A Modular Static Analysis Approach to Affine Loop Invariants Detection

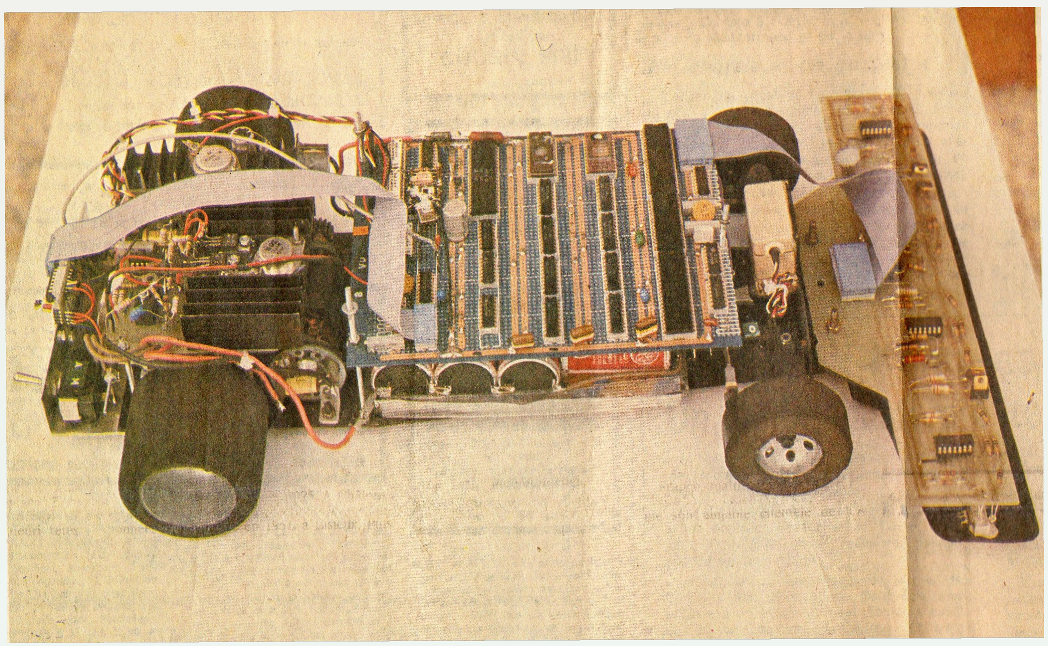
Corinne Ancourt, Fabien Coelho and Francois Irigoin

2nd International Workshop on Numerical and Symbolic Abstract Domains
NSAD 2010

Car racing ....

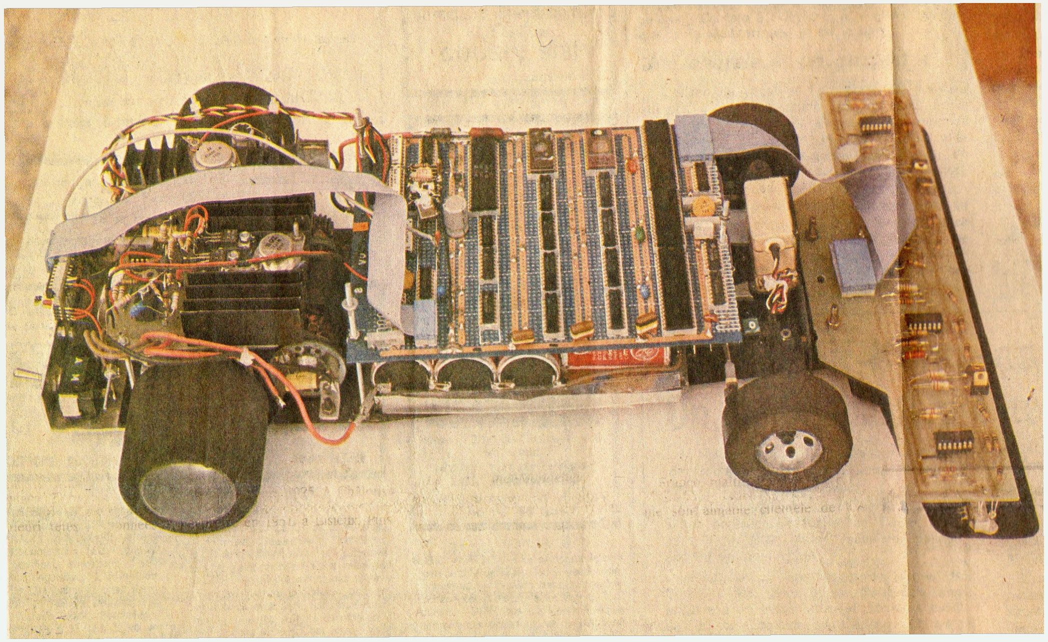
I.1.1

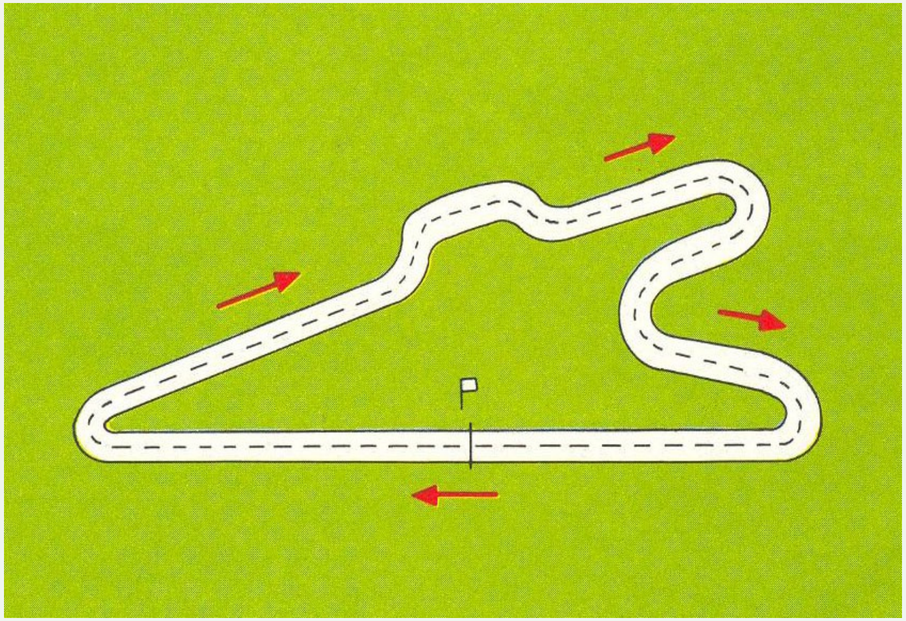
Intelligent Car - 1980



Car racing ....

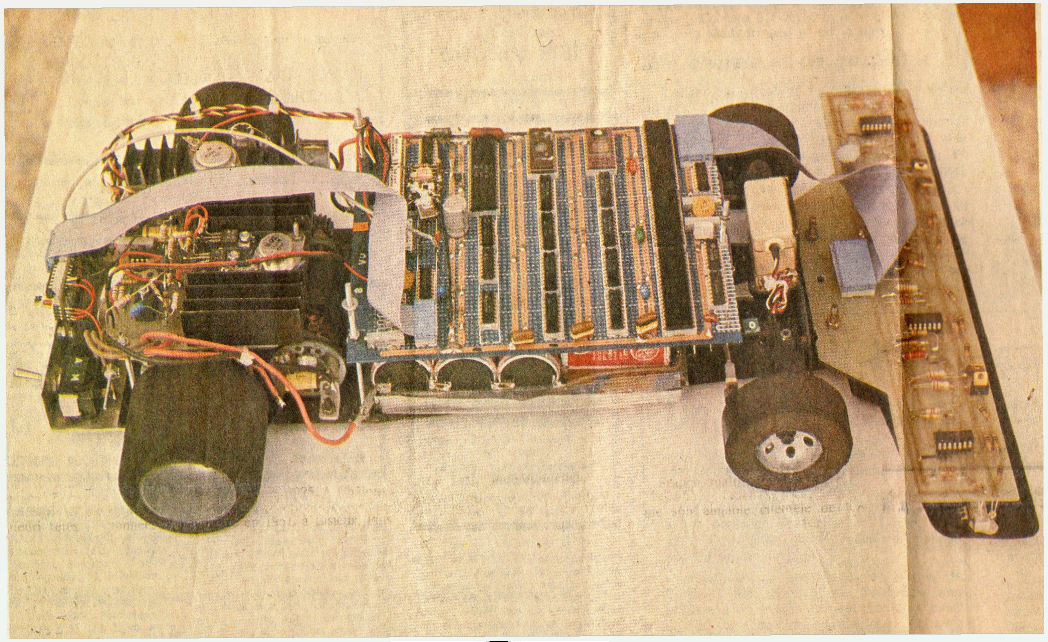
Intelligent Car - 1980

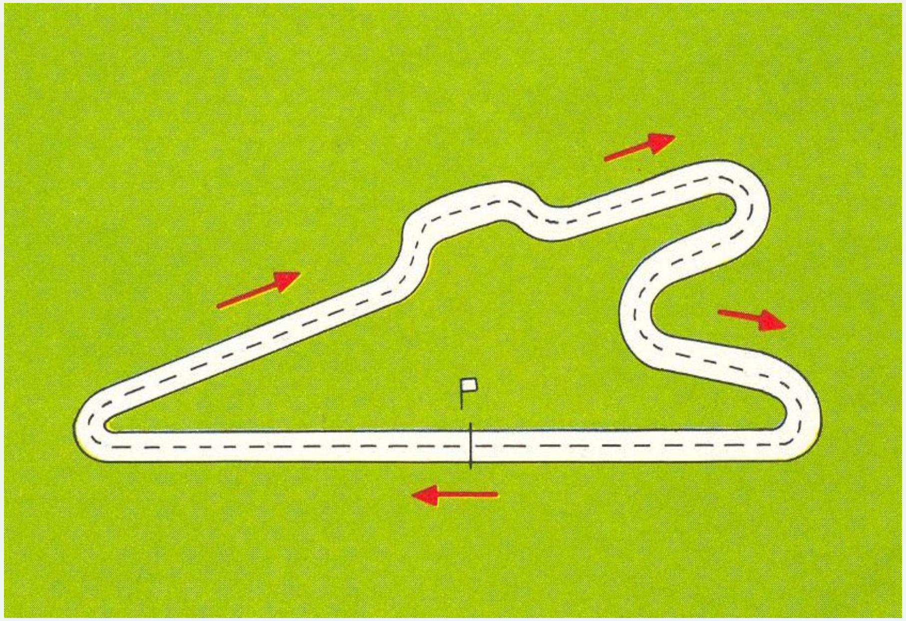


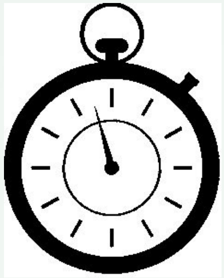


Car racing ....

