6.0 Removal of partial redundancies (PRE) in a compiler with SSA format (SSA-PRE)

5 (easy) steps

1. Insert \( \Phi \) nodes (record when different versions of expressions join)

--- break ---

2. Identify places (basic blocks) where it is legal to insert a copy of an expression \( E \)

3. Find place where insertion of \( E \) is profitable

4. Transform the program (with \( E \) that is fully redundant) to exploit redundancies (i.e., eliminate computations)
6.5 Identify places where it is legal to insert a copy of E

E: Some expression

Starting point: Consider E nodes where (at least) one operand is the "L" symbol - then exists one path along