Scalar Evolutions

Sébastian Pop

Centre de recherche en informatique

École des mines de Paris

Evolution in a loop

```
j = 3
for (i = 1; i <= 123; i+=5)
  {
    j = i + 7 + j;
  }
```

```
j = 3
i = 1
loop_1
  if (i > 123) goto fin
  t = i + 7
  j = t + j
  i = i + 5
endloop
fin:
```

$$i = 1, 6, 11, \cdots, 126$$

$$j = 3, 11, 24, \cdots, 1703$$

Polynomial Interpolation

Input: $j(n) = 3, 11, 24, 42, 65, \ldots$

Differentiation table:

k	0	1	2	3	4
Δ_k^0	3	11	24	42	65
Δ_k^1	8	13	18	23	
Δ_k^2	5	5	5		
Δ_k^3	0	0			

Newton formula for interpolation:

$$j(n) = \sum_{i=0}^{p} \Delta_0^i \binom{n}{i}$$

Chains of recurrences

Δ_0^i coefficients of a chain of recurrence.

Chrecs = chains of recurrences

Syntax for scalar functions.

Chrecs: examples

$$\textbf{Syntax} \longrightarrow \textbf{Semantics}$$

$$\{3, +, 8, +, 5\}(x) \longrightarrow f(x) = 3\binom{x}{0} + 8\binom{x}{1} + 5\binom{x}{2}$$

$$f(x) = 3 + 8x + 5\frac{x(x-1)}{2}$$

$$f(x) = \frac{5}{2}x^2 + \frac{11}{2}x + 3$$

$$\{5, +, 2\}(x) \longrightarrow f(x) = 2x + 5$$

$$\{1, *, 2\}(x) \longrightarrow f(x) = 2^x = e^{x \cdot ln(2)}$$

$$\{1, *, 1, +, 1\}(x) \longrightarrow f(x) = x!$$

Extraction algorithm

- symbolic for all the initial conditions,
- symbolic for polynomials of degree > 2,
- proof by structural induction.

Algorithm:

1. Walk the def-uses,
2. Reconstruct the update expression,
3. Translate the loop-phi into a chrec,
4. Instantiate parameters (optional).

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

The initial condition is kept under a symbolic form.

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

Walk the use-defs to a loop-phi node.

$$c \longrightarrow f \longrightarrow e \longrightarrow d$$

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

$c \neq d$, walk back:

$$d \longrightarrow e \longrightarrow f$$

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

search for another loop-phi:

$$f \longrightarrow c$$

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

Found the starting loop-phi. Reconstruct the update expression.

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

$$c = phi\,(a,\ c + e)$$

$$c \rightarrow \{a, +, e\}$$

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

$$c \rightarrow \{a, +, e\} \xrightarrow{Instantiate} \cdots$$

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

$$c \rightarrow \{a, +, e\} \xrightarrow{Instantiate} \{3, +, e\}$$

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

$$c \longrightarrow \{a, +, e\} \xrightarrow{Instantiate} \{3, +, d + 7\}$$

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

$$c \rightarrow \{a, +, e\} \xrightarrow{Instantiate} \{3, +, d + 7\}$$

$$d \rightarrow \{1, +, 5\}$$

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

$$c \rightarrow \{a, +, e\} \xrightarrow{Instantiate} \{3, +, d + 7\}$$

$$d \rightarrow \{1, +, 5\}$$

$$e \rightarrow \{8, +, 5\}$$

Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto fin
  e = d + 7
  f = e + c
  g = d + 5
endloop
fin:
```

$$c \rightarrow \{a, +, e\} \xrightarrow{Instantiate} \{3, +, 8, +, 5\}$$

$$d \rightarrow \{1, +, 5\}$$

$$e \rightarrow \{8, +, 5\}$$

Semantics of SSA

Syntax of the SSA program:

```
loop_1
  a = phi (init, a + update)
endloop
```

Semantics:

$$a(x) = init + \sum_{j=0}^{x-1} update(j)$$

Proof by induction

Degree 0: *update* is a constant function.

$$
\begin{aligned}
a(x) &= init + \sum_{j=0}^{x-1} update \\
a(x) &= init + x * update \\
a(x) &= init \binom{x}{0} + update \binom{x}{1} \\
a(x) &\rightarrow \{init, +, update\}(x)
\end{aligned}
$$

Proof by induction

Degree n: *update* is a polynomial of degree *n*.

$$
\begin{aligned}
a(x) &= init + \sum_{j=0}^{x-1} update(j) \\
a(x) &= init + \sum_{j=0}^{x-1} \{b_0, +, \ldots, +, b_{n-1}\}(j) \\
a(x) &= init + \sum_{j=0}^{x-1} \sum_{k=0}^{n-1} b_k \binom{j}{k} \\
a(x) &= init + \sum_{k=0}^{n-1} b_k \sum_{j=0}^{x-1} \binom{j}{k} \\
a(x) &= init + \sum_{k=0}^{n-1} b_k \binom{x}{k+1} \\
a(x) &= init \binom{x}{0} + b_0 \binom{x}{1} + \ldots + b_{n-1} \binom{x}{n} \\
a(x) &\rightarrow \{init, +, b_0, +, \ldots, +, b_{n-1}\}_x
\end{aligned}
$$

Summary

From the SSA program:

