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

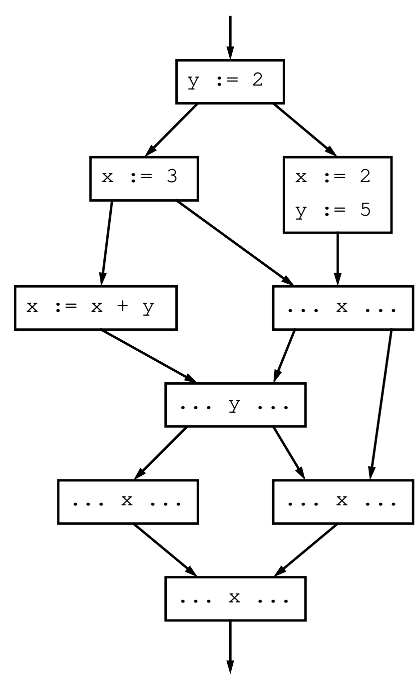
Def/use chains directly linking defs to uses

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

– ignores control flow
    - misses some optimization opportunities
    - not executable by itself

– space-consuming, in worst case: $O(E^2V)$
– 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

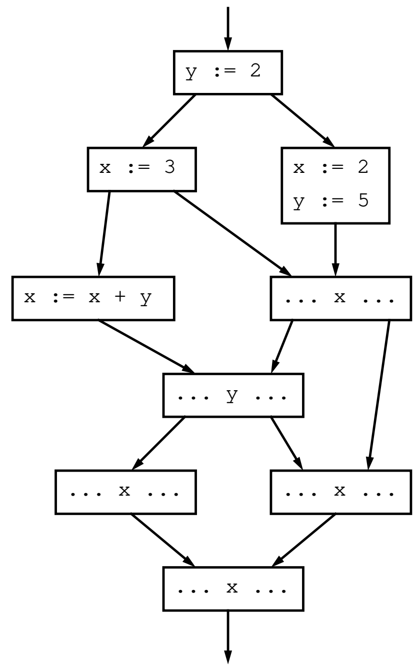
+ lower worst-case space cost: O(*EV*)
+ algorithms simplified by exploiting single assignment property:
  - can treat variable & value synonymously
  - don't have to worry about updates to variables as much

– still not executable by itself

– 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` $\rightarrow$ `x, -c` $\rightarrow$ `y, *p` $\rightarrow$ `z}`

If `a+b`$\rightarrow$`x` available for `y := a+b`, transform to `y := x`

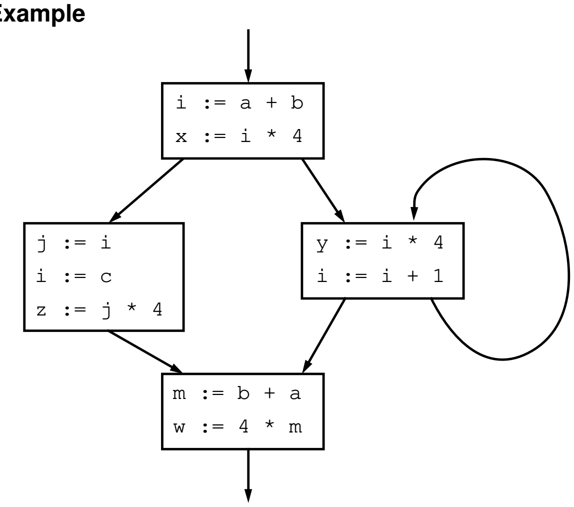
What direction to do analysis?

Flow function:

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

What merge function?

Example



Value graph representation

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

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

$\texttt{unknown}_{\texttt{v}}$ is same as $\texttt{unknown}_{\texttt{v}}$

$\texttt{unknown}_{\texttt{v1}}$ might be same or different than $\texttt{unknown}_{\texttt{v2}}$, $\texttt{v1} \neq \texttt{v2}$

[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)
```


CSE with SSA form and values

If using SSA form and value graphs,
compute mappings between variables and values

- e.g. $\{\texttt{x} \leftrightarrow \texttt{v}_a\texttt{+v}_b\texttt{, y} \leftrightarrow \texttt{(v}_a\texttt{+v}_b\texttt{)*2, z} \leftrightarrow \texttt{v}_z\}$

If at `x := y op z`,
in $\supseteq \{\texttt{y} \leftrightarrow \texttt{v}_y\texttt{, z} \leftrightarrow \texttt{v}_z\texttt{, q} \leftrightarrow \texttt{v}_y \texttt{ op v}_z\}$,
replace with `x := q`

Flow function:

`x := y op z`

What merge function?

Example, again