Une analyse interprocédurale modulaire des relations affines entre variables

François IRIGOIN
Centre de Recherche en Informatique
École des mines de Paris, Fontainebleau

Pourquoi?

1. parallélisation
   - intraprocédurale
   - interprocédurale
   - privatisation de tableau
2. tests de débordement de tableaux
   - élimination statique
   - réduction du surcoût dynamique
3. spécialisation de code:
   - *évaluation partielle*, constant folding,...

4. optimisation de code:
   - connaissance de valeurs maximales et de bornes de boucles
5. maintenance et retroingénierie:
   - suppression de code mort
6. preuve de propriétés convexes
   - exclusion mutuelle, producteur-consommateur,...

Parallélisation automatique: transposition de matrice

```
DO I = 1, N
   DO J = I+1, N
      T = A(I,J)
      A(I,J) = A(J,I)
      A(J,I) = T
   ENDDO
ENDDO
```

Spécialisation de code: CURMOD

```
C  P(I,II,INC,ISIG,J,K,NFREE,NTERMS)
C    {IC==10, ILIM==0, INCMAX==10,
C    IPRIN==0, MODE==0, NFREE+NLS==NPTS,
C    NTERMS==NLS, NSEARC==0}

31     DO 34 J = 1, NSEARC

C  P() {0==-1}
          BETA(J) = 0.0

C  P() {0==-1}
          DO 34 K = 1, J

C  P() {0==-1}
34             ALPHA(J,K) = 0.0
```

Simplification de test de débordement: extrait de TOMCATV

```
      PROGRAM TOMCATV
      PARAMETER (NMAX = 513, ITMAX = 1000)
      REAL*8 RX(NMAX,NMAX), RXM(ITMAX)

      READ (5, *) N, ITACT
      IF (ITACT.GT.ITMAX.OR.N.GT.NMAX) THEN
         PRINT *, 'Please recompile: This version is lacking Storage'
         STOP
      ENDIF
C  P(ITACT,N) {ITACT<=1000, N<=513}
      DO 140 ITER = 1, ITACT
         DO 80 J = 2, N-1
            DO 80 I = 2, N-1
C  P(I,ITACT,ITER,J,N) {2<=I, I+1<=N, ITACT<=1000, ITER<=ITACT,
C     1<=ITER, 2<=J, J+1<=N, N<=513}
               RXM(ITER) = MAX(RXM(ITER), ABS(RX(I,J)))
80          CONTINUE
140   CONTINUE
      END
```

Preuve de propriété: exemple Lambert

```
C  T(A,B,C,D,E,F,G) {A+C#init+E+G==A#init+C+E#init+G#init,
C    B+C+E#init+G#init==B#init+C#init+E+G,
C    C+D+E==C#init+D#init+E#init, F+G==F#init+G#init}
     DO WHILE (.TRUE.)
C     Transition sc
       IF (A.GE.1.AND.D.GE.1) THEN
C          T(A,B,D,E) {A==A#init-1, B==B#init+1, D==D#init-1, E==E#init+1}
          A = A-1, D = D-1, B = B+1, E = E+1
C     Transition rc
       ELSEIF (E.GE.1.AND.F.GE.1) THEN
C          T(D,E,F,G) {D==D#init+1, E==E#init-1, F==F#init-1, G==G#init+1}
          E = E-1, F = F-1, D = D+1, G = G+1
C     Transition sd
       ELSEIF (B.GE.1.AND.D.GE.1) THEN
C          T(A,B,C,D) {A==A#init+1, B==B#init-1, C==C#init+1, D==D#init-1}
          B = B-1, D = D-1, A = A+1, C = C+1
C     Transition rd
       ELSEIF (C.GE.1.AND.G.GE.1) THEN
C          T(C,D,F,G) {C==C#init-1, D==D#init+1, F==F#init+1, G==G#init-1}
          C = C-1, G = G-1, D = D+1, F = F+1
       ELSE
          PRINT *, A, B, C, D, E, F
       ENDIF
```