Intelligent Car - 1980

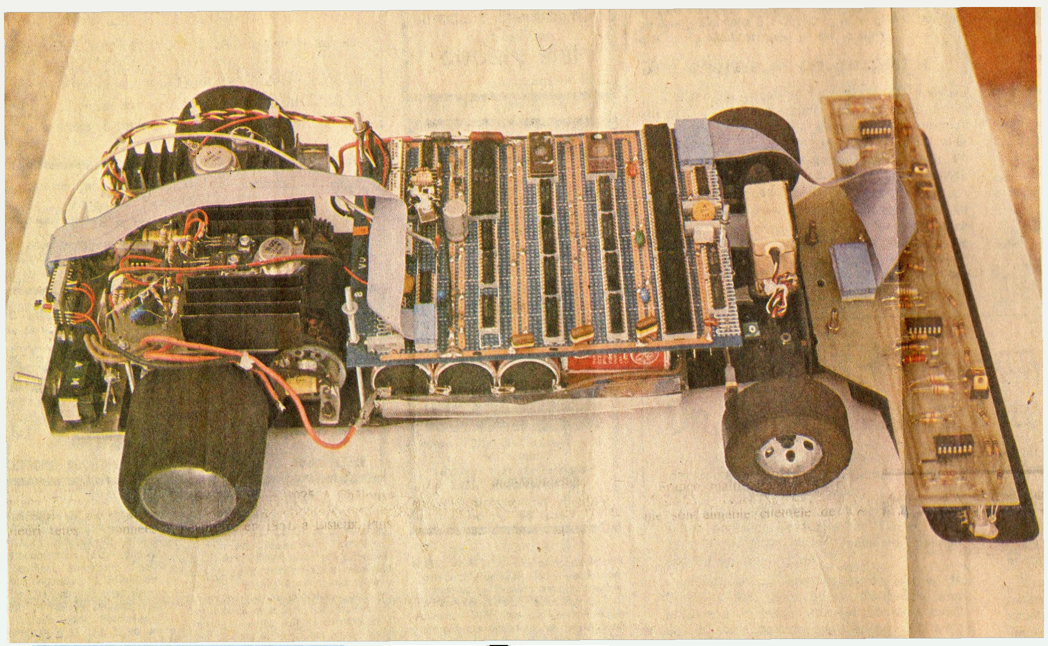


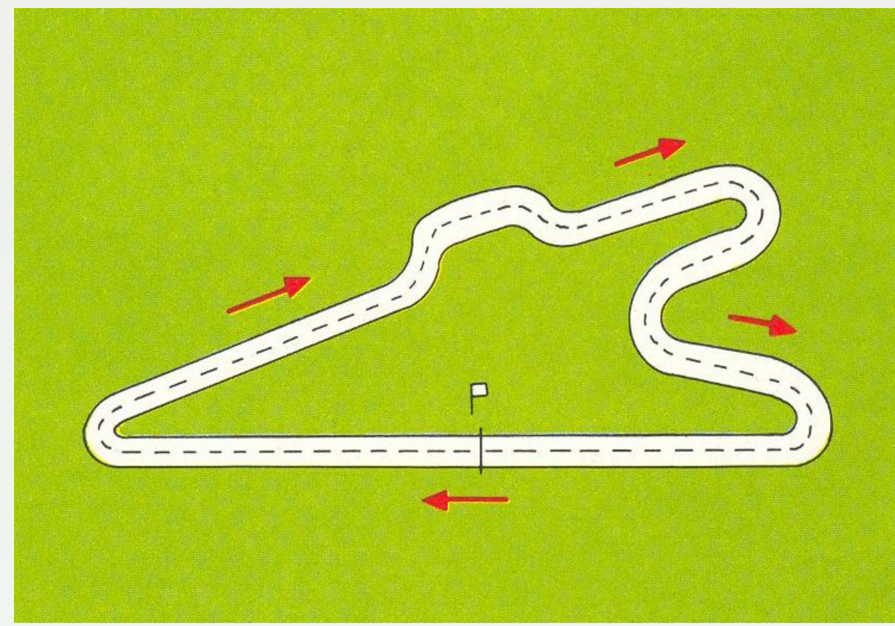


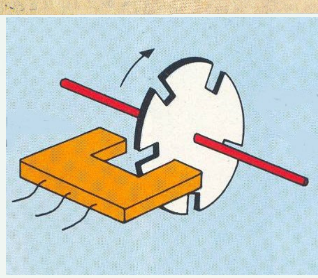


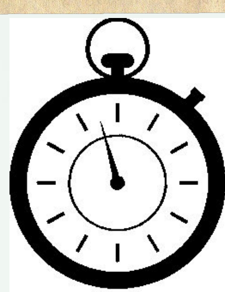
Car racing ....

Intelligent Car - 1980





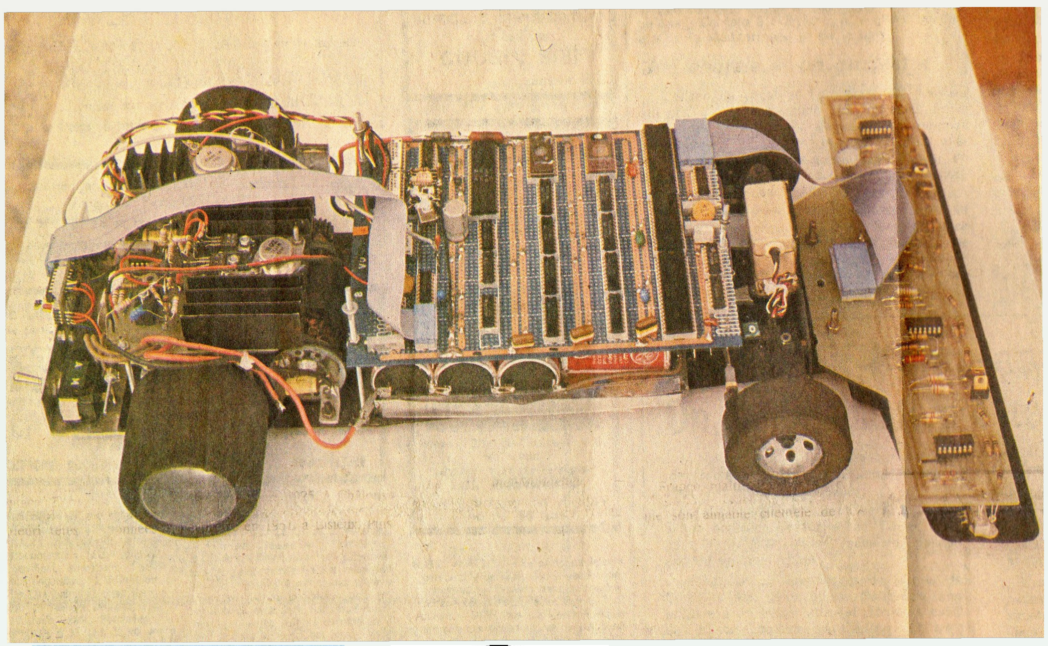


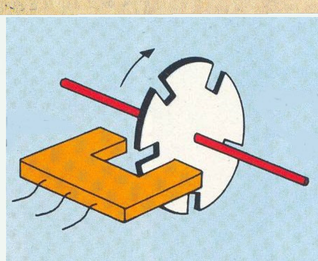


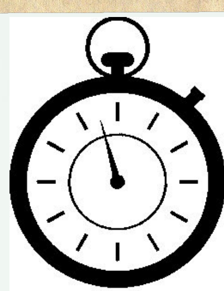
Car racing ....

I.1.1

Intelligent Car - 1980







Robot Car safety – Halbwachs 93 extended

```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```

Scientific Computing

I.2.1

```
      SUBROUTINE COSTI (N,WSAVE)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DIMENSION  WSAVE(*)
      PARAMETER (PI = 3.14159265358979)
      IF (N .LE. 3) THEN
      RETURN
      ENDIF
      NM1 = N-1
      NP1 = N+1
      NS2 = N/2
      DT = PI/FLOAT(NM1)
      KC = NP1-1
      DO 101 K=2,NS2
        KC = KC-1
        FK = FLOAT(K-1)
        WSAVE(K) = 2.D0*DSIN(FK*DT)
        WSAVE(KC) = 2.D0*DCOS(FK*DT)
  101 CONTINUE
C     CALL RFFTI (NM1,WSAVE(N+1))
      RETURN
      END
```

Computing Invariants for

- **Optimization**
- **Dependence analysis**
- **Parallelization**
- **Array access bound checking**
- **Array access initialization checking**
- **Property verification**
- **...**

Scientific Computing

I.2.1

```
      SUBROUTINE COSTI (N,WSAVE)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DIMENSION  WSAVE(*)
      PARAMETER (PI = 3.14159265358979)
      IF (N .LE. 3) THEN
      RETURN
      ENDIF
      NM1 = N-1
      NP1 = N+1
      NS2 = N/2
      DT = PI/FLOAT(NM1)
      KC = NP1-1
      DO 101 K=2,NS2
        KC = KC-1
        FK = FLOAT(K-1)
        WSAVE(K) = 2.D0*DSIN(FK*DT)
        WSAVE(KC) = 2.D0*DCOS(FK*DT)
  101 CONTINUE
C       CALL RFFTI (NM1,WSAVE(N+1))
      RETURN
      END
```

```
C  P(KC,NM1,NP1,NS2){KC==N, KC==NM1+1,
C  KC==NP1-1, 4<=KC, 2NS2<=KC, KC<=2NS2+1}

      DO 101 K = 2, NS2

C  P(K,KC,NM1,NP1,NS2){K+KC==N+2, N==NM1+1,
C  N==NP1-1, K<=NS2, KC<=N, 4<=N, 2NS2<=N,
N<=2NS2+1}

        KC = KC-1

C  P(K,KC,NM1,NP1,NS2){K+KC==N+1, N==NM1+1,
C  N==NP1-1, K<=NS2, KC+1<=N, 4<=N, 2NS2<=N,
N<=2NS2+1}

        FK = FLOAT(K-1)
        WSAVE(K) = 2.D0*DSIN(FK*DT)
        WSAVE(KC) = 2.D0*DCOS(FK*DT)
101       CONTINUE
C       CALL RFFTI (NM1,WSAVE(N+1))
```

Scientific Computing

I.2.1

```
      SUBROUTINE COSTI (N,WSAVE)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DIMENSION  WSAVE(*)
      PARAMETER (PI = 3.14159265358979)
      IF (N .LE. 3) THEN
      RETURN
      ENDIF
      NM1 = N-1
      NP1 = N+1
      NS2 = N/2
      DT = PI/FLOAT(NM1)
      KC = NP1-1
      DO 101 K=2,NS2
        KC = KC-1
        FK = FLOAT(K-1)
        WSAVE(K) = 2.D0*DSIN(FK*DT)
        WSAVE(KC) = 2.D0*DCOS(FK*DT)
  101 CONTINUE
C     CALL RFFTI (NM1,WSAVE(N+1))
      RETURN
      END
```

```
C  P(KC,NM1,NP1,NS2){KC==N, KC==NM1+1,
C  KC==NP1-1, 4<=KC, 2NS2<=KC, KC<=2NS2+1}

    DO 101 K = 2, NS2
      KC = KC-1

C  P(K,KC,NM1,NP1,NS2){K+KC==N+1, N==NM1+1,
C  N==NP1-1, K<=NS2, KC+1<=N, 4<=N, 2NS2<=N,
N<=2NS2+1}

      FK = FLOAT(K-1)
      WSAVE(K) = 2.D0*DSIN(FK*DT)
      WSAVE(KC) = 2.D0*DCOS(FK*DT)
101     CONTINUE
```

Scientific Computing

I.2.1

```
      SUBROUTINE COSTI (N,WSAVE)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DIMENSION  WSAVE(*)
      PARAMETER (PI = 3.14159265358979)
      IF (N .LE. 3) THEN
      RETURN
      ENDIF
      NM1 = N-1
      NP1 = N+1
      NS2 = N/2
      DT = PI/FLOAT(NM1)
      KC = NP1-1
      DO 101 K=2,NS2
        KC = KC-1
        FK = FLOAT(K-1)
        WSAVE(K) = 2.D0*DSIN(FK*DT)
        WSAVE(KC) = 2.D0*DCOS(FK*DT)
  101 CONTINUE
C     CALL RFFTI (NM1,WSAVE(N+1))
      RETURN
      END
```

```
C  P(KC,NM1,NP1,NS2){KC==N, KC==NM1+1,
C  KC==NP1-1, 4<=KC, 2NS2<=KC, KC<=2NS2+1}

      DO 101 K = 2, NS2
        KC = KC-1

C  P(K,KC,NM1,NP1,NS2){K+KC==N+1, N==NM1+1,
C  N==NP1-1, K<=NS2, KC+1<=N, 4<=N, 2NS2<=N,
N<=2NS2+1}
```

Dependence Test for WSAVE:
K == KC , K+KC==N+1
2K == N+1, K<=NS2,
N+1 <= 2NS2 , 2NS2<=N
N+1 <= N

```
        FK = FLOAT(K-1)
        WSAVE(K) = 2.D0*DSIN(FK*DT)
        WSAVE(KC) = 2.D0*DCOS(FK*DT)
101     CONTINUE
```

Modularity: Preconditions and Transformers

I.3.1

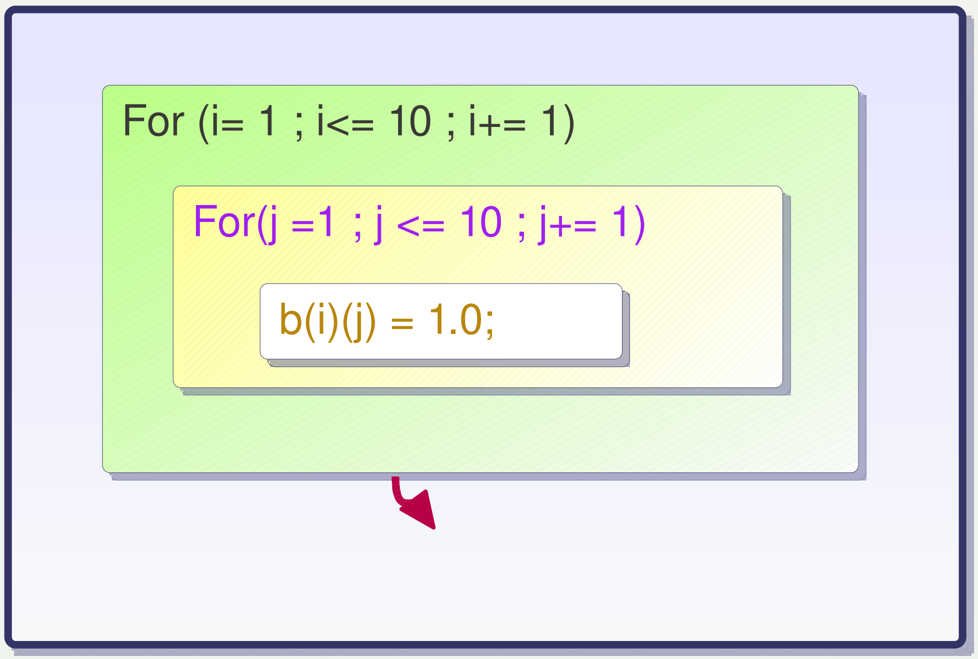
```
int main()
{
  float a[10][10], b[10][10], h;
  int i, j;

  for(i = 1; i <= 10; i += 1)

    for(j = 1; j <= 10; j += 1)

      b[i][j] = 1.0;

  h = 2.0;

  func1(10, 10, a, b, h);

  for(i = 1; i <= 10; i += 1)

    for(j = 1; j <= 10; j += 1)

      fprintf(stderr, "a[%d] = %f \n", i, a[i][j]);
}
```

Modularity: Preconditions and Transformers

I.3.1

```
// P() {}
int main()
{
   float a[10][10], b[10][10], h;
   int i, j;
// P() {}
   for(i = 1; i <= 10; i += 1)
// P(i,j) {1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
// P(i,j) {1<=i, i<=10, 1<=j, j<=10}
         b[i][j] = 1.0;
// P(i,j) {i==11, j==11}
   h = 2.0;
// P(h,i,j) {2.0==h, i==11, j==11}
   func1(10, 10, a, b, h);
// P(h,i,j) {2.0==h, i==11, j==11}
   for(i = 1; i <= 10; i += 1)
// P(h,i,j) {2.0==h, 1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
// P(h,i,j) {2.0==h, 1<=i, i<=10, 1<=j, j<=10}
         fprintf(stderr, "a[%d] = %f \n", i, a[i][j]);
}
```

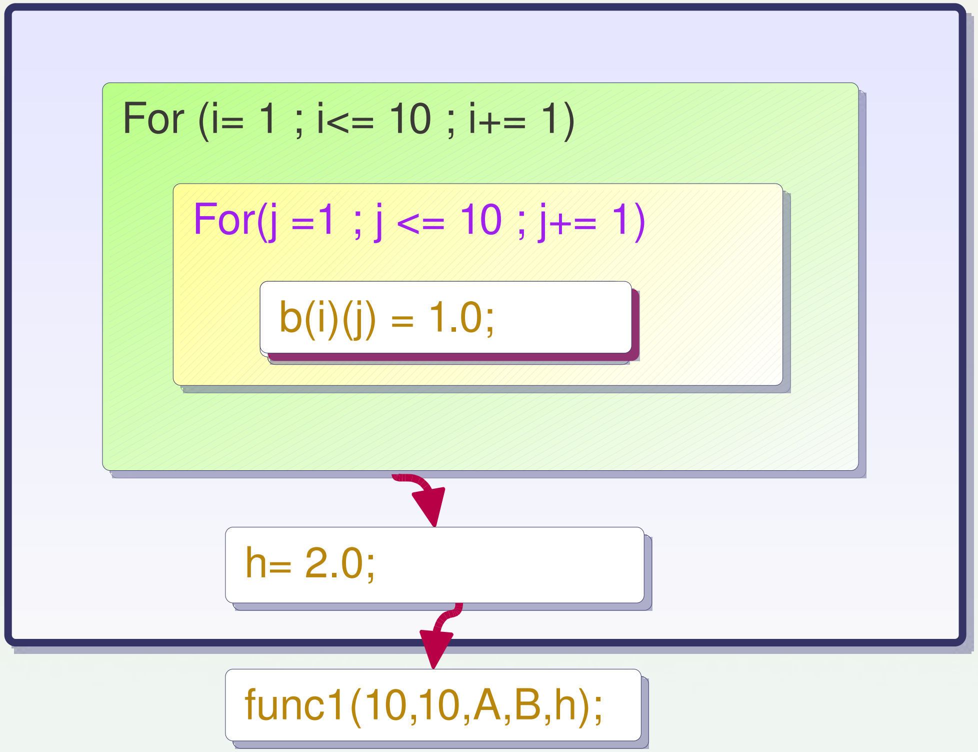
Modularity: Preconditions and Transformers




```
// P() {}
int main()
{
   float a[10][10], b[10][10], h;
   int i, j;
// P() {}
   for(i = 1; i <= 10; i += 1)
// P(i,j) {1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
// P(i,j) {1<=i, i<=10, 1<=j, j<=10}
         b[i][j] = 1.0;
// P(i,j) {i==11, j==11}
   h = 2.0;
// P(h,i,j) {2.0==h, i==11, j==11}
   func1(10, 10, a, b, h);
// P(h,i,j) {2.0==h, i==11, j==11}
   for(i = 1; i <= 10; i += 1)
// P(h,i,j) {2.0==h, 1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
// P(h,i,j) {2.0==h, 1<=i, i<=10, 1<=j, j<=10}
         fprintf(stderr, "a[%d] = %f \n", i, a[i][j]);
}
```

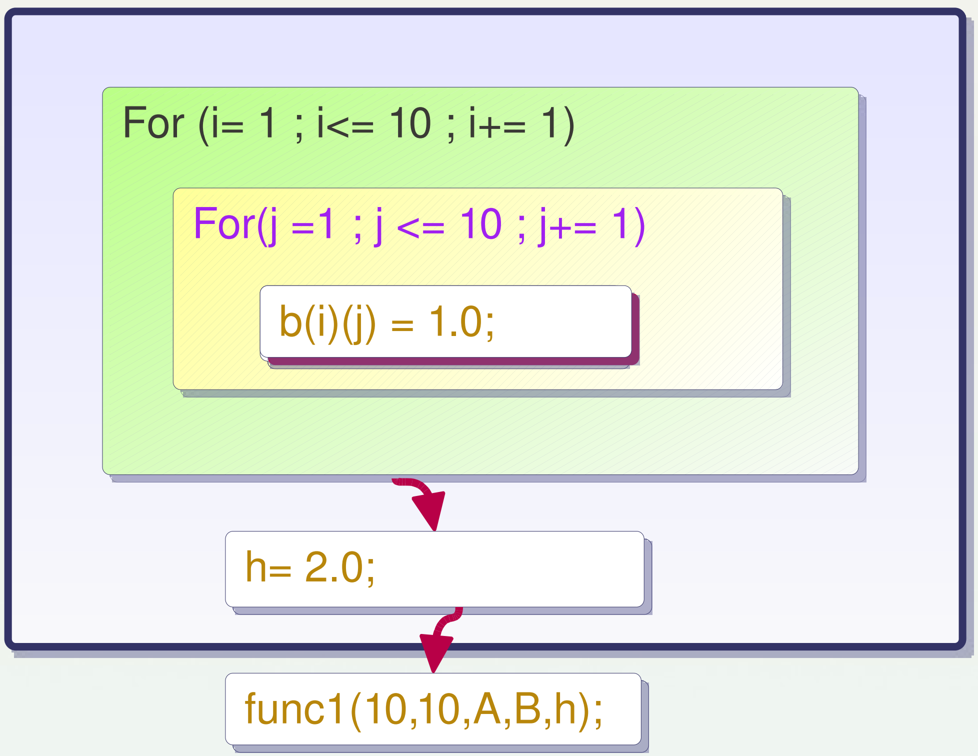
Modularity: Preconditions and Transformers

I.3.1




```
//  P() {}
int main()
{
   float a[10][10], b[10][10], h;
   int i, j;
//  P() {}
   for(i = 1; i <= 10; i += 1)
//  P(i,j) {1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
//  P(i,j) {1<=i, i<=10, 1<=j, j<=10}
         b[i][j] = 1.0;
//  P(i,j) {i==11, j==11}
   h = 2.0;
//  P(h,i,j) {2.0==h, i==11, j==11}
   func1(10, 10, a, b, h);
//  P(h,i,j) {2.0==h, i==11, j==11}
   for(i = 1; i <= 10; i += 1)
//  P(h,i,j) {2.0==h, 1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
//  P(h,i,j) {2.0==h, 1<=i, i<=10, 1<=j, j<=10}
         fprintf(stderr, "a[%d] = %f \n", i, a[i][j]);
}
```

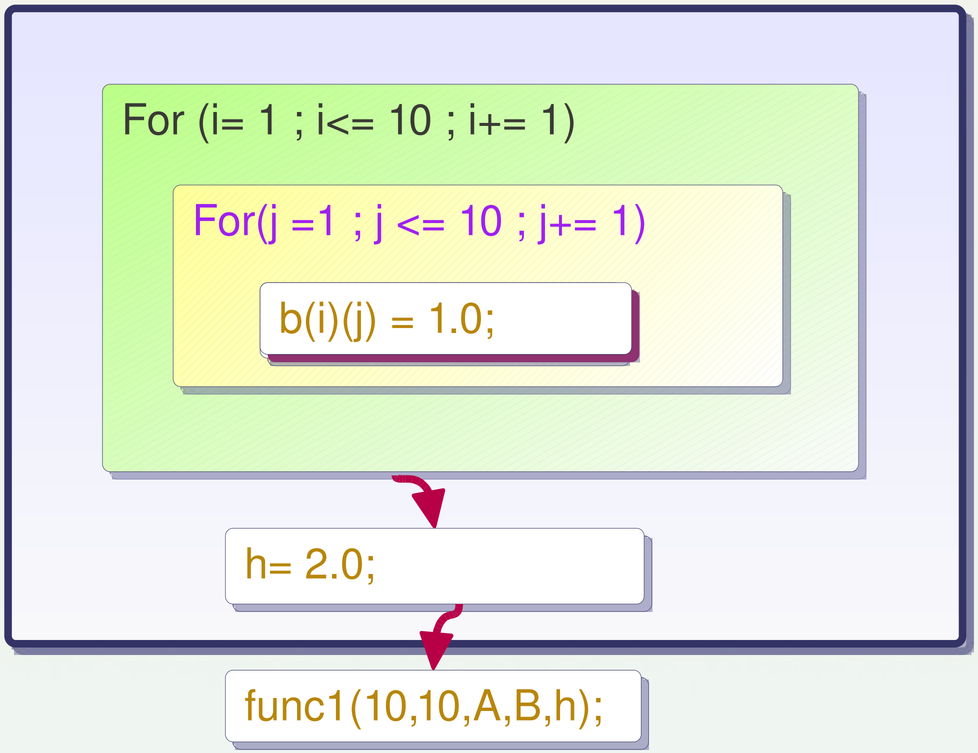
Modularity: Preconditions and Transformers




```
// P() {}
int main()
{
   float a[10][10], b[10][10], h;
   int i, j;
// P() {}
   for(i = 1; i <= 10; i += 1)
// P(i,j) {1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
// P(i,j) {1<=i, i<=10, 1<=j, j<=10}
         b[i][j] = 1.0;
// P(i,j) {i==11, j==11}
   h = 2.0;
// P(h,i,j) {2.0==h, i==11, j==11}
   func1(10, 10, a, b, h);
// P(h,i,j) {2.0==h, i==11, j==11}
   for(i = 1; i <= 10; i += 1)
// P(h,i,j) {2.0==h, 1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
// P(h,i,j) {2.0==h, 1<=i, i<=10, 1<=j, j<=10}
         fprintf(stderr, "a[%d] = %f \n", i, a[i][j]);
}
```

Modularity: Preconditions and Transformers

I.3.1




```
//  P() {}
int main()
{
   float a[10][10], b[10][10], h;
   int i, j;
//  P() {}
   for(i = 1; i <= 10; i += 1)
//  P(i,j) {1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
//  P(i,j) {1<=i, i<=10, 1<=j, j<=10}
         b[i][j] = 1.0;
//  P(i,j) {i==11, j==11}
   h = 2.0;
//  P(h,i,j) {2.0==h, i==11, j==11}
   func1(10, 10, a, b, h);
//  P(h,i,j) {2.0==h, i==11, j==11}
   for(i = 1; i <= 10; i += 1)
//  P(h,i,j) {2.0==h, 1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
//  P(h,i,j) {2.0==h, 1<=i, i<=10, 1<=j, j<=10}
         fprintf(stderr, "a[%d] = %f \n", i, a[i][j]);
}
```

Principle: Each Function is Analyzed Once
Summaries must be built

Affine Transitive Closure Algorithm

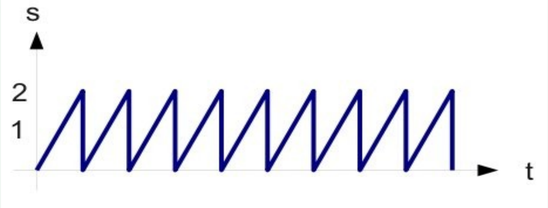
II.0.1

Affine Transitive Closure Algorithm

Car safety

II.1.1

```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```

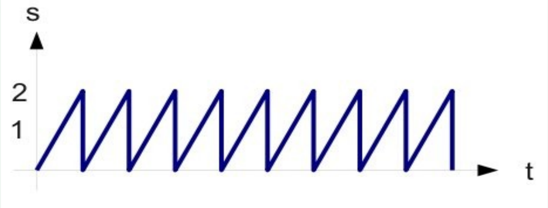


Car safety

II.1.1

```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```

T(d,s) {d==d'+1, s==s'+1, 0<=n, t<=n, s<=3}



Bottom-up propagation of Transformers

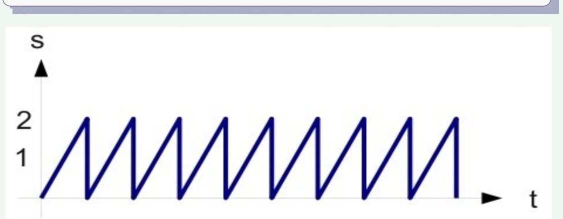
Car safety

II.1.1

```
void voiture05(int n)
{
 int s = 0, t = 0, d = 0;

 assert(n>=0);
 while(s <= 2 && t <= n) {
  if(alea() > 0.)
    t++, s = 0;
  else
    d++, s++;
 }

 if(d <= 2*n+3)
  printf("safe");
 else
  printf("crashed!");
}
```

T(s,t) {s==0, t==t'+1, 0<=n, t<=n+1, s'<=2}

T(d,s) {d==d'+1, s==s'+1, 0<=n, t<=n, s<=3}



Bottom-up propagation of Transformers

Car safety

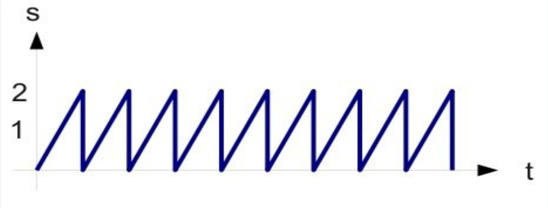
II.1.1

```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```

T(d,s,t) {d+t==d'+t'+1, 0<=n, t'<=n, s'+3t'+1<=s+3t, s+3t<=3t'+3, t'<=t, t<=t'+1}

T(s,t) {s==0, t==t'+1, 0<=n, t<=n+1, s'<=2}

T(d,s) {d==d'+1, s==s'+1, 0<=n, t<=n, s<=3}



Bottom-up propagation of Transformers

Car safety

II.1.1

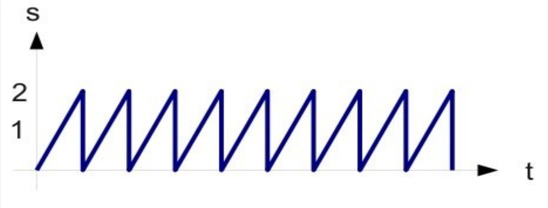
```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```

T(d,s,t) {d'==0, s'==0, t'==0, 0<=d, s<=3d, s<=2d+2t, d<=s+2t, 0<=n, t<=n, s<=2, 0<=t}

T(d,s,t) {d+t==d'+t'+1, 0<=n, t'<=n, s'+3t'+1<=s+3t, s+3t<=3t'+3, t'<=t, t<=t'+1}

T(s,t) {s==0, t==t'+1, 0<=n, t<=n+1, s'<=2}

T(d,s) {d==d'+1, s==s'+1, 0<=n, t<=n, s<=3}



Bottom-up propagation of Transformers

Car safety

II.1.1

Top-down propagation of Preconditions

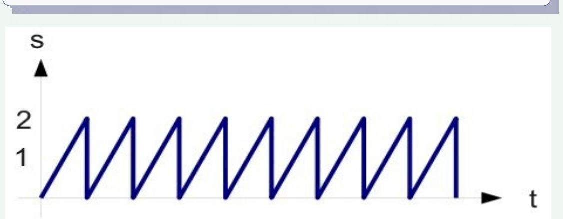
```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```

P(d,s,t) {d==0, s==0, t==0, 0<=n}

T(d,s,t) {d+t==d'+t'+1, 0<=n, t'<=n, s'+3t'+1<=s+3t, s+3t<=3t'+3, t'<=t, t<=t'+1}

T(s,t) {s==0, t==t'+1, 0<=n, t<=n+1, s'<=2}

T(d,s) {d==d'+1, s==s'+1, 0<=n, t<=n, s<=3}



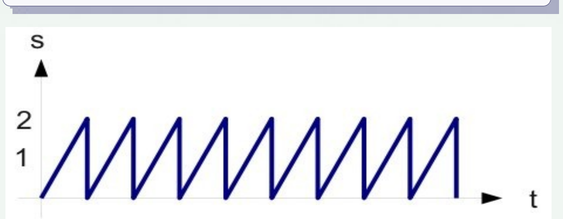
Bottom-up propagation of Transformers

Car safety

II.1.1

Top-down propagation of Preconditions

```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```



P(d,s,t) {d==0, s==0, t==0, 0<=n}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, **t<=n,** **s<=2,** 0<=t}

T(s,t) {s==0, t==t'+1, 0<=n, t<=n+1, s'<=2}

T(d,s) {d==d'+1, s==s'+1, 0<=n, t<=n, s<=3}

Bottom-up propagation of Transformers

Car safety

II.1.1

Top-down propagation of Preconditions

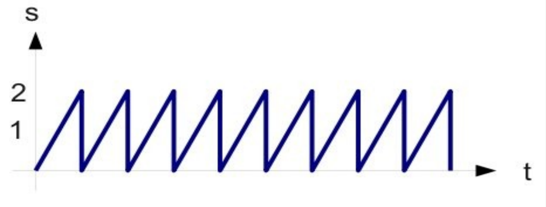
```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```

P(d,s,t) {d==0, s==0, t==0, 0<=n}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, **t<=n, s<=2,** 0<=t}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, t<=n, s<=2, 0<=t}

T(d,s) {d==d'+1, s==s'+1, 0<=n, t<=n, s<=3}



Bottom-up propagation of Transformers

Car safety

Top-down propagation of Preconditions

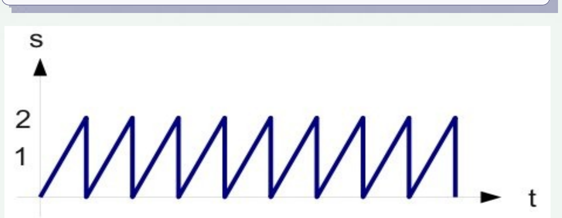
```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```

P(d,s,t) {d==0, s==0, t==0, 0<=n}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, **t<=n, s<=2,** 0<=t}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, t<=n, s<=2, 0<=t}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, t<=n, s<=2, 0<=t}

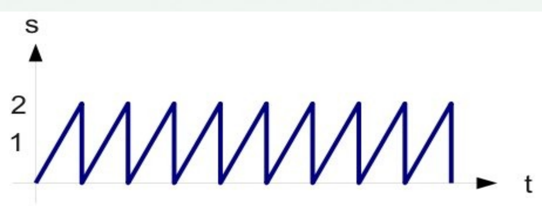


Bottom-up propagation of Transformers

Car safety

Top-down propagation of Preconditions

```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```



P(d,s,t) {d==0, s==0, t==0, 0<=n}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, **t<=n, s<=2,** 0<=t}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, t<=n, s<=2, 0<=t}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, t<=n, s<=2, 0<=t}

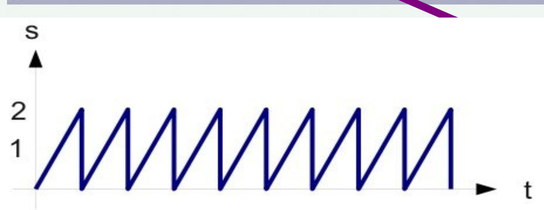
P(d,s,t) {0<=d, 5s<=9d, s+3<=3d+3t, d<=s+2t, s+3t<=3n+3, **t<=n+1, s<=3**, 3<=s+3t, 2<=s+2t}

Bottom-up propagation of Transformers

Car safety

Top-down propagation of Preconditions

```
void voiture05(int n)
{
  int s = 0, t = 0, d = 0;

  assert(n>=0);
  while(s <= 2 && t <= n) {
    if(alea() > 0.)
      t++, s = 0;
    else
      d++, s++;
  }

  if(d <= 2*n+3)
    printf("safe");
  else
    printf("crashed!");
}
```



P(d,s,t) {d==0, s==0, t==0, 0<=n}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, **t<=n,** **s<=2,** 0<=t}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, t<=n, s<=2, 0<=t}

P(d,s,t) {0<=d, s<=3d, s<=2d+2t, d<=s+2t, t<=n, s<=2, 0<=t}

P(d,s,t) {0<=d, 5s<=9d, s+3<=3d+3t, d<=s+2t, s+3t<=3n+3, **t<=n+1,** **s<=3**, 3<=s+3t, 2<=s+2t}

P() {0==-1}

Algorithm for Transitive Closure of Transformers

$\delta x^i = x^i - x^{i-1}$

Algorithm for Transitive Closure of Transformers

$\delta x^i = x^i - x^{i-1}$

$$x^k = x^0 + \sum_{i=1, k} \delta x^i$$

Algorithm for Transitive Closure of Transformers

II.2.1

$\delta x^i = x^i - x^{i-1}$

$$x^k = x^0 + \sum_{i=1,\, k} \delta x^i$$

$T(x^{i-1}, x^i)$

Algorithm for Transitive Closure of Transformers

II.2.1

$\delta x^i = x^i - x^{i-1}$

$$x^k = x^0 + \sum_{i=1, k} \delta x^i$$

$$T(x^{i-1}, x^i) \longrightarrow T'(\delta x^i) = \left\{ \delta x^i \mid \begin{array}{l} A\,\delta x^i = b \\ A'\,\delta x^i \leqslant b' \end{array} \right\}$$

Algorithm for Transitive Closure of Transformers

II.2.1

$\delta x^i = x^i - x^{i-1}$

$x^k = x^0 + \sum_{i=1,k} \delta x^i$ $\xrightarrow{\times A}$

$$A\,x^k = A\,x^0 + \sum_{i=1,k} A\,\delta x^i$$

$$A'\,x^k = A'\,x^0 + \sum_{i=1,k} A'\,\delta x^i$$

$T(x^{i-1}, x^i) \longrightarrow T'(\delta x^i) = \left\{ \delta x^i \mid \begin{array}{l} A\,\delta x^i = b \\ A'\,\delta x^i \leqslant b' \end{array} \right\}$

Algorithm for Transitive Closure of Transformers

II.2.1

$\delta x^i = x^i - x^{i-1}$

$x^k = x^0 + \sum_{i=1,k} \delta x^i$ —(× A)→

$A\, x^k = A\, x^0 + \sum_{i=1,k} A\, \delta x^i$

$A'\, x^k = A'\, x^0 + \sum_{i=1,k} A'\, \delta x^i$

$T(x^{i-1}, x^i) \rightarrow T'(\delta x^i) = \left\{ \delta x^i \mid A\, \delta x^i = b,\ A'\, \delta x^i \leqslant b' \right\}$ →

$A\, x^k = A\, x^0 + k\, b$

$A'\, x^k \leqslant A'\, x^0 + k\, b'$

Algorithm for Transitive Closure of Transformers

II.2.1

$\delta x^i = x^i - x^{i-1}$

$x^k = x^0 + \sum_{i=1,k} \delta x^i$ → $\times A$ →

$$A x^k = A x^0 + \sum_{i=1,k} A\,\delta x^i$$
$$A' x^k = A' x^0 + \sum_{i=1,k} A'\,\delta x^i$$

$T(x^{i-1}, x^i) \rightarrow T'(\delta x^i) = \left\{ \delta x^i \mid A\,\delta x^i = b,\ A'\,\delta x^i \leqslant b' \right\}$ →

$$A x^k = A x^0 + k\,b$$
$$A' x^k \leqslant A' x^0 + k\,b'$$

$$T^*(x^0, x) \subseteq \{ (x^0, x) \mid \exists k \in [0, \infty[\quad A x = A x^0 + kb \ \wedge\ A' x \leqslant A' x^0 + kb' \}$$

From Transformer T* to Precondition

II.3.1

- **Loop body Precondition from transformer T*:**

$$P^*(x^0) \subseteq \{ x \mid T^*(x^0,x) \}$$

- **Using T^+ instead of T*:**

$$P^* = P^0 \cup T^+(P^0)$$

- **Information loss due to convex hulls can be postponed even more:**

$$P^* = P^0 \cup T(P^0) \cup T^{2+}(P^0) \cup T(T^{2+}(P^0))$$

Improving Preconditions

Improving Preconditions

Periodic Behavior

Is it possible to parallelize Loop i ?

```
void flipflop(n,m)
{
  double x[2][m];
  int old = 0, new = 1, i, t = 0;

  while(t++<n) {
   for(i=0;i<m-1;i++)
     x[new][i] = g(x[old][i+1]);
   old = new;
   new = 1 - old;
  }
```

Periodic Behavior

Is it possible to parallelize Loop i ?

```
void flipflop(n,m)
{
  double x[2][m];
  int old = 0, new = 1, i, t = 0;

  while(t++<n) {
    for(i=0;i<m-1;i++)
      x[new][i] = g(x[old][i+1]);
    old = new;
    new = 1 - old;
  }
```

```
delete flipflop1

create flipflop1 flipflop1.c

setproperty
SEMANTICS_COMPUTE_TRANSFORMERS_IN_CONTEXT TRUE

setproperty SEMANTICS_FIX_POINT_OPERATOR "derivative"
setproperty PRETTYPRINT_ANALYSES_WITH_LF FALSE

echo TRANSFORMERS

activate PRINT_CODE_TRANSFORMERS
display PRINTED_FILE[flipflop]

echo PRECONDITIONS

activate PRINT_CODE_PRECONDITIONS
display PRINTED_FILE[flipflop]
```

Periodic Behavior


```
void flipflop(n,m)
{
  double x[2][m];
  int old = 0, new = 1, i, t = 0;

  while(t++<n) {
   for(i=0;i<m-1;i++)
     x[new][i] = g(x[old][i+1]);
   old = new;
   new = 1 - old;
  }
```

Is it possible to parallelize Loop i ?

```
void flipflop(int n, int m)
{
  double x[2][m];
  int old = 0, new = 1, i, t = 0;
// P(i,new,old,t) {new==1, old==0, t==0}

  while (t++<n) {
// P(i,new,old,t) {new+old==1, t<=n, 1<=t}
    for(i = 0; i <= m-2; i += 1)
// P(i,new,old,t) {new+old==1, 0<=i, i+2<=m, t<=n, 1<=t}
      x[new][i] = g(x[old][i +1]);
// P(i,new,old,t) {new+old==1, 0<=i, m<=i, t<=n, 1<=t}
    old = new;
// P(i,new,old,t) {new==old, 0<=i, m<=i, t<=n, 1<=t}
    new = 1-old;
// P(i,new,old,t) {new+old==1, 0<=i, m<=i, t<=n, 1<=t}
  }
```

Periodic Behavior

Is it possible to parallelize Loop i ?

```
void flipflop(n,m)
{
  double x[2][m];
  int old = 0, new = 1, i, t = 0;

  while(t++<n) {
   for(i=0;i<m-1;i++)
     x[new][i] = g(x[old][i+1]);
   old = new;
   new = 1 - old;
  }
```

Dependence test for x:
new == old
new == old+1
i' == i+1
$0 \leq i' \leq m - 2$
$0 \leq i \leq m - 2$
==> no solution

```
void flipflop(int n, int m)
{
  double x[2][m];
  int old = 0, new = 1, i, t = 0;
//  P(i,new,old,t) {new==1, old==0, t==0}

  while (t++<n) {
//  P(i,new,old,t) {new+old==1, t<=n, 1<=t}
     for(i = 0; i <= m-2; i += 1)
//  P(i,new,old,t) {new+old==1, 0<=i, i+2<=m, t<=n, 1<=t}
        x[new][i] = g(x[old][i +1]);
//  P(i,new,old,t) {new+old==1, 0<=i, m<=i, t<=n, 1<=t}
     old = new;
//  P(i,new,old,t) {new==old, 0<=i, m<=i, t<=n, 1<=t}
     new = 1-old;
//  P(i,new,old,t) {new+old==1, 0<=i, m<=i, t<=n, 1<=t}
  }
```

Virtual Unrolling

III.2.1

Virtual Unrolling


```
#include <stdio.h>
#include <assert.h>

void flipflop(n)
{
  int t, new,old;
  t=0,new=0,old=1;

  while (t<n) {
    new = 1 - new;
    old = 1 - old;
    t++;
  }
  assert(new+old==1);
}

int main(){
  int n;
  scanf("%d", &n);
  flipflop(n);
}
```

Virtual Unrolling


```
#include <stdio.h>
#include <assert.h>

void flipflop(n)
{
  int t, new,old;
  t=0,new=0,old=1;

  while (t<n) {
    new = 1 - new;
    old = 1 - old;
    t++;
  }
  assert(new+old==1);
}

int main(){
  int n;
  scanf("%d", &n);
  flipflop(n);
}
```

```
void flipflop(int n)
{
   int t, new, old;
//  P(new,old,t) {}
   t = 0, new = 0, old = 1;
//  P(new,old,t) {new==0, old==1, t==0}
   while (t<n) {
//  P(new,old,t) {t+1<=n, 0<=t}
      new = 1-new;
//  P(new,old,t) {1<=n, t+1<=n, 0<=t}
      old = 1-old;
//  P(new,old,t) {1<=n, t+1<=n, 0<=t}
      t++;
   }
//  P(new,old,t) {n<=t, 0<=t}
   new+old==1?(void) 0:__assert_fail("new+old==1", "./flipflop2.c",
14, (const char *) 0);
}
```

Virtual Unrolling

III.2.1

```
#include <stdio.h>
#include <assert.h>

void flipflop(n)
{
  int t, new,old;
  t=0,new=0,old=1;

  while (t<n) {
    new = 1 - new;
    old = 1 - old;
    t++;
  }
  assert(new+old==1);
}

int main(){
  int n;
  scanf("%d", &n);
  flipflop(n);
}
```

```
//  T() {}
void flipflop(int n)
{
//  T(new,old,t) {}
   int t, new, old;
//  T(new,old,t) {new==0, old==1, t==0}
   t = 0, new = 0, old = 1;
//  T(new,old,t) {new'==0, old'==1, t'==0, 1<=n, t+1<=n, 0<=t}
   while (t<n) {
//  T(new) {new+new'==1, 1<=n, t+1<=n}
      new = 1-new;
//  T(old) {old+old'==1, 1<=n, t+1<=n}
      old = 1-old;
//  T(t) {t==t'+1, 1<=n, t<=n}
      t++;
   }
}
```

Virtual Unrolling


```
#include <stdio.h>
#include <assert.h>

void flipflop(n)
{
  int t, new,old;
  t=0,new=0,old=1;

  while (t<n-1) {
    new = 1 - new;
    old = 1 - old;
    t++;
    new = 1 - new;
    old = 1 - old;
    t++;
  }
  if (t<n) {
    new = 1 - new;
    old = 1 - old;
    t++;
  }
    assert(new+old==1);
}
```

```
//  T() {}
void flipflop(int n)
{
//  T(new,old,t) {}
   int t, new, old;
//  T(new,old,t) {new==0, old==1, t==0}
   t = 0, new = 0, old = 1;
//  T(new,old,t) {new'==0, old'==1, t'==0, 1<=n, t+1<=n, 0<=t}
   while (t<n) {
//  T(new) {new+new'==1, 1<=n, t+1<=n}
      new = 1-new;
//  T(old) {old+old'==1, 1<=n, t+1<=n}
      old = 1-old;
//  T(t) {t==t'+1, 1<=n, t<=n}
      t++;
   }
}
```

Virtual Unrolling


```
#include <stdio.h>
#include <assert.h>

void flipflop(n)
{
 int t, new,old;
 t=0,new=0,old=1;

 while (t<n-1) {
  new = 1 - new;
  old = 1 - old;
  t++;
  new = 1 - new;
  old = 1 - old;
  t++;
 }
 if (t<n) {
  new = 1 - new;
  old = 1 - old;
  t++;
 }
  assert(new+old==1);
}
```

```
// P(new,old,t) {new==0, old==1, t==0}
  while (t<n-1) {
// P(new,old,t) {new==0, old==1, t+2<=n, 0<=t}
     new = 1-new;
// P(new,old,t) {new==1, old==1, 2<=n, t+2<=n, 0<=t}
     old = 1-old;
// P(new,old,t) {new==1, old==0, 2<=n, t+2<=n, 0<=t}
     t++;
// P(new,old,t) {new==1, old==0, 2<=n, t+1<=n, 1<=t}
     new = 1-new;
// P(new,old,t) {new==0, old==0, 2<=n, t+1<=n, 1<=t}
     old = 1-old;
// P(new,old,t) {new==0, old==1, 2<=n, t+1<=n, 1<=t}
     t++;
  }
// P(new,old,t) {new==0, old==1, n<=t+1, 0<=t}
  if (t<n) {
// P(new,old,t) {n==t+1, new==0, old==1, 1<=n}
     new = 1-new;
// P(new,old,t) {n==t+1, new==1, old==1, 1<=n}
     old = 1-old;
// P(new,old,t) {n==t+1, new==1, old==0, 1<=n}
     t++;  }
```

Virtual Unrolling


```
#include <stdio.h>
#include <assert.h>

void flipflop(n)
{
  int t, new,old;
  t=0,new=0,old=1;

  while (t<n-1) {
    new = 1 - new;
    old = 1 - old;
    t++;
    new = 1 - new;
    old = 1 - old;
    t++;
  }
  if (t<n) {
    new = 1 - new;
    old = 1 - old;
    t++;
  }
    assert(new+old==1);
}
```

```
delete flipflop2b
create flipflop2b flipflop2.c

setproperty
SEMANTICS_COMPUTE_TRANSFORMERS_IN_CONTEXT TRUE
setproperty PRETTYPRINT_ANALYSES_WITH_LF FALSE
setproperty SEMANTICS_FIX_POINT_OPERATOR "derivative"

setproperty SEMANTICS_K_FIX_POINT 2

echo TRANSFORMERS

activate PRINT_CODE_TRANSFORMERS
display PRINTED_FILE[flipflop]

echo PRECONDITIONS

activate PRINT_CODE_PRECONDITIONS
display PRINTED_FILE[flipflop]

close
quit
```

```
      t++;   }
```

Virtual Unrolling

III.2.1

```
#include <stdio.h>
#include <assert.h>

void flipflop(n)
{
  int t, new,old;
  t=0,new=0,old=1;

  while (t<n-1) {
    new = 1 - new;
    old = 1 - old;
    t++;
    new = 1 - new;
    old = 1 - old;
    t++;
  }
  if (t<n) {
    new = 1 - new;
    old = 1 - old;
    t++;
  }
    assert(new+old==1);
}
```

```
delete flipflop2b
create flipflop2b flipflop2.c

setproperty SEMANTICS_COMPUTE_TRANSFORMERS_IN_CONTEXT TRUE
setproperty PRETTYPRINT_ANALYSES_WITH_LF FALSE
setproperty SEMANTICS_FIX_POINT_OPERATOR "derivative"

setproperty SEMANTICS_K_FIX_POINT 2

echo TRANSFORMERS

activate PRINT_CODE_TRANSFORMERS
display PRINTED_FILE[flipflop]

echo PRECONDITIONS

activate PRINT_CODE_PRECONDITIONS
display PRINTED_FILE[flipflop]

close
quit
```

$$P^* = P^0 \cup T(P^0) \cup T^{2+}(P^0) \cup T(T^{2+}(P^0))$$

```
      t++;   }
```

Virtual Unrolling


```
#include <stdio.h>
#include <assert.h>

void flipflop(n)
{
  int t, new,old;
  t=0,new=0,old=1;

  while (t<n) {
    new = 1 - new;
    old = 1 - old;
    t++;
  }
  assert(new+old==1);
}

int main(){
  int n;
  scanf("%d", &n);
  flipflop(n);
}
```

```
delete flipflop2b
create flipflop2b flipflop2.c

setproperty
SEMANTICS_COMPUTE_TRANSFORMERS_IN_CONTEXT TRUE
setproperty PRETTYPRINT_ANALYSES_WITH_LF FALSE
setproperty SEMANTICS_FIX_POINT_OPERATOR "derivative"

setproperty SEMANTICS_K_FIX_POINT 2

echo TRANSFORMERS

activate PRINT_CODE_TRANSFORMERS
display PRINTED_FILE[flipflop]

echo PRECONDITIONS

activate PRINT_CODE_PRECONDITIONS
display PRINTED_FILE[flipflop]

close
quit
```

$$P^* = P^0 \cup T(P^0) \cup T^{2+}(P^0) \cup T(T^{2+}(P^0))$$

```
t++;   }
```

Virtual Unrolling


```
#include <stdio.h>
#include <assert.h>

void flipflop(n)
{
  int t, new,old;
  t=0,new=0,old=1;

  while (t<n) {
    new = 1 - new;
    old = 1 - old;
    t++;
  }
  assert(new+old==1);
}

int main(){
  int n;
  scanf("%d", &n);
  flipflop(n);
}
```

```
void flipflop(int n)
{
   int t, new, old;
//  P(new,old,t) {}
   t = 0, new = 0, old = 1;
//  P(new,old,t) {new==0, old==1, t==0}
   while (t<n) {
//  P(new,old,t) {new+old==1, t+1<=n, 0<=new, new<=1, new<=t}
      new = 1-new;
//  P(new,old,t) {new==old, 1<=n, t+1<=n, 0<=new, new<=1,
1<=new+t}
      old = 1-old;
//  P(new,old,t) {new+old==1, 1<=n, t+1<=n, 0<=new, new<=1,
1<=new+t}
      t++;
   }
}
```

```
      t++;   }
```

Iterative Refinment


```
int main()
{
  int i = 0, j = 0, n;
  if(n<0) exit(1);
  while(i<=n) {
    i++;
    j+=i;
  }
}
```

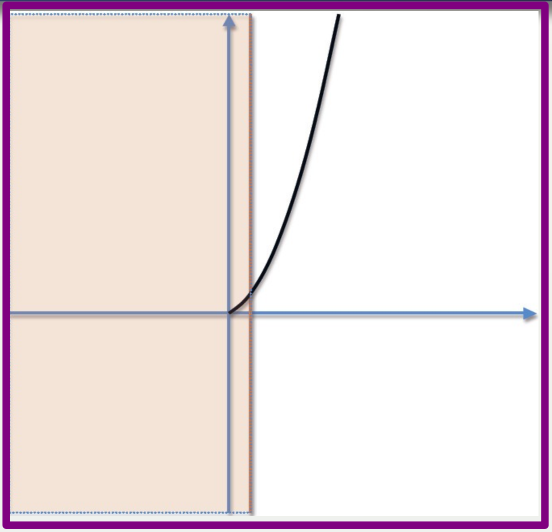
Iterative Refinment


```
int main()
{
  int i = 0, j = 0, n;
  if(n<0) exit(1);
  while(i<=n) {
    i++;
    j+=i;
  }
}
```

```
//  T(i,j) {i'==0, j'==0, 0<=i, i<=n, 0<=n}
   while (i<=n) {
//  T(i) {i==i'+1, i<=n+1, 0<=n}
      i++;
//  T(j) {i+j'==j, i<=n+1, 0<=n}
      j += i;
   }
//  T() {i==n+1, 1<=i}
```

Iterative Refinment


```
int main()
{
  int i = 0, j = 0, n;
  if(n<0) exit(1);
  while(i<=n) {
    i++;
    j+=i;
  }
}
```

