Domain-specific languages

- **also called application specific languages, little languages**

- **narrow domain of applicability**
- **not necessarily programmable or Turing-complete**
  - often declarative, not imperative

- **sometimes small enough that you could build one yourself**

- **examples:**
  - regular expressions
  - shell, Awk
  - XML, HTML, Troff, (LA)TEX, Markdown: markup languages
  - SQL: database access
  - R: statistics
  - AMPL: mathematical optimization
  - …

Example: Markup / 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, Markdown, etc.

Unix document preparation: *roff

- **text interspersed with formatting commands on separate lines**

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

- **originally just ASCII output, fixed layout, single column**
- **nroff: macros, a event mechanism for page layout (Turing complete)**
- **troff: version of nroff for phototypesetters**
  - adds features for size, font, precise positioning, bigger character sets
  - originally by Joe Ossanna (~1972); inherited by BWK ~1977
- **photypesetter produces output on photographic 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
  - klunky, slow, messy, expensive media
- **complex program, complex language**
  - language reflects many of the weirdnesses of first typesetter
  - macro packages make it usable by mortals for standard tasks
- **troff + phototypesetter enables book-quality output**
  - *Elements of Programming Style, Software Tools*, K&R, …

Extension to complex specialized material

- **mathematics**
  - called “penalty copy” in the printing industry
- **tables**
- **drawings**
- **graphs**
- **references**
- **indexes**
- **etc.**

- **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**
- **how to mechanize the production**

EQN: a language for typesetting mathematics

- **BWK, 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`
- **math mode in TEX (1978) inspired by 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?**

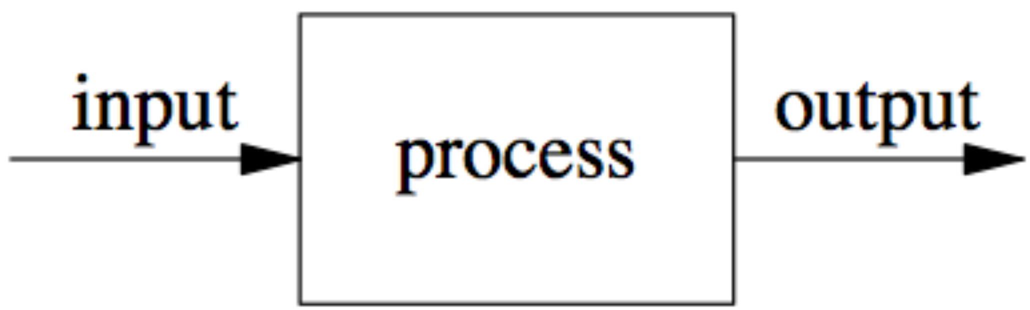
- **answer: invent another language, again a preprocessor**
  - add simple line-drawing primitives to troff: line, arc, spline

- **advantages of text descriptions of pictures**
  - systematic changes are easy, always have 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

