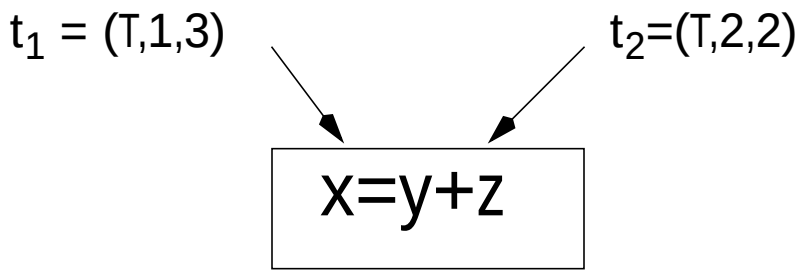
Not all Data Flow Problems are Distributive

Constant propagation is *not* distributive.

Consider the following (with variables (x,y,z)):



Now f(t)=t' where

t'(y) = t(y), t'(z) = t(z),

t'(x) = if t(y)=⊥ or t(z) = ⊥
then ⊥
elseif t(y)=T or t(z) =T
then T
else t(y)+t(z)

Now $f(t_1 \wedge t_2) = f(\top, \bot, \bot) = (\bot, \bot, \bot)$

$f(t_1) = (4,1,3)$

$f(t_2) = (4,2,2)$

$f(t_1) \wedge f(t_2) = (4, \bot, \bot) \geq (\bot, \bot, \bot)$

Why does it Matter if a Data Flow Problem isn't Distributive?

Consider actual program execution paths from b_0 to (say) b_k.

One path might be $b_0,b_{i_1},b_{i_2},...,b_{i_n}$

where $b_{i_n}=b_k$.

At b_k the Data Flow information we want is

$f_{i_n}(...f_{i_2}(f_{i_1}(f_0(T)))...) \equiv f(b_0,b_1,...,b_{i_n})$

On a different path to b_k, say $b_0,b_{j_1},b_{j_2},...,b_{j_m}$, where $b_{j_m}=b_k$

the Data Flow result we get is

$f_{j_m}(...f_{j_2}(f_{j_1}(f_0(T)))...) \equiv$

$f(b_0,b_{j_1},...,b_{j_m})$.

Since we can't know at compile time which path will be taken, we must *combine* all possible paths:

$$\bigwedge_{p \in \text{ all paths to } b_k} f(p)$$

This is the *meet over all paths* (MOP) solution. It is the *best possible* static solution. (Why?)

As we shall see, the meet over all paths solution can be computed efficiently, using standard Data Flow techniques, if the problem is Distributive.

Other, non-distributive problems (like Constant Propagation) can't be solved as precisely.

Explicitly computing and meeting all paths is prohibitively expensive.

Conditional Constant Propagation

We can extend our Constant Propagation Analysis to determine that some paths in a CFG aren't executable. This is *Conditional Constant Propagation*.

Consider

```
i = 1;
if (i > 0)
      j = 1;
else j = 2;
```

Conditional Constant Propagation can determine that the else part of the if is unreachable, and hence `j` must be 1.

The idea behind Conditional Constant Propagation is simple. Initially, we mark all edges out of conditionals as "not reachable."

Starting at b_0, we propagate constant information *only* along edges considered reachable.

When a boolean expression $b(v_1,v_2,...)$ controls a conditional branch, we evaluate $b(v_1,v_2,...)$ using the t(v) mapping that identifies the "constant status" of variables.

If $t(v_i)=\top$ for any v_i, we consider all out edges unreachable (for now).

Otherwise, we evaluate $b(v_1,v_2,...)$ using t(v), getting true, false or $\perp$.

Note that the short-circuit properties of boolean operators may yield true or false even if $t(v_i)=\perp$ for some v_i.

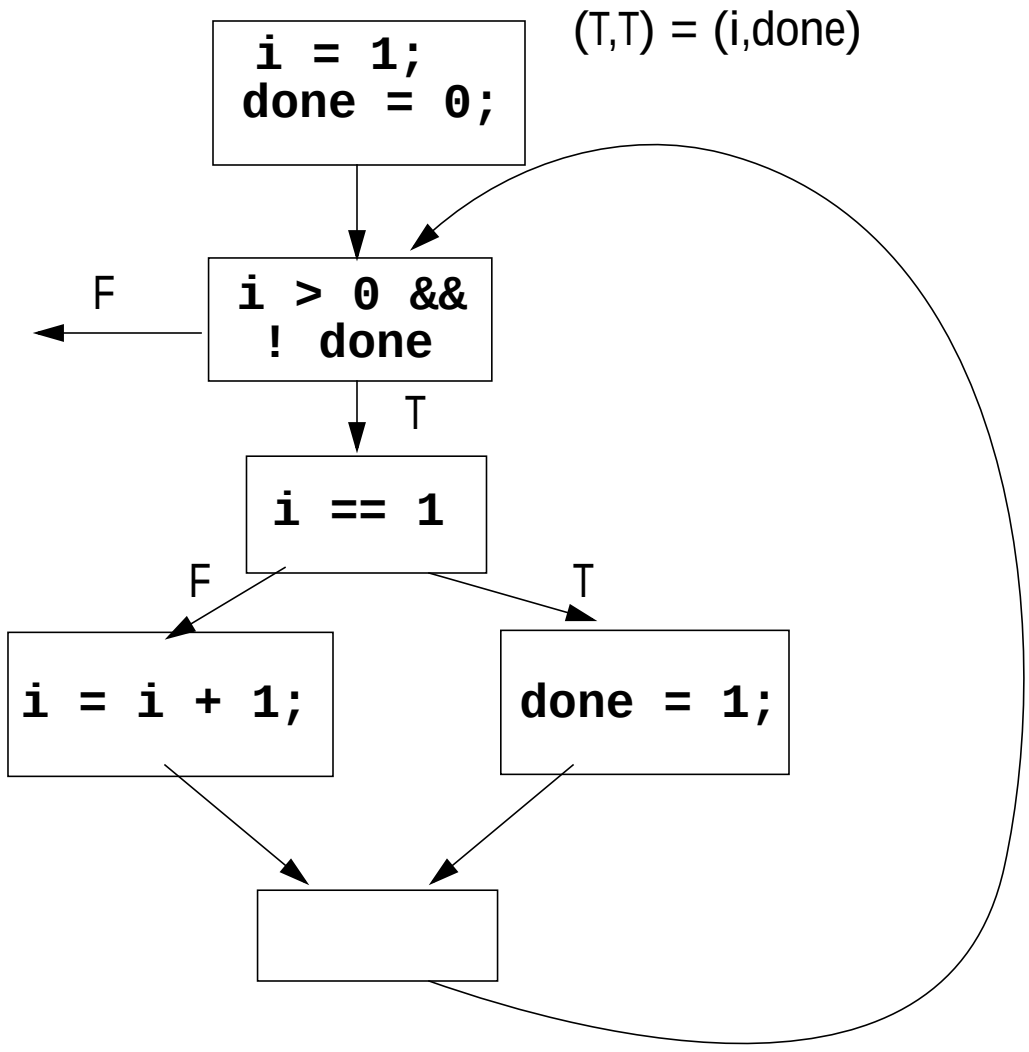
If $b(v_1,v_2,...)$ is true or false, we mark only one out edge as reachable.