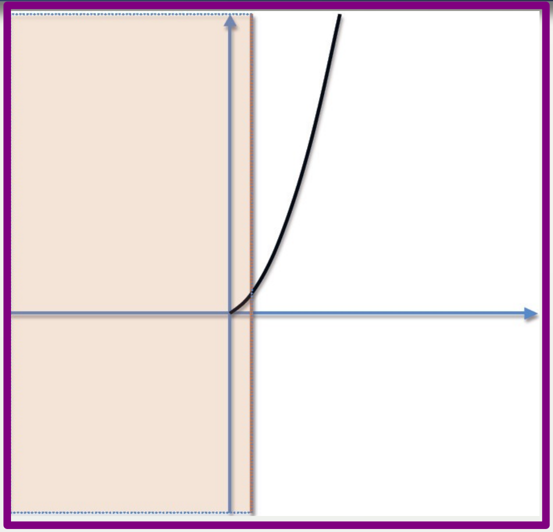


```
   int i = 0, j = 0, n;
//  P(i,j,n) {i==0, j==0}
   if (n<0)
//  P(i,j,n) {i==0, j==0, n+1<=0}
      exit(1);
//  P(i,j,n) {i==0, j==0, 0<=n}
   while (i<=n) {
//  P(i,j,n) {0<=i, i<=n}
      i++;
//  P(i,j,n) {1<=i, i<=n+1, 0<=n}
      j += i;
   }
//  P(i,j,n) {i==n+1, 1<=i}
```

```
//  T(i,j) {i'==0, j'==0, 0<=i, i<=n, 0<=n}
   while (i<=n) {
//  T(i) {i==i'+1, i<=n+1, 0<=n}
      i++;
//  T(j) {i+j'==j, i<=n+1, 0<=n}
      j += i;
   }
//  T() {i==n+1, 1<=i}
```

Iterative Refinment


```
int main()
{
  int i = 0, j = 0, n;
  if(n<0) exit(1);
  while(i<=n) {
    i++;
    j+=i;
  }
}
```

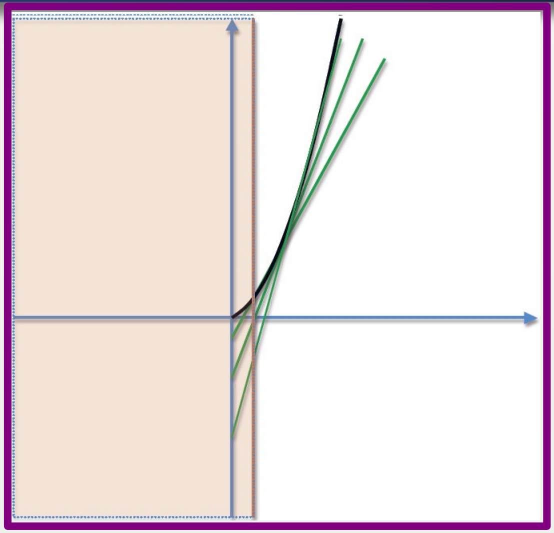


```
  int i = 0, j = 0, n;
//  P(i,j,n) {i==0, j==0}
  if (n<0)
//  P(i,j,n) {i==0, j==0, n+1<=0}
     exit(1);
//  P(i,j,n) {i==0, j==0, 0<=n}
  while (i<=n) {
//  P(i,j,n) {0<=i, i<=n}
     i++;
//  P(i,j,n) {1<=i, i<=n+1, 0<=n}
     j += i;
  }
//  P(i,j,n) {i==n+1, 1<=i}
```

```
//  T(i,j) {i'==0, j'==0, 0<=i, i<=j, 2i<=j+1, i<=n, 0<=n}
  while (i<=n) {
//  T(i) {i==i'+1, 1<=i, i<=n+1, 0<=n}
     i++;
//  T(j) {i+j'==j, 1<=i, i<=n+1, 0<=n}
     j += i;
  }
//  T() {i==n+1, 1<=i, 2i<=j+1, 3i<=j+3}
```

Iterative Refinment


```
int main()
{
  int i = 0, j = 0, n;
  if(n<0) exit(1);
  while(i<=n) {
    i++;
    j+=i;
  }
}
```



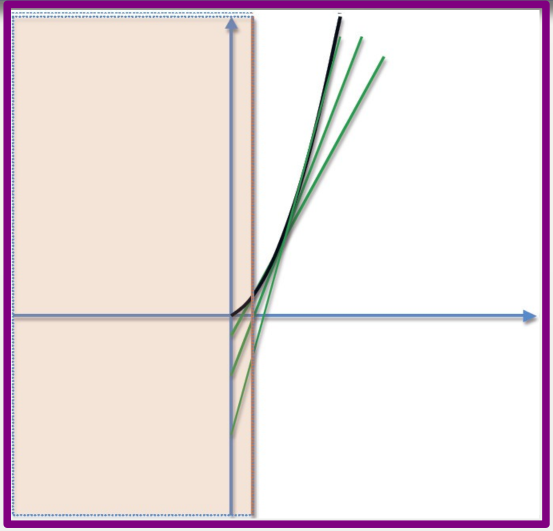
```
  int i = 0, j = 0, n;
// P(i,j,n) {i==0, j==0}
  if (n<0)
// P(i,j,n) {i==0, j==0, n+1<=0}
    exit(1);
// P(i,j,n) {i==0, j==0, 0<=n}
  while (i<=n) {
// P(i,j,n) {0<=i, i<=j, 2i<=j+1, 3i<=j+3, i<=n}
    i++;
// P(i,j,n) {1<=i, i<=j+1, 2i<=j+3, 3i<=j+6,
i<=n+1, 0<=n}
    j += i;
  }
// P(i,j,n) {i==n+1, 1<=i, 2i<=j+1, 3i<=j+3,
4i<=j+6}
```

```
// T(i,j) {i'==0, j'==0, 0<=i, i<=j, 2i<=j+1, i<=n, 0<=n}
  while (i<=n) {
// T(i) {i==i'+1, 1<=i, i<=n+1, 0<=n}
    i++;
// T(j) {i+j'==j, 1<=i, i<=n+1, 0<=n}
    j += i;
  }
// T() {i==n+1, 1<=i, 2i<=j+1, 3i<=j+3}
```

Iterative Refinment

III.3.1

```
int main()
{
  int i = 0, j = 0, n;
  if(n<0) exit(1);
  while(i<=n) {
    i++;
    j+=i;
  }
}
```



```
  int i = 0, j = 0, n;
// P(i,j,n) {i==0, j==0}
  if (n<0)
// P(i,j,n) {i==0, j==0, n+1<=0}
    exit(1);
// P(i,j,n) {i==0, j==0, 0<=n}
  while (i<=n) {
// P(i,j,n) {0<=i, i<=j, 2i<=j+1, 3i<=j+3, i<=n}
    i++;
// P(i,j,n) {1<=i, i<=j+1, 2i<=j+3, 3i<=j+6,
i<=n+1, 0<=n}
    j += i;
  }
// P(i,j,n) {i==n+1, 1<=i, 2i<=j+1, 3i<=j+3,
4i<=j+6}
```

```
// T(i,j) {i'==0, j'==0, 0<=i, i<=j, 2i<=j+1, 3i<=j+3,
//   4i<=j+6, i<=n, 0<=n}
  while (i<=n) {
// T(i) {i==i'+1, 1<=i, i<=j+1, 2i<=j+3, 3i<=j+6, i<=n+1, 0<=n}
    i++;
// T(j) {i+j'==j, 1<=i, i<=j'+1, 2i<=j'+3, 3i<=j'+6,
//   i<=n+1, 0<=n}
    j += i;
  }
// T() {i==n+1, 1<=i, 2i<=j+1, 3i<=j+3, 4i<=j+6, 5i<=j+10}
```

Iterative Refinment

III.3.1

```
int main()
{
  int i = 0, j = 0, n;
  if(n<0) exit(1);
  while(i<=n) {
    i++;
    j+=i;
  }
}
```

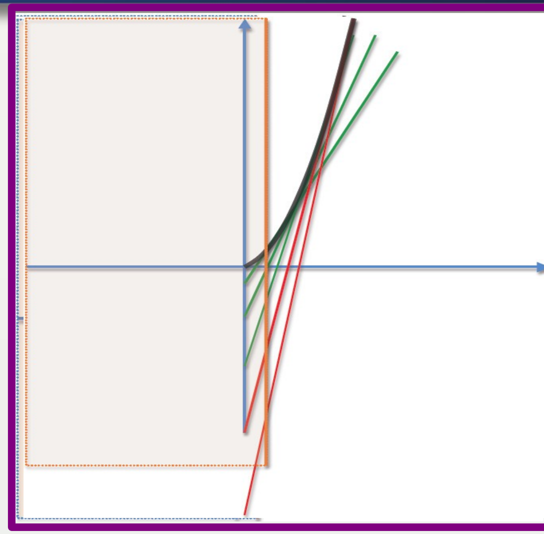
```
  int i = 0, j = 0, n;
//  P(i,j,n) {i==0, j==0}
  if (n<0)
//  P(i,j,n) {i==0, j==0, n+1<=0}
     exit(1);
//  P(i,j,n) {i==0, j==0, 0<=n}
  while (i<=n) {
//  P(i,j,n) {0<=i, i<=j, 2i<=j+1, 3i<=j+3,
4i<=j+6, 5i<=j+10, i<=n}
     i++;
//  P(i,j,n) {1<=i, i<=j+1, 2i<=j+3, 3i<=j+6,
4i<=j+10, 5i<=j+15,
//     i<=n+1, 0<=n}
     j += i;
  }
//  P(i,j,n) {i==n+1, 1<=i, 2i<=j+1, 3i<=j+3,
4i<=j+6, 5i<=j+10,6i<=j+15}
```

```
//  T(i,j) {i'==0, j'==0, 0<=i, i<=j, 2i<=j+1, 3i<=j+3,
//    4i<=j+6, i<=n, 0<=n}
   while (i<=n) {
//  T(i) {i==i'+1, 1<=i, i<=j+1, 2i<=j+3, 3i<=j+6, i<=n+1, 0<=n}
     i++;
//  T(j) {i+j'==j, 1<=i, i<=j'+1, 2i<=j'+3, 3i<=j'+6,
//    i<=n+1, 0<=n}
     j += i;
   }
//  T() {i==n+1, 1<=i, 2i<=j+1, 3i<=j+3, 4i<=j+6, 5i<=j+10}
```

Iterative Refinment

III.3.1

```
int main()
{
  int i = 0, j = 0, n;
  if(n<0) exit(1);
  while(i<=n) {
    i++;
    j+=i;
  }
}
```



