Advanced program representations

Goal:

- more effective analysis
- faster analysis
- easier transformations

Approach:

more directly capture important program properties

- e.g. data flow, independence

Examples

CFG:

+ simple to build
+ complete

– computing info is slow and/or ineffective
– doesn't support incremental update of computed info
– doesn't support staging of analysis and transformations

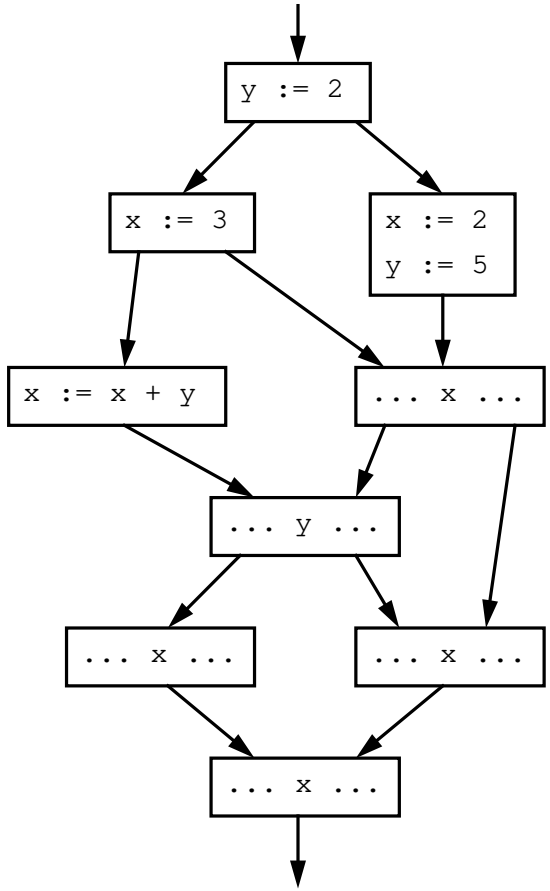
Def/use chains

Def/use chains directly linking defs to uses

+ directly captures data flow for analysis
  - e.g. constant propagation easy

– ignores control flow
  - misses some optimization opportunities
  - not executable by itself
– space-consuming: O(E^2V)
– can have multiple defs of same variable in program, multiple defs can reach a use

Example



Static Single Assignment (SSA) form

Invariant: at most one definition reaches each use

Constructing SSA form:

1. Create new target names for all definitions
2. Insert **pseudo-assignments** at merge points:
   x_n := $\phi(x_1, \ldots, x_n)$
3. Adjust uses to refer to appropriate new names

\+ less asymptotic space cost: O(*EV*)

\+ algorithms simplified by exploiting single assignment property

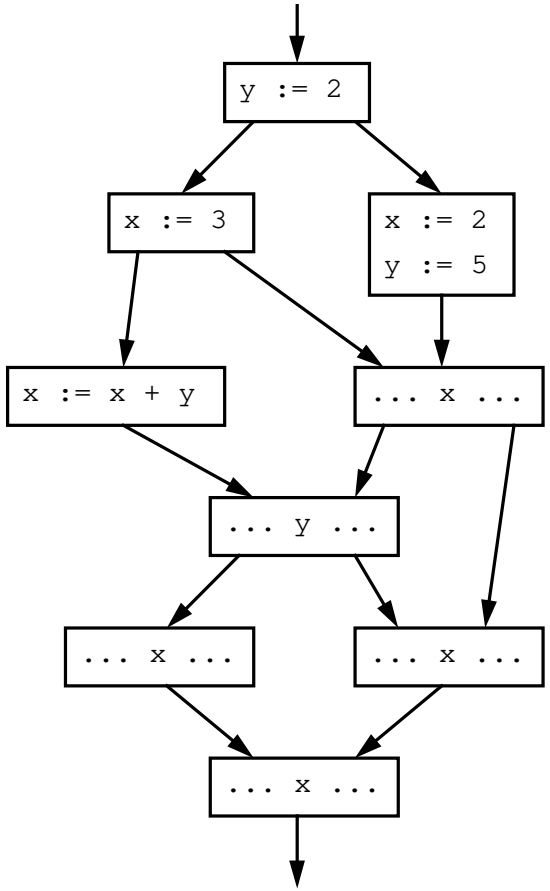
– still not executable

– inverse property not provided

- DFG's [Pingali *et al.*] fix this

Very popular in research compilers

Example



Common subexpression elimination

At each program point, compute set of **available expressions**

- e.g. `{a+b → x, -c → y, *p → z}`

If `a+b`→`x` available for `y := a+b`, transform to `y := x`

What direction to do analysis?

Flow functions for:

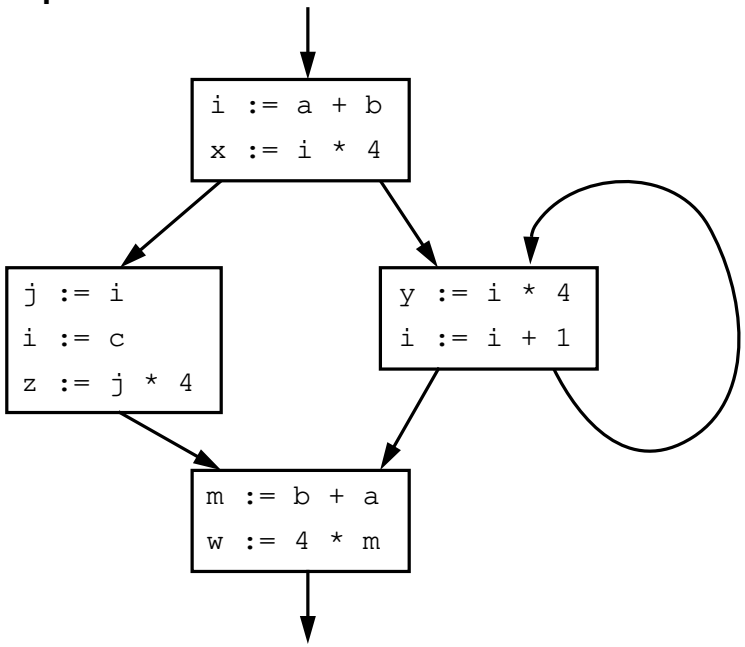
```
x := y op z
```

```
x := *(p + o)
```

```
*(p + o) := x
```

What merge function?

Example



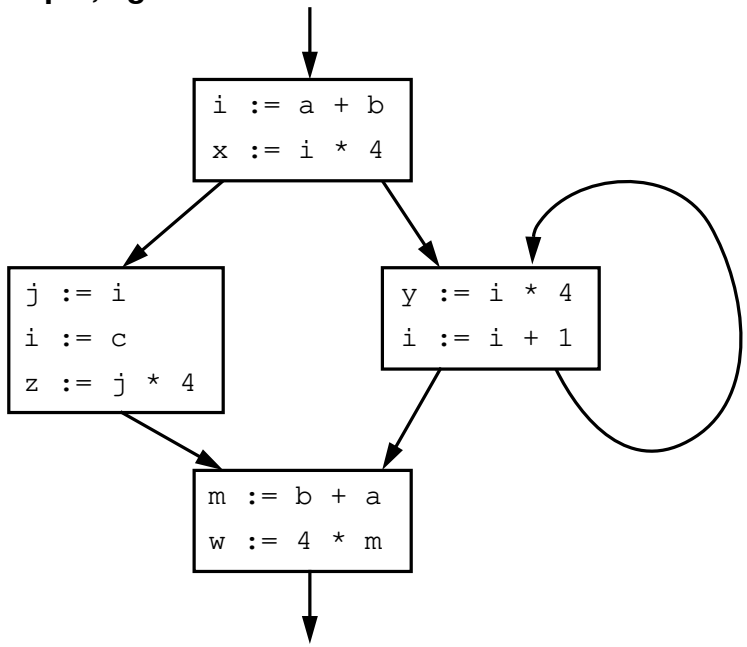
Value graph representation

Revised available expressions:
convert to SSA form;
associate a **value** with each name

```
value  ::=  constant
       |    atomic_value(identity)
       |    op(value_op1, ..., value_opN)
       |    φ(value_pred1, ..., value_predN)
```

[AWZ88]: advanced value graphs,
incorporating control dependences

```
value  ::=  ...
       |    φ_if(value_then, value_else, value_test)
       |    φ_enter(value_head, value_tail)
       |    φ_exit(value_exit-test, value)
```


Example, again