5.8 - Dead code elimination

Dominance

\( n \text{ dom } m \quad n \text{ on every path from } s \text{ to } t \)

Post-Dominance

\( p \text{ pdom } i \quad p \text{ is on every path from } i \text{ to } \text{Exit} \)

- Post dominator tree
- Immediate post dominator
- Strict post dom.
Exit pdom start, B₀, B₁...
  B₇ pdom B₂, B₅, B₄, B₆, B₃, B₂, B₁, B₀
B₅ pdom B₄, B₆
B₄ pdom B₆, B₃
B₁ pdom B₀
How to get this?
Build reverse control flow graph (R CFG) CFG⁻¹

DT ...

Reverse dominator tree R DT = post dom tree of CFG

Reverse dominator for list RDF

DF⁻¹
why do we care?

post dominance relation is useful to determine control dependence.

if block i determines if block k is executed

A block y is control dependent on block x if

1) ∃ path P from x to y

(P is non-null P: x⇝y)

P: x⇝x₁⇝x₂⇝⋯⇝y

such that y pred xᵢ (xᵢ: on path)

2) y does not strictly post dominate x
Idea:

Determine control dependence
- build podom tree (RDT)
- compute reverse dominance frontier (RDF)

Look at stmts (in blocks) to find out which stmts are necessary to compute output of a given method.

Approach:

Initially, all stmts are marked dead.
Mark stmt "live" if needed to compute visible output.
Find stmt s that must be live
stmt \( S \) must be live

1) \( S \) may affect program output
   (e.g., I/O statement, assignment to a reference parameter, call to fct with side effects, assignment to keep variable, static, ... long-lived object.)

2) \( S \) is an assignment stmt and some stmt \( S' \) (that is live) may use the output of \( S \)

3) \( S \) is a conditional branch and then exists some stmt \( S' \) (that is live) is control-dependent on \( S \).
What do we need?

PreLive - set of stmts that affect program output

Live(S) - S is live

Worklist -

Definers(S) - set of stmts that provide values used by S

Last(B) - stmt that terminates a block B

Block(S) - basic block that contains S

 updom(B) - immediate postdominator of B

CD^{-1}(B) - set of blocks that are control dependence predecessors of B

( = RDF(B)) ( = DF^+(B))
for (each stmt S) do {
  if (S ∈ Prelive) & Live(S) = true { 
    else ? Live(S) = false }
}

Worklist = Prelive;

while (Worklist ≠ ∅) do { 
  pick S from Worklist  // (removes S)
  for (each D ∈ Definers(S)) do { 
    if (!Live(D)) { 
      Live(D) = true; 
      add D to Worklist;
    }
  }
  // for each D...

  for (each B ∈ CD⁻¹(Blocks(S))) do { 
    if (!Live(B)) { 
      Live(Last(B)) = true; 
      add Last(B) to Worklist;
    }
  }
  // for each B
\texttt{for ( each stmt s) do }
\texttt{
  if (! live(s)) delete s from Block(s)}

Removes unnecessary stmts by finding live stmts.
\texttt{(not marked live: can be removed)}
Sources of dead code ...

```
if ( )
  if ( )
    x = ...
    x = x
    println x = ...
    x = x
```

```