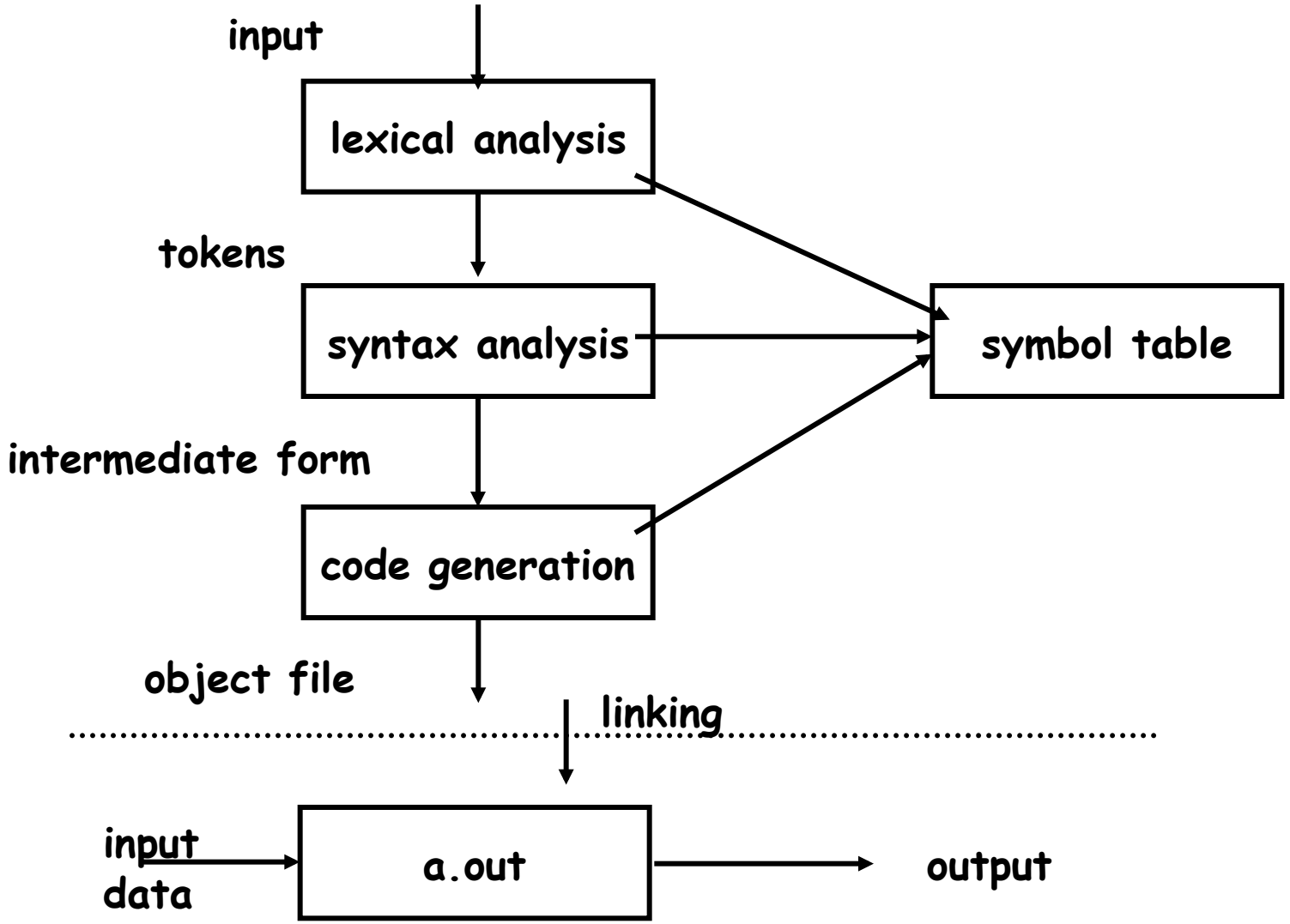
Little languages

- **also called specialized, application-specific, domain-specific, ...**
- **focused on a single area, not trying to be general purpose**
- **often declarative (though not always)**
  - some are Turing complete, many are not
- **examples**
  - regular expressions, shell, AWK, XML, AMPL, …
- **definition is fuzzy**

Anatomy of a compiler



YACC and LEX

- **languages/tools for building [parts of] compilers and interpreters**

- **YACC: "yet another compiler compiler" (Steve Johnson, ~1972)**
  - converts a grammar and semantic actions into a parser for that grammar

- **LEX: lexical analyzer generator (Mike Lesk, ~1974)**
  - converts regular expressions for tokens into a lexical analyzer that recognizes those tokens

- **parser calls lexer each time it needs another input token**
- **lexer returns a token and its lexical type**

- **when to think of using them:**
  - real grammatical structures (e.g., recursively defined)
  - complicated lexical structures
  - rapid development time is important
  - language design might change

YACC overview

- **YACC converts grammar rules & semantic actions into parsing fcn yyparse()**
  - yyparse parses programs written in that grammar, performs semantic actions as grammatical constructs are recognized

- **semantic actions usually build a parse tree**
  - each node represents a particular syntactic type, children are components

- **actions could anything**
  - run the program directly
  - interpret directly from the tree
    at each node, interpret children (recursion), do operation of node itself, return result
  - generate byte code output to run elsewhere
  - generate internal byte code
  - generate some other language to be processed later

Grammar specified in YACC

- **grammar rules give syntax**
- **action part of a rule gives semantics, usually used to build a parse tree**

statement :
    IF (*expression*) *statement*
        create node(IF, expr, stmt, 0)
    IF (*expression*) *statement* **ELSE** *statement*
        create node(IF, expr, stmt1, stmt2)
    WHILE (*expression*) *statement*
        create node(WHILE, expr, stmt)
    variable = *expression*
        create node(ASSIGN, var, expr)
    …
expression :
    expression + *expression*
    expression - *expression*
    …

- **YACC creates a parser from this**
- **when the parser runs, it creates a parse tree**
- **a compiler walks the tree to generate code**
- **an interpreter walks the tree to execute it**
- **can even execute or generate code on the fly**

Excerpts from a real grammar

```
term:
 | term '+' term            { $$ = op2(ADD, $1, $3); }
 | term '-' term            { $$ = op2(MINUS, $1, $3); }
 | term '*' term            { $$ = op2(MULT, $1, $3); }
 | term '/' term            { $$ = op2(DIVIDE, $1, $3); }
 | term '%' term            { $$ = op2(MOD, $1, $3); }
 | '-' term %prec UMINUS    { $$ = op1(UMINUS, $2); }
 | INCR var                 { $$ = op1(PREINCR, $2); }
 | var INCR                 { $$ = op1(POSTINCR, $1); }

stmt:
 | while {inloop++;} stmt   {--inloop; $$ = stat2(WHILE,$1,$3);}
 | if stmt else stmt        { $$ = stat3(IF, $1, $2, $4); }
 | if stmt                  { $$ = stat3(IF, $1, $2, NIL); }
 | lbrace stmtlist rbrace   { $$ = $2; }

while:
   WHILE '(' pattern rparen    { $$ = notnull($3); }
```

Excerpt from a real grammar

- **precedence and associativity specified separate from grammar**

```
%right  ASGNOP
%left   OR
%left   AND
%nonassoc APPEND EQ GE GT LE LT NE MATCHOP IN
%left   CAT
%left   '+' '-'
%left   '*' '/' '%'
%left   NOT UMINUS
%right  POWER
%right  DECR INCR
```

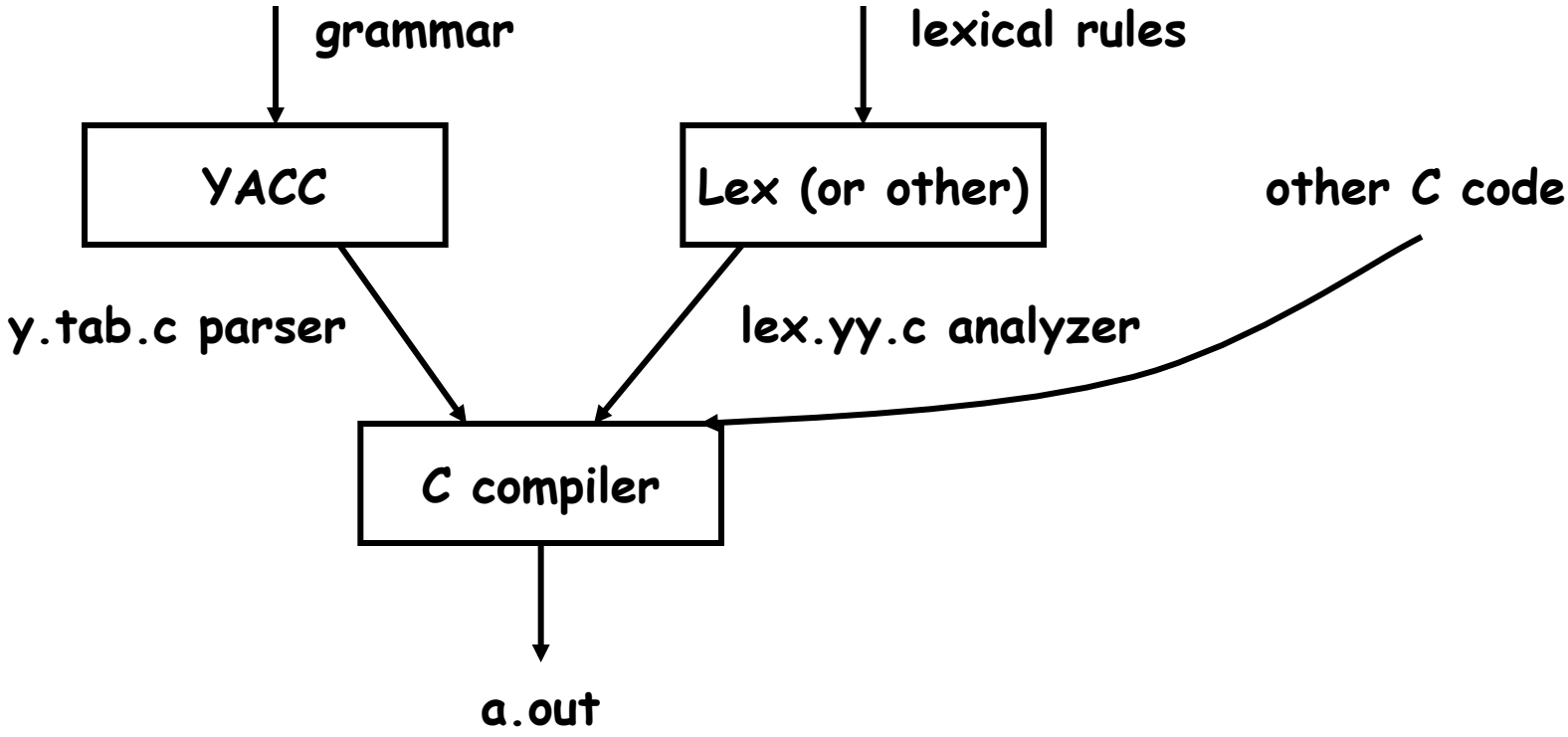
Excerpts from a LEX analyzer

```
"++"          { yylval.i = INCR; RET(INCR); }
"--"          { yylval.i = DECR; RET(DECR); }

([0-9]+(\.?)[0-9]*|\.[0-9]+)([eE](\+|-)?[0-9]+)? {
    yylval.cp = setsymtab(yytext, tostring(yytext),
                    atof(yytext), CON|NUM, symtab);
    RET(NUMBER); }

while    { RET(WHILE); }
for      { RET(FOR); }
do       { RET(DO); }
if       { RET(IF); }
else     { RET(ELSE); }
return   { if (!infunc)
              ERROR "return not in function" SYNTAX;
           RET(RETURN);
         }
•        { RET(yylval.i = yytext[0]); /* everything else */ }
```

The whole process



Example: Document preparation languages

- **illustrates topics of 333 in a different setting**
  - tools
  - language design (good and bad); notation
  - evolution of software systems; maintenance
  - personal interest, research area for 10-20 years, heavy use in books

- **examples:**
  - roff and related early formatters
  - nroff (Unix man command still uses it)
  - troff
  - TEX
  - HTML, etc.

  - all of these are "batch" commandline programs, not WYSIWYG

The roff family

- **commands on separate lines**

```
.sp 2
.in 5
This is a paragraph ...
```

- **originally just for output on line printers (ASCII)**
- **layout originally fixed**
  - e.g., only one-column output
- **nroff added macros for notational convenience**
- **and a trap mechanism for specifying page layout**
  - awkward and tricky event-based programming model
  - Turing complete!

- **how much should be built in and how much programmable?**
  - features versus extensibility

Troff: formatting for a (photo)typesetter

- **photypesetter produces output on photorgraphic paper or film**
- **first high-quality output device at a reasonable price (~$15K)**
  - predates laser printers by 5-10 years
  - predates Postscript (1982) by 10 years, PDF (1993) by 21 years
  - very klunky, slow, messy, expensive

- **troff: version of nroff for typesetters**
  - adds features for size, font, precise positioning, bigger character sets
  - originally by Joe Ossanna (~1972); inherited by BWK ~1977
- **very complex program, very complex language**
  - language reflects many of the weirdnesses of first typesetter

- **troff + phototypesetter produces book-quality output**
  - *Elements of Programming Style, Software Tools*, …

More complicated and difficult material

- **mathematics**
  - called "penalty copy" in the printing industry
- **tables**
- **drawings**
- **graphs**
- **references**
- **indexes**

- **at the time, done by hand composition**
  - not much better than medieval technology

- **Bell Labs authors writing papers and books with all of these**
- **being done by manual typewriters**
  - XXX can I find the paper with handwritten Greek letters?
- **how to handle them?**

EQN: a language for typesetting mathematics

- **with Lorinda Cherry ~1974**

- **idea: a language that matches the way mathematics is spoken aloud**
- **translate that into troff commands**
  - since the language is so orthogonal, it wouldn't fit directly
  - and there isn't room anyway, since program has to be less than 65KB
  - troff is powerful enough
- **use a pipeline eqn | troff**

- **like TEX, but simpler, easier (though not as systematic or powerful)**
  - math mode in TEX comes from EQN

EQN examples

```
x sup 2 + y sup 2 = z sup 2
```

$$x^2 + y^2 = z^2$$

```
f(t) = 2 pi int sin ( omega t ) dt
```

$$f(t) = 2\pi \int \sin(\omega t)\ dt$$

```
lim from {x -> pi / 2} ( tan x ) = inf
```

$$\lim_{x \to \pi/2} (\tan x) = \infty$$

```
x = {-b +- sqrt {b sup 2 - 4ac} over 2a }
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

EQN implementation

- **based on a YACC grammar**
  - first use of YACC outside mainstream compilers
- **grammar is simple**
  - box model
  - just combine boxes in various ways:
    concatenate, above/below, sub and superscript, sqrt, ...

eqn: box | eqn box
box: text | { eqn } | box over box | sqrt box
| box sub box | box sup box | box from box to box | ...

- **YACC makes experimental language design easy**

Pic: a language for pictures (line drawings)

- **new typesetter has more capabilities (costs more too: $50K in 1977)**
- **can we use troff to do line drawings?**

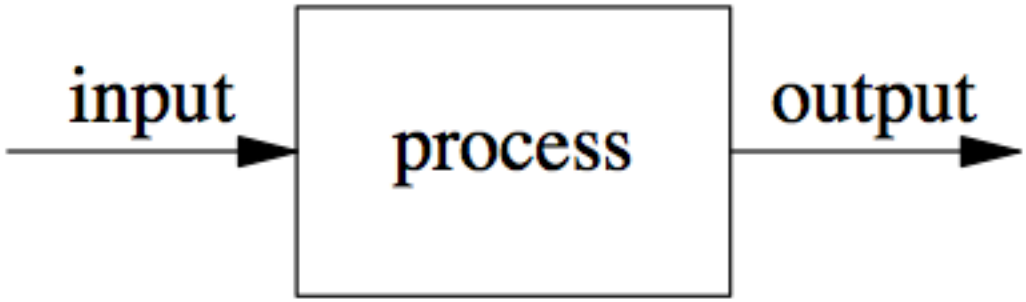
- **answer: invent another language, again a preprocessor**
  - add simple line-drawing primitives to troff: line, arc, spline

- **advantages of text descriptions of pictures**
  - systematic changes easy, always correct dimensions,
  - Pic has loops, conditionals, etc., for repetitive structures
    Turing complete!

- **implemented with YACC and LEX**
  - makes it easy to experiment with syntax
  - human engineering:
    free-form English-like syntax
    implicit positioning: little need for arithmetic on coordinates

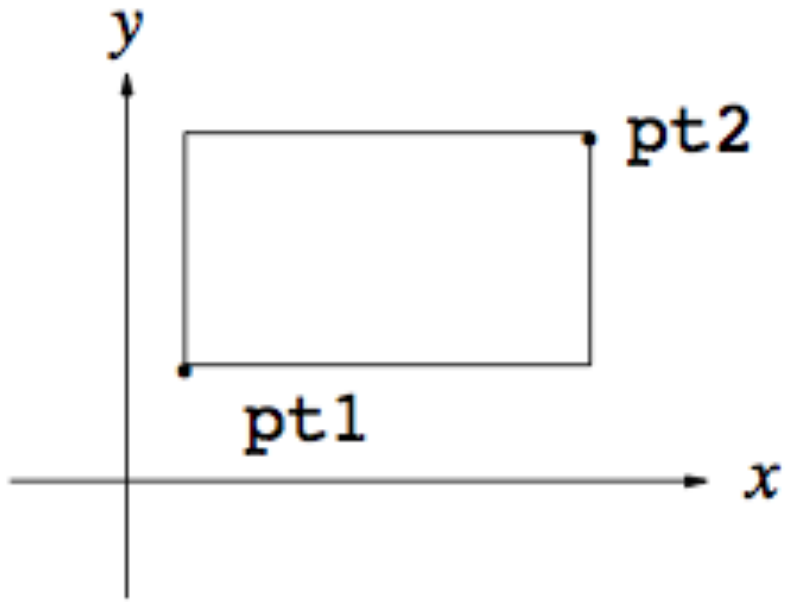
Pic examples

```
.PS
arrow "input" above
box "process"
arrow "output" above
.PE
```



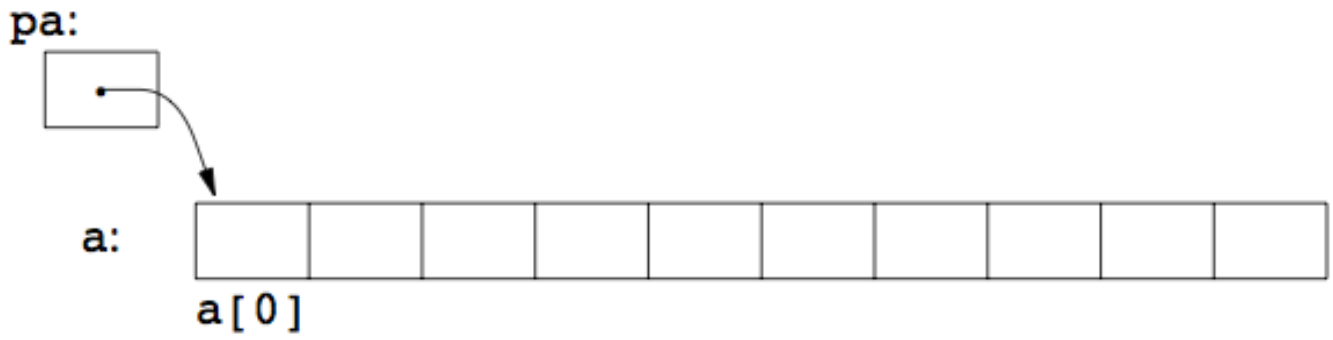
Pic examples

```
.PS
line from (-.2,0) to (1,0) ->
        "  $x$" ljust at last line.end
line from (0,-.2) to (0,.7) ->
        "$y$" at last line.end above
line from .1,.2 to .8,.2 to .8,.6 to .1,.6 to .1,.2
bullet at .1,.2
"\f(CWpt1\fP" ljust at (.2,.1)
bullet at .8,.6
"  \f(CWpt2\fP" ljust at .8,.6
.PE
```



Pic examples

```
.PS
define L { line from $1<B.nw,B.ne> to $1<B.sw,B.se> }
A: "\f(CWa\fP:" wid .5
B: box wid 3  ht .2 with .w at A.e; # "..." at .6<B.w,B.e>
L(.1); L(.2); L(.3); L(.4); L(.5)
L(.6); L(.7); L(.8); L(.9)
"\f(CWa[0]\fP" ht .18 wid .3 with .nw at B.sw
PA: box ht .2 wid .3 bullet at A + (0,.4)
"\f(CWpa\fP:" wid .1 ht .15 with .s at PA.nw
spline -> from PA right .2 then to B.nw +(.05,0.02)
.PE
```

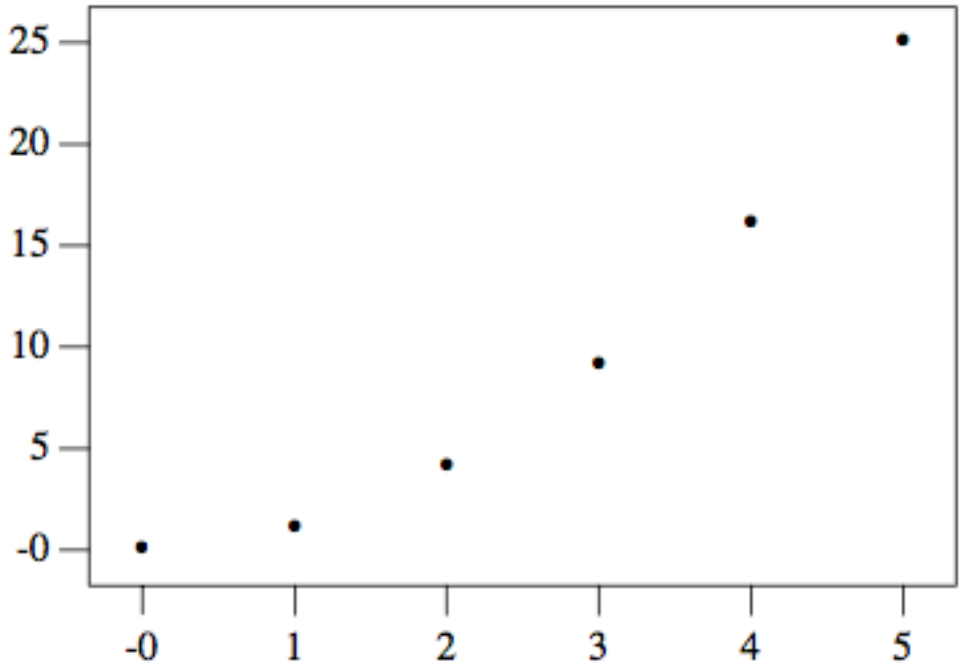


Grap: a language for drawing graphs

- line drawings, not "charts" in the Excel sense
- with Jon Bentley, ~1984

- a Pic preprocessor: grap | pic | troff

```
.G1
0 0
1 1
2 4
3 9
4 16
5 25
.G2
```



Notation matters

- **each of these languages has its own fairly natural notation**
  - doesn't work as well when force everything into one notation
  - but also can be hard to mix, e.g., equations in diagrams in tables

- **TEX/LATEX:**
  - "math mode" is a different language
  - tables are mostly the same as underlying language
  - there are no drawings (?)

- **XML vocabularies put everything into a single notation**
  - except for the specific tags and attributes
  - bulky, inconvenient, but uniform

HTML / XHTML / XML

- **HTML is a batch-mode markup language**
- **similar to TEX except very simple**
- **layout control is tricky**
  - as in troff and TEX
- **tables, but no math, no drawings**

- **MathML: XML vocabulary for mathematical expressions**

- **SVG (Scalable Vector Graphics): XML vocabulary for drawings (and more)**

- **two problems at least**
  - MathML and SVG are unusable by humans
  - MathML doesn't work consistently (if at all) in current browsers