DMS

Program Transformations for Practical Scalable Software Evolution

Ira Baxter   Chris Pidgeon   Michael Mehlich

Semantic Designs, Inc.

www.semanticdesigns.com

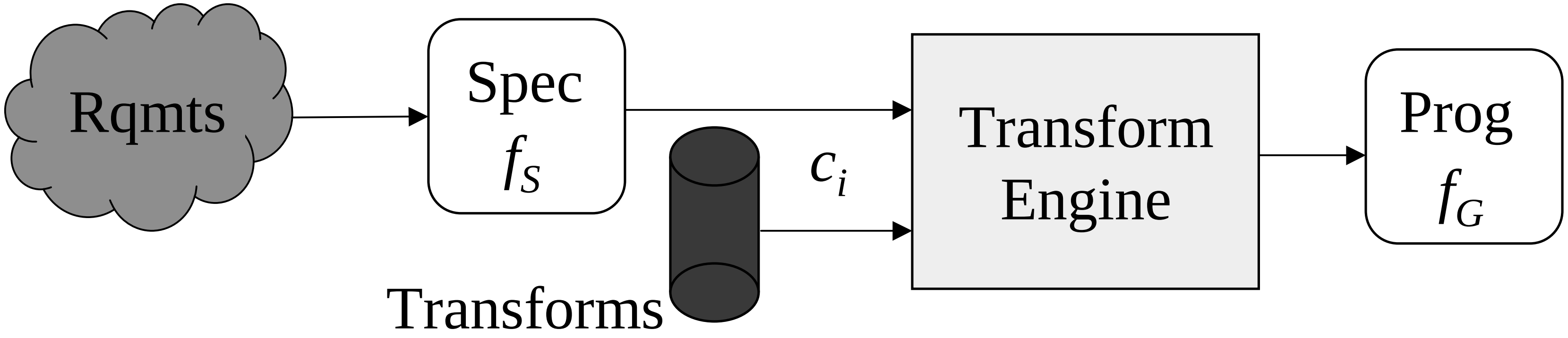
May 2004

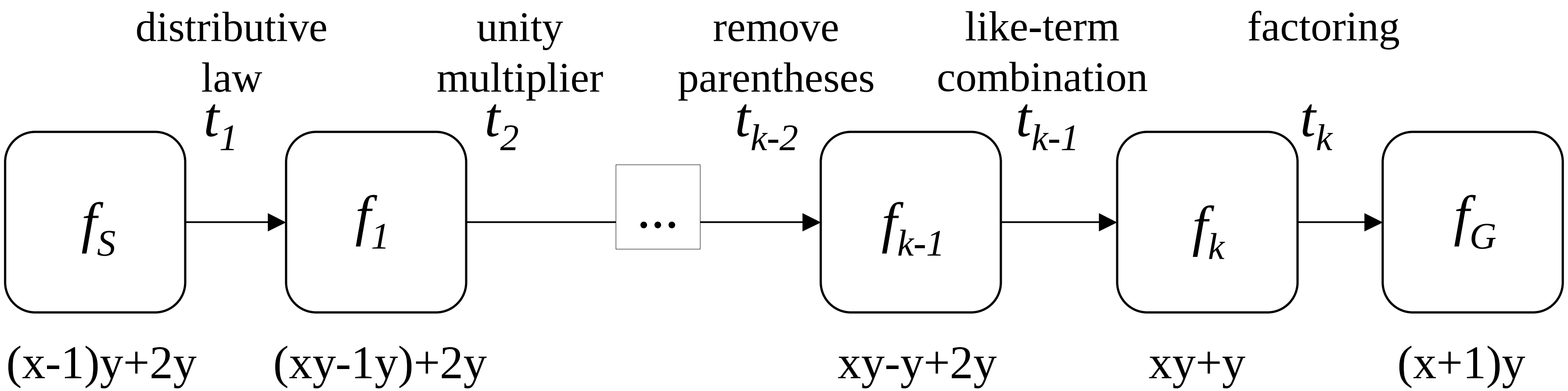
Software Change Tools Need Infrastructure Too

- To build a conventional application
  - Define specifications
  - Implement application
  - Use Operating System to provide standard services
    - File/Terminal I/O, CPU management, Security
- To build an automated software change tool
  - Define specifications
  - Implement tool
  - Use …?… to provide standard services
    - Lexing/Parsing
    - *Life after Parsing*: Tree/Symbol table building, Analysis management, Transformation, PrettyPrinting
- This talk: a practical change-tool infrastructure