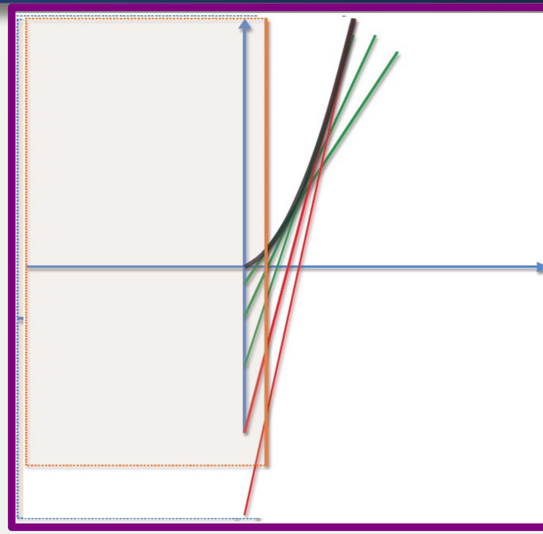
```
  int i = 0, j = 0, n;
// P(i,j,n) {i==0, j==0}
  if (n<0)
// P(i,j,n) {i==0, j==0, n+1<=0}
     exit(1);
// P(i,j,n) {i==0, j==0, 0<=n}
  while (i<=n) {
// P(i,j,n) {0<=i, i<=j, 2i<=j+1, 3i<=j+3,
4i<=j+6, 5i<=j+10, i<=n}
     i++;
// P(i,j,n) {1<=i, i<=j+1, 2i<=j+3, 3i<=j+6,
4i<=j+10, 5i<=j+15,
//   i<=n+1, 0<=n}
     j += i;
  }
// P(i,j,n) {i==n+1, 1<=i, 2i<=j+1, 3i<=j+3,
4i<=j+6, 5i<=j+10,6i<=j+15}
```

```
// T(i,j) {i'==0, j'==0, 0<=i, i<=j, 2i<=j+1, 3i<=j+3,
//   4i<=j+6, 5i<=j+10, 6i<=j+15, i<=n, 0<=n}
  while (i<=n) {
// T(i) {i==i'+1, 1<=i, i<=j+1, 2i<=j+3, 3i<=j+6, 4i<=j+10,
//   5i<=j+15, i<=n+1, 0<=n}
     i++;
// T(j) {i+j'==j, 1<=i, i<=j'+1, 2i<=j'+3, 3i<=j'+6,
//   4i<=j'+10, 5i<=j'+15, i<=n+1, 0<=n}
     j += i;
  }
// T() {..., 2i<=j+1, 3i<=j+3, 4i<=j+6, 5i<=j+10,
6i<=j+15,7i<=j+21}
```

Iterative Refinment

III.3.1

```
int main()
{
  int i = 0, j = 0, n;
  if(n<0) exit(1);
  while(i<=n) {
    i++;
    j+=i;
  }
}
```



```
  if (n<0)
// P(i,j,n) {i==0, j==0, n+1<=0}
     exit(1);
// P(i,j,n) {i==0, j==0, 0<=n}
  while (i<=n) {
// P(i,j,n) {0<=i, i<=j, 2i<=j+1, 3i<=j+3,
4i<=j+6, 5i<=j+10,
//    6i<=j+15, 7i<=j+21, i<=n}
     i++;
// P(i,j,n) {1<=i, i<=j+1, 2i<=j+3, 3i<=j+6,
4i<=j+10, 5i<=j+15,
//    6i<=j+21, 7i<=j+28, i<=n+1, 0<=n}
     j += i;
  }
// P(i,j,n) {i==n+1, 1<=i, 2i<=j+1, 3i<=j+3,
4i<=j+6, 5i<=j+10, 6i<=j+15, 7i<=j+21,
8i<=j+28}
```

```
// T(i,j) {i'==0, j'==0, 0<=i, i<=j, 2i<=j+1, 3i<=j+3,
//   4i<=j+6, 5i<=j+10, 6i<=j+15, i<=n, 0<=n}
  while (i<=n) {
// T(i) {i==i'+1, 1<=i, i<=j+1, 2i<=j+3, 3i<=j+6, 4i<=j+10,
//   5i<=j+15, i<=n+1, 0<=n}
     i++;
// T(j) {i+j'==j, 1<=i, i<=j'+1, 2i<=j'+3, 3i<=j'+6,
//   4i<=j'+10, 5i<=j'+15, i<=n+1, 0<=n}
     j += i;
  }
// T() {..., 2i<=j+1, 3i<=j+3, 4i<=j+6, 5i<=j+10,
6i<=j+15,7i<=j+21}
```

$$T^*_{n+1} = \tau(B, P^*_n) \wedge P^*_n$$

Control Restructuring


```
while {
  if(t) a;
  else b;
}
```

→

```
while(c) {
  while (c&&t) a;
  while (c&&!t) b;
}
```

Control Restructuring

Example – Gulwani 2007

```
#include <stdio.h>

int main()
{
  int x,y,z;

  x=0;
  y=50;
  while(x<100) {
   if (x<50)
     x++;
   else  {
     x++; y++;
   }
  }
  if (y==100)
   printf("property  verified\n");
}
```

```
#include <stdio.h>
int main() {
  int x,y,z;

  x=0;
  y=50;
  while(x<100)
   {
     while ( x<50)
       x++;
     while (x<100 && x>=50){
       x++; y++;
     }
   }
  if (x ==100 && y==100)
   printf("property  verified\n");
}
```

Control Restructuring

Example – Gulwani 2007

```
//  T(x,y) {x'==0, y'==50, x<=99, y<=x+50,
x+2450<=50y, 50<=y}
   while (x<100)
//  T(x,y) {x==x'+1, x+50y'<=50y+50, y'<=y, y<=y'+1}
      if (x<50)
//  T(x) {x==x'+1, x<=50}
         x++;
      else {
//  T(x,y) {x==x'+1, y==y'+1, 51<=x, x<=100}
         // BEGIN BLOCK
//  T(x) {x==x'+1, 51<=x, x<=100}
         x++;
//  T(y) {y==y'+1, 51<=x, x<=100}
         y++;
         // END BLOCK
      }
```

```
//  T() {x==0, y==50}
   while (x<100) {
//  T(x,y) {x==100, x'+y<=y'+100, y'+1<=y, y<=y'+50}
      // BEGIN BLOCK
//  T(x) {x<=49, x'<=x, x'<=49}
      while (x<50)
//  T(x) {x==x'+1, x<=50}
         x++;
//  T(x,y) {x+y'==x'+y, 50<=x, x<=99, 50<=x', x'<=99,
//     y'<=y}
      while (x<100&&x>=50) {
//  T(x,y) {x==x'+1, y==y'+1, 51<=x, x<=100}
         // BEGIN BLOCK
//  T(x) {x==x'+1, 51<=x, x<=100}
         x++;
//  T(y) {y==y'+1, 51<=x, x<=100}
         y++;
      }
```

Control Restructuring

Postponing convex hull

Example – Gulwani 2007

```
// T(x,y) {x'==0, y'==50, x<=99, y<=x+50,
x+2450<=50y, 50<=y}
  while (x<100)
// T(x,y) {x==x'+1, x+50y'<=50y+50, y'<=y, y<=y'+1}
    if (x<50)
// T(x) {x==x'+1, x<=50}
      x++;
    else {
// T(x,y) {x==x'+1, y==y'+1, 51<=x, x<=100}
      // BEGIN BLOCK
// T(x) {x==x'+1, 51<=x, x<=100}
      x++;
// T(y) {y==y'+1, 51<=x, x<=100}
      y++;
      // END BLOCK
    }
```

```
// T() {x==0, y==50}
  while (x<100) {
// T(x,y) {x==100, x'+y<=y'+100, y'+1<=y, y<=y'+50}
    // BEGIN BLOCK
// T(x) {x<=49, x'<=x, x'<=49}
    while (x<50)
// T(x) {x==x'+1, x<=50}
      x++;
// T(x,y) {x+y'==x'+y, 50<=x, x<=99, 50<=x', x'<=99,
//   y'<=y}
    while (x<100&&x>=50) {
// T(x,y) {x==x'+1, y==y'+1, 51<=x, x<=100}
      // BEGIN BLOCK
// T(x) {x==x'+1, 51<=x, x<=100}
      x++;
// T(y) {y==y'+1, 51<=x, x<=100}
      y++;
    }
```

Control Restructuring

Example – Gulwani 2007

```
// T(x,y) {x'==0, y'==50, x<=99, y<=x+50,
x+2450<=50y, 50<=y}
   while (x<100)
// T(x,y) {x==x'+1, x+50y'<=50y+50, y'<=y, y<=y'+1}
      if (x<50)
// T(x) {x==x'+1, x<=50}
         x++;
      else {
// T(x,y) {x==x'+1, y==y'+1, 51<=x, x<=100}
         // BEGIN BLOCK
// T(x) {x==x'+1, 51<=x, x<=100}
         x++;
// T(y) {y==y'+1, 51<=x, x<=100}
         y++;
         // END BLOCK
      }
```

```
// T() {
  while
// T(x,y
    // B
// T(x)
    wh
// T(x)
      x
// T(x,y
//   y'<
    wh
// T(x,y
      //
// T(x)
      x
// T(y)
      y
    }
```

```
// P(x,y,z) {x==0, y==50}
   while (x<100) {

// P(x,y,z) {x==0, y==50}
      while (x<50)
// P(x,y,z) {y==50, 0<=x, x<=49}
         X++;

// P(x,y,z) {x==50, y==50}
      while (x<100&&x>=50) {
// P(x,y,z) {x==y, 50<=x, x<=99}
         x++;
// P(x,y,z) {x==y+1, 51<=x, x<=100}
         y++;
      }
   }

// P(x,y,z) {x==100, x==y}
   if (y==100)
      printf("property  verified\n");}
```

Control Restructuring

III.4.1

Example – Gulwani 2007

```
// P(x,y,z) {x==0, y==50}
   while (x<100)

// P(x,y,z) {x<=99, y<=x+50, x+1200<=25y, 50<=y}
      if (x<50)
// P(x,y,z) {x<=49, y<=x+50, x+1200<=25y, 50<=y}
         x++;
      else {
// P(x,y,z) {50<=x, x<=99, y<=x+50, x+1200<=25y,50<=y}
         x++;
//P(x,y,z) {51<=x, x<=100, y<=x+49, x+1199<=25y, 50<=y}
         y++;
      }

// P(x,y,z) {x==100, 53<=y, y<=150}
   if (y==100)
      printf("property  verified\n");
}
```

```
// P(x,y,z) {x==0, y==50}
   while (x<100) {

// P(x,y,z) {x==0, y==50}
      while (x<50)
// P(x,y,z) {y==50, 0<=x, x<=49}
         X++;

// P(x,y,z) {x==50, y==50}
      while (x<100&&x>=50) {
// P(x,y,z) {x==y, 50<=x, x<=99}
         x++;
// P(x,y,z) {x==y+1, 51<=x, x<=100}
         y++;
      }
   }

// P(x,y,z) {x==100, x==y}
   if (y==100)
      printf("property  verified\n");}
```

Related Work

IV.0.1

- Many examples from Chaochen, Gonnord, Gopan, Gulavani, Gulwani, Halbwachs, Merchat,... processed successfully
- Gonnord: abstract acceleration
- Kelly & al.: transitive closure computation of relation encoded by a Presburger formulae. Heuristic uses the *d-form* relation
- Bielecki & al.: exact non linear transitive closure for normalized relations, written as systems of recurrence equations
- Paige & Koenig: finite differencing of source code
  - *Extension to predicates over arrays ?*
- Spezialetti & Gupta: monotonicity analysis

Conclusion

IV.1.1

- **New transitive closure algorithm, implemented in PIPS**
- **Input : programs in Fortran and C**
  - Large scientific codes up to hundreds of functions, 100KLOCS
  - Bourdoncle's algorithm used to deal with unstructured control flow graph
    - But not optimal
  - Structured loops converted into while loops
- **Modular and interprocedural Analysis**
- **Results are equivalent to related work examples**
  - Halbwachs extended with symbolic time bound
  - Beyond counters: multiply, divide, affine assignments
- **Extensions for non affine program behavior:**
  - Periodic
  - Iterative transformer refinement using preconditions