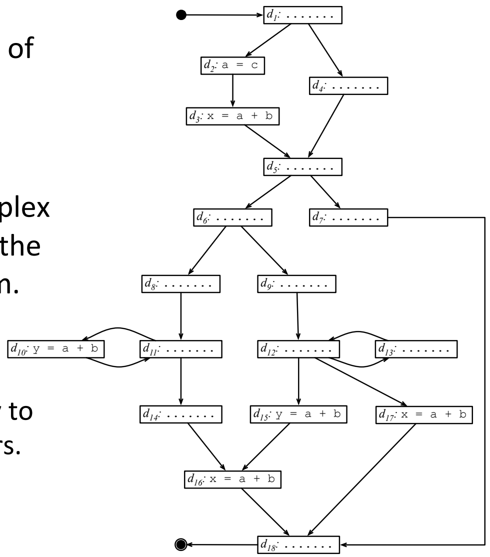
An Example to Conquer them All

- The original formulation of Lazy Code Motion was published in a paper by Knoop *et al*◇.
- The authors used a complex example to illustrate all the phases of their algorithm.
- Many papers are build around examples.
  - That is a good strategy to convey ideas to readers.



◇: Lazy Code Motion, PLDI (1992)

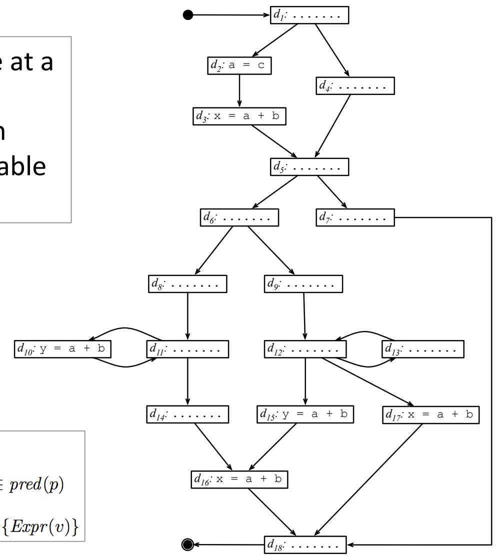
Available Expressions

An expression *e* is available at a program point *p* if e is computed along every path from p_{start} to *p*, and no variable in *e* is redefined until *p*.

What is the OUT set of available expressions in the example?

$p : v = E$

$$IN(p) = \bigcap OUT(p_s), p_s \in pred(p)$$
$$OUT(p) = (IN(p) \cup \{E\}) \setminus \{Expr(v)\}$$

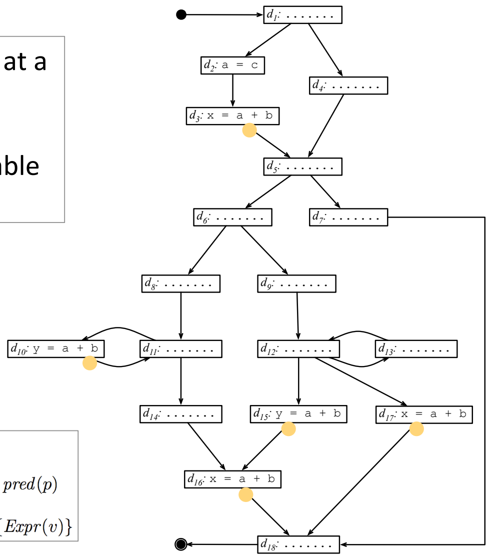


Available Expressions

An expression *e* is available at a program point *p* if e is computed along every path from p_{start} to *p*, and no variable in *e* is redefined until *p*.

$p : v = E$

$$IN(p) = \bigcap OUT(p_s), p_s \in pred(p)$$
$$OUT(p) = (IN(p) \cup \{E\}) \setminus \{Expr(v)\}$$



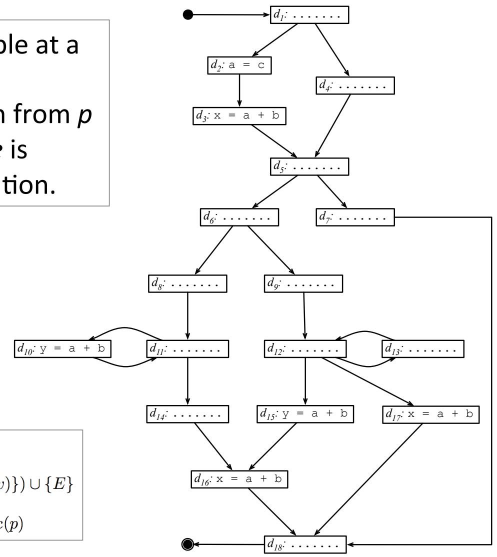
Anticipable Expressions

An expression *e* is anticipable at a program point *p* if *e* will be computed along every path from *p* to p_{end}, and no variable in *e* is redefined until its computation.

What is the IN set of anticipable expressions in the example?

$p : v = E$

$$IN(p) = (OUT(p) \setminus \{Expr(v)\}) \cup \{E\}$$
$$OUT(p) = \bigcap IN(p_s), p_s \in succ(p)$$



Anticipable Expressions

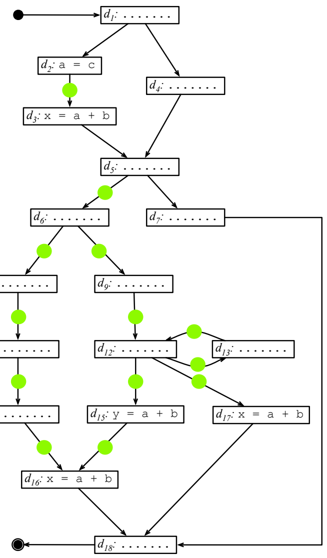
An expression *e* is anticipable at a program point *p* if *e* will be computed along every path from *p* to p_{end}, and no variable in *e* is redefined until its computation.

What is the IN set of anticipable expressions in the example?

$p : v = E$

$$IN(p) = (OUT(p) \setminus \{Expr(v)\}) \cup \{E\}$$

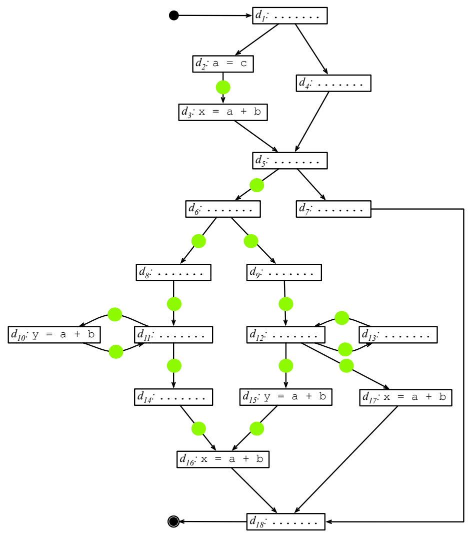
$$OUT(p) = \bigcap IN(p_s), p_s \in succ(p)$$



$EARLIEST(i, j) = IN_{ANTICIPABLE}(j) \cap \overline{OUT_{AVAILABLE}(i)} \cap (KILL(i) \cup \overline{OUT_{ANTICIPABLE}(i)})$

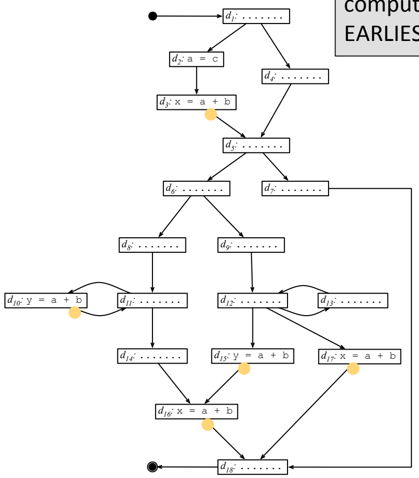
What is $KILL(i) \cup \overline{OUT_{ANTICIPABLE}(i)}$ in our running example?

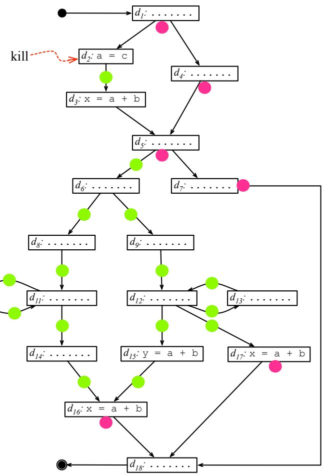
This figure shows the IN sets of anticipability analysis.



$EARLIEST(i, j) = IN_{ANTICIPABLE}(j) \cap \overline{OUT_{AVAILABLE}(i)} \cap (KILL(i) \cup \overline{OUT_{ANTICIPABLE}(i)})$

Can you compute EARLIEST now?





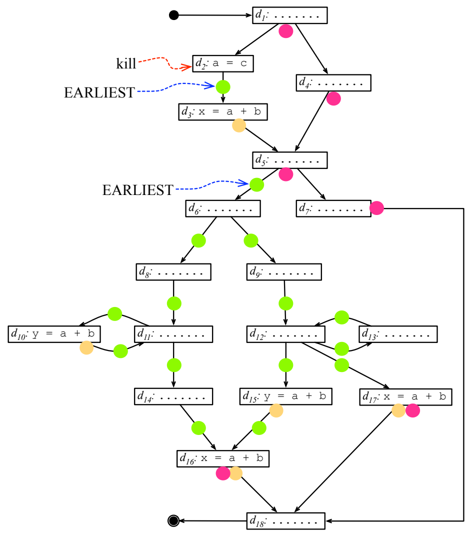
$EARLIEST(i, j) = IN_{ANTICIPABLE}(j) \cap \overline{OUT_{AVAILABLE}(i)} \cap (KILL(i) \cup \overline{OUT_{ANTICIPABLE}(i)})$

We have two EARLIEST edges in this CFG.

Anticipable at IN(j)

Not anticipable at OUT(i)

Available at OUT(i)

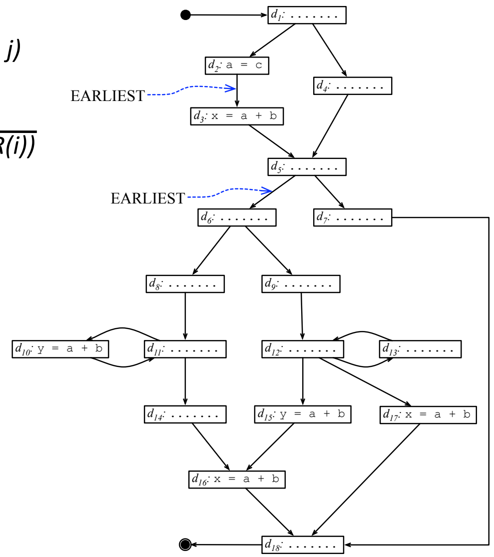


Latest

$$IN_{LATER}(j) = \cap_{i \in pred(j)} LATER(i, j)$$

$$LATER(i, j) = EARLIEST(i, j) \cup (IN_{LATER}(i) \cap \overline{EXPR(i)})$$

The goal now is to compute the latest IN sets, and the latest edges. Can you do it?