Transformation Systems

Stepwise Conversion of Specs to Code



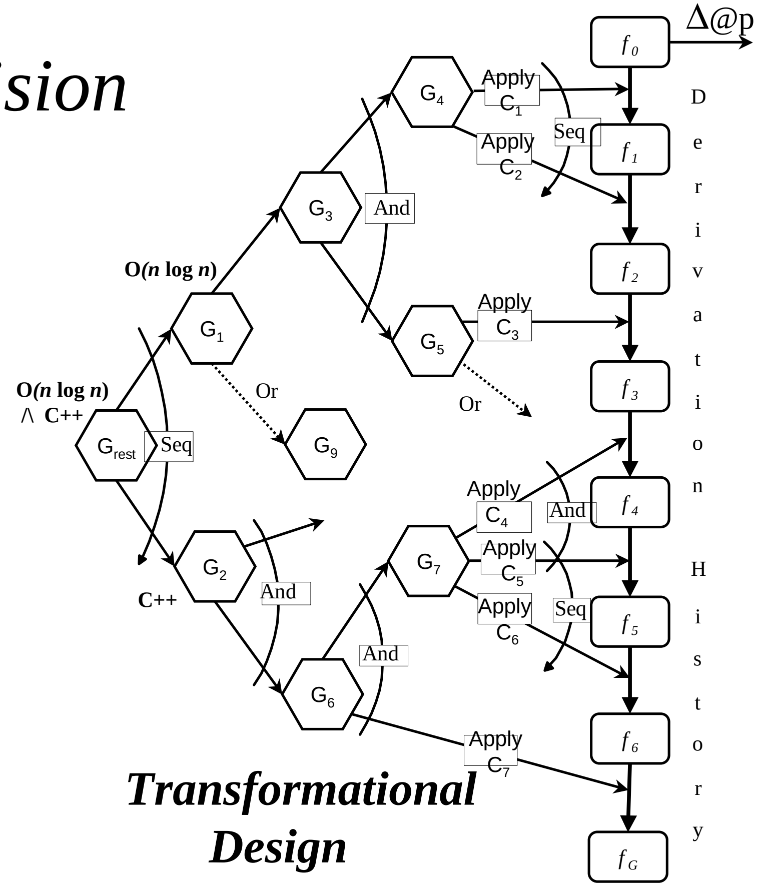


Practical Transformation Machinery

- Theory long established
  - Source to Source transforms ("rewrites")
  - Refinement theory
- Emphasis now focused on *change*
- Need integrated mechanisms
  - Parsing, PrettyPrinting
  - Name/Type Resolution, Rewriting, Analyzers
  - Essential technology already developed
- *Practical tools need support for scale*
  - Large real codes
  - Multiple languages
  - Symbolic Computation for analysis and change

The Design Maintenance System *Vision*

- Transformational Design
  - Functionality Spec (f_0) + Performance Spec (G_{rest}) + Derivation + Justification + Alternatives
- Metaprogram driven automation



Transformational Design

- Incremental Updates as Δs
  - Specification, Performance, Technology Δs
  - Δs drive design revision:
    retain transforms that commute with delta

$$\Delta@p(C_i@q(f_i)) = C_i@q'(\Delta'@p'(f_i))$$

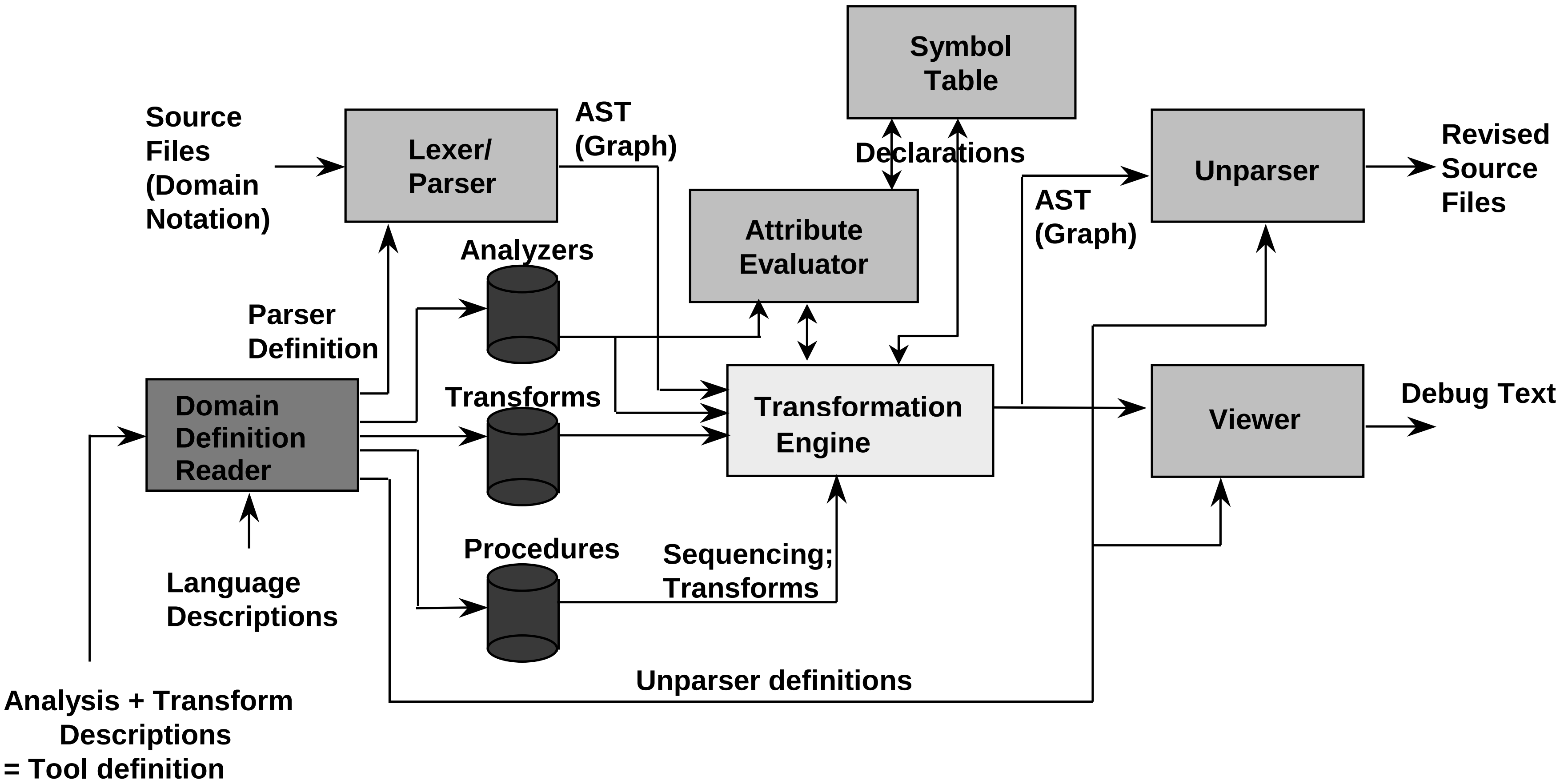
$$\begin{tikzcd} f_i \arrow[r, "\Delta@p"] \arrow[d, "C_i@q"'] & f_i' \arrow[d, "C_i@q'"] \\ f_{i+1} \arrow[r, "\Delta'@p'"'] & f_{i+1}' \end{tikzcd}$$