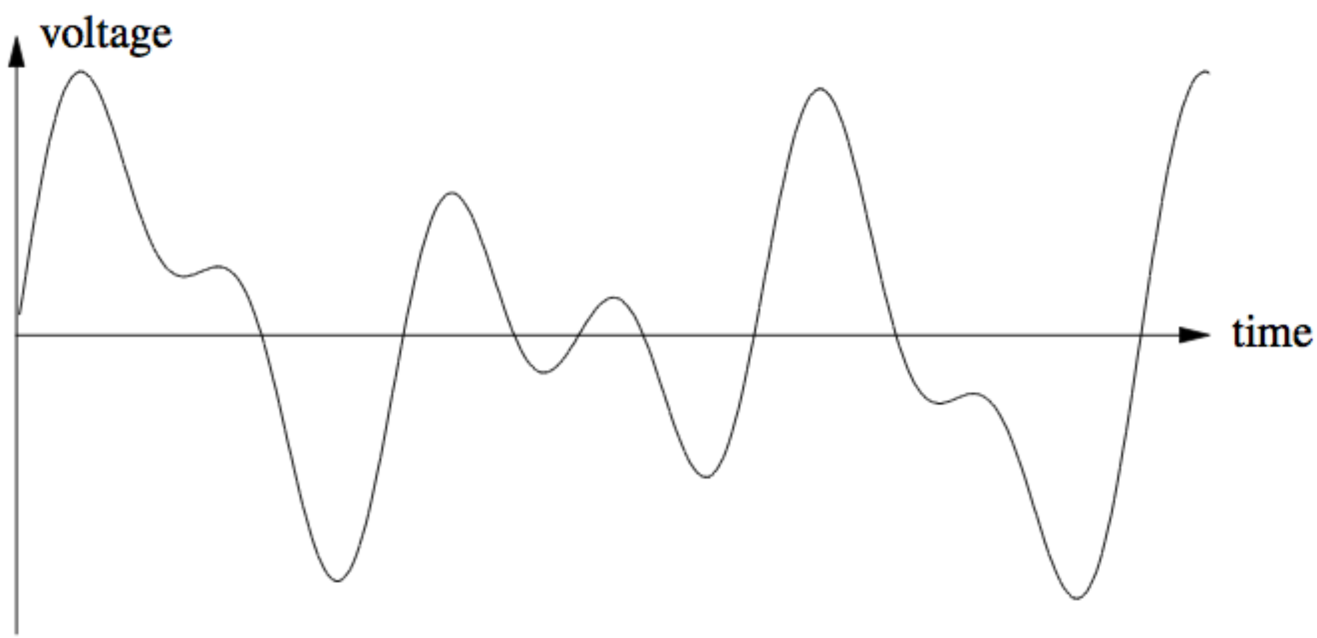
Pic examples

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



Pic examples

```
.PS
V: arrow from 0,-1 to 0,1;  "  voltage" ljust at V.end
L: arrow from 0,0 to 4,0;   "  time" ljust at L.end
for i = 1 to 399 do X
    j = i+1
    line from (L + i/100, sin(i/10) / 3 + sin(i/20) / 2
     + sin(i/30) / 4) to (L + j/100, sin(j/10) / 3
     + sin(j/20) / 2 + sin(j/30) / 4)
X
.PE
```



Markup languages

- **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 (eXtensible Markup Language) is a meta-language for markup**
  - a text-only language for describing grammar and vocabularies of other markup languages that deal with hierarchical textual data
  - a notation for describing trees
  - internal nodes are elements; leaves are Unicode text
  - element: data surrounded by markup that describes it
- **XML vocabularies put everything into a single notation**
  - except for the specific tags and attributes
  - bulky, inconvenient, but uniform

XML vocabularies and namespaces

- **a *vocabulary* is an XML description for a specific domain**

  - Schema
  - XHTML
  - RSS (really simple syndication)
  - SVG (scalable vector graphics)
  - MathML (mathematics)
  - EPUB (electronic book format)
  - Android screen layout
  - ...

- **namespaces**
  - mechanism for handling name collisions between vocabularies

```
<ns:some_tag> ... </ns:some_tag>
<ns2:some_tag> ... </ns2:some_tag>
```

