Scalable Conditional Induction Variables (CIV) Analysis.

Cosmin E. Oancea and Lawrence Rauchwerger
cosmin.oancea@diku.dk, rwerger@cse.tamu.edu

University of Copenhagen and Texas A&M University

11th of February 2015

Setting the Stage

Low-level analysis of array subscripts typically assumes subscripts to be affine expressions of loop indices.

```
k = 0
DO i = 1, N
  k = k + 2
  A(k) = ...
ENDDO
```

- Loop-carried dependences on k,
- Any Dependences on A?

Ind Var $\Longrightarrow$ Subst

```
k = 0
DO i = 1, N
  A(2*i) = ...
ENDDO
k = MAX(0,2*N)
```

- substituting k→2*i:
1. eliminates dependences on k
2. allows easy reasoning for A: $i_1 \neq i_2 \;\Rightarrow\; 2 * i_1 \neq 2 * i_2$

Setting the Stage

Low-level analysis of array subscripts typically assumes subscripts to be affine expressions of loop indices.

```
k = 0
DO i = 1, N
  k = k + 2
  A(k) = ...
ENDDO
```

- Loop-carried dependences on k,
- Any Dependences on A?

Ind Var $\Longrightarrow$ Subst

```
k = 0
DO i = 1, N
  A(2*i) = ...
ENDDO
k = MAX(0,2*N)
```

- substituting k→2*i:

1. eliminates dependences on k
2. allows easy reasoning for A: $i_1 \neq i_2 \;\Rightarrow\; 2 * i_1 \neq 2 * i_2$

Setting the Stage

Low-level analysis of array subscripts typically assumes subscripts to be affine expressions of loop indices.

```
k = 0
DO i = 1, N
  k = k + 2
  A(k) = ...
ENDDO
```

- Loop-carried dependences on k,
- Any Dependences on A?

Ind Var $\Longrightarrow$ Subst

```
k = 0
DO i = 1, N
  A(2*i) = ...
ENDDO
k = MAX(0,2*N)
```

- substituting k→2*i:

1 eliminates dependences on k

2 allows easy reasoning for A: $i_1 \neq i_2 \;\Rightarrow\; 2 * i_1 \neq 2 * i_2$

Problem Statement & Related Work

Challenging case of subscripts using "conditional induction variables"
CIV: monotonic, but conditionally incremented (NO closed-form sol).

Filter, scan, push vector abstractions create CIV patterns.

Five out of thirty ($\sim$ 17%) benchmarks require CIV analysis.

```
civ = 0
DO i = 1, N
  IF ( B(i).GT.0 ) THEN
    civ = civ + 1
    A(civ) = ...
  ENDIF
ENDDO
```

Related work: specialized dependency test (at pair-of-accesses level)

- consecutively-written, single-index access pattern [Lin,Padua]
- accesses of shape {`X(civ)`, `X(civ+K)`} [Wu,Cohen,Padua]
- assume that CIV used only to index $\Rightarrow$ CIV computed at the end of loop.

Problem Statement & Related Work

Challenging case of subscripts using "conditional induction variables"
CIV: monotonic, but conditionally incremented (NO closed-form sol).

Filter, scan, push vector abstractions create CIV patterns.

Five out of thirty ($\sim 17\%$) benchmarks require CIV analysis.

```
civ = 0
DO i = 1, N
  IF ( B(i).GT.0 ) THEN
    civ = civ + 1
    A(civ) = ...
  ENDIF
ENDDO
```

Related work: specialized dependency test (at pair-of-accesses level)

- consecutively-written, single-index access pattern [Lin,Padua]
- accesses of shape {`X(civ)`, `X(civ+K)`} [Wu,Cohen,Padua]
- assume that CIV used only to index ⇒ CIV computed at the end of loop.

Problem Statement & Our Approach

Challenging case of subscripts using "conditional induction variables", i.e., monotonic, but conditionally incremented (NO closed-form sol).

```
civ = 0
DO i = 1, N
  IF ( B(i).GT.0 )
  THEN
    civ = civ + 1
    A(civ) = ...
  ENDIF
ENDDO
```

Key difference: see it as a summarization of array references problem

- CIV monotonicity $\Rightarrow$ summary monotonicity (?)
- constructive rather than existential proof,
- common representation for affine and CIV-based summary,
- dependency test is modeled as an equation on summaries & requires no modification,
- summary-based techniques better suited for larger loops.

Problem Statement: CIV computation

1 How to compute CIVs in parallel?

```
civ = civ0;
DO i = 1, N
  IF ( B(i).GT.0 ) THEN
    civ = civ ⊕ 1
    A(civ) = ...
  ENDIF
ENDDO
```

Conceptually, parallel CIV computation is:

```
X ← map(\b→if b > 0 then 1 else 0, B)
y ← scan^exc(⊕, n_el, X)
in map((⊕ civ0), Y)
```

$\oplus$: any associative operator,
n_{el} its neutral element.

Also solves the cases when `civ` is **not** used for indexing!

$\text{map}(f, \{a_1, a_2, \ldots, a_n\}) \equiv \{f(a_1), f(a_2), \ldots, f(a_n)\}$

$\text{scan}^{exc}(\odot, e, \{a_1, a_2, \ldots, a_n\}) \equiv \{e, e \odot a_1, \ldots, e \odot a_1 \ldots \odot a_{n-1}\}$

Problem Statement: CIV computation

1 How to compute CIVs in parallel?

```
civ = civ0;
DO i = 1, N
  IF ( B(i).GT.0 ) THEN
    civ = civ ⊕ 1
    A(civ) = ...
  ENDIF
ENDDO
```

Conceptually, parallel CIV computation is:

```
X ← map(\b→if b > 0 then 1 else 0, B)
y ← scan^exc(⊕, n_el, X)
in map((⊕ civ0), Y)
```

$\oplus$: any associative operator,

n_{el} its neutral element.

Also solves the cases when `civ` is **not** used for indexing!

$$\text{map}(f, \{a_1, a_2, \ldots, a_n\}) \equiv \{f(a_1), f(a_2), \ldots, f(a_n)\}$$

$$\text{scan}^{exc}(\odot, e, \{a_1, a_2, \ldots, a_n\}) \equiv \{e, e \odot a_1, \ldots, e \odot a_1 \ldots \odot a_{n-1}\}$$

Problem Statement: Summary Computation

2 How to summarize CIV-based subscripts?

```
civ@1 = civ0
DO i = 1, N
  civ@2=γ(civ@1,civ@4)
  IF ( B(i).GT.0 ) THEN
    civ@3 = civ@2 + 1
    A(civ@3) = ...
  ELSE

  ENDIF
  civ@4=γ(civ@1,civ@4)
ENDDO
civ@5 = γ(civ@4,civ@1)
```

1 gated SSA representation

- CIV evolution on each path known ⇒

2 express each path summary in terms of civ@2 and civ@4

   a then: {civ$_3$}≡[civ@2+1,civ@4]

   b else: $\emptyset$ ≡ [civ@2+1,civ@4], because civ@2+1 > civ@4.

3 Iteration: W_i =[civ@2+1,civ@4] (all paths identical formula).

4 Loop: $\cup_{i=1}^{N} W_i$ = [civ@1+1,civ@5]

5 $\cup_{k=1}^{i-1} W_k$ = [civ@1+1,civ@4^{i-1}] = [civ@1+1,civ@2^{i}]

Problem Statement: Summary Computation

2 How to summarize CIV-based subscripts?

```
civ@1 = civ0
DO i = 1, N
  civ@2=γ(civ@1,civ@4)
  IF ( B(i).GT.0 ) THEN
    civ@3 = civ@2 + 1
    A(civ@3) = ...
  ELSE

  ENDIF
  civ@4=γ(civ@1,civ@4)
ENDDO
civ@5 = γ(civ@4,civ@1)
```

1. gated SSA representation

- CIV evolution on each path known ⇒

2. express each path summary in terms of civ@2 and civ@4
   - a then: $\{civ_3\} \equiv$ [civ@2+1,civ@4]
   - b else: $\emptyset \equiv$ [civ@2+1,civ@4], because civ@2+1 > civ@4.
3. Iteration: $W_i =$ [civ@2+1,civ@4] (all paths identical formula).
4. Loop: $\cup_{i=1}^{N} W_i =$ [civ@1+1,civ@5]
5. $\cup_{k=1}^{i-1} W_k =$ [civ@1+1,civ@4^{i-1}] $=$ [civ@1+1,civ@2^{i}]

Problem Statement: Summary Computation

2 How to summarize CIV-based subscripts?

```
civ@1 = civ0
DO i = 1, N
  civ@2=γ(civ@1,civ@4)
  IF ( B(i).GT.0 ) THEN
    civ@3 = civ@2 + 1
    A(civ@3) = ...
  ELSE

  ENDIF
  civ@4=γ(civ@1,civ@4)
ENDDO
civ@5 = γ(civ@4,civ@1)
```

1 gated SSA representation

- CIV evolution on each path known ⇒

2 express each path summary in terms of civ@2 and civ@4
   a then: $\{civ_3\} \equiv$ [civ@2+1,civ@4]
   b else: $\emptyset \equiv$ [civ@2+1,civ@4], because civ@2+1 > civ@4.

3 Iteration: $W_i =$ [civ@2+1,civ@4] (all paths identical formula).

4 Loop: $\cup_{i=1}^{N} W_i =$ [civ@1+1,civ@5]

5 $\cup_{k=1}^{i-1} W_k =$ [civ@1+1,civ@4^{i-1}] $=$ [civ@1+1,civ@2^{i}]

Problem Statement: Summary Computation

2 How to summarize CIV-based subscripts?

```
civ@1 = civ0
DO i = 1, N
  civ@2=γ(civ@1,civ@4)
  IF ( B(i).GT.0 ) THEN
    civ@3 = civ@2 + 1
    A(civ@3) = ...
  ELSE

  ENDIF
  civ@4=γ(civ@1,civ@4)
ENDDO
civ@5 = γ(civ@4,civ@1)
```

1 gated SSA representation

- CIV evolution on each path known ⇒

2 express each path summary in terms of civ@2 and civ@4
   a then: $\{civ_3\} \equiv$ [civ@2+1,civ@4]
   b else: $\emptyset \equiv$ [civ@2+1,civ@4], because civ@2+1 > civ@4.

3 Iteration: $W_i =$ [civ@2+1,civ@4] (all paths identical formula).

4 Loop: $\cup_{i=1}^{N} W_i =$ [civ@1+1,civ@5]

5 $\cup_{k=1}^{i-1} W_k =$ [civ@1+1,civ@4^{i-1}]
$=$ [civ@1+1,civ@2^{i}]

Problem Statement: Summary Computation

2 How to summarize CIV-based subscripts?

```
civ@1 = civ0
DO i = 1, N
  civ@2=γ(civ@1,civ@4)
  IF ( B(i).GT.0 ) THEN
    civ@3 = civ@2 + 1
    A(civ@3) = ...
  ELSE

  ENDIF
  civ@4=γ(civ@1,civ@4)
ENDDO
civ@5 = γ(civ@4,civ@1)
```

1 gated SSA representation

- CIV evolution on each path known ⇒

2 express each path summary in terms of civ@2 and civ@4
   a then: $\{civ_3\} \equiv$ [civ@2+1,civ@4]
   b else: $\emptyset \equiv$ [civ@2+1,civ@4], because civ@2+1 > civ@4.

3 Iteration: $W_i =$ [civ@2+1,civ@4] (all paths identical formula).

4 Loop: $\cup_{i=1}^{N} W_i =$ [civ@1+1,civ@5]

5 $\cup_{k=1}^{i-1} W_k =$ [civ@1+1,civ@4^{i-1}]
$=$ [civ@1+1,civ@2^{i}]

Problem Statement: Independence Equations

3 How to disambiguate CIV-based subscripts?

```
civ@1 = civ0
DO i = 1, N
  civ@2=γ(civ@1,civ@4)
  IF ( B(i).GT.0 ) THEN
    civ@3 = civ@2 + 1
    A(civ@3) = ...
  ELSE

  ENDIF
  civ@4=γ(civ@1,civ@4)
ENDDO
civ@5 = γ(civ@4,civ@1)
```

1 Loop independence by set equations:

- Output independence:
  $\cup_{i=1}^{n}(\cup_{k=1}^{i-1} W_k \cap W_i) = \emptyset$
- $\cup_{i=1}^{n}($ [civ@1+1,civ@2^i] $\cap$ [civ@2^i+1,civ@4^i] $) = \emptyset$

2 Other uses: per-iteration copy-in/out

Problem Statement: Independence Equations

3 How to disambiguate CIV-based subscripts?

```
civ@1 = civ0
DO i = 1, N
  civ@2=γ(civ@1,civ@4)
  IF ( B(i).GT.0 ) THEN
    civ@3 = civ@2 + 1
    A(civ@3) = ...
  ELSE

  ENDIF
  civ@4=γ(civ@1,civ@4)
ENDDO
civ@5 = γ(civ@4,civ@1)
```

1 Loop independence by set equations:

- Output independence: $\cup_{i=1}^{n}(\cup_{k=1}^{i-1} W_k \ \cap \ W_i) \ = \emptyset$
- $\cup_{i=1}^{n}(\ [\texttt{civ@1+1,civ@2}^i] \ \cap \ [\texttt{civ@2}^i\texttt{+1,civ@4}^i] \ ) = \emptyset$

2 Other uses: per-iteration copy-in/out

A Nontrivial Loop: CORREC_do401 (BDNA,PERFECT-CLUB)

```
civ@1 = Q
DO i = M, N, 1
 civ@2=γ(civ@1,civ@4)
 .. =  X(i) ..
 IF C(i) .GT. 0 THEN
  DO j = 1, C(i), 1
   IF(..)X(j+civ@2       )=..
   IF(..)X(j+civ@2+  C(i))=..
   IF(..)X(j+civ@2+2*C(i))=..
  ENDDO
  civ@3 = 3*C(i) + civ@2
 ENDIF
 civ@4=γ(civ@3,civ@2)
ENDDO
civ@5=γ(civ@4,civ@1)
```

- CIV may have non-constant evolution through loop
- CIV subscripts:
  - neither single indexed
  - nor consecutively written,
  - nor of shape: {X(civ), X(civ+K)}
- summary may contain holes ⇒ over/underestimate approx.

A Nontrivial Loop: CORREC_do401 (BDNA,PERFECT-CLUB)

```
civ@1 = Q
DO i = M, N, 1
 civ@2=γ(civ@1,civ@4)
 .. = X(i) ..
 IF C(i) .GT. 0 THEN
  DO j = 1, C(i), 1
   IF(..)X(j+civ@2       )=..
   IF(..)X(j+civ@2+  C(i))=..
   IF(..)X(j+civ@2+2*C(i))=..
  ENDDO
  civ@3 = 3*C(i) + civ@2
 ENDIF
 civ@4=γ(civ@3,civ@2)
ENDDO
civ@5=γ(civ@4,civ@1)
```

- CIV may have non-constant evolution through loop
- CIV subscripts:
  - neither single indexed
  - nor consecutively written,
  - nor of shape: {X(civ), X(civ+K)}
- summary may contain holes ⇒ over/underestimate approx.

