortran
business applications
Cobol
systems programming
C
C++
XQuery: Querying XML
Java
simulations
Perl
SQL: Querying relational data
Postscript: Printing
XSLT: Transforming XML
Latex: Typesetting

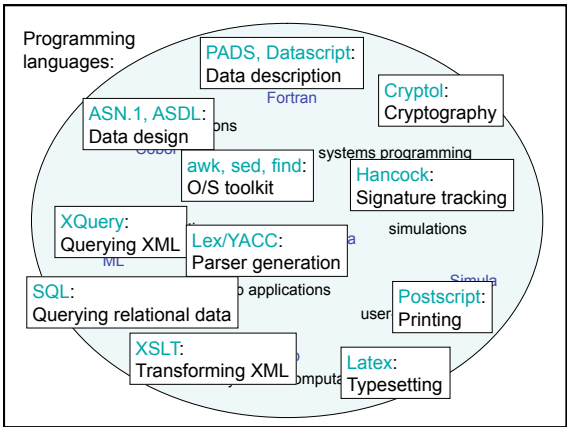

Programming languages:
scientific computation
Fortran
Cryptol: Cryptography
business applications
Cobol
systems programming
C
C++
XQuery: Querying XML
Java
simulations
Perl
SQL: Querying relational data
Postscript: Printing
XSLT: Transforming XML
Latex: Typesetting

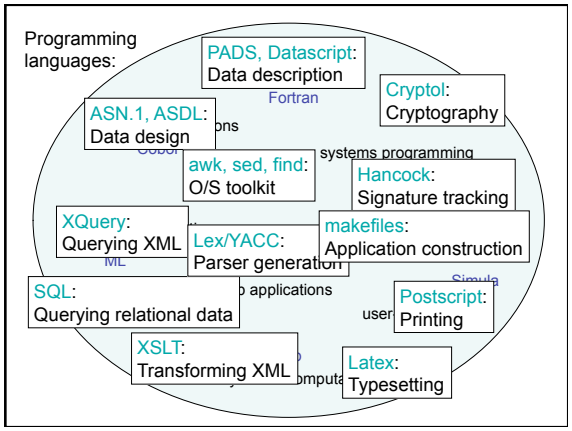

Programming languages:
scientific computation
Fortran
Cryptol: Cryptography
business applications
Cobol
systems programming
Hancock: Signature tracking
XQuery: Querying XML
Java
simulations
Perl
SQL: Querying relational data
Postscript: Printing
XSLT: Transforming XML
Latex: Typesetting

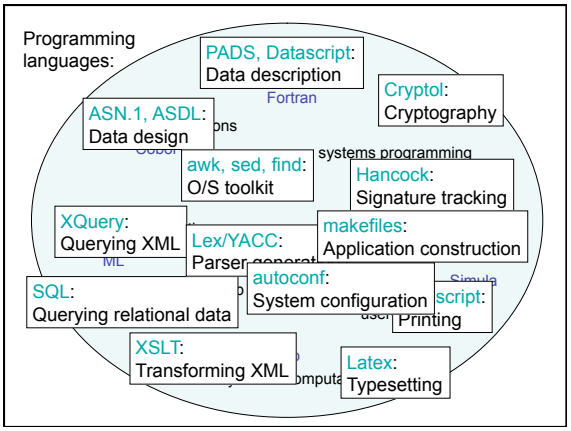

Programming languages:
PADS, Datascript: Data description
Fortran
Cryptol: Cryptography
business applications
Cobol
systems programming
Hancock: Signature tracking
XQuery: Querying XML
Java
simulations
Perl
SQL: Querying relational data
Postscript: Printing
XSLT: Transforming XML
Latex: Typesetting

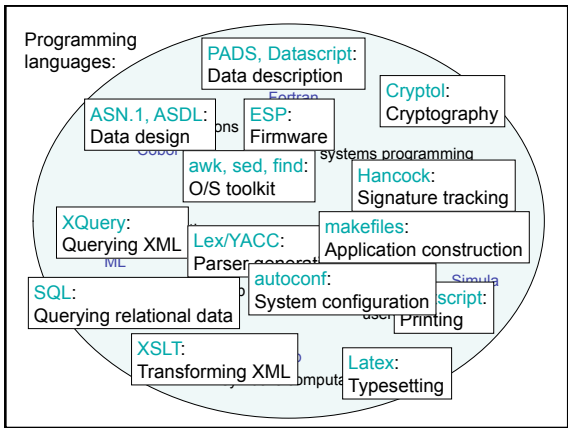

Programming languages:
PADS, Datascript: Data description
Fortran
Cryptol: Cryptography
ASN.1, ASDL: Data design
systems programming
Hancock: Signature tracking
XQuery: Querying XML
Java
simulations
Perl
SQL: Querying relational data
Postscript: Printing
XSLT: Transforming XML
Latex: Typesetting

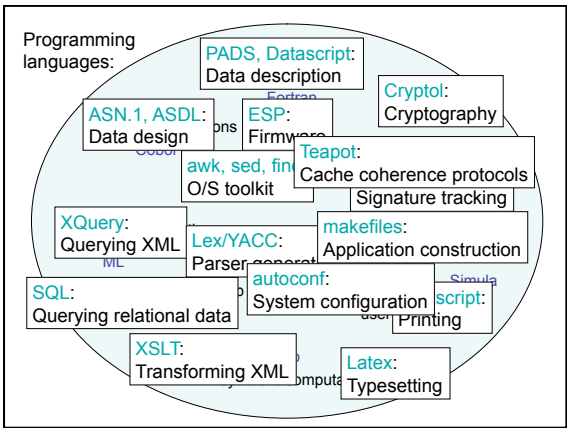

Programming languages:
PADS, Datascript: Data description
Fortran
Cryptol: Cryptography
ASN.1, ASDL: Data design
systems programming
Hancock: Signature tracking
XQuery: Querying XML
Lex/YACC: Parser generation
simulations
SQL: Querying relational data
applications
Postscript: Printing
user
XSLT: Transforming XML
Latex: Typesetting

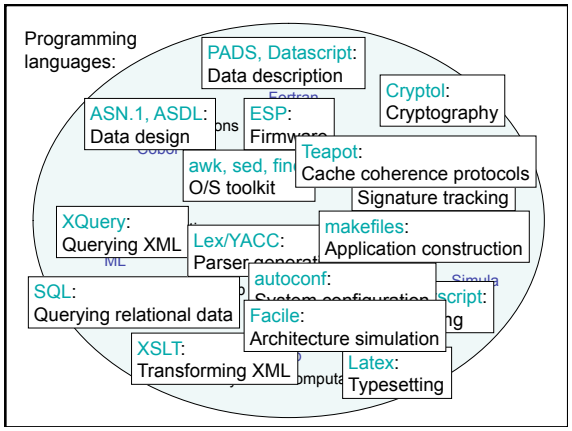

Programming languages:
PADS, Datascript: Data description
Fortran
Cryptol: Cryptography
ASN.1, ASDL: Data design
awk, sed, find: O/S toolkit
systems programming
Hancock: Signature tracking
XQuery: Querying XML
Lex/YACC: Parser generation
simulations
SQL: Querying relational data
applications
Postscript: Printing
user
XSLT: Transforming XML
Latex: Typesetting

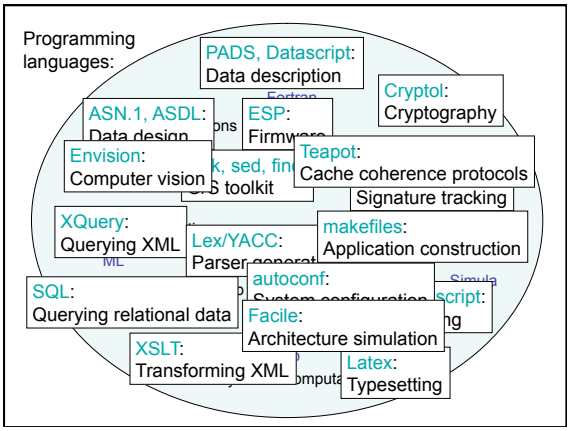

Programming languages:
PADS, Datascript: Data description
Fortran
Cryptol: Cryptography
ASN.1, ASDL: Data design
awk, sed, find: O/S toolkit
systems programming
Hancock: Signature tracking
XQuery: Querying XML
Lex/YACC: Parser generation
makefiles: Application construction
SQL: Querying relational data
applications
Postscript: Printing
user
XSLT: Transforming XML
Latex: Typesetting

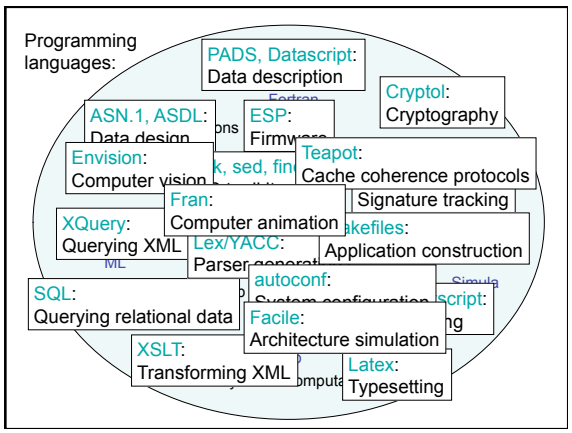

Programming languages:
PADS, Datascript: Data description
Fortran
Cryptol: Cryptography
ASN.1, ASDL: Data design
awk, sed, find: O/S toolkit
systems programming
Hancock: Signature tracking
XQuery: Querying XML
Lex/YACC:
makefiles: Application construction
autoconf: System configuration
SQL: Querying relational data
XSLT: Transforming XML
Latex: Typesetting