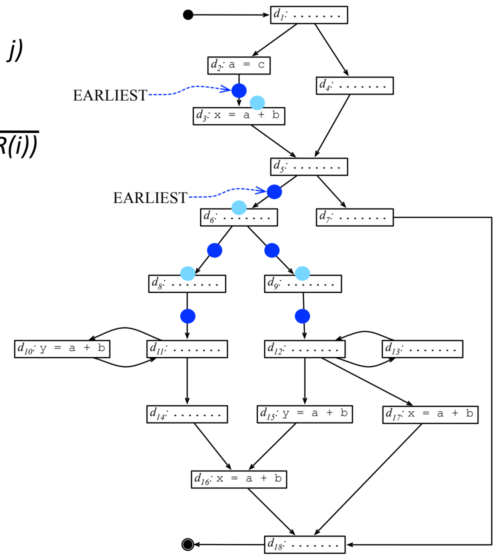
Latest

$$IN_{LATER}(j) = \cap_{i \in pred(j)} LATER(i, j)$$

$$LATER(i, j) = EARLIEST(i, j) \cup (IN_{LATER}(i) \cap \overline{EXPR(i)})$$

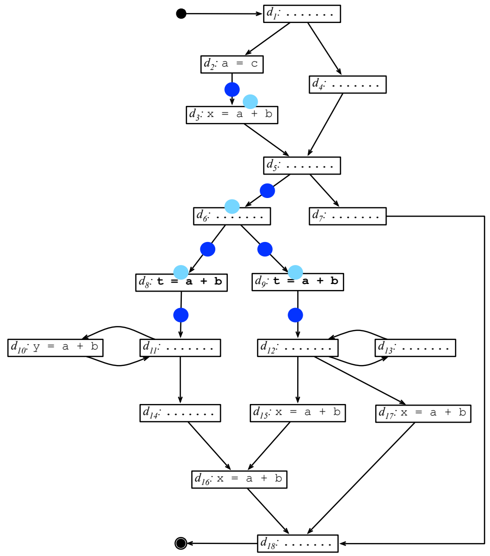
The LATEST edges are marked with the dark blue circles.

The LATEST IN sets are marked with the light blue circles.



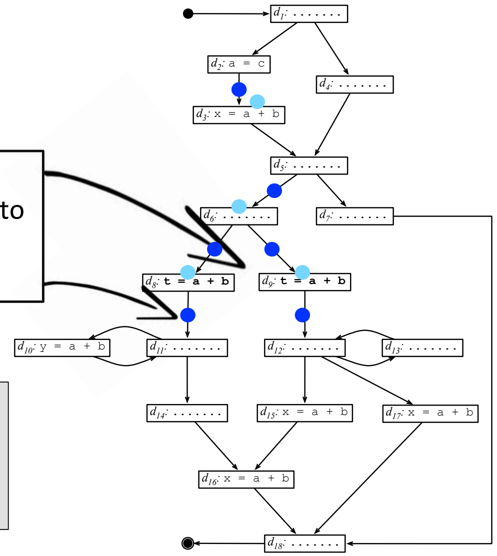
$INSERT(i, j) = LATER(i, j) \cap \overline{IN_{LATER}(j)}$

We must now find the sites where we can insert new computations of a + b. Can you compute INSERT(i, j)?



$INSERT(i, j) = LATER(i, j) \cap \overline{IN_{LATER}(j)}$

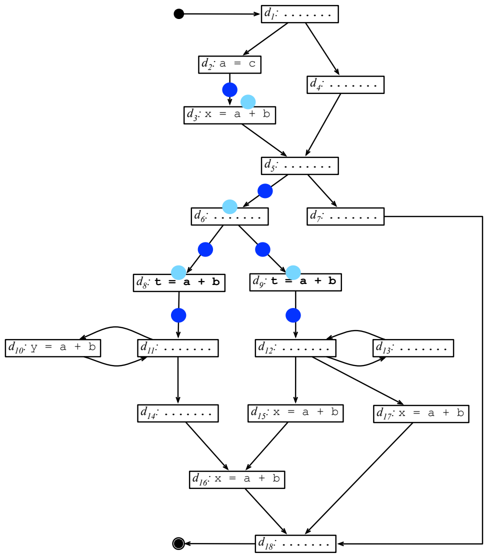
In this example, we only have two sites to insert new computations.



Do you remember why we insert the computation of a + b at d_8, instead of d_{11}?

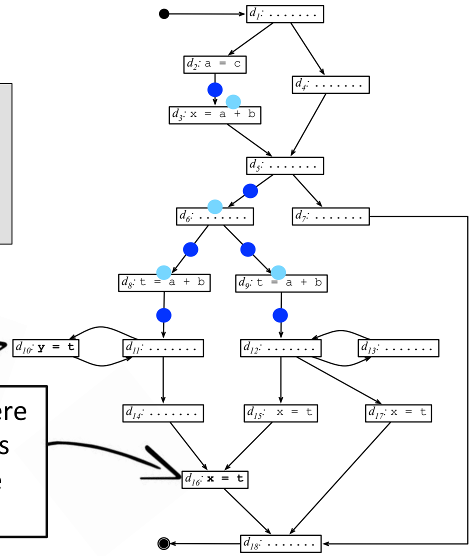
$DELETE(i) = EXPR(i) \cap \overline{IN_{LATER}(i)}$

We must now delete redundant computations that exist at program points p. Can you determine DELETE(p)?



$DELETE(i) = EXPR(i) \cap \overline{IN_{LATER}(i)}$

Is it clear why all the other computations of a + b must be kept unchanged?



In this example, there are four expressions that we can replace with a temporary.