A Nontrivial Loop: CORREC_do401 (BDNA,PERFECT-CLUB)

```
civ@1 = Q
DO i = M, N, 1
 civ@2=γ(civ@1,civ@4)
 .. =  X(i) ..
 IF C(i) .GT. 0 THEN
  DO j = 1, C(i), 1
   IF(..)X(j+civ@2       )=..
   IF(..)X(j+civ@2+  C(i))=..
   IF(..)X(j+civ@2+2*C(i))=..
  ENDDO
  civ@3 = 3*C(i) + civ@2
 ENDIF
 civ@4=γ(civ@3,civ@2)
ENDDO
civ@5=γ(civ@4,civ@1)
```

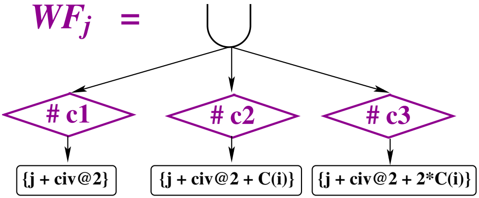
- CIV may have non-constant evolution through loop
- CIV subscripts:
  - neither single indexed
  - nor consecutively written,
  - nor of shape: {X(civ), X(civ+K)}
- summary may contain holes ⇒ over/underestimate approx.

Preliminaries: Exact USR Summarization (1)

Summaries (RO, RW, WF) are

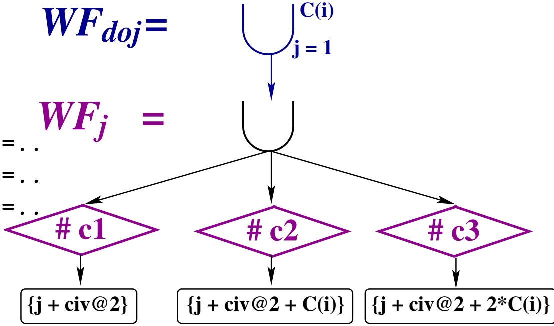
- constructed via a bottom-up parse of the AbSyn,
- structural data-flow equations dictate how to compose consecutive regions, aggregate/translate across loops/callsites, ...

```
IF(..)X(j+civ@2       )=..
IF(..)X(j+civ@2+  C(i))=..
IF(..)X(j+civ@2+2*C(i))=..
```



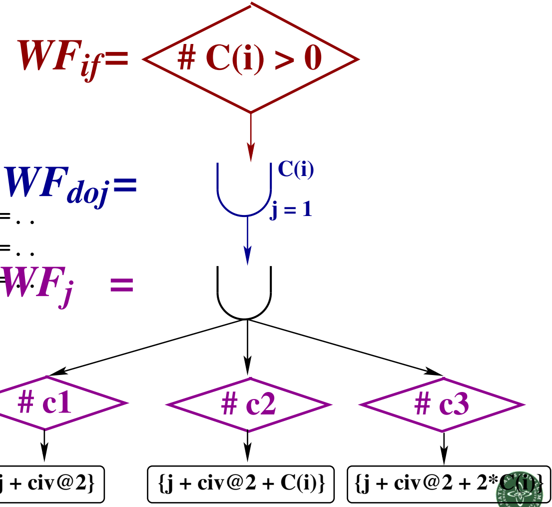
Preliminaries: Exact USR Summarization (2)

```
DO j = 1, C(i), 1
 IF(..)X(j+civ@2       )=..
 IF(..)X(j+civ@2+  C(i))=..
 IF(..)X(j+civ@2+2*C(i))=..
ENDDO
```



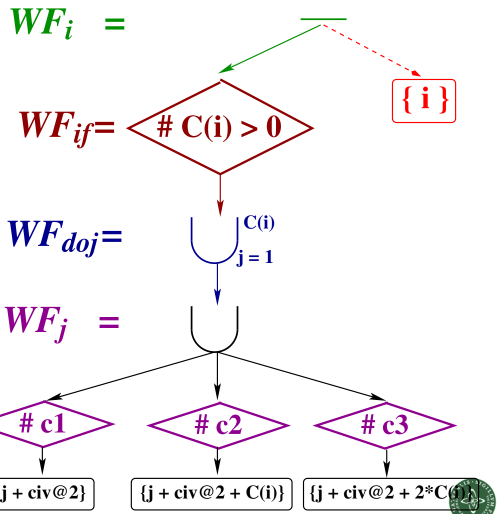
Preliminaries: Exact (USR) Summarization (3)

```
IF C(i) .GT. 0 THEN
 DO j = 1, C(i), 1
  IF(..)X(j+civ@2      )=..
  IF(..)X(j+civ@2+  C(i))=..
  IF(..)X(j+civ@2+2*C(i))=..
 ENDDO
 civ@3 = 3*C(i) + civ@2
ENDIF
```



Preliminaries: Exact (USR) Summarization (4)

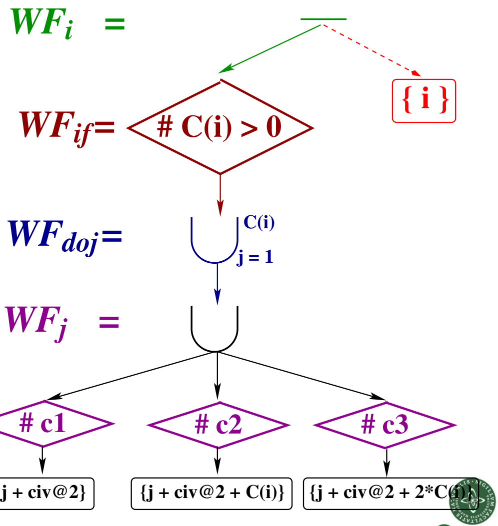
```
civ@1 = Q
DO i = M, N, 1
 civ@2=γ(civ@1,civ@4)
 .. =  X(i) ..
 IF C(i) .GT. 0 THEN
  DO j = 1, C(i), 1
   IF(..)X(j+civ@2       )=..
   IF(..)X(j+civ@2+  C(i))=..
   IF(..)X(j+civ@2+2*C(i))=..
  ENDDO
  civ@3 = 3*C(i) + civ@2
 ENDIF
 civ@4=γ(civ@3,civ@2)
ENDDO
civ@5=γ(civ@4,civ@1)
```



$RO_i = \{i\} - WF_{if}$ & $RW_i = WF_{if} \cap \{i\}$

Preliminaries: Exact (USR) Summarization (4)

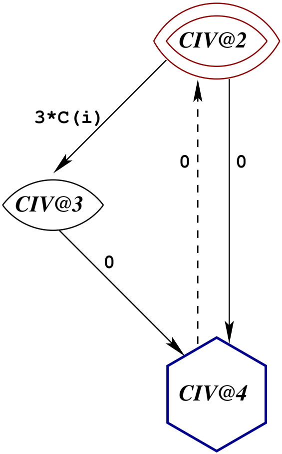
```
civ@1 = Q
DO i = M, N, 1
 civ@2=γ(civ@1,civ@4)
 .. =  X(i) ..
 IF C(i) .GT. 0 THEN
  DO j = 1, C(i), 1
   IF(..)X(j+civ@2       )=..
   IF(..)X(j+civ@2+  C(i))=..
   IF(..)X(j+civ@2+2*C(i))=..
  ENDDO
  civ@3 = 3*C(i) + civ@2
 ENDIF
 civ@4=γ(civ@3,civ@2)
ENDDO
civ@5=γ(civ@4,civ@1)
```



$$RO_i = \{i\} - WF_{if} \;\&\; RW_i = WF_{if} \cap \{i\}$$

Preliminaries: Value Evolution Graph (VEG)

```
civ@1 = Q
DO i = M, N, 1
 civ@2=γ(civ@1,civ@4)
 .. =  X(i) ..
 IF C(i) .GT. 0 THEN
  DO j = 1, C(i), 1
   IF(..)X(j+civ@2       )=..
   IF(..)X(j+civ@2+  C(i))=..
   IF(..)X(j+civ@2+2*C(i))=..
  ENDDO
  civ@3 = 3*C(i) + civ@2
 ENDIF
 civ@4=γ(civ@3,civ@2)
ENDDO
civ@5=γ(civ@4,civ@1)
```



VEG constructed at loop and subroutine call level.

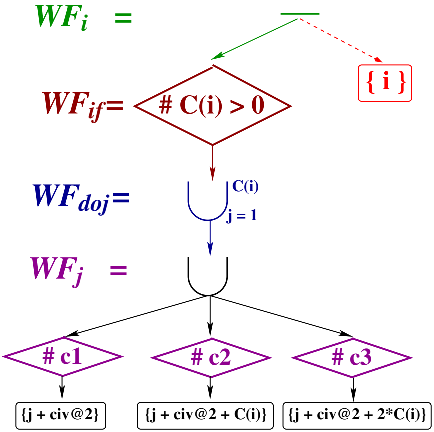
Represents the flow of values between gated-SSA CIV names.

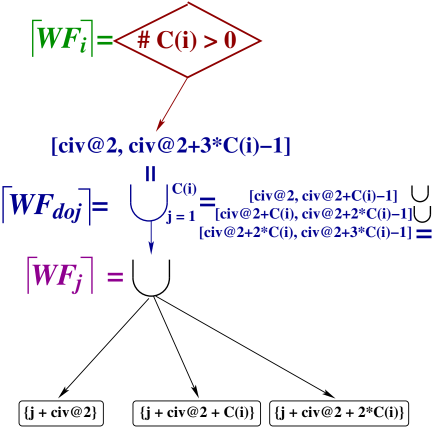
CIV-Summarization

Refinement of exact summarization: computes over/underestimates

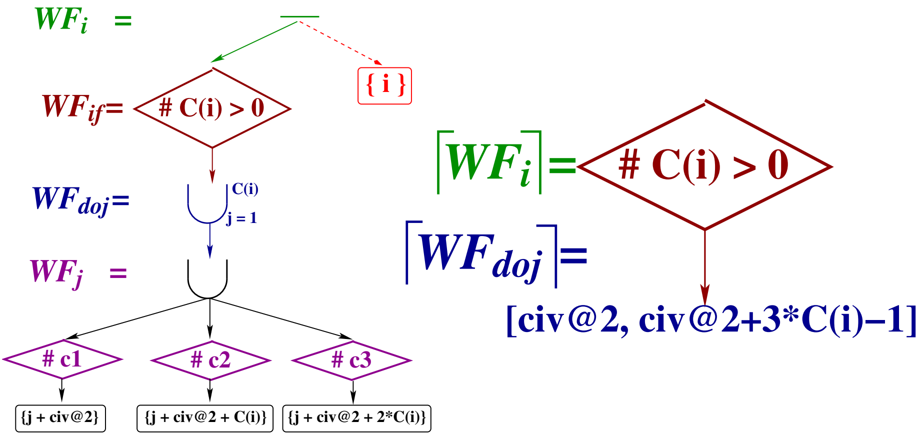
1. Approximate USR with a union of gated intervals.
2. Associate each gated interval with a VEG node.
3. Summarize each path in terms of start and end CIV node.
   - for underestimate check that the condition of the path
   - implies the gates of each of the interval on that path.
4. Merge across all paths of an iteration.
5. Total/partial aggregation across loops.

1. Union of Gated-Intervals Approximation



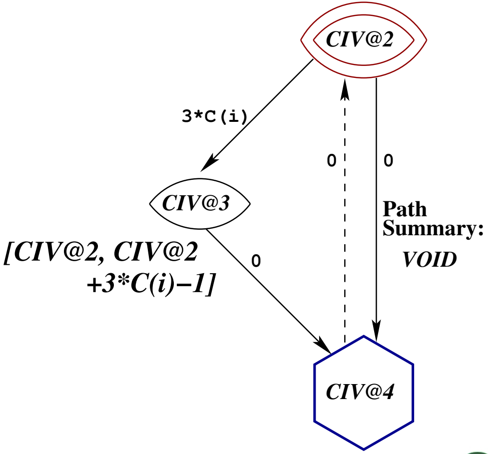


1. Union of Gated-Intervals Approximation



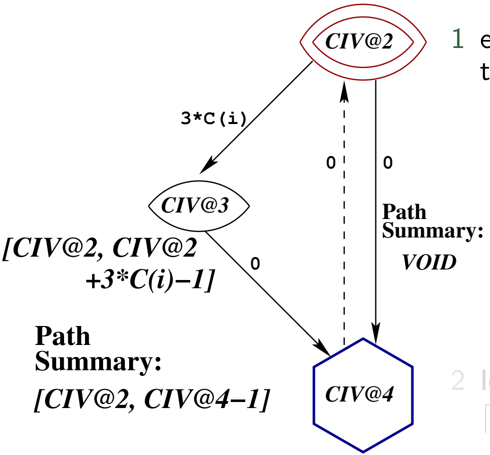
2. Project Gated Intervals to VEG Nodes

```
civ@1 = Q
DO i = M, N, 1
 civ@2=γ(civ@1,civ@4)
 .. =  X(i) ..
 IF C(i) .GT. 0 THEN
  DO j = 1, C(i), 1
   IF(..)X(j+civ@2        )=..
   IF(..)X(j+civ@2+  C(i))=..
   IF(..)X(j+civ@2+2*C(i))=..
  ENDDO
  civ@3 = 3*C(i) + civ@2
 ENDIF
 civ@4=γ(civ@3,civ@2)
ENDDO
civ@5=γ(civ@4,civ@1)
```



Find the CIV *node that describes best the summarization program point: either the "immediate"* CIV*-node dominator or postdominator.*

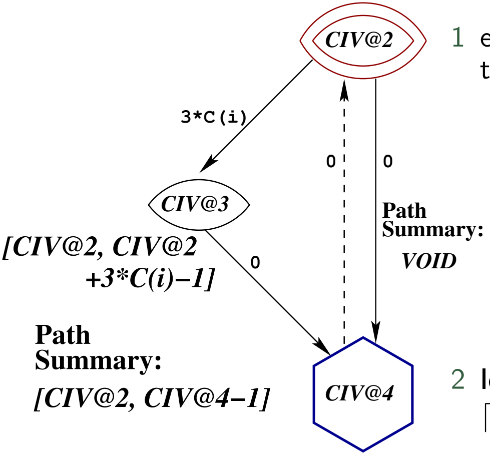
3,4. Summarize and Merge Paths



1 express each path summary in terms of `civ@2` and `civ@4`
   a then path: `[civ@2,civ@2+3*C(i)-1]` ≡ `[civ@2,civ@4-1]`
   b else path: $\emptyset$ ≡ `[civ@2,civ@4-1]`, because `civ@2 > civ@4-1` (0 evolution).

2 Identical Formula, hence $\lceil WF_i \rceil$ = `[civ@2,civ@4-1]`

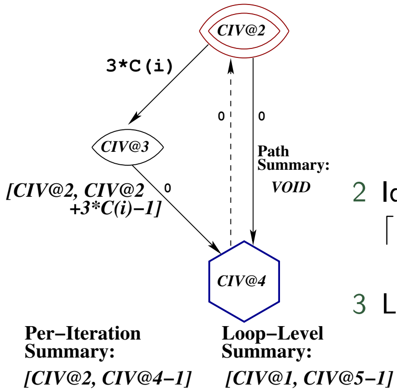
3,4. Summarize and Merge Paths



1 express each path summary in terms of `civ@2` and `civ@4`
   a then path: `[civ@2,civ@2+3*C(i)-1]` $\equiv$ `[civ@2,civ@4-1]`
   b else path: $\emptyset \equiv$ `[civ@2,civ@4-1]`, because `civ@2 > civ@4-1` (0 evolution).

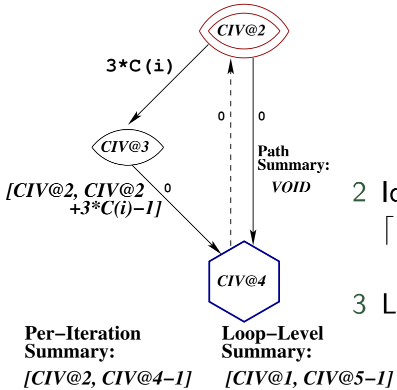
2 Identical Formula, hence $\lceil WF_i \rceil =$ `[civ@2,civ@4-1]`

5. Total Partial Aggregation Across Loop



1. express each path summary in terms of `civ@2` and `civ@4`
   - a then path: `[civ@2,civ@2+3*C(i)-1]` $\equiv$ `[civ@2,civ@4-1]`
   - b else path: $\emptyset$ $\equiv$ `[civ@2,civ@4-1]`, because `civ@2>civ@4-1` (0 evol).
2. Identical Formula, hence $\lceil WF_i \rceil =$ `[civ@2,civ@4-1]`
3. Loop: $\cup_{i=1}^{N} \lceil WF_i \rceil =$ `[civ@1,civ@5-1]`
4. $\cup_{k=1}^{i-1} \lceil WF_k \rceil =$ [civ@1,civ@4$^{i-1} - 1$] $=$ [civ@1,civ@2$^{i} - 1$]

5. Total Partial Aggregation Across Loop



1 express each path summary in terms of `civ@2` and `civ@4`

   a `then path:` `[civ@2,civ@2+3*C(i)-1]` $\equiv$ `[civ@2,civ@4-1]`

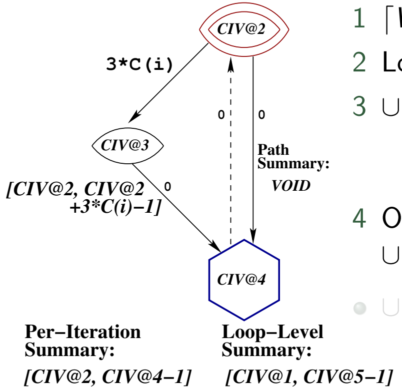
   b `else path:` $\emptyset \equiv$ `[civ@2,civ@4-1]`, because `civ@2>civ@4-1` (0 evol).

2 Identical Formula, hence $\lceil WF_i \rceil =$ `[civ@2,civ@4-1]`

3 Loop: $\cup_{i=1}^{N} \lceil WF_i \rceil =$ `[civ@1,civ@5-1]`

4 $\cup_{k=1}^{i-1} \lceil WF_k \rceil = [\mathtt{civ@1,civ@4}^{i-1} - 1]$
$= [\mathtt{civ@1,civ@2}^{i} - 1]$

6. Satisfiability of Independence Equations



1 $\lceil WF_i \rceil$ = [civ@2,civ@4-1]

2 Loop: $\cup_{i=1}^{N} \lceil WF_i \rceil$ = [civ@1,civ@5-1]

3 $\cup_{k=1}^{i-1} \lceil WF_k \rceil$ = [civ@1,civ@4^{i-1} − 1]
= [civ@1,civ@2^{i} − 1]

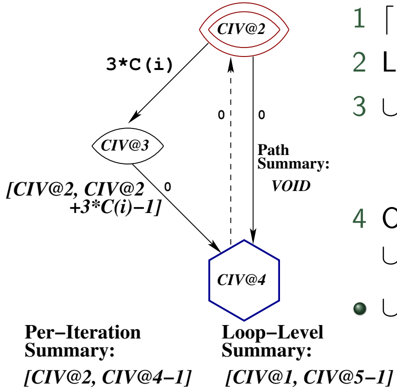
4 Output independence:
$\cup_{i=1}^{n}(\cup_{k=1}^{i-1} W_k \cap W_i) = \emptyset$

- $\cup_{i=1}^{n}($ [civ@1,civ@2^i − 1] $\cap$ [civ@2^i,civ@4^i − 1]$) = \emptyset$

True/Anti Independence:
$(\cup_{i=M}^{N} \lceil R_i \rceil) \cap (\cup_{i=M}^{N} \lceil WF_i \rceil) = [M-1, N-1] \cap [civ@1, civ@5-1] = \emptyset?$

Sufficient Condition: $Q \geq N \ \vee \ M > civ@5$

6. Satisfiability of Independence Equations



1 $\lceil WF_i \rceil =$ `[civ@2,civ@4-1]`

2 Loop: $\cup_{i=1}^{N} \lceil WF_i \rceil = [\texttt{civ@1}, \texttt{civ@5}-1]$

3 $\cup_{k=1}^{i-1} \lceil WF_k \rceil = [\texttt{civ@1}, \texttt{civ@4}^{i-1} - 1]$

$= [\texttt{civ@1}, \texttt{civ@2}^{i} - 1]$

4 Output independence:

$\cup_{i=1}^{n} (\cup_{k=1}^{i-1} W_k \;\cap\; W_i) = \emptyset$

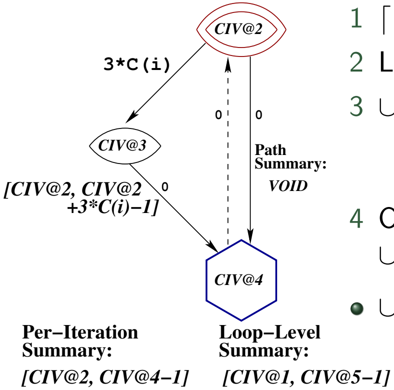
- $\cup_{i=1}^{n} ([\texttt{civ@1}, \texttt{civ@2}^{i} - 1] \;\cap\; [\texttt{civ@2}^{i}, \texttt{civ@4}^{i} - 1]) = \emptyset$