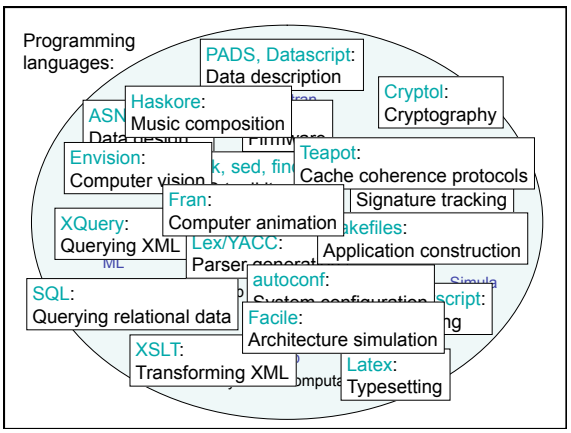

Programming languages:
PADS, Datascript: Data description
Cryptol: Cryptography
ASN.1, ASDL: Data design
ESP: Firmware
awk, sed, find: O/S toolkit
systems programming
Hancock: Signature tracking
XQuery: Querying XML
Lex/YACC:
makefiles: Application construction
autoconf: System configuration
SQL: Querying relational data
XSLT: Transforming XML
Latex: Typesetting

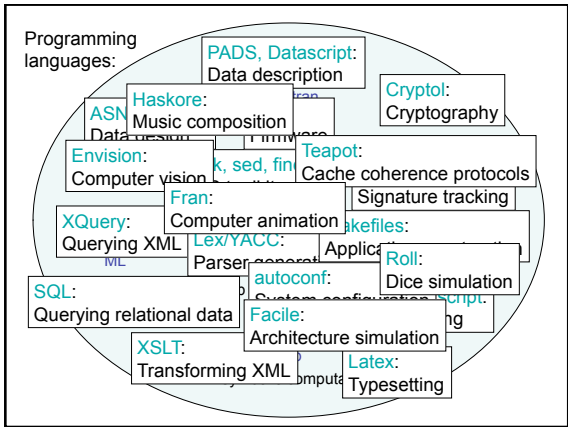

Programming languages:
PADS, Datascript: Data description
Cryptol: Cryptography
ASN.1, ASDL: Data design
ESP:
Teapot: Cache coherence protocols
Signature tracking
XQuery: Querying XML
Lex/YACC:
makefiles: Application construction
autoconf: System configuration
SQL: Querying relational data
XSLT: Transforming XML
Latex: Typesetting

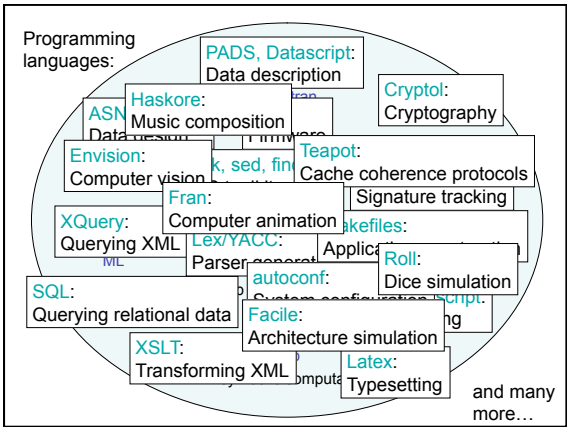

Programming languages:
PADS, Datascript: Data description
Cryptol: Cryptography
ASN.1, ASDL: Data design
ESP:
Teapot: Cache coherence protocols
awk, sed, fin
O/S toolkit
Signature tracking
XQuery: Querying XML
Lex/YACC:
makefiles: Application construction
autoconf:
SQL: Querying relational data
Facile: Architecture simulation
XSLT: Transforming XML
Latex: Typesetting


Programming languages:
PADS, Datascript: Data description
Cryptol: Cryptography
ESP:
Envision: Computer vision
Teapot: Cache coherence protocols
Signature tracking
XQuery: Querying XML
Lex/YACC:
makefiles: Application construction
autoconf:
SQL: Querying relational data
Facile: Architecture simulation
XSLT: Transforming XML
Latex: Typesetting


Programming languages:
PADS, Datascript: Data description
Cryptol: Cryptography
ESP:
Envision: Computer vision
Teapot: Cache coherence protocols
Fran: Computer animation
Signature tracking
XQuery: Querying XML
Lex/YACC:
makefiles: Application construction
autoconf:
SQL: Querying relational data
Facile: Architecture simulation
XSLT: Transforming XML
Latex: Typesetting


Programming languages:
PADS, Datascript: Data description
Haskore: Music composition
Cryptol: Cryptography
Envision: Computer vision
Teapot: Cache coherence protocols
Fran: Computer animation
Signature tracking
XQuery: Querying XML
Lex/YACC:
Application construction
autoconf:
SQL: Querying relational data
Facile: Architecture simulation
XSLT: Transforming XML
Latex: Typesetting


Programming languages:
PADS, Datascript: Data description
Haskore: Music composition
Cryptol: Cryptography
Envision: Computer vision
Teapot: Cache coherence protocols
Fran: Computer animation
Signature tracking
XQuery: Querying XML
Lex/YACC:
Roll: Dice simulation
autoconf:
SQL: Querying relational data
Facile: Architecture simulation
XSLT: Transforming XML
Latex: Typesetting


Programming languages:
PADS, Datascript: Data description
Haskore: Music composition
Cryptol: Cryptography
Envision: Computer vision
Teapot: Cache coherence protocols
Fran: Computer animation
Signature tracking
XQuery: Querying XML
Lex/YACC:
Roll: Dice simulation
autoconf:
SQL: Querying relational data
Facile: Architecture simulation
XSLT: Transforming XML
Latex: Typesetting
and many more…

Why DSLs?

- Why a language at all?
  - Languages supply a rich interface to computers.

vs

  - Languages directly provide a model of the computational domain.

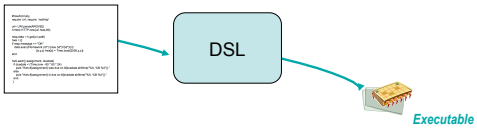
Tailored abstractions

- Increase accessibility for domain experts
- Improve reliability
  - Programs are shorter.
  - Compiler generates tedious boilerplate code.
- Allow programs to serve as "living documentation"

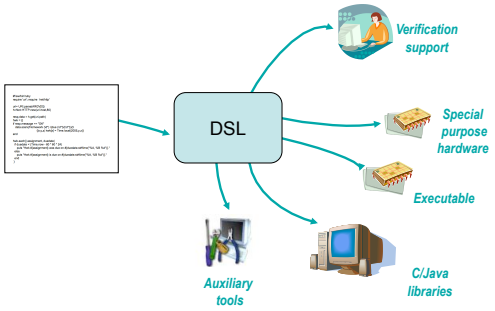
More for less

- Restricting expressiveness enables validation and optimization at domain-level.
  - SQL programs are guaranteed to terminate.
  - YACC specifications are guaranteed to compile into PDAs.
  - Cryptol programs are guaranteed to require only finite space.

Two for one specials



Two for one specials



Outline

- Introduction
  - Language domains
  - The case for domain specific languages
- Examples:
  - ESP, SQL
  - PADS
  - Cryptol
- Conclusion

ESP

- Language for programming device firmware
- Computational model:
  - Event-driven state-machines (based on CSP)
  - Much easier to express in ESP than when coded in C (Code is an order of magnitude smaller).
- Compiler generates:
  - C code to compile to produce firmware
  - SPIN input to model check program for concurrency and memory errors.

Teapot is a similar DSL
for writing cache coherence protocols

SQL

Language for querying relational data bases.

Students

ID	NAME
01	Harry Potter
02	Hermione Granger
03	Ronald Weasley

Potions

ID	GRADE
01	Satisfactory
02	Outstanding
03	Satisfactory

```
SELECT Students.NAME,
       Potions.GRADE
FROM Students, Potions
WHERE Students.ID = Potions.ID
```

NAME	GRADE
Harry Potter	Satisfactory
Hermione Granger	Outstanding
Ronald Weasley	Satisfactory

SQL

- SQL compiles into relational algebra with select, project, and join logical operators.
- Query engine chooses corresponding physical operators based on indices and other statistics about the data.
- Years of research have gone into the best query plan selection and join algorithms.
- Data analyst can be blissfully ignorant of details under the covers.

PADS

- Data description language in development at AT&T, Princeton, and University of Michigan.
- More information:

http://www.padsproj.org

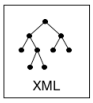
Disclaimer: This is my project.

Data, data, everywhere!

Incredible amounts of data stored in well-behaved formats:

Databases:

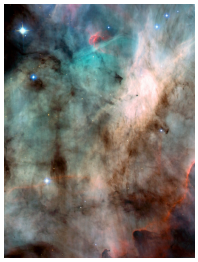
XML:



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

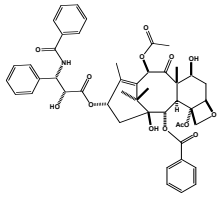
```
!date: Fri Mar 18 21:00:28 PST 2005
!version: $Revision: 3.223 $
!type: % is_a is a
!type: < part_of part of
!type: ^ inverse_of inverse of
!type: | disjoint_from disjoint from $Gene_Ontology ;
GO:0003673 <biological_process ;
GO:0008150 %behavior ;
GO:0007610 ; synonym:behaviour %adult behavior ;
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   Company                                  Price     Price     Volume    EPS      RS
Symbol  Name                          Price  Change  % Change  % Change  Rating  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

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.

Prior approaches

- Lex/Yacc
  - No one uses them for ad hoc data.
- Perl/C
  - Code brittle with respect to changes in input format.
  - Analysis 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.

PADS

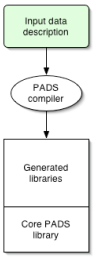
Data expert writes declarative description of data source:

- Physical format information
- Semantic constraints

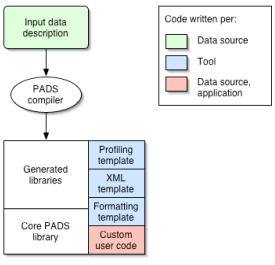
Many data consumers use description and generated parser.

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

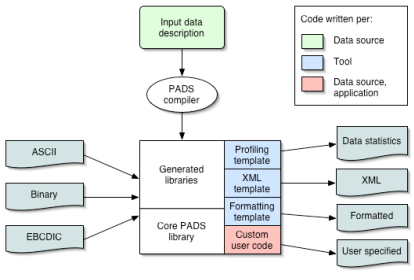
PADS architecture



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

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.

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

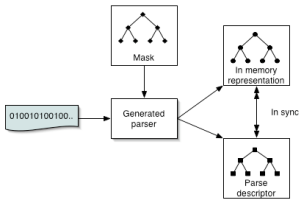
CLF in PADS

```
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 [];
}
```

PADS parsing

```
Perror_t entry_t_read(P_t *pdc, entry_t_m* mask,
                      entry_t_pd* pd, entry_t* rep);
```



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 the 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:

```
207.136.97.49 - - [15/Oct/1997:18:46:51 -0700] "GET /tk/p.txt HTTP/1.0" 200 30
tj62.aol.com - - [16/Oct/1997:14:32:22 -0700] "POST /scpt/dd@grp.org/confirm HTTP/1.0" 200 941
```

Normalize time zones
Normalize delimiters

Drop unnecessary values
Filter/repair errors

```
207.136.97.49|-|-|10/16/97:01:46:51|GET|/tk/p.txt|1|0|200|30
tj62.aol.com|-|-|10/16/97:21:32:22|POST|/scpt/dd@grp.org/confirm|1|0|200|941
```

- 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.

Why a DSL?

- Dramatically shorter code (68 versus ~7.9K lines).
- Description is short enough to serve as documentation.
- **Safer**: error code inserted automatically and completely (as long as the is compiler right…).
- **Leverage**: produce value-added tools.

Cryptol

- A language, developed at Galois, for expressing cryptographic algorithms.
- More information:

http://www.cryptol.net

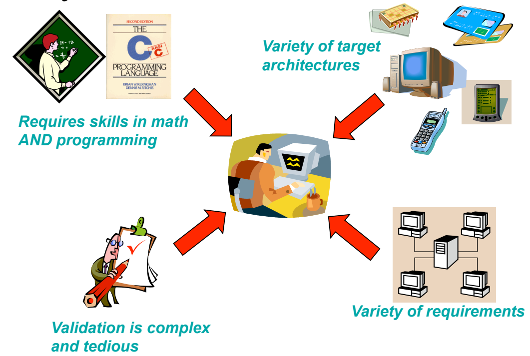
Cryptol
The Language of Cryptography

Thanks to Galois for contributing material for these slides.

Crypto-algorithm correctness

- Verification and validation a critical piece of crypto-modernization programs.
- Exploding complexity and requirements
  - Number of algorithms, hardware platforms
  - High assurance requirements
- 25% of algorithms submitted for FIPS validation had flaws (according to Director of NIST CMVP, 2002)

Why is this hard?



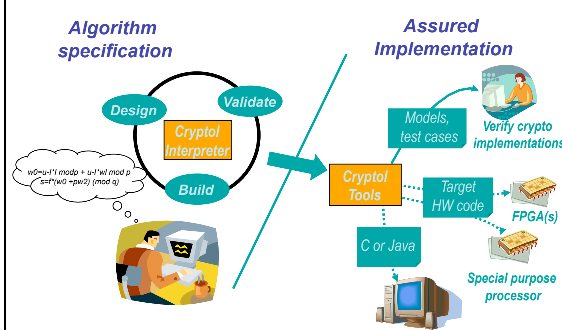
Lack of clear reference implementations

$$x_i = \lfloor X/2^{8i} \rfloor \bmod 2^8 \qquad i = 0,\ldots,3$$
$$y_i = s_i[x_i] \qquad i = 0,\ldots,3$$
$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} \cdot & \cdots & \cdot \\ \vdots & \mathrm{MDS} & \vdots \\ \cdot & \cdots & \cdot \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$
$$Z = \sum_{i=0}^{3} z_i \cdot 2^{8i}$$

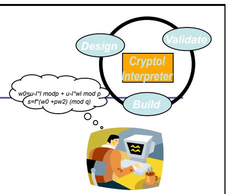
```
#define        MDS_GF_FDBK    0x169
#define        LFSR1(x) ( ((x) >> 1)  ^ (((x) &
MDS_GF_FDBK/2 : 0))
               LFSR2(x) ( ((x) >> 2)  ^ (((x) &

                   MDS_GF_FDBK/2 :
                        ^ (((x) & 0x01) ?
                   MDS_GF_FDBK/4 : 0))
               Mx_1(x) ((DWORD)  (x))
               Mx_X(x) ((DWORD) ((x) ^ LFSR2(x)))
               Mx_Y(x) ((DWORD) ((x) ^ LFSR1(x) ^
 LFSR2(x)))
#define        M00            Mul_1
#define        M01            Mul_Y
return  ((M00(b[0]) ^ M01(b[1]) ^
        ((M20(b[0]) ^ M21(b[1]) ^
         M22(b[2])^ M23(b[3])) <<16)^
        ((M30(b[0]) ^ M31(b[1]) ^
```

It's hard to relate implementations to the underlying math

One Specification - Many Uses



Algorithm specification




```
rc6ks : {a} (w >= width a) =>
        [a][8] -> [r+2][2][w];
rc6ks key = split (rs >>> (v - 3 * nk))
  where {
    c = max (1, (width key + 3) / (w / 8));
    v = 3 * max (c, nk);
    initS =  [pw (pw+qw) ..]@@[0 .. (nk-1)];
    padKey : [4*c][8];
    padKey = key # zero;
    initL : [c][w];
    initL = split (join padKey);
    ss = [| (s+a+b) <<< 3
          || s <- initS # ss
          || a <- [0] # ss
          || b <- [0] # ls |];
    ls = [| (l+a+b) <<< (a+b)
          || l <- initL # ls
          || a <- ss
          || b <- [0] # ls |];
    rs = ss @@ [(v-nk) .. (v-1)];
  };
```

- Models crypto-algorithm
- Natural expression
- Clear and unambiguous
- Structure and guide an implementation

Key ideas in Cryptol

- Domain-specific data and control abstractions
  - Sequences
  - Recurrence relations (not for-loops)
- Powerful data transformations
  - Data may be viewed in many ways
  - Machine independent
- Flexible sizes
  - Algorithms parameterized on size
    - Size constraints are explicit in many specs
    - Number of iterations may depend on size
  - A Size-Type system captures and maintains size constraints

Choosing what to leave out is critical

Cryptol programs

- File of mathematical definitions
  - Two kinds of definitions: values and functions
  - Definitions may be accompanied by a type
- Definitions are computationally neutral
  - Cryptol tools provide the computational content (interpreters, compilers, code generators, verifiers)

```
x : [4][32];
x = [23 13 1 0];

F : ([16],[16]) -> [16];
F (x,x') = 2 * x + x';
```

Data types

- Homogeneous sequences

```
[False True False True False False True]
[[1 2 3 4] [5 6 7 8]]
```

- Numbers are represented as sequences of bits
  - Aka "words"
  - Decimal, octal (`0o`), hex (`0x`), binary (`0b`)

```
123, 0xF4, 0b11110100
```

- Quoted strings are just syntactic sugar for sequences of 8-bit words

```
"abc" = [0x61 0x62 0x63]
```

- Heterogenous data can be grouped together into *tuples*

```
(13, "hello", True)
```

Standard operations

- Arithmetic operators
  - Result is modulo the word size of the arguments
  - + - * / % **
- Boolean operators
  - From bits, to arbitrarily nested matrices of the same shape
  - & | ^ ~
- Comparison operators
  - Equality, order
  - == != < <= > >=
  - returns a Bit
- Conditional operator
  - Expression-level if-then-else
  - Like C's a?b:c

Sequences

- Sequence operators
  - Concatenation (#), indexing (@), size

```
[1..5] # [3 6 8]  =  [1 2 3 4 5 3 6 8]
[50 .. 99] @ 10  =  60
```

- Shifts and Rotations
  - Shifts (`<<, >>`), Rotations (`<<<, >>>`)

```
[0 1 2 3] <<  2  =  [2 3 0 0]
[0 1 2 3] <<< 2  =  [2 3 0 1]
```

Cryptol types

- Types express size and shape of data

```
[[0x1FE 0x11] [0x132 0x183]
 [0x1B4 0x5C] [ 0x26  0x7A]]   has type   [4][2][9]
```

- Strong typing
  - The types provide mathematical guarantees on interfaces
- Type inference
  - Use type declarations for active documentation
  - All other types computed
- Parametric polymorphism
  - Express size parameterization of algorithms

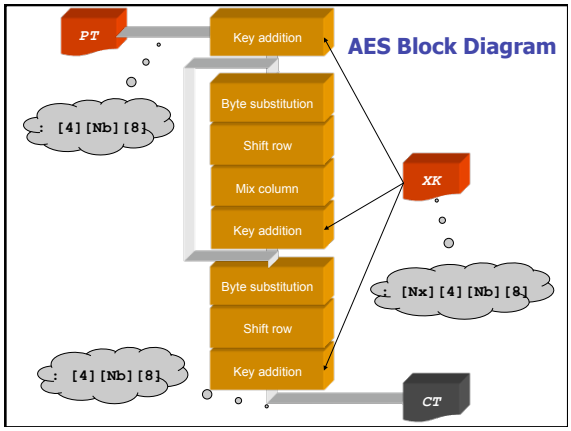
AES Types

- "The State can be pictured as a rectangular array of bytes. This array has four rows, the number of columns is denoted by Nb and is equal to the block length divided by 32."

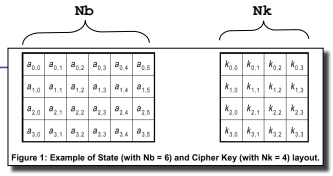
```
state : [4][Nb][8];
```

- "The input and output used by Rijndael at its external interface are considered to be one-dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4*Nb-1. The Cipher Key is considered to be a one-dimensional array of 8-bit bytes numbered upwards from 0 to the 4*Nk-1."

```
input : [4 * Nb][8];
key   : [4 * Nk][8];
```



AES API




```
keySchedule : [4*Nk][8] -> Xkey
encrypt : (Xkey,[4*Nb][8]) -> [4*Nb][8]
decrypt : (Xkey,[4*Nb][8]) -> [4*Nb][8]

Xkey = ([4][Nb][8],
        [max(Nb,Nk)+5][4][Nb][8],
        [4][Nb][8])
```

Splitting and Joining sequences

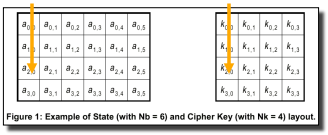
```
0x99FAC6F975BABB3E
        split
[0x99 0xFA 0xC6 0xF9 0x75 0xBA 0xBB 0x3E]
        join
0x99FAC6F975BABB3E
```

Polymorphic operation: use a type to resolve how many terms in the split list

Striping

2D sequences considered to be row major

```
stripe : [4*Nb][8] -> [4][Nb][8];
stripe(block) = transpose(split(block));

unstripe : [4][Nb][8] -> [4*Nb][8];
unstripe(state) = join(transpose(state));
```



AES encryption

```
encrypt : (Xkey,[4*Nb][8]) -> [4*Nb][8];

encrypt(XK,PT) = unstripe(Rounds(State,XK))
  where {
    State : [4][Nb][8];
    State = stripe(PT);
  };
```

Sequence Comprehensions

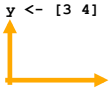
- The comprehension notion borrowed from set theory
  - **{ a+b | a ∈ A, b ∈ B}**
  - Adapted to sequences
- Applying an operation to each element

```
[| 2*x + 3  ||  x <- [1 2 3 4] |]
= [ 5 7 9 11 ]
```

Traversals

- Cartesian traversal

```
[| [x y] ||  x <- [0 1 2], y <- [3 4] |]
   = [[0 3] [0 4]
      [1 3] [1 4]
      [2 3] [2 4]]
```



- Parallel traversal

```
[| x + y  ||  x <- [1 2 3]
          ||  y <- [3 4 5 6 7] |]
  =  [4 6 8]
```

Row traversals in AES

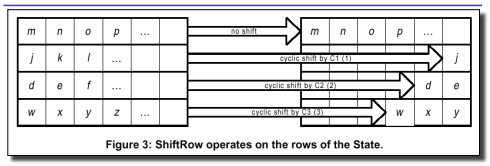


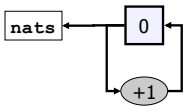
Figure 3: ShiftRow operates on the rows of the State.

```
ShiftRow : [4][Nb][8] -> [4][Nb][8];
ShiftRow(state)
  = [| row <<< i || row <- state
                 || i <- [0 1 2 3] |]
```

Recurrence

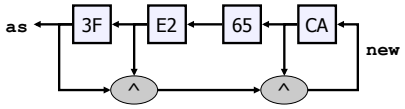
Textual description of shift circuits
- Follow mathematics: use stream-*equations*
- Stream-definitions can be *recursive*

```
nats = [0] # [| y+1 || y <- nats |];
```



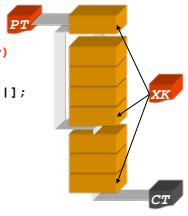
More complex stream equations

```
as  = [0x3F 0xE2 0x65 0xCA] # new;
new = [| a ^ b ^ c || a <- as
                   || b <- drop(1,as)
                   || c <- drop(3,as)|];
```

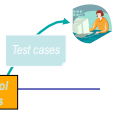


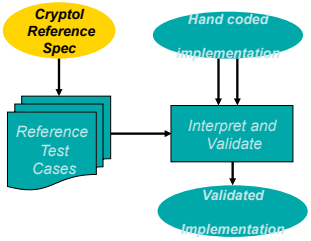
AES rounds

```
Rounds(State,(initialKey,rndKeys,finalKey)) = final
 where {
   istate = State ^ initialKey;
   rnds = [istate] # [| Round(state,key)
                      || state <- rnds
                      || key <- rndKeys |];
   final = FinalRound(last(rnds),
                      finalKey);
};
```



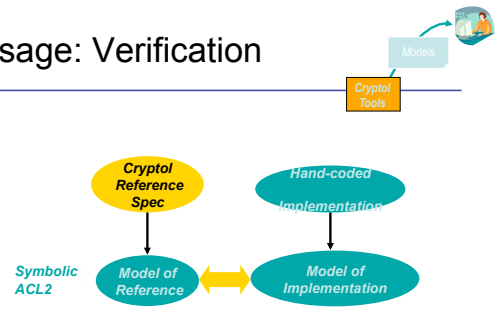
Usage: Testing



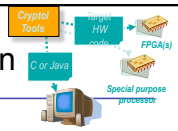


- Generates "known good tests"
- Built-in capture of intermediate vectors simplifies debugging
- Easy to generate new intermediate vectors as needed

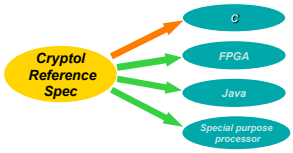
Usage: Verification



Usage: Code Generation



A single correct, executable Cryptol specification can be deployed to a variety of target platforms…



- **One specification to 'get right'**
- **Many targets for use**

Key Observation

- Sequences are descriptions only
- Implementation of sequences can be:
  - Laid out in time
    - Loops and/or state machines
  - Laid out in space
    - Parallel and/or pipeline
  - Or a mixture of both
    - The mathematical specification is the same

> Henceforth space by itself, and time by itself, are doomed to fade away into mere shadows, and only a kind of union of the two will preserve an independent reality.
> *Minkowski, Space and Time, Sept. 21, 1908*

Sequentialization in Cryptol comes only from data-dependency — just like hardware

Ideal for reference implementations

- Domain Specific
  - Naturally understandable to developers
  - Simplifies expression, inspection, reuse
- Executable
  - Run tests and debug for correctness
  - Generate test cases
- Declarative
  - Not implementation-specific, concise
  - Multiple uses – test, generation, model building, etc.
  - Highly retargetable to any architecture
- Unambiguous
  - Formal basis
  - Precise syntax and semantics
  - Independent of underlying machine models

Outline

- Introduction
  - Language domains
  - The case for domain specific languages
- Examples:
  - ESP, SQL
  - PADS
  - Cryptol
- Conclusion

Tailored abstractions

- Accessible to domain experts
  - Cryptol: cryptographers
  - SQL: data analysts
- Program reliability (code size reduction)
  - PADS generates error detection code
  - ESP generates state machine context switching
- Living documentation
  - Cryptol implementations as reference specifications
  - PADS descriptions document ad hoc data formats

More for less

- Cryptol leaves out
  - recursion to support compilation in finite space.
  - imperative variables to support mathematical reasoning.
- ESP leaves out recursive data structures and buffered channels to facilitate model checking.
- SQL and YACC restrict control flow to ensure efficient compilation.

Two for one specials

- Specify once, reap multiple rewards
  - Cryptol: reference implementation, testing support, theorem proving support, implementations for special purpose hardware.
  - PADS: parser, pretty printer, statistical profiler, formatting tool, integration with Xquery.
  - ESP: firmware code and model checking input
  - Teapot: cache coherency protocol implementation and model checking input.
  - Roll: generates dice rolls and probability distributions.

Why not libraries?

- Some DSLs are in fact libraries
  - Example: Haskore, a language for composing music.
  - Fits best in languages with good control-flow abstractions and overloading mechanisms, eg, Haskell and C++.
- But:
  - Complex libraries can be hard to use.
  - Type checking only at host language level.
  - More difficult to leverage domain knowledge or to generate more than one artifact.

Disadvantages of DSLs

- Users have to learn a new language.
- Implementation and maintenance of DSL are daunting, particularly for narrowly focused domain.
- Tool support can be lacking:
  - Debuggers, profilers, interactive development environments, …

Summary

- All languages have a domain.
- Languages provide a rich interface to computers.
- Tailored abstractions are powerful:
  - support domain experts
  - make code more reliable (shorter: boring code is generated)
- Less is more
  - Extra reasoning principles & more optimization opportunities
- Two for one specials
  - Executable, verification support, auxiliary tools, …
- Languages are constantly being designed, implemented, and used.