4.0 Practical issues

So far: scalars, method-local

4.1 Arrays

Why are arrays a problem?

Most (common) languages allow access to individual array elements.

Example

```c
int A[50]; int i, k, j, ...
read (i, j, k)

A[i][j] = ...
A[j][i] = ...
```

← how could you turn these structs into SSA format
Different elements of the array are used/written.

- Exact element is known only at runtime.

> Sometimes it may be possible to show

\[ i = j = k \quad \forall \text{ executions} \]

... transformations are possible...

In general, not possible

(\( i, j, k \) can be parameter input... results from method...)

\[
\begin{align*}
\text{\{ } & \\
\text{ } & a = \text{foo}(); \quad \text{ } & a = \text{foo}(); \\
\text{ } & b = \text{bar}(); \quad \text{ } & b = \text{bar}(); \\
\text{\}} & \quad \text{\}}
\end{align*}
\]

order of execution is fixed
Approach:

We extend SSA format to deal with arrays but restrict analysis & optimization to scalars.
- opt. of scalars (method local)
  - most profitable (e.g., ren amthmalch)

Arrays: we introduce two functions

- access: to model reading of an array element
- update: to model writing an array element

Compiler must keep order of access / update inocations unchanged.
Example

\[ is = \text{read}(i) \]
\[ j_0 = \text{read}(j) \]
\[ k_0 = \text{read}(k) \]

\[ \text{read } A[i] \]
\[ \text{access } (A, is) \]

\[ \text{update}(A, i, j) \]
\[ \text{value written into } A[j] \]
\[ \text{access } (A, k_0) \]

**Update**: changes the Array, one element is modified.

- update can happen “in place”
  - i.e. memory cell for \( A[j] \) is modified.

\( \text{update}(A, \text{array}, \text{index}, \text{value}) \)

easy to implement...
too restrictive?

can you change the order of

access (x, index)

update (y, index, ...) too...

without SSA

x [index]

y [index]

Later topic: analyses of subscripts to determine exact dependences.

effectiveness depends on domain
You can extend handling SSA arrays in the SSA form by adding "update & copy" to your compiler.

```c
updatedCopy( array, subscript, array, value )
```

new array, modification not done in place

many copies ... space ...
4.2 Dealing with object instances (records, structs)

Idea: map fields to array

```plaintext
Class X &
  int a, b, c;
  
  static int xref[] = ...

  -> xref.a = xref.b = xref.c
```

Model object instance
```
data[3]
a = data[0]
b = data[1]
c = data[2]
```

Treat object as array
```
update(xref, 0, ...) = access(xref, 1)
update(xref, 1, ...) = access(xref, 2)
```

→ order fixed
5.0 Optimization

Given a variable $v$

Algorithm (3.0) inserts $\phi$ nodes wherever two paths $P_1$, $P_2$ meet s.t. $P_1$ and $P_2$ contain different assignments to $v$.

meet: $\phi$ node

$\phi$ node = meet/join of paths with different assignments

Alg. insert minimal number of $\phi$ nodes.

Question. Does Alg. 3.0 produce an optimal solution?