```
loop_1
  a = phi (init, a + update)
endloop
```

Extract the symbolic chrec:

$$a(x) \rightarrow \{init, +, update\}_1(x)$$

Finally, instantiate the parameters.

Static analyzes

For some $i, j \in IterationDomain$ determine:

- $f(i) = g(i)$
- $f(i) = g(j)$

Uses in:

- number of iterations in a loop,
- check elimination,
- data dependence analysis.

Number of iterations

```
if (chrec_1 > chrec_2) goto endloop;
...
if (chrec_1 == chrec_2) goto endloop;
```

Find the first iteration $i \in IterationDomain$ that satisfies:

$$chrec_1(i) \; op \; chrec_2(i)$$

Check elimination

```
if (chrec_1 > chrec_2) foo; else bar;
...
if (chrec_1 == chrec_2) foo; else bar;
```

Prove statically that for all $i \in IterationDomain$:

$$chrec_1(i) \; op \; chrec_2(i)$$

Data dependences

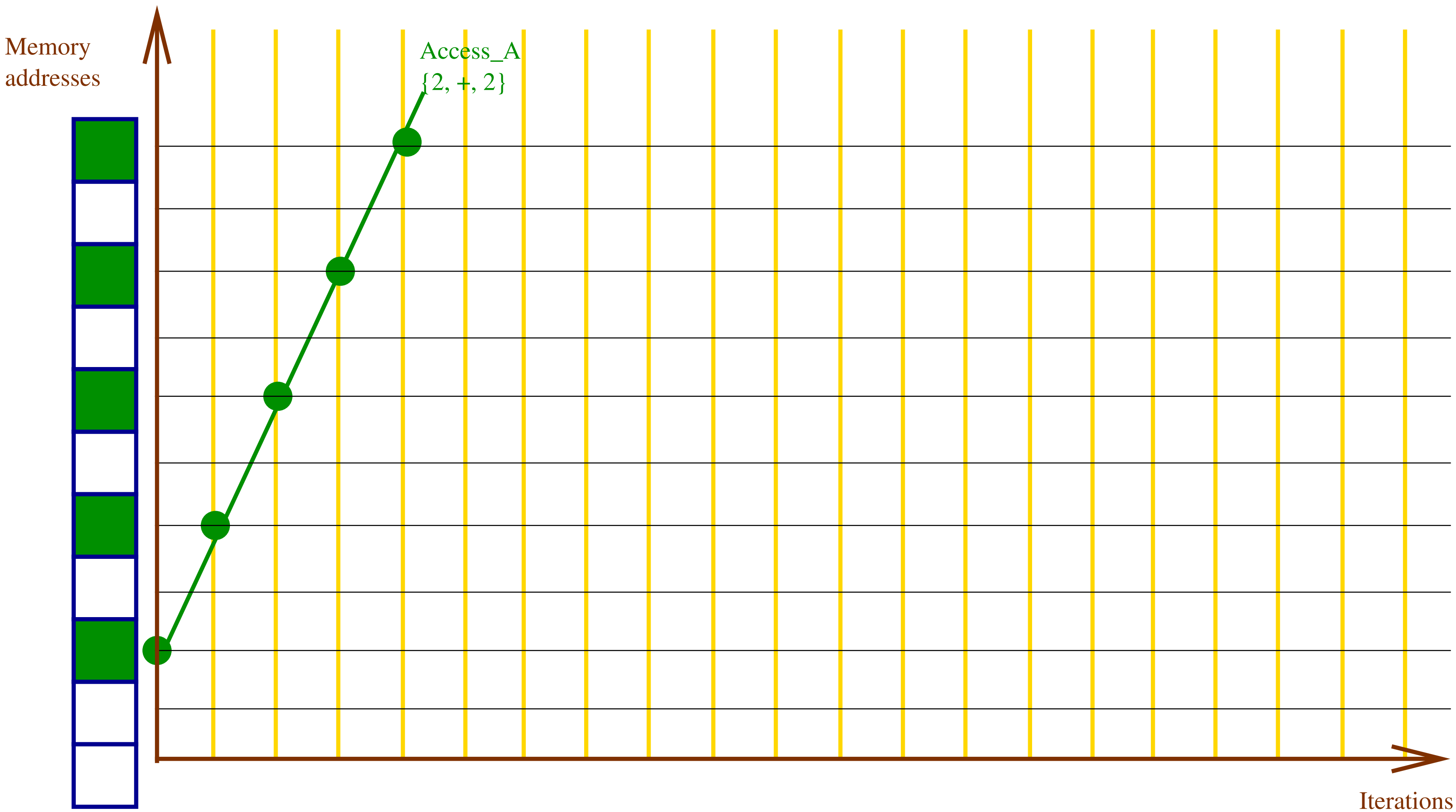
```
T[chrec_1] = ...
... = T[chrec_2]
```

Given two data accesses to the same array, there is a dependence if:

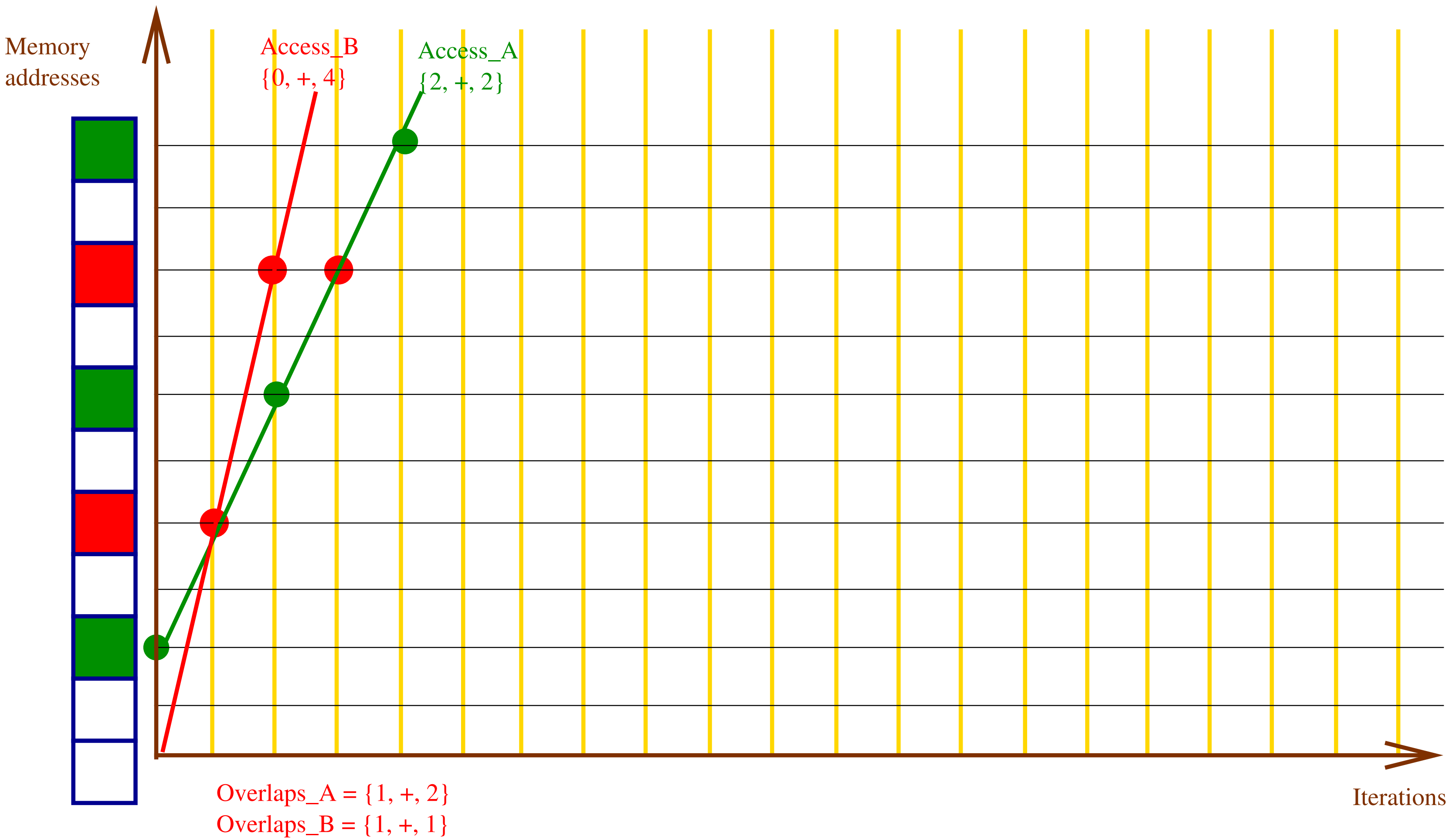
$$chrec_1(i) = chrec_2(j)$$

for two iterations $i, j \in IterationDomain$.

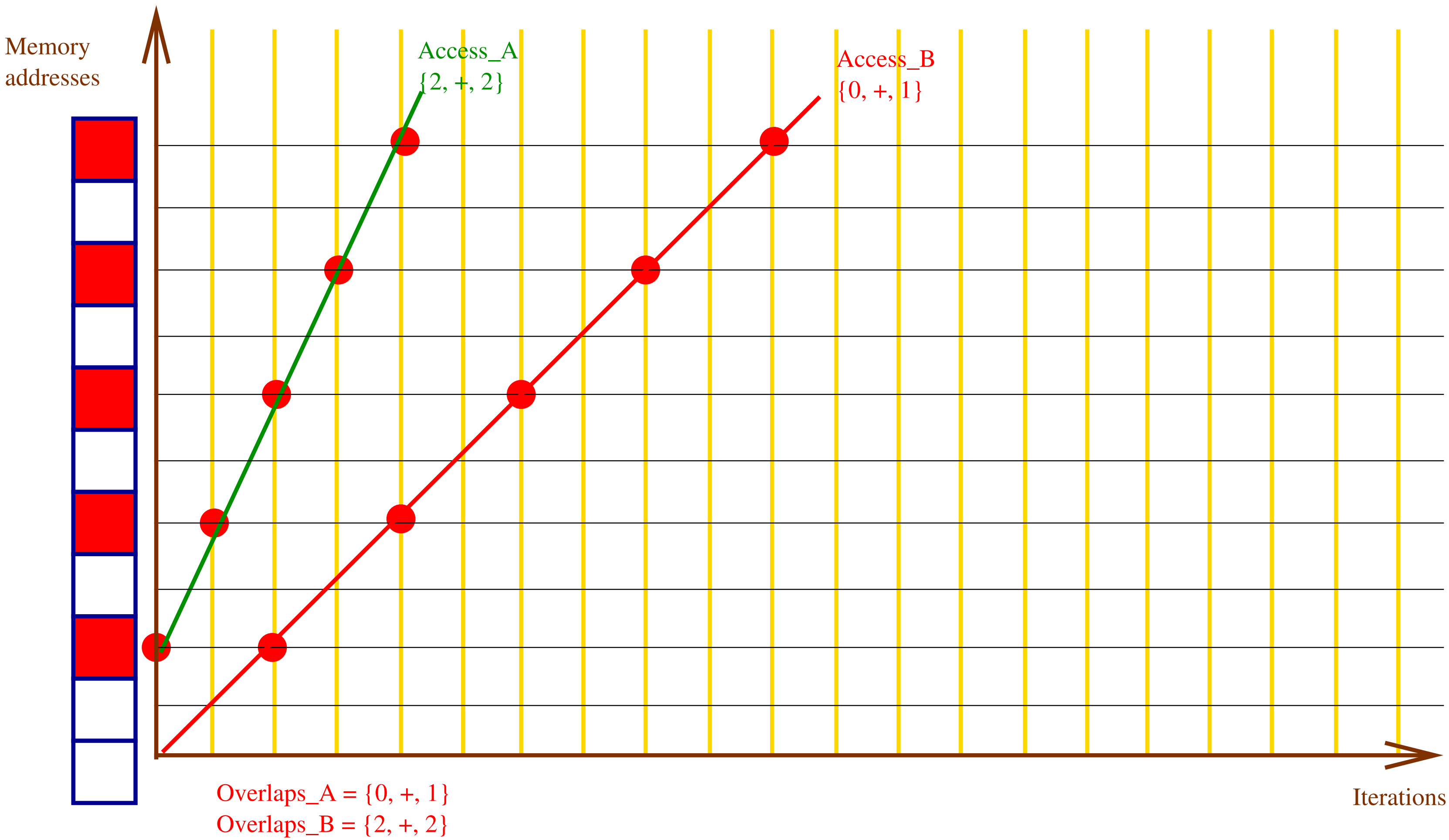
Array Accesses



Array Accesses



Array Accesses



Array Accesses

For two access functions, determine:

- the accessed elements (green dots),
- the conflicting elements (red dots),
- a description of the conflict iterations: for $k \geq 0$,

$$Access_A(Overlap_A(k)) = Access_B(Overlap_B(k))$$

- when the overlaps have same evolution, compute the distance.
- based on the distance, compute the direction.

Modulo arithmetics

Syntax of the SSA program:

```
unsigned char a;
loop_1
  a = phi (init, a + update)
endloop
```

Semantics:

$$a(x) = \left(init + \sum_{j=0}^{x-1} update(j) \right) \pmod{256}$$