Preuve de propriété:exemple Lambert (2)

```
C  P(A,B,C,D,E,F,G) {A==1, B==0, C==0, D==1, E==0, F==1, G==0}
      DO WHILE (.TRUE.)
C       Transition sc
          IF (A.GE.1.AND.D.GE.1) THEN
             A = A-1, D = D-1, B = B+1, E = E+1
C       Transition rc
          ELSEIF (E.GE.1.AND.F.GE.1) THEN
             E = E-1, F = F-1, D = D+1, G = G+1
C       Transition sd
          ELSEIF (B.GE.1.AND.D.GE.1) THEN
             B = B-1, D = D-1, A = A+1, C = C+1
C       Transition rd
          ELSEIF (C.GE.1.AND.G.GE.1) THEN
             C = C-1, G = G-1, D = D+1, F = F+1
          ELSE
C  P(A,B,C,D,E,F,G) {A+B==1, B+2C+D==G+1, C+D+E==1, F+G==1}
             PRINT *, A, B, C, D, E, F
          ENDIF
      ENDDO
```

Une précondition dans D2ESP de FPPP

```
C  P(I,IACC,IAJ,IAJ1,IAK,IAK1,IBETA,ICHECK,ICRIT,IDM,IDN,IDUMP,IEND,
C    IF1,IFHAM,IFIRST,IFLAGS,IFRC,IFROZE,IFXYZ,IHDIAG,IMO,INCONF,
C    INDDO,INEQ,INITF,IPRINT,IREST,ISCF,ISCFP,ISCVEE,ISCVEN,ISCVNN,
C    ISHLBF,ISYMGR,IVEE,JFXYZ,LENB,LENHAM,LENMO,LENNDO,LRWSY1,NAT3,
C    NDEN,NHAM,NONC1,NOP1,NORB,NOSYM,NTT,USESYM) {IAJ==IFHAM,
C    IAJ==INCONF, IAJ1==IEND+2, IAJ1==IF1+1, IFIRST+ISCVNN==2IBETA-1,
C    IBETA+NBASIS==IFIRST, IDM==6NATOMS+2, IDN==2NAT3+2,
C    IDN==6NATOMS+2, ISCVNN+LENNDO==IF1+3, IFLAGS==IHDIAG-4,
C    IFLAGS==ISCVEN-1, IFRC==0, IFXYZ==2, IHDIAG==IMO+4,
C    IHDIAG==ISCVEE+2, INCONF+LENHAM==ISCVNN-3, 3NSHELL+NTT==INEQ-2,
C    IOUT==6, IREST==0, ISCF==0, ISCFP==1, ISCVEN==ISCVNN-2, IVEE==1,
C    JFXYZ==NAT3+2, 2JFXYZ==3NSHELL+4, LENB==7401, LENMO==0,
C    LRWSY1==13, MAXSHL==400, NDEN==1, NHAM==0, NONC1==1, NOP1==1,
C    NORB==0, NOSYM==0, 1<=I, NTT+1<=I, IACC<=0,
C    3INCONF+2ISHLBF+3LENHAM+11<=5IBETA,
C    5IBETA<=3INCONF+2ISHLBF+3LENHAM+12,
C    3INCONF+3LENHAM+2LENNDO+3<=IBETA+2ISHLBF,
C    IBETA+2ISHLBF<=3INCONF+3LENHAM+2LENNDO+4, 0<=ICRIT, ICRIT<=9,
C    4INCONF<=4IFROZE+NSHELL+2, 4IFROZE+NSHELL<=4INCONF}

       IF (.NOT.USESYM) THEN
```

Interprocédularité: *transformer* dans l'appelant D2ESP

Chaque module n'est analysé qu'une fois:

```
C  T(LENB) {37MAXSHL+2<=2LENB, 2LENB<=37MAXSHL+3}

C
C     INITIALIZE THIS LINK.
C
C***      CALL DRUM
C***      CALL TRAKIO(6HD2ESP ,1)
      LENB = 15*MAXSHL+INTOWP(2+7*MAXSHL)
```

Transformer de l'appelé INTWOP

```
C  T(INTOWP) {2INTOWP<=I+1, I<=2INTOWP}

      INTEGER FUNCTION INTOWP(I)
      INTOWP = (I+1)/2
      END
```

Modularité: transformers et preconditions

- abstraction de commandes, $Cmd = Store \rightarrow Store$, et non de stores σ (avec $Store = Loc \rightarrow Value$)
- sémantique: fonction de store à store
- abstraction: relation polyèdrique entre ensembles de stores
- minimisation du nombre d'équations explicites:

```
C  T(I,J) {I+J==I#init+J#init}
      DO WHILE (I.LT.N)

C  P(IXX,J,NN) {J==1, 2N==NN}
      DO 11 I = 1, NN, 2
```

- impact sur la précision: la meilleure fonction de transfert convexe dépend de l'espace de départ