MathML examples

- **Firefox 28.0**

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

- **Safari 6.1.3**

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

- **EQN**

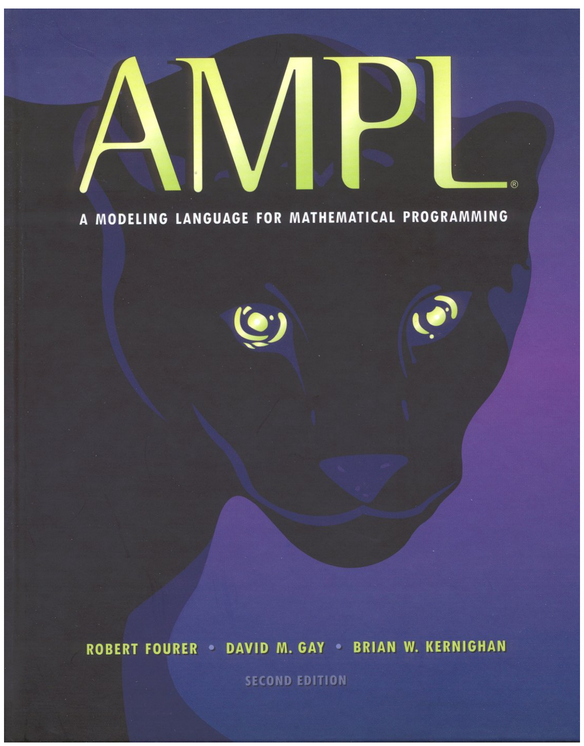
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Input is not fit for humans:

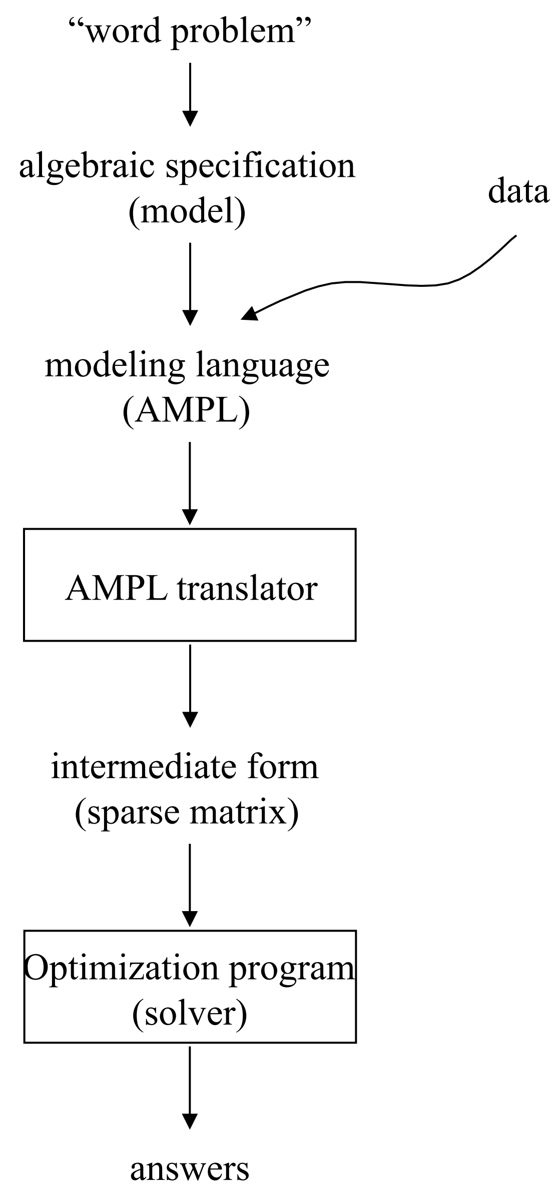
```
<mrow>
  <mi>x</mi>
  <mo>=</mo>
  <mfrac>
   <mrow>
    <mo form="prefix">&#x2212;</mo>
    <mi>b</mi>
    <mo>&#x00B1;</mo>
    <msqrt>
     <msup>
      <mi>b</mi>
      <mn>2</mn>
     </msup>
     <mo>&#x2212;</mo>
<mn>4</mn>
     <mo>&#x2062;</mo>
     <mi>a</mi>
     <mo>&#x2062;</mo>
     <mi>c</mi>
    </msqrt>
   </mrow>
   <mrow>
    <mn>2</mn>
    <mo>&#x2062;</mo>
    <mi>a</mi>
   </mrow>
  </mfrac>
 </mrow>
</math>
```

AMPL: A big DSL that got bigger

- **a language and system for**
  - describing optimization problems in a uniform, natural way
  - compiling descriptions into form needed by solver programs
  - controlling execution of solvers
  - displaying results in problem terms