Otherwise, if $b(v_1,v_2,...)$ evaluates to $\perp$, we mark all out edges as reachable.

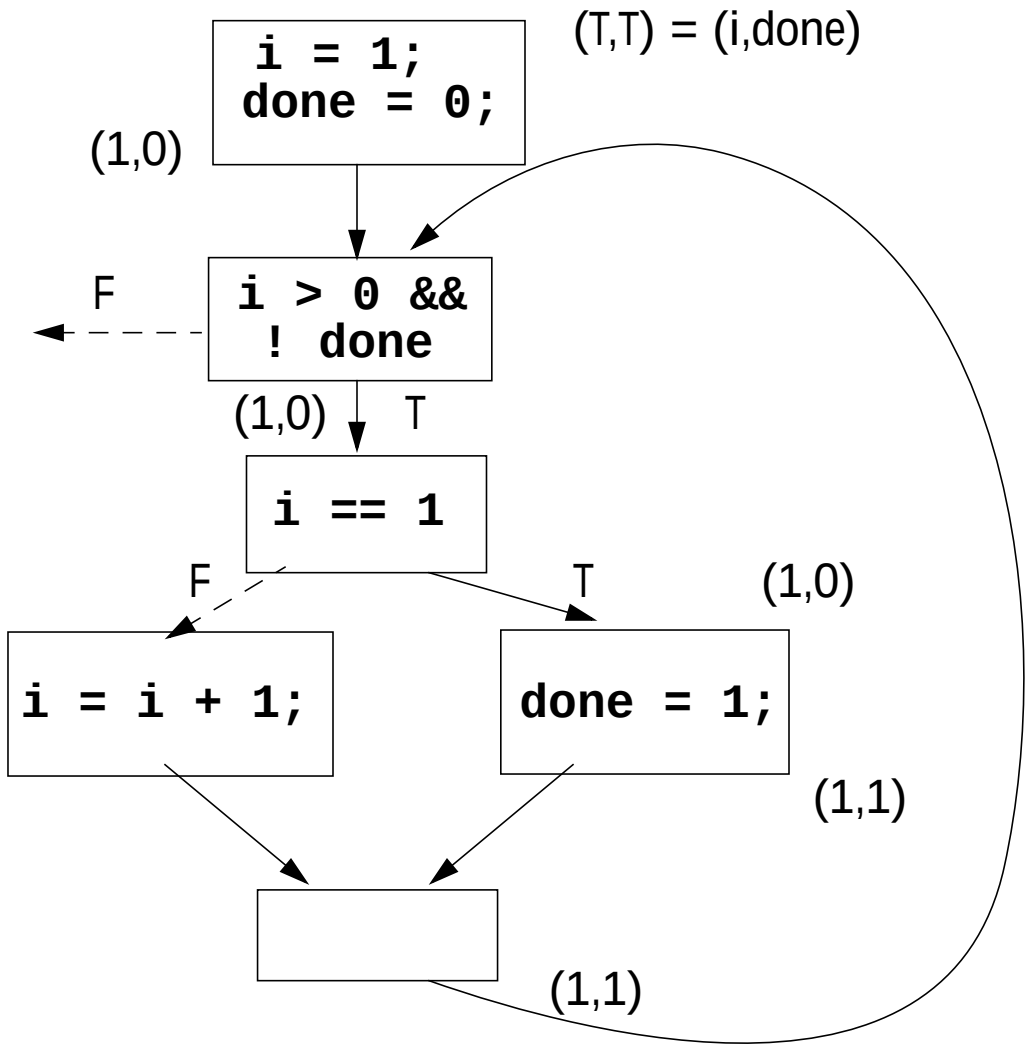
We propagate constant information only along reachable edges.

Example

```
i = 1;
done = 0;
while ( i > 0 && ! done) {
  if (i == 1)
       done = 1;
  else i = i + 1;
}
```



Pass 1:



Pass 2: