Semantic Array Dataflow Analysis

Paul Iannetta

UCBL 1, CNRS, ENS de Lyon, Inria,
LIP, F-69342, LYON Cedex 07, France

Laure Gonnord

UCBL 1, CNRS, ENS de Lyon, Inria,
LIP, F-69342, LYON Cedex 07, France

Lionel Morel

Univ Grenoble Alpes, CEA, List
F-38000 Grenoble, France

Tomofumi Yuki

Inria, Univ Rennes, CNRS, IRISA
F-35000 Rennes, France

January 23, 2019

If you think I missed a reference please tell me!

Thesis Context (ANR CoDaS: [Gonnord 2017])

Inspiration[Alias et al. 2010]:

- Termination: generates affine schedules (ranking functions) with classical Polyhedral Model computations.
- Program semantics: approximated with (polyhedral) Abstract Interpretation.

Thesis' subject:

- A Polyhedral Model Extension which supports:
  - Trees [Cohen 1999]
  - Maps = allow to index arrays by array cells
- No closed form to access elements
- Need to make approximations

▶ First step here: general control flow.

A Semantic Ground For Abstract Intrepretation

- Not rely on syntax
- Set as few as possible restrictions

A Semantic Ground For Abstract Intrepretation

- Not rely on syntax
- Set as few as possible restrictions

Too constrained syntax (iteration variable is apparant)

- `for i from 0 to N` [Feautrier 1991]
- `for i from 0 while` *cond(i)* [Griebl 1997]

A Semantic Ground For Abstract Intrepretation

- Not rely on syntax
- Set as few as possible restrictions

Too constrained syntax (iteration variable is apparant)

- `for i from 0 to N` [Feautrier 1991]
- `for i from 0 while` *cond(i)* [Griebl 1997]

Our target (general while loops)

```
while cond(i,j,k,l) {
    ...
}
```

A Semantic Ground For Abstract Intrepretation

- Not rely on syntax
- Set as few as possible restrictions

Too constrained syntax (iteration variable is apparant)

- `for i from 0 to N` [Feautrier 1991]
- `for i from 0 while` *cond(i)* [Griebl 1997]

Our target (general while loops)

```
while cond(i,j,k,l) {
    ...
}
```

- Iteration variable is not visible anymore
- Leads to non polyhedral programs
- Polyhedral approximation

Benefits of a Semantic - of Abstract Interpretation

- Dissociate definitions from computations:
  - Computations are expressed within the model
  - Can characterize dependences within the model
  - Allows verification.
  - Allows precise characterisations of where abstractions/approximations are made.

A Semantic Ground for Earlier Projects

- Be a model for compiler IR, LLVM [Grosser et al. 2012] or GCC [Trifunović et al. 2010]
  - Integration within real compiler
  - Composition with other optimizations
- Would *a posteriori* justify the implementation on top of a compiler IR.

Steps of the Approach

1. Define a barebone language
   - Allow general programs on arrays
   - Can be computed from a CFG

Steps of the Approach

1. Define a barebone language
   - Allow general programs on arrays
   - Can be computed from a CFG

2. Equip it with a dependence-enabled semantic

Steps of the Approach

1. Define a barebone language
   - Allow general programs on arrays
   - Can be computed from a CFG
2. Equip it with a dependence-enabled semantic
3. Show that dependences can be statically computed (equivalence with previous work).

A Barebone Language

$$\langle Aexp \rangle ::= \langle Num \rangle \mid \langle Aexp \rangle \ \langle Aop \rangle \ \langle Aexp \rangle \mid \langle Vexp \rangle$$

$$\langle Aop \rangle ::= \text{`}\texttt{+}\text{'} \mid \text{`}\texttt{*}\text{'} \mid \text{`}\texttt{-}\text{'} \mid \text{`}\texttt{/}\text{'} \mid \text{`}\texttt{mod}\text{'}$$

$$\begin{array}{l} \langle Bexp \rangle ::= \text{`}\texttt{true}\text{'} \mid \text{`}\texttt{false}\text{'} \mid !(\langle Bexp \rangle) \\ \quad \mid \ \langle Bexp \rangle \ \langle Bop \rangle \ \langle Bexp \rangle \mid \langle Aexp \rangle \ \langle Cop \rangle \ \langle Aexp \rangle \end{array}$$

$$\langle Bop \rangle ::= \text{`}\texttt{or}\text{'} \mid \text{`}\texttt{and}\text{'}$$

$$\langle Cop \rangle ::= \text{`}\texttt{<}\text{'} \mid \text{`}\texttt{==}\text{'}$$

$$\langle Vexp \rangle ::= \mathsf{X} \mid \mathsf{X}\text{`}\texttt{[}\text{'}\langle Aexp \rangle\text{`}\texttt{]}\text{'}$$

$$\begin{array}{l} \langle Sexp \rangle ::= \kappa_n \ \text{`}\texttt{:begin}\text{'} \mid \text{`}\texttt{skip}\text{'} \mid \langle Sexp \rangle \ \text{`}\texttt{;}\text{'} \ \langle Sexp \rangle \\ \quad \mid \ \kappa_n \ \text{`}\texttt{:if}\text{'} \ \langle Bexp \rangle \ \text{`}\texttt{then}\text{'} \ \langle Sexp \rangle \ \text{`}\texttt{else}\text{'} \ \langle Sexp \rangle \ \text{`}\texttt{fi}\text{'} \\ \quad \mid \ \kappa_n \ \text{`}\texttt{:while}\text{'} \ \langle Bexp \rangle \ \text{`}\texttt{do}\text{'} \ \langle Sexp \rangle \ \text{`}\texttt{done}\text{'} \\ \quad \mid \ \langle Vexp \rangle \ \text{`}\texttt{:=}\text{'} \ \langle Aexp \rangle \end{array}$$

A Barebone Language

$\langle Aexp \rangle ::= \langle Num \rangle \mid \langle Aexp \rangle \langle Aop \rangle \langle Aexp \rangle \mid \langle Vexp \rangle$

What is important about that syntax is that:

- Allow arrays (scalars = 1-length array)
- Allow conditional tests to reference array cells
- Allow array cells to be referenced by other array cells
- Allow while loops with no restrictions on conditions

$\mid \quad \kappa_n$ ':while' $\langle Bexp \rangle$ 'do' $\langle Sexp \rangle$ 'done'

$\mid \quad \langle Vexp \rangle$ ':=' $\langle Aexp \rangle$

An Example Program

```
i = 0
while i < N
  j = 0
  while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

- Iteration variables are not visible
- Add annotation to keep track of operations

Annotation

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

- Add variables which counts operations on a hierarchical level

Iteration variables κ_i in the semantics

What the semantic is about?

Describe the evolution of an augmented state:

- Standard state: snapshot of the memory at time t
- Augmented Memory: value and last modification time
- The current timestamp : a vector of κs

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	0

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j		
A[1]		
A[2]		
k		
A[3]		
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j		
A[1]		
A[2]		
k		
A[3]		
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	0

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	0	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 0 \rangle]$
A[1]		
A[2]		
k		
A[3]		
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	1

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	0	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 0 \rangle]$
A[1]		
A[2]		
k		
A[3]		
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	1
κ_2	0

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	0	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 0 \rangle]$
A[1]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 0 \rangle]$
A[2]		
k		
A[3]		
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	1
κ_2	1

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 1 \rangle]$
A[1]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 0 \rangle]$
A[2]		
k		
A[3]		
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	1
κ_2	2

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 1 \rangle]$
A[1]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 0 \rangle]$
A[2]	A[1] + 1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 2 \rangle]$
k		
A[3]		
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	1
κ_2	3

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	2	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 3 \rangle]$
A[1]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 0 \rangle]$
A[2]	A[1] + 1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 2 \rangle]$
k		
A[3]		
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	2

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	2	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 3 \rangle]$
A[1]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 0 \rangle]$
A[2]	A[1] + 1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 2 \rangle]$
k	0	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 2 \rangle]$
A[3]		
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	3

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	2	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 3 \rangle]$
A[1]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 0 \rangle]$
A[2]	A[1] + 1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 2 \rangle]$
k	0	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 2 \rangle]$
A[3]		
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	3
κ_3	0

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 3 \rangle]$
A[1]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 0 \rangle]$
A[2]	A[1] + 1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 2 \rangle]$
k	0	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 2 \rangle]$
A[3]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 3 \rangle, \langle \kappa_3 = 0 \rangle]$
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	3
κ_3	1

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	2	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 3 \rangle]$
A[1]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 0 \rangle]$
A[2]	A[1] + 1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 2 \rangle]$
k	1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 3 \rangle, \langle \kappa_3 = 1 \rangle]$
A[3]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 3 \rangle, \langle \kappa_3 = 0 \rangle]$
A[4]		

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	3
κ_3	2

Cell	Value	Last access
i	0	$[\langle\kappa_0 = 0\rangle]$
j	2	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 3\rangle]$
A[1]	A[0]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 0\rangle]$
A[2]	A[1] + 1	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 2\rangle]$
k	1	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 1\rangle]$
A[3]	A[0]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 0\rangle]$
A[4]	A[1]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 2\rangle]$

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	3
κ_3	3

Cell	Value	Last access
i	0	$[\langle \kappa_0 = 0 \rangle]$
j	2	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 3 \rangle]$
A[1]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 0 \rangle]$
A[2]	A[1] + 1	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 1 \rangle, \langle \kappa_2 = 2 \rangle]$
k	2	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 3 \rangle, \langle \kappa_3 = 3 \rangle]$
A[3]	A[0]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 3 \rangle, \langle \kappa_3 = 0 \rangle]$
A[4]	A[1]	$[\langle \kappa_0 = 1 \rangle, \langle \kappa_1 = 3 \rangle, \langle \kappa_3 = 2 \rangle]$

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	4

Cell	Value	Last access
i	1	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 4\rangle]$
j	2	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 3\rangle]$
A[1]	A[0]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 0\rangle]$
A[2]	A[1] + 1	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 2\rangle]$
k	2	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 3\rangle]$
A[3]	A[0]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 0\rangle]$
A[4]	A[1]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 2\rangle]$

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	5

Cell	Value	Last access
i	1	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 4\rangle]$
j	0	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 5\rangle]$
A[1]	A[0]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 0\rangle]$
A[2]	A[1] + 1	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 2\rangle]$
k	2	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 3\rangle]$
A[3]	A[0]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 0\rangle]$
A[4]	A[1]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 2\rangle]$

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	6

Cell	Value	Last access
i	0	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 4\rangle]$
j	0	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 5\rangle]$
A[1]	A[0]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 0\rangle]$
A[2]	A[1] + 1	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 2\rangle]$
k	2	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 3\rangle]$
A[3]	A[0]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 0\rangle]$
A[4]	A[1]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 2\rangle]$

Table: Memory State

Unrolling of an Execution

```
κ0:begin
i = 0
κ1:while i < N
  j = 0
  κ2:while j < 2
    A[i+j+1] = A[j] + j
    j = j + 1
  k = 0
  κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
  i = i + 1
```

Table: Timestamp

κ_0	1
κ_1	6
κ_2	0

Cell	Value	Last access
i	1	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 4\rangle]$
j	0	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 5\rangle]$
A[1]	A[0]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 6\rangle, \langle\kappa_2 = 0\rangle]$
A[2]	A[1] + 1	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 1\rangle, \langle\kappa_2 = 2\rangle]$
k	2	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 3\rangle]$
A[3]	A[0]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 0\rangle]$
A[4]	A[1]	$[\langle\kappa_0 = 1\rangle, \langle\kappa_1 = 3\rangle, \langle\kappa_3 = 2\rangle]$

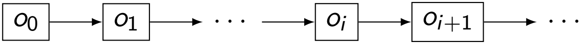
Table: Memory State

Trace

Operation

An operation is a *tuple*: $(s, \mathbf{t})$ where

- s is a *statement* (*i.e.*, `A[i] = A[i-1] + i`)
- $\mathbf{t}$ is a *timestamp* (*i.e.*, $\left[\langle\kappa_0, 3\rangle, \langle\kappa_1, 1\rangle\right]$)



Trace

Operation

An operation is a *tuple*: $(s, \mathbf{t})$ where

- s is a *statement* (*i.e.*, `A[i] = A[i-1] + i`)
- $\mathbf{t}$ is a *timestamp* (*i.e.*, $\left[\langle\kappa_0, 3\rangle, \langle\kappa_1, 1\rangle\right]$)

$$o_0 \longrightarrow o_1 \longrightarrow \cdots \longrightarrow o_i \longrightarrow o_{i+1} \longrightarrow \cdots$$

Zoom on the state of the memory at o_i

$o_i = (s, \mathbf{t})$	
...	...
$A[21]$	$\left[\langle\kappa_0, 3\rangle, \langle\kappa_1, 1\rangle\right]$
...	...

Dependence Definition

Inspired from [Feautrier 1991].

Dependence: o_2 depends on an operation o_1

1. o_1 is *valid* (*i.e.*, it belongs to a trace)
2. $o_1 = (s_1, t_1)$ occurs before $o_2 = (s_2, t_2)$
   - $t_1 <_{lex} t_2$
3. $o_2 = (s_2, t_2)$ is reading and/or writing a cell that $o_1 = (s_1, t_1)$ wrote
   - s_1 is "$A[f(i,j,k)] \; = \; \ldots$"
   - s_2 is "$\ldots \; = \; A[g(l,r)]$"
   - "$f(i,j,k) \; = \; g(l,r)$"

In (3), the access uses real variables

Dependence Computation I

```
[...]
κ3:while k < 2
    A[k+3+i] = A[k] + i
    k = k + 1
[...]
```

1. Express timestamps as function of real variables
    1. Express the relation between variables before and after a loop step
        - $k \sim k+1$
        - $\kappa_3 \sim \kappa_3 + 2$
    2. Compute the transitive closure (if the loop is affine) [Verdoolaege et al. 2011]
        - $\kappa_3 = 3k$

Dependence Computation II

```
[...]
κ3:while k < 2
  A[k+3+i] = A[k] + i
  k = k + 1
[...]
```

2. Solve the parametrized integer linear programs
   - Parameters: i, k
   - Conditions:
     - $0 \leq i,\ k,\ i',\ k'$
     - $k + 3 + i = k'$
3. Express back the dependences within our model

Conclusion

The ideas are not new. However,

- We got rid of the syntax
- We have a new dependence front-end to an integer linear program

Future work,

- Non polyhedral programs
  - Find reasonable approximations as polyhedral programs

References I

C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *Proceedings of the 17th International Conference on Static Analysis*, SAS '10, pages 117–133, 2010.

A. Cohen. *Program Analysis and Transformation: From the Polytope Model to Formal Languages*. Theses, Université de Versailles-Saint Quentin en Yvelines, Dec. 1999. URL https://tel.archives-ouvertes.fr/tel-00550829.

P. Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–53, 1991.

L. Gonnord. Codas: Complex data-structure scheduling, April 2017.

M. Griebl. *The mechanical parallelization of loop nests containing while loops*. PhD thesis, University of Passau, 1997. URL http://d-nb.info/950009474.

References II

T. Grosser, A. Groesslinger, and C. Lengauer. Polly - performing polyhedral optimizations on a low-level intermediate representation. *Parallel Processing Letters*, 22(04):1250010–1–1250010–28, 2012.

K. Trifunović, A. Cohen, D. Edelsohn, F. Li, T. Grosser, H. Jagasia, R. Ladelsky, S. Pop, J. Sjödin, and R. Upadrasta. GRAPHITE two years after: First lessons learned from eal-world polyhedral compilation. In *2nd GCC Research Opportunities Workshop (GROW)*, 2010. URL http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.220.3386.

S. Verdoolaege, A. Cohen, and A. Beletska. Transitive Closures of Affine Integer Tuple Relations and their Overapproximations. In E. Yahav, editor, *Proceedings of the 18th International Static Analysis Symposium (SAS'11)*, volume 6887 of *LNCS*, pages 216–232, Venice, Italy, Sept. 2011. Springer. doi: $10.1007/978\text{-}3\text{-}642\text{-}23702\text{-}7\backslash\_18$. URL https://hal.inria.fr/hal-00645221.

Annexe: Semantics 1/3

$$\text{skip}\ \frac{}{\langle \sigma, \mathrm{skip} \rangle}$$

$$\text{begin}\ \frac{}{\langle \sigma, \kappa_0 : \mathrm{begin}; s \rangle \rightarrow \langle \mathrm{upd}(\sigma, \kappa_0, 0), s \rangle}$$

$$\text{Assign}\ \frac{}{\langle \sigma, v := e \ ; \ s \rangle \rightarrow \langle \mathrm{incr}(\sigma[v := e]), s \rangle}$$

incr and upd

- incr: increments the timestamp
- upd: create a fresh κ or does nothing
- $\sigma \setminus \kappa_n$ remove κ_n from the state

Annexe: Semantics 2/3

$$\text{WhT}\ \frac{\langle \sigma, b_0 \rangle \to \text{true} \qquad \langle \text{upd}(\sigma, \kappa_n, 0), s_1 \rangle \to^+ \langle \sigma', \text{skip} \rangle}{\begin{array}{c}\langle \sigma, \kappa_n : \text{while } b_0 \text{ do } s_1 \text{ done }; \ s \rangle \to \\ \langle \text{incr}(\sigma'), \kappa_n : \text{while } b_0 \text{ do } s_1 \text{ done }; \ s \rangle\end{array}}$$

$$\text{WhF}\ \frac{\langle \sigma, b_0 \rangle \to \text{false}}{\langle \sigma, \kappa_n : \text{while } b_0 \text{ do } s_1 \text{ done }; \ s \rangle \to \langle \text{incr}(\sigma \setminus \kappa_n), s \rangle}$$

$$\text{IT}\ \frac{\langle \sigma, b_0 \rangle \to \text{true} \qquad \langle \text{upd}(\sigma, \kappa_n, -\text{length}(s_1)), s_1 \rangle \to^+ \langle \sigma', \text{skip} \rangle}{\langle \sigma, \kappa_n : \text{if } b_0 \text{ then } s_1 \text{ else } s_2 \text{ fi}; s \rangle \to \langle \text{incr}(\sigma' \setminus \kappa_n); s \rangle}$$

$$\text{IF}\ \frac{\langle \sigma, b_0 \rangle \to \text{false} \qquad \langle \text{upd}(\sigma, \kappa_n, 0), s_2 \rangle \to^+ \langle \sigma', \text{skip} \rangle}{\langle \sigma, \kappa_n : \text{if } b_0 \text{ then } s_1 \text{ else } s_2 \text{ fi}; s \rangle \to \langle \text{incr}(\sigma' \setminus \kappa_n), s \rangle}$$