True/Anti Independence:

$(\cup_{i=M}^{N} \lceil R_i \rceil) \cap (\cup_{i=M}^{N} \lceil WF_i \rceil) = [M-1, N-1] \cap [civ@1, civ@5-1] = \emptyset ?$

Sufficient Condition: $Q \geq N \;\vee\; M > civ@5$

6. Satisfiability of Independence Equations



1 $\lceil WF_i \rceil = $`[civ@2,civ@4-1]`

2 Loop: $\cup_{i=1}^{N} \lceil WF_i \rceil = $ `[civ@1,civ@5-1]`

3 $\cup_{k=1}^{i-1} \lceil WF_k \rceil = [\texttt{civ@1}, \texttt{civ@4}^{i-1} - 1]$
$= [\texttt{civ@1}, \texttt{civ@2}^{i} - 1]$

4 Output independence:
$\cup_{i=1}^{n} (\cup_{k=1}^{i-1} W_k \ \cap \ W_i) = \emptyset$

- $\cup_{i=1}^{n} ([\texttt{civ@1}, \texttt{civ@2}^{i} - 1] \ \cap$
  $[\texttt{civ@2}^{i}, \texttt{civ@4}^{i} - 1]) = \emptyset$

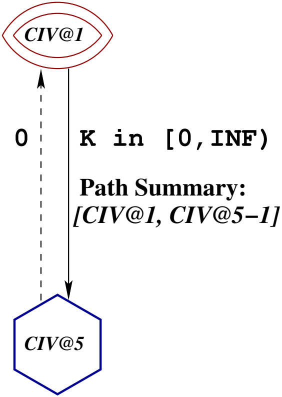
True/Anti Independence:
$(\cup_{i=M}^{N} \lceil R_i \rceil) \cap (\cup_{i=M}^{N} \lceil WF_i \rceil) = [M-1, N-1] \cap [civ@1, civ@5-1] = \emptyset ?$

Sufficient Condition: $Q \geq N \ \vee \ M > civ@5$

Compobility of the Technique

```
civ@0 = Q
DO l = 1, L
 civ@1 = γ(civ@0,civ@5)
 DO i = M, N, 1
  civ@2=γ(civ@1,civ@4)
  .. = X(i) ..
  IF C(i) .GT. 0 THEN
   DO j = 1, C(i), 1
    IF(..)X(j+civ@2       )=..
    IF(..)X(j+civ@2+  C(i))=..
    IF(..)X(j+civ@2+2*C(i))=..
   ENDDO
   civ@3 = 3*C(i) + civ@2
  ENDIF
  civ@4=γ(civ@3,civ@2)
 ENDDO
 civ@5=γ(civ@4,civ@1)
ENDDO
civ@7=γ(civ@5,civ@0)
```

OVERESTIMATE CASE:



Per–Iteration Summary:
[CIV@1, CIV@5−1]

Loop–Level Summary:
[CIV@0, CIV@6−1]

Experimental Multi-Core Setup

- Texas A&M's Polaris compiler :
  Sequential Fortran77 $\Rightarrow$ OpenMP (parallel) Fortran77.
- AMD Opteron(TM) 6274 system with 128GB memory.
- `gfortran -O3`, and run on a 16-core.
- Empirical evaluation on 30 PERFECT-CLUB and SPEC bench:
  - analyzed 2100 loops, measured 380 loops covering 92% runtime,
  - Five out of thirty benchmarks ($\sim$17%) exhibit important loops that require CIV-based analysis.

Benchmark and Loops Characterization

Properties of Benchmarks Exhibiting Important Loops That Use CIVs					
BENCH	PROPERTIES	DO LOOP	LSC%	$T^L_{P/S}$(s)	TYPE
BDNA P=8	$T_{P/S}$=.19/.65 s SC=87%,OV=0%	ACTFOR_500	47.8	.05/.31	ST-PAR
		ACTFOR_240	35.6	.04/.23	CIV_{AGG}
NASA7 P=8	$T_{P/S}$=1.14/3.1 s SC=98%,OV=0%	GMTTST_120	17.4	.27/.54	FI O(1)
		EMIT_5	13.6	.09/.42	CIV_{COMP} OI O(N)
		BTRTST_120	10.1	.05/.31	FI O(1)
TRACK P=8	$T_{P/S}$=6.6/16.8 s SC=97%,OV=45%	FPTRAK_300	52.8	3.6/8.9	CIV_{COMP}
		EXTEND_400	43.9	2.3/7.4	CIV_{COMP}
TREE P=8	$T_{P/S}$=12.8/59 s SC=91%,OV=0%	ACCEL_10	91.2	7.6/54	CIV_{AGG}
PRICE_I P=8	$T_{P/S}$=.29/2.0 s SC=99%,OV=0%	PRICE_I_10	99	.29/2.0	CIV_{AGG}
PRICE_R P=8	$T_{P/S}$=.17/.98 s SC=99%,OV=6.4%	PRICE_R_10	99	.17/.98	CIV_{COMP}

Experimental Results: Perfect-Club & SPEC Bench