"Design Maintenance Systems" CACM April 92

DMS Software Reengineering Toolkit

- *Automated* source code analysis and *modification*
  - *Leverage transformation machinery needed to build DMS vision*
- Enables wide variety of SE tasks to be automated
  - *Commercial applications*
    - Source Formatters, Hyperlinked Source Browsers
    - Documentation extraction
    - Metrics
    - Preprocessor conditional simplification
    - Test Coverage and Profiling tools
    - Clone Detection and removal
    - XML DTD compilation
    - DSL code generation: Factory Automation
    - Migrations (JOVIAL to C)
  - *Research applications*
    - Aspect-weaving (U. Alabama Birmingham)
    - Large-scale C++ component restructuring (SD/Boeing)
    - Code generation/quality checking for spacecraft (NASA/JPL)

DMS *Implementation*
"Software Reengineering Toolkit"



Coded in PARLANSE

DMS Toolkit = Generalized Compiler

- Underlying Hypergraph representation: trees, graphs, …
- Parsing/Prettyprinting
  - UNICODE lexer with binary conversions, lexical format/comment capture
  - GLR (context-free) parser with automatic tree builder
  - Prettyprinting by "Text Box" building language
- Analysis
  - Multipass attribute grammars
  - Generalized symbol table support: inheritance, overloading, …
  - *Next:* Generic Control/Data Flow
- Transformation
  - Complete procedural AST interface => procedural transforms (& analyzers)
  - Conditional Source to Source transforms w/ associative/commutative laws
  - *Next:* Goal-directed metaprogramming
- Predefined Domains
  - Specification, Technology, and Implementation languages

    Spectrum, .MDL XML, SQL, IDL C/C++, C#, Java, COBOL, …

Fundamental Issue: *Scale*

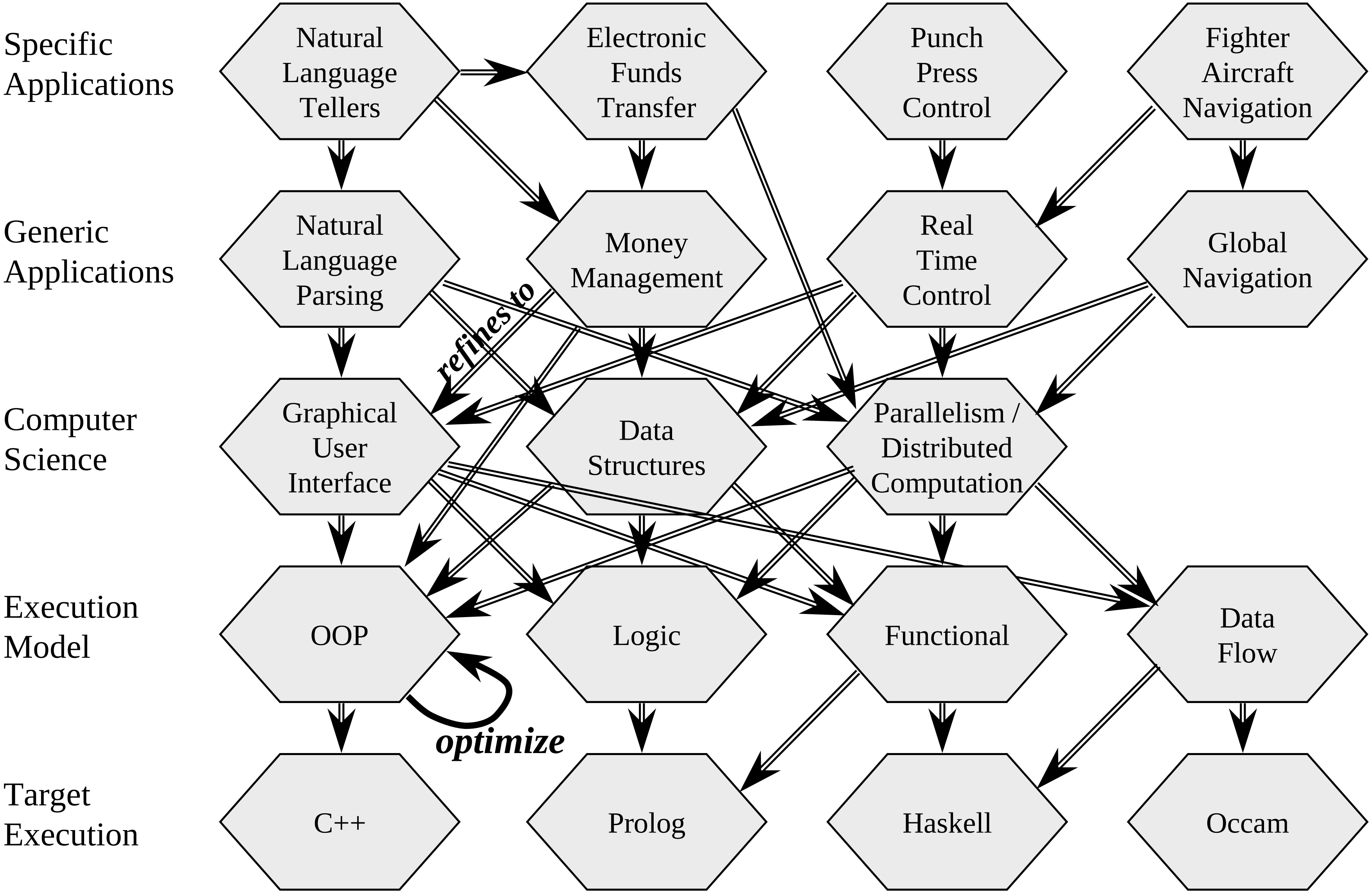
- Engineering hard but straightforward
  - Ugly details: C preprocessors, etc.
- Reasoning/Analysis costs
  - "Incremental" Knowledge capture => domains
  - Computers do Symbolic computation slowly => Parallel foundations
  - Future: RuleSet compilers
- Legacy Systems are huge
  - MSLOC + tens of thousands of files
    - Careful design of hypergraph to conserve space
  - Arbitrary languages => robust parsing technology
    - Need for domain agility: fast domain/dialect definition
    - *Use domain notations define knowledge*
  - Applications use multiple languages
    - reasoning and transforms must work with mix
- Other scale issues
  - Software Versions, Large Engineering Teams, Long term Transactions