Precondition imprécise (exemple TME)

Postcondition souhaitée: $0 \leq \sigma(\rho(A)) < \sigma(\rho(T))$

```
      PROGRAM NON_CONVEX
      INTEGER I, A ,T
      A = 0
      T = 0

C  P(A,T) {A==0, T==0}
      DO I = 1, N

C  P(A,I,T) {1<=I, I<=N, T+1<=I, 0<=T}
         READ *, X
         IF (X.GT.X0) THEN
            A = T
C  P(A,I,T) {A==T, I<=N, T+1<=I, 0<=T}
            T = T+1
         ENDIF
      ENDDO

C  P(A,I,T) {N+1<=I, T+1<=I, 0<=T}
      PRINT *, A, T
      END
```

Transformer convexe imprécis

$$(A = A' \wedge T = T') \vee (A = T' \wedge T = T' + 1)$$

```
      PROGRAM NON_CONVEX
      INTEGER I, A ,T
      A = 0
      T = 0

C  T(A,I,T) {T+1<=I+T#init, T#init<=T}
      DO I = 1, N
         READ *, X

C  T(A,T) {T#init<=T, T<=T#init+1}
         IF (X.GT.X0) THEN
            A = T
            T = T+1
         ENDIF
      ENDDO

      PRINT *, A, T
      END
```

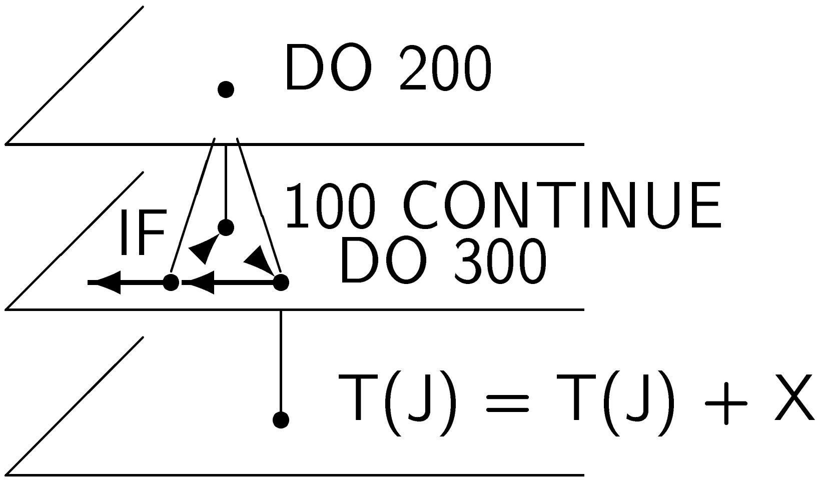
Le calcul direct de la postcondition convexe est possible.

Graphe de contrôle hiérarchique et AST

- exploiter les parties structurées:
  - équations sémantiques simples
- minimiser la taille des graphes de contrôles quand ils sont nécessaires
  - complexité dépendant du nombre de nœuds et d'arcs
- emboitement récursif de zones structurées et non-structurées

Exemple de décomposition

```
      DO 200 I = 1, N
100      CONTINUE
         DO 300 J = 1, M
           T(J) = T(J) + X
300      CONTINUE
         IF(X.GT.T(I)) GO TO 100
200   CONTINUE
```



Opérateurs de base

- call:
  - assignment
  - intrinsic
  - user call
- sequence: opérateur *rond*
- test: enveloppe convexe
- loop, whileloop: point fixe
- unstructured (CFG): point fixe

Équations sémantiques

Algèbre linéaire

- satisfiabilité: Fourier-Motzkin et simplexe
  (avec extensions entières)
- élimination de redondance: dual et satisfiabilité
- enveloppe convexe: Chernikova (Halbwachs)
- heuristiques