Robert Fourer
David Gay
Brian Kernighan



Cost minimization: a diet model

- **Find a minimum-cost mix of TV dinners that satisfies requirements on the minimum and maximum amounts of certain nutrients.**
- **Given:**

  F, a set of foods

  N, a set of nutrients

  a_{ij} = amount of nutrient i in a package of food j

  c_j = cost of package of food j, for each $j \in F$

  f_j^- = minimum packages of food j, for each $j \in F$

  f_j^+ = maximum packages of food j, for each $j \in F$

  n_i^- = minimum amount of nutrient i, for each $i \in N$

  n_i^+ = maximum amount of nutrient i, for each $i \in N$
- **Define variables:**

  X_j = packages of food j to buy, for each $j \in F$
- **Minimize:** $\sum_{j \in F} c_j X_j$
- **Subject to:**

  $n_i^- \leq \sum_{j \in F} a_{ij} X_j \leq n_i^+$, for each $i \in N$

  $f_j^- \leq X_j \leq f_j^+$, for each $j \in F$

AMPL version of the diet model

```
set FOOD;
set NUTR;

param amt {NUTR,FOOD} >= 0;
param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];
param n_min {NUTR} >= 0;
param n_max {i in NUTR} >= n_min[i];

var Buy {j in FOOD} >= f_min[j], <= f_max[j];

minimize total_cost:  sum {j in FOOD} cost[j] * Buy[j];

subject to diet {i in NUTR}:
   n_min[i] <= sum {j in FOOD} amt[i,j] * Buy[j] <= n_max[i];
```

Diet data:

```
set NUTR := A B1 B2 C ;
set FOOD := BEEF CHK FISH HAM MCH MTL SPG TUR ;
param amt (tr):
           A    C   B1   B2 :=
   BEEF   60   20   10   15
   CHK     8    0   20   20
   FISH    8   10   15   10
   HAM    40   40   35   10
   MCH    15   35   15   15
   MTL    70   30   15   15
   SPG    25   50   25   15
   TUR    60   20   15   10 ;
param:   cost  f_min  f_max :=
  BEEF   3.19    0     100
  CHK    2.59    0     100
  FISH   2.29    0     100
  HAM    2.89    0     100
  MCH    1.89    0     100
  MTL    1.99    0     100
  SPG    1.99    0     100
  TUR    2.49    0     100 ;
param:   n_min  n_max :=
   A      700   20000
   C      700   20000
   B1     700   20000
   B2     700   20000 ;
```

Running AMPL:

```
$ ampl
ampl: model diet.mod;
ampl: data diet.dat;
ampl: solve;
MINOS 5.4: optimal solution found.
6 iterations, objective 88.2
ampl: display Buy;
Buy [*] :=
BEEF   0
 CHK   0
FISH   0
 HAM   0
 MCH  46.6667
 MTL   0
 SPG   0
 TUR   0
;
```

AMPL: moderately successful

- **a big frog in quite a small pond**
  - widely used optimization tool
  - taught in courses
  - supports a small company (~5 employees)
- **language started out purely declarative**
- **gradually has added all the trappings of programming languages**
  - conditionals
  - loops
  - functions/procedures
- **but with odd, irregular and unconventional syntax**

Why languages succeed

- **solve real problems in a clearly better way**

- **culturally compatible and familiar**
  - familiar syntax helps (e.g., C-like)
  - easy to get started with
  - portable to new environments

- **environmentally compatible**
  - don't have to buy into an entire new environment to use it
  - e.g., can use standard tools and link to existing libraries
  - open source, not proprietary

- **weak competition**
- **good luck**

Why languages fail to thrive

- **niche or domain disappears**

- **poor engineering**
  - too big, too complicated, too slow, too late
  - incompatible with environments

- **poor philosophical choices**
  - ideology over functionality
  - single programming paradigm
  - too "mathematical"
  - too different, too incompatible