Knowledge Capture:

DMS Domain Parts

- Syntax (domain notation)
  - External Form (what you can say: string or graphical)
  - Internal Form (How DMS stores it)
  - Parser (how to convert external form to internal form)
  - PrettyPrinter (how to display the Internal Form)
- Semantics (what the domain *means*)
  - Analyzers (how to analyze in the domain)
  - Optimizations (how to optimize in the domain)
  - Refinements (how to transform one domain to another)
  - (Attachments) (procedures to enhance DMS efficiency)

A Domain Network



DMS Domain for Java
Parser + Pretty Printer

```
nested_class_declaration = nested_class_modifiers class_header class_body ;
  <<PrettyPrinter>>:  { V(H(nested_class_modifiers,class_header),class_body); }

class_header = 'class' IDENTIFIER ;
  <<PrettyPrinter>>:  { H('class',IDENTIFIER); }
class_header = 'class' IDENTIFIER 'implements' name_list ;
  <<PrettyPrinter>>:  { H('class',IDENTIFIER,'implements',name_list); }
class_header = 'class' IDENTIFIER 'extends' name;
  <<PrettyPrinter>>:  { H('class',IDENTIFIER,'extends',name); }
class_header = 'class' IDENTIFIER 'extends' name 'implements' name_list ;
  <<PrettyPrinter>>:  { H('class',IDENTIFIER,'extends',name,'implements',name_list);

class_body = '{' class_body_declarations '}' ;
  <<PrettyPrinter>>:  { V(H('{',STRING(" "),class_body_declarations),'}'); }

nested_class_modifiers = nested_class_modifiers nested_class_modifier ;
  <<PrettyPrinter>>:  { H(CH(nested_class_modifiers[1]),nested_class_modifier); }
```

… + 300 more rules…(COBOL is 3500!)

Parsing to Abstract Syntax Trees

A Program Representation analyzable by Computers

- Use DMS grammar domain to define language syntax
- DMS generates lexer/parser automatically
- *Reengineering* parser reads source file(s)
  - Carries out lexical conversions (e.g, FP text -> IEEE binary fp)
  - Captures comments and formats of literals
  - Builds Abstract Syntax Tree
  - Records Position of every node (file, line, col)
- Present capability for the following domains
  - *Specification:* Spectrum, BNF, Rose Models
  - *Technology:* XML, IDL, SQL
  - *Implementation:* C/C++, C#, COBOL, Java, Ada, VB6, Fortran, Verilog

Winner: Obfuscated “C” Contest

The Maintenance Programmer’s Nightmare

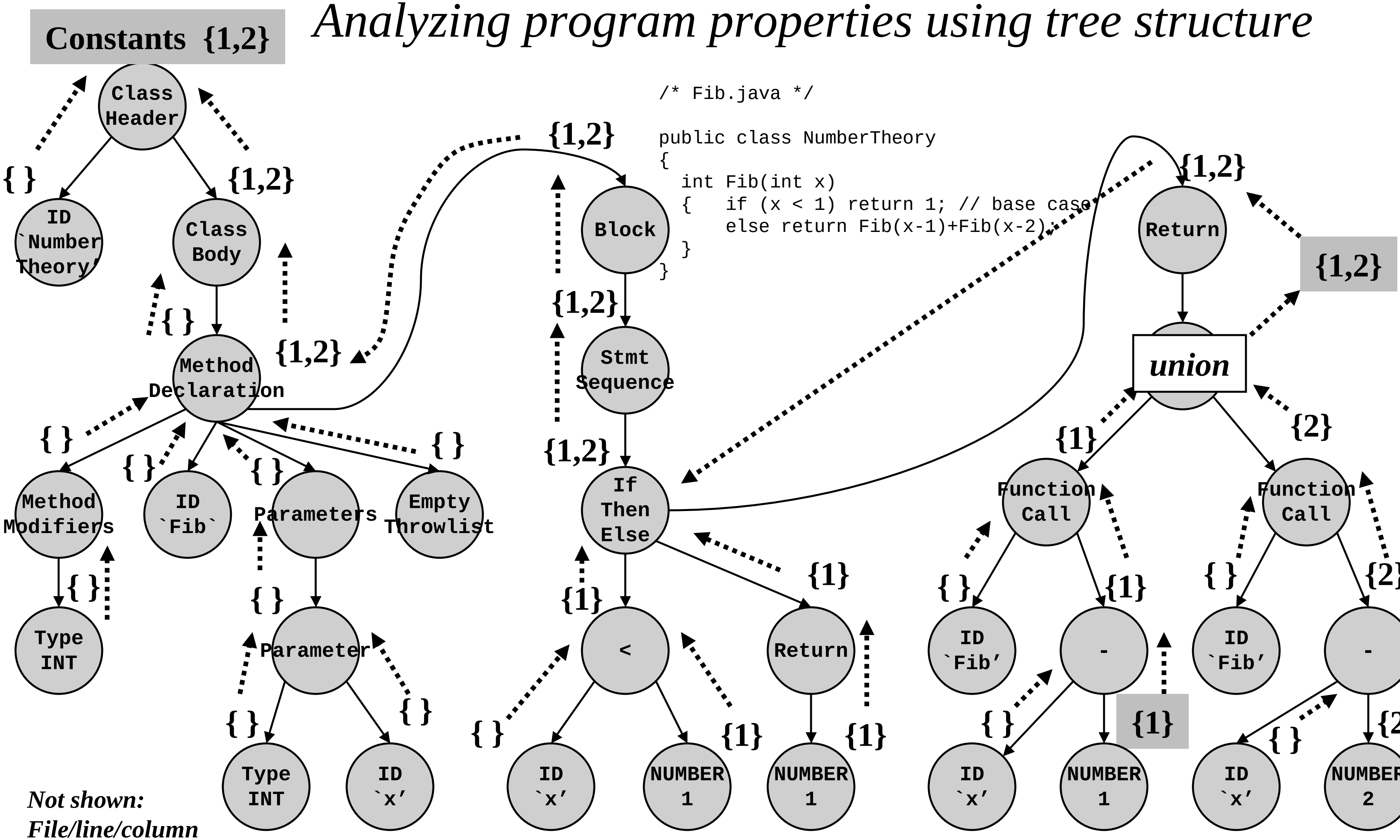
```
#include                                     <math.h>
#include                                   <sys/time.h>
#include                                   <X11/Xlib.h>
#include                                  <X11/keysym.h>
                                          double L ,o ,P
                                         ,_=dt,T,Z,D=1,d,
                                         s[999],E,h= 8,I,
                                         J,K,w[999],M,m,O
                                        ,n[999],j=33e-3,i=
                                        1E3,r,t, u,v ,W,S=
                                        74.5,l=221,X=7.26,
                                        a,B,A=32.2,c, F,H;
                                        int N,q, C, y,p,U;
                                       Window z; char f[52]
                                    ; GC k; main(){ Display*e=
 XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC (e,z,0,0),BlackPixel(e,0))
; scanf("%lf%lf%lf",y +n,w+y, y+s)+1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400,
0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval G={ 0,dt*1e6}
; K= cos(j); N=1e4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*T; O+=D*_*F/ K+d/K*E*_; B=
sin(j); a=B*T*D-E*W; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D-_*F*E; P=W*E*B-T*D; for (o+=(I=D*W+E
*T*B,E*d/K *B+v+B/K*F*D)*_; p<y; ){ T=p[s]+i; E=c-p[w]; D=n[p]-L; K=D*m-B*T-H*E; if(p [n]+w[ p]+p[s
]== 0|K <fabs(W=T*r-I*E +D*P) |fabs(D=t *D+Z *T-a *E)> K)N=1e4; else{ q=W/K *4E2+2e2; C= 2E2+4e2/ K
 *D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } L+=_* (X*t +P*M+m*l); T=X*X+ l*l+M *M;
  XDrawString(e,z,k ,20,380,f,17); D=v/l*15; i+=(B *l-M*r -X*Z)*_; for(; XPending(e); u *=CS!=N){
                                   XEvent z; XNextEvent(e ,&z);
                                       ++*((N=XLookupKeysym
                                         (&z.xkey,0))-IT?
                                         N-LT? UP-N?& E:&
                                         J:& u: &h); --*(
                                         DN -N? N-DT ?N==
                                         RT?&u: & W:&h:&J
                                          ); } m=15*F/l;
                                          c+=(I=M/ l,l*H
                                          +I*M+a*X)*_; H
                                          =A*r+v*X-F*l+(
                                          E=.1+X*4.9/l,t
                                          =T*m/32-I*T/24
                                           )/S; K=F*M+(
                                           h* 1e4/l-(T+
                                           E*5*T*E)/3e2
                                           )/S-X*d-B*A;
                                           a=2.63 /l*d;
                                           X+=( d*l-T/S
                                            *(.19*E +a
                                            *.64+J/1e3
                                            )-M* v +A*
                                            Z)*_; l +=
                                            K *_; W=d;
                                            sprintf(f,
                                            "%5d  %3d"
                                            "%7d",p =l
                                           /1.7,(C=9E3+
                              O*57.3)%0550,(int)i); d+=T*(.45-14/l*
                             X-a*130-J* .14)*_/125e2+F*_*v; P=(T*(47
                             *I-m* 52+E*94 *D-t*.38+u*.21*E) /1e2+W*
                             179*v)/2312; select(p=0,0,0,0,&G); v-=(
                              W*F-T*(.63*m-I*.086+m*E*19-D*25-.11*u
                               )/107e2)*_; D=cos(o); E=sin(o); } }
```

Pretty Printing

```
#include <math.h>
#include <sys/time.h>
#include <X11/Xlib.h>
#include <X11/keysym.h>
double L, o, P, _ = dt, T, Z, D = 1, d, s[999], E, h = 8, I, J, K, w[999],
       M, m, O, n[999], j = 3.3e-2, i = 1e3, r, t, u, v, W, S = 7.45e1,
       l = 221, X = 7.26, a, B, A = 3.22e1, c, F, H;
int N, q, C, y, p, U;
Window z;
char f[52];
GC k;
main()
{
  Display * e = XOpenDisplay(0);
  z = RootWindow(e, 0);
  for (XSetForeground(e, k = XCreateGC(e, z, 0, 0), BlackPixel(e, 0));
  scanf("%lf%lf%lf", y + n, w + y, y + s) + 1; y++);
  XSelectInput(e, z = XCreateSimpleWindow(e, z, 0, 0, 400, 400,
               0, 0, WhitePixel(e, 0)), KeyPressMask);
  for (XMapWindow(e, z);; T = sin(O))
    {
      struct timeval G = { 0, dt * 1e6 };
      K = cos(j);
      N = 1e4;
      M += H * _;
      Z = D * K;
      F += _ * P;
      r = E * K;
      W = cos(O);
      m = K * W;
      H = K * T;
      O += D * _ * F / K + d / K * E * _;
      B = sin(j);
      a = B * T * D - E * W;
      XClearWindow(e, z);
      t = T * E + D * B * W;
      j += d * _ * D - _ * F * E;
      P = W * E * B - T * D;
      for (o += (I = D * W + E * T * B, E * d / K * B + v + B / K * F * D) * _; p < y;)
        {
          T = p[s] + i;
          E = c - p[w];
          D = n[p] - L;
          K = D * m - B * T - H * E;
          if (p[n] + w[p] + p[s] == 0 | K < fabs(W = T * r - I * E + D * P) | fabs(D = t * D + Z * T - a * E) > K)
            N = 1e4;
          else
            {
              q = W / K * 4e2 + 2e2;
              C = 2e2 + 4e2 / K * D;
              N - 1e4 && XDrawLine(e, z, k, N, U, q, C);
              N = q;
              U = C;
            }
          ++p;
        }
      L += _ * (X * t + P * M + m * l);
      T = X * X + l * l + M * M;
      XDrawString(e, z, k, 20, 380, f, 17);
      D = v / l * 15;
      i += (B * l - M * r - X * Z) * _;
      for (; XPending(e); u *= CS != N)
        {
          XEvent z;
          XNextEvent(e, & z);
          ++ * ((N = XLookupKeysym(& z.xkey, 0)) - IT ? N - LT ? UP - N ? & E : & J : & u : & h);
          -- * (DN - N ? N - DT ? N == RT ? & u : & W : & h : & J);
        }
      m = 15 * F / l;
      c += (I = M / l, l * H + I * M + a * X) * _;
      H = A * r + v * X - F * l + (E = 1e-1 + X * 4.9 / l, t = T * m / 32 - I * T / 24) / S;
      K = F * M + (h * 1e4 / l - (T + E * 5 * T * E) / 3e2) / S - X * d - B * A;
      a = 2.63 / l * d;
      X += (d * l - T / S * (1.9e-1 * E + a * 6.4e-1 + J / 1e3) - M * v + A * Z) * _;
      l += K * _;
      W = d;
      sprintf(f, "%5d  %3d"
              "%7d", p = l / 1.7, (C = 9e3 + O * 5.73e1) % 0550, (int) i);
      d += T * (4.5e-1 - 14 / l * X - a * 130 - J * 1.4e-1) * _ / 1.25e4 + F * _ * v;
      P = (T * (47 * I - m * 52 + E * 94 * D - t * 3.8e-1 + u * 2.1e-1 * E) / 1e2 + W * 179 * v) /
2312;
      select(p = 0, 0, 0, 0, & G);
      v -= (W * F - T * (6.3e-1 * m - I * 8.6e-2 + m * E * 19 - D * 25 - 1.1e-1 * u) / 1.07e4) * _;
      D = cos(o);
      E = sin(o);
    }
}
```

Analyzers using Attribute Evaluation

Analyzing program properties using tree structure



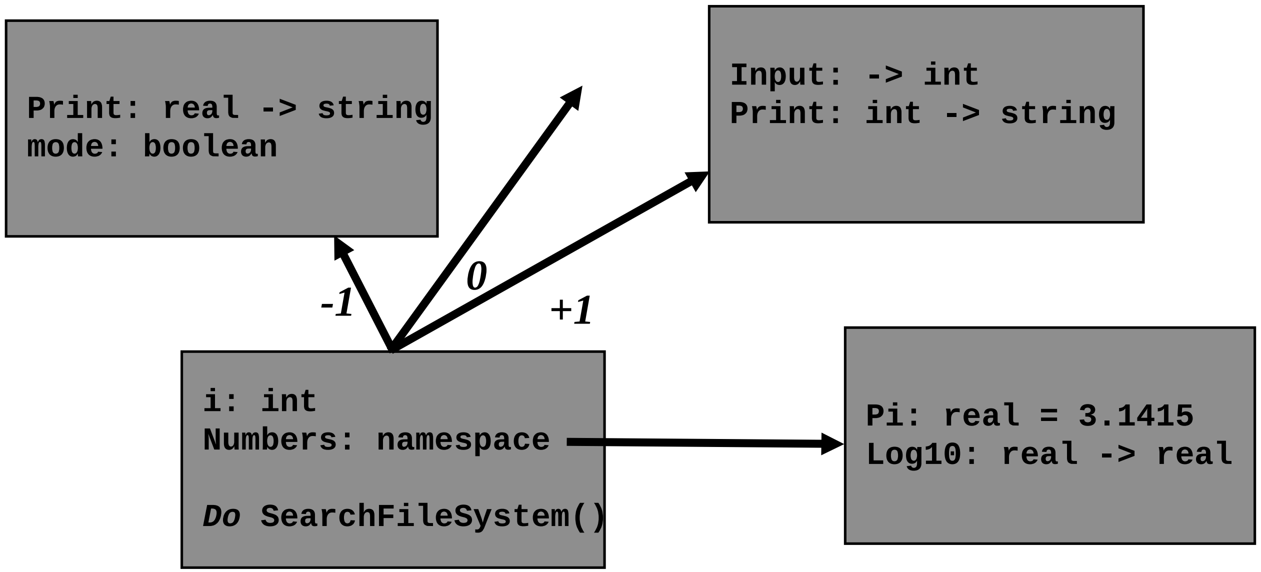
Not shown: File/line/column annotation on each node

Attribute Evaluator Specification for determining Constants used

```
ATTRIBUTES { SetOfReal Constants; }
Statement = ‘return’ Expression ;
  <<UseConstants>> { Statement.Constants = Expression.Constants; }
Expression = Sum ;
  <<UseConstants>> { Expression.Constants = Sum.Constants; }
Sum = Sum ‘-’ Term ;
  <<UseConstants>> { Sum[0].Constants = UnionReals(Sum[1].Constants,Term.Constants); }
Sum = Sum ‘+’ Term ;
  <<UseConstants>> { Sum[0].Constants = UnionReals(Sum[1].Constants,Term.Constants); }
Term = ‘(‘ Expression ‘)’ ;
  <<UseConstants>> { Term.Constants = Expression.Constants };
Term = IDENTIFIER ;
  <<UseConstants>> { Term.Constants = EmptySetOfReals(); }
Term = NATURAL ;
  <<UseConstants>> { Term.Constants = SingletonRealSetFromNatural(NATURAL.); }
Term = DOUBLE ;
  <<UseConstants>> { Term.Constants = SingletonRealSetFromDouble(DOUBLE.); }
Term = IDENTIFER ‘(‘ ArgumentList ‘)’ ;
  <<UseConstants>> { Term.Constants = ArgumentList.Constants; }
ArgumentList = Expression ;
  <<UseConstants>> { ArgumentList.Constants = Expression.Constants; }
ArgumentList = ArgumentList ‘,’ Expression ;
  <<UseConstants>> { ArgumentList[0].Constants = UnionReals(ArgumentList[1].Constants,
    Expression.Constants); }
```

Symbol Table Support

- Needed to support non-context-free transformations
- = Set of Symbol Spaces
- Each Symbol Space
  - Map from identifier to arbitrary value
    - Domain specific matching used to implement overloading
  - Multiple lexical-scope parent links with integer priorities
  - Lexical-scope search mediated by attached actions
- Typically constructed by attribute evaluation
  - *Fully* implemented: C, C++, Java, COBOL



Optimization transform in DMS Rewrite Rule Language

```
default base domain C;

rule use-auto-increment(v: lvalue):
        statement -> statement
   “\v = \v +1”
   rewrites to
   “\v++”
   if no_side_effects(v);
```

Domain Name

Domain Syntax

Rule Condition

Before: `(*Z)[a>>2] = (*Z)[a>>2]+1;`

After: `(*Z)[a>>2]++;`

Refinement transforms

Jovial to C

```
default source domain Jovial;
default target domain C;

private rule refine_data_reference_dereference_NAME
                (n1:identifier@C,n2:identifier@C)
                   :data_reference->expression
  = "\n1\:NAME @ \n2\:NAME" -> "\n2->\n1".

private rule refine_for_loop_letter_2
               (lc:identifier@C,f1:expression@C,
                f2:expression@C,s:statement@C)
                     :statement->statement
  = "FOR \lc\:loop_control :
         \f1\:formula BY \f2\:formula; \s\:statement"
    ->
       "{ int \lc = (\f1);
          for(;;\lc += (\f2)) { \s } }"
        if is_letter_identifier(lc).
```

Domain Name

Pattern Variables

Source Domain Syntax

Target Domain Syntax

Refinement Transforms in Action

Jovial to C

JOVIAL Source:

```
FOR i: j*3 BY 2 ;
    x@mydata = x@mydata+I;
```

Translated C Result:

```
{ int i = j*3;
  for (;;i+=2)
     { mydata->x = mydata->x + i}
}
```

Typically lots of small transforms for full translation
~2500 rules to translate full Jovial

A More Complex Example

Jovial to C

```
START
  TABLE TFP'D'TWRDET (1:109,12:37);
    BEGIN
      % Main status boolean %
      ITEM TFP'G'TWRDET STATUS (V(YES),V(NO));
    END
    TYPE TFP'D'TWRDET'TABLE TABLE (7:23) W 3;
      BEGIN
        ITEM TFP'ITM S 3 POS(0,3); "cube axis"
      END

  %begin proc%
  PROC PROC'A(c1) S;
    BEGIN
      ITEM match'count U 6;
        %an item%
      ITEM c1 C 5; "parameter value"
      ITEM c2 C 7;
      IF c1 <= c2 AND c2 > c1;
         match'count = UBOUND(TFP'D'TWRDET,0) +
                       UBOUND(TFP'D'TWRDET'TABLE,0);
      "result off by 1 so adjust"
      match'count = match'count+1;
      BEGIN
        match'count=match'count/2;
        PROC'A = match'count; % return answer %
      END "cleanup and exit";
    END "end proc"

TERM
```

packed tables with bit offsets, typedefs, functions, string operations, comments

```
#include "jovial.h"
static struct
  { /* Main status boolean */
    enum { V(yes$OF$tfp_g_twrdet$OF$tfp_d_twrdet),
           V(no$OF$tfp_g_twrdet$OF$tfp_d_twrdet) }
                          tfp_g_twrdet _size_as_word;
          } tfp_d_twrdet[109][26];
typedef union
  { W(3);
    struct
      { POS(0, 3) S(3) tfp_itm:4 _align_to_bit; /* cube axis */
      };
   } tfp_d_twrdet_table[17];

static S proc_a(C(5) c1);
/* begin proc */
static S proc_a(C(5) c1)
{ __typeof__(proc_a(c1)) RESULT(proc_a);
  _main:
    { U(6) match_count;
      C(7) c2;
      if (CHARACTER_COMPARE(BYTE_CONVERT(C(7), c1, 7), c2) <= 0
          && CHARACTER_COMPARE(c2, BYTE_CONVERT(C(7), c1, 7)) > 0)
        match_count = UBOUND(tfp_d_twrdet, 2, 0) + 16;
      /* result off by 1 so adjust */
      match_count = (S(6))match_count + 1;
      { match_count = (S(6))match_count / 2;
        RESULT(proc_a) = (S(6))match_count; /* return answer */
      } /* cleanup and exit */
      ;
    }
  _return:
    return RESULT(proc_a);
} /* end proc */
```

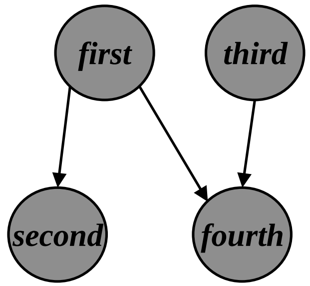
Equivalent C (used with hand-coded macro library)

PARLANSE

PARallel LANguage for Symbolic Expression

- Used to implement DMS & DMS-based applications
  - Goal: provide support expensive, large-scale symbolic computation
- Compiled, C-like programming language
  - Scalar and compound data types; pointers without casts
- Parallelism Support for SMP workstations; 2-3x speedup NOW
  - Explicitly specified parallelism
    - Dynamic forks
    - Static partially-ordered computations
    - Compiler chooses how much to implement
  - `signal` and `wait` primitives
- Software Engineering Support
  - Dynamic strings and arrays
  - Storage allocation pools with block release
  - Exception handling integrated with parallelism
  - Debug time checking: array indices, union tags, bad pointers, …

```
(|; first(<< second fourth)
                     (+= x)
    second           (= z (fib x))
    third            (sort (. y))
    fourth (>> third) (= f y:x) )|;
```



Uses of PARLANSE in DMS

- DMS core implementation
  - 250K SLOC; SE support was essential!
  - Parallel safe data structures (e.g., symbol tables)
- Application tool "metaprogramming" language
  - Can often parallelize actions of tools at high level
- Custom domain support/escapes
  - Parallel Symbol table construction for Java (3500 files!)
  - Parallel procedural Clone analysis
- Target of DMS attribute rule compiler
  - Reliable partial-order parallel programs by construction!
    - C++ Name resolver: 40K SLOC .ATG --> 250K SLOC PARLANSE
- Future: fine-grain parallel rewriting/inference
- *Importance grows with scale of automation ambitions*
  - *You can't decide to parallelize when you hit a scale barrier!*

DMS: Conclusion

- Transformation technology maturing
  - Practical value ***now*** for
    - Massive change
    - High quality code generation
- Need generalized compiler-like infrastructure
  - Definable parsers, prettyprinters, transforms, analyzers
  - Must *scale* to application systems with MSLOCs
  - ***Amortizes tool costs over many custom tools***
- Much work remaining to achieve DMS *vision*
  - Capture of transform sequence and rationale
  - Implementation of design revision
  - Transactioning to support teams of engineers