Problèmes: explosion en magnitude (64 bits), en espace et en temps

Impact d'overflows à répétition

```
C  P(A1,A2,I,N,PC1,PC2,S,T) {I<=N, 1<=2I+PC1, PC1+1<=2I,
C    7PC1+6PC2+21S+42<=29I+14T, 9PC1+6PC2+18S+27<=21I+11T,
C    4PC1+9PC2+8S+15<=14I+8T, 2PC1+3PC2+6S+9<=7I+4T,
C    PC1+2PC2+4S+6<=5I+2T, 2PC1+PC2+4S+6<=5I+2T, PC1+PC2+2S+2<=2I+T,
C    PC1+2S+3<=4I, PC1+3S+6<=5I+2T, PC1+2S+3<=3I+T, T<=I+PC1+2S+1,
C    T+1<=3I+PC1, PC1+1<=I+T, 1<=2I+PC2, PC2+1<=2I, PC2+2S+3<=4I,
C    PC2+3S+6<=5I+2T, PC2+2S+3<=3I+T, T<=I+PC2+2S+1, T+1<=3I+PC2,
C    PC2+1<=I+T, S+2<=2I, T<=2I, T<=PC1+PC2+2S+2, 0<=PC1+2S+1,
C    0<=PC2+2S+1, 0<=S, 0<=T}

          IF (PC1.EQ.1) THEN
c            end of critical section for process 1
            PC1 = -1
            S = S+1
          ENDIF

C  P(A1,A2,I,N,PC1,PC2,S,T) {I<=N, 1<=2I+PC2, PC2+1<=2I, T+1<=3I+PC2,
C    PC2+1<=I+T, T<=2I, 0<=T}
```

Les points fixes pour boucles

Ils sont calculés sur les transformers et éventuellement recalculés avec les preconditions:

– recherche du noyau: équations invariantes

– pattern-matching: FORMA

– dérivation: recherche de contraintes sur les différences

– Halbwachs/Cousot: deux algorithmes (exponentiel par rapport à l'imbrication), un implémenté, jamais utilisé

– Talpin & al.: non implémenté

Problème: fonctions involutives, $f^k = I$

Exemple de point fixe: *precondition*

```
C  P(IJ,IX,IY) {1<=IX, NX2+1<=IX}
      DO IX = 1, NX2

C  P(IJ,IX,IY) {1<=IX, IX<=NX2}
         IJ = IX

C  P(IJ,IX,IY) {IJ==IX, 1<=IX, IX<=NX2}
         DO IY = 1, NY2

C  P(IJ,IX,IY) {IX+IY<=IJ+1, 1<=IX, IX<=NX2, 1<=IY, IY<=NY2}
            A(IJ) = 0.

C  P(IJ,IX,IY) {IX+IY<=IJ+1, 1<=IX, IX<=NX2, 1<=IY, IY<=NY2}
            IJ = IJ+NX2
         ENDDO
      ENDDO
```

Exemple de point fixe: *transformer*

```
C  T(IJ,IX,IY) {1<=IX}
      DO IX = 1, NX2
         IJ = IX
C  T(IJ,IY) {1<=IY}
         DO IY = 1, NY2
            A(IJ) = 0.
C  T(IJ) {IJ#init+NX2==IJ}
            IJ = IJ+NX2
         ENDDO
      ENDDO
```

Il faut réévaluer ce point fixe en prenant en compte les préconditions pour obtenir des préconditions précise: IJ est croissant.

Les points fixes pour CFG

- décomposition en composantes fortement connexes
- suppression des arcs de retour
- si DAG, calcul des fonctions de transfert le long de chaque chemin (pas très efficace)
- remontée avec calcul de point fixe sur l'union des fonctions de transferts et remplacement de la fonction de transfert associée au point d'entrée de la composante fortement connexe
- un point fixe par circuit
- même algorithme pour transformers et preconditions

Exemple de point fixe sur CFG

```
    PROGRAM UNSTRUC11

    K = 0
    I = 0
    J = 0
    L = 0

100 CONTINUE
    IF(X.GT.0.) GO TO 300
    L = L + 4
200 CONTINUE
    K = K + 3
    I = I + 1
    IF(Y.GT.0.) GO TO 100
    J = J + 2
    GO TO 200

300 CONTINUE
    PRINT *, I, J, K, L
    END
```

Exemple de point fixe sur CFG (suite)

```
C  T(I,J,K,L) {4I+2J#init+L#init==4I#init+2J+L, 3I+K#init==3I#init+K,
C    0.<=X, 2I#init+J<=2I+J#init, J#init<=J}
C     BEGIN UNSTRUCTURED
99998 CONTINUE
      IF (X.GT.0.) GOTO 300
C  T(L) {L==L#init+4}
      L = L+4

C  T(I,K) {I==I#init+1, K==K#init+3}
200   CONTINUE
      K = K+3
      I = I+1

      IF (Y.GT.0.) GOTO 99998
C  T(J) {J==J#init+2}
      J = J+2
      GOTO 200
300   CONTINUE
C     END UNSTRUCTURED
```

Pour aller au-delà de Fortran

- environnement, $\rho = Id \rightarrow Loc$:
  - aliasing statique
  - structure de données tableaux
  - absence de pointeurs
- exceptions vs pas d'effets de contrôle:
  - traitement du STOP seulement
- récursivité
- paramètres formels fonctionnels non traités

Extensions récentes ou en cours

– transformer pour expressions

– booléens: encodés comme des entiers

– réels: constantes entières à éviter

x=i, y=j, x+y!=i+j

Fourier-Motzkin *adapté* pour les entiers

précision numérique potentiellement excessive

– chaînes de caractères: deux variables au plus (à prouver)

– programmes non conformes à la norme: ajout de tests

– point fixe amélioré pour les CFGs

Exemple de prédicats multitypes

Transformer:

```
C  T(I,L1,S1,X) {S1==S2, U+V==X, 1<=I, I<=2, 0<=L1, 0<=L1, L1<=1,
C    L1<=L2, L2+L3<=L1+1, L1<=L3}
     CALL COPY_FOUR(I, 1+MOD(J, 2), X, U+V, S1, S2, L1, L2.AND.L3)
     PRINT *, I, X, S1, L1
```

Postconditions:

```
     CALL COPY_FOUR(I, 1+MOD(J, 2), X, U+V, S1, S2, L1, L2.AND.L3)
C  P(I,L1,L2,S1,S2,U,V,X) {"Hi!"==S2, 2.==U, 3.==V, L2==1, S1==S2,
C    U+V==X, 1<=I, I<=2, 0<=L1, L1<=1, L3<=L1, L1<=L3}

     PRINT *, I, X, S1, L1

     READ *, J, U, V, S2, L2
C  P(I,J,L1,L2,S1,S2,U,V,X) {"Hi!"==S1, 2.+3.==X, 1<=I, I<=2, 0<=L1,
C    L1<=1, L3<=L1, L1<=L3}
```

Exemple de prédicats logiques dans TWLDRV (FPPP)

```
C  P(A1,A12,A1234,A1234I,A12I,A2,A3,A34,A34I,A4,ABVE00,ABX,ABY,ABZ,AX,
C    AX1,AY,AZ,BX,BY,BZ,C1110,C1120,C1130,C1210,C1220,C1230,C1310,
C    C1320,C1330,C2110,C2120,C2130,C2210,C2220,C2230,C2310,C2320,
C    C2330,C3110,C3120,C3130,C3210,C3220,C3230,C3310,C3320,C3330,
C    CDVE00,CDX,CDY,CDZ,CPA,CPB,CPC,CPD,CPP,CPPP,CPPPP,CPPPS,CPPS,
C    CPPSP,CPPSS,CPS,CPSP,CPSPP,CPSPS,CPSS,CPSSP,CPSSS,CSA,CSB,CSC,
C    ...
C    V1200,V1210,V1220,V1230,V1300,V1310,V1320,V1330,V2000,V2010,
C    V2020,V2030,V2100,V2110,V2120,V2130,V2200,V2210,V2220,V2230,
C    V2300,V2310,V2320,V2330,V3000,V3010,V3020,V3030,V3100,V3110,
C    V3120,V3130,V3200,V3210,V3220,V3230,V3300,V3310,V3320,V3330,VE00,
C    VE00S,VE11,VE11S,VE12,VE12S,VE13,VE14,VE21,VE21S,VE22,VE22S,VE23,
C    VE24,VE31,VE31S,VE32,VE32S,VE33,VE34,VEE,VTEST,VTOL1,VTOL2,VTOLS)
C    {FMTGEN:FOUR==2FMTGEN:TWO, 2FMTGEN:ONE==FMTGEN:TWO,
C    FMTGEN:TEN==5FMTGEN:TWO, ICRIT==ICRITP-1, IOUT==6, ISCF==0,
C    2NATOMS==NSHELL, NONC1==1, NSYMOP==1, FI<=1, FIJ<=FI, FIJK<=FIJ,
C    0<=FIJK, 1<=ICRITP, ICRITP<=10, 0<=IFMT, JSHELL<=ISHELL,
C    KSHELL<=ISHELL, ISHELL<=2NATOMS, 1<=JSHELL, LSHELL<=KSHELL,
C    1<=LSHELL, 0<=USESYM, USESYM<=1}
```

Entiers et flottants en analyse multitype

Les constantes flottantes sont traitées comme des variables:

```
      PROGRAM INTFLOAT
      A = 0.
      J = 0

C  T(A,I,J) {2I+J#init==J+2, 1<=I}
      DO I = 1, N
C  T(A,J) {2.+A#init==A, J==J#init+2}
         J = J+2
         A = A+2.
      ENDDO
      PRINT *, A, J
      END
```

```
      PROGRAM INTFLOAT
      A = 0.
      J = 0
      DO I = 1, N
        J = J+2
        A = A+2.
      ENDDO

C  P(A,I,J) {2I==J+2, 1<=I, N+1<=I}
      PRINT *, A, J
      END
```

Pas de point fixe, pas (trop) de problèmes de précision numérique!

Améliorations prévues ou envisagées

- évaluation ou ré-évaluation des transformers avec les préconditions
- linéarisation des expressions non-linéaires
- analyse arrière
- détection des variables à valeurs bornées
- analyse des tableaux: $b > n \; \equiv \; \forall i \; b[i] > n$

Exemple de linéarisation: accès à Q(L)

```
    DO 210 I=2,NX1
      Q1=0.D0
      DO 200 J=2,NY1
        XX=(I-1.5D0)*HX
        YY=(J-1.5D0)*HY
        L=I+(J-1)*NX2
        DO 200 KSP=1,NSP
          TM=DENS(XX,YY)*DSIGN(1.0D0,QSPEC(KSP))
          IF (NPX(KSP)*NPY(KSP).EQ.0) TM=0.D0
          Q(L) = Q(L) - TM
          Q1=Q1+TM
200     CONTINUE
      QAV=QAV-Q1
210 CONTINUE
```

Encodage d'une union

```
      PARAMETER (MAXVALUE=10)
      READ *, I, J, K
      IF (I.LT.-MAXVALUE.OR.I.GT.MAXVALUE.OR.J.LT.-MAXVALUE.OR.J.GT.
     &MAXVALUE.OR.M.LT.-MAXVALUE.OR.M.GT.MAXVALUE) STOP
C  P(I,J,K) 0<=I+10, I<=10, 0<=J+10, J<=10, 0<=M+10, M<=10
      IF (X.GT.0.) THEN
         I = M, K = 1
      ELSE
         J = M, K = 0
      ENDIF
C  P(I,J,K) 0<=I+10, I<=10, M<=I+J+10, I+J<=M+10, 20K+M<=I+20,
C     I+20K<=M+20, 0<=J+10, J<=10, M<=J+20K, J<=20K+M, 0<=M+10, M<=10
      IF (K.EQ.1) THEN
C       Can we guess the relation between i and n?
         IF (I.EQ.M) THEN
C  P(I,J,K) I==M, K==1, 0<=J+10, J<=10, M<=J+20, J<=M+20, 0<=M+10,
C     M<=10
            PRINT *, I, J, K
         ELSE
C  P(I,J,K) 0==-1
            PRINT *, I, J, K
         ENDIF
      ENDIF
```

État de l'art

- constant propagation/constant folding, PRE,...
- égalités: Karr
- Rice:
  - Callahan, Cooper, Torczon, Kennedy
  - Havlak: GSA
- IBM Research:
  - Wegman, Zadeck
  - SSA
- gardes: Dehbonei (Bull)
- polynômes: Haghighat (UIUC CSRD)
- Polyspace Technologie: Alain Deutsch?!?

Implémentation

- bibliothèque linéaire creuse (44.5 K avec le dense):
  - vecteur (3 K), contrainte (2K)
  - systeme de contraintes (12 K)
  - système générateur (0.6 K)
  - polyèdre (6 K)
- modules linéaire dense: Simplexe, Chernikova (Polylib)
- bibliothèque transformer + environnement: 4.5 K

– bibliothèque semantics: implémentation des équations sémantiques, 12 K lignes de C
  – dbm_interface.c: 790
  – ri_to_transformers.c: 1511
  – ri_to_preconditions.c: 693
  – loop.c: 906
  – expression.c: 2017
  – interprocedural.c: 1322
  – unstructured.c: 2519

Conclusion

- déjà 12 ans... robustesse et vitesse
- modularité: un point fixe par circuit
- interprocéduralité: une analyse par module
- preuves en tous genres
- traitement de programmes de 1 à 160 K lignes
- centaines de contraintes et de variables
- vitesse et précision: améliorations progressives en fonction des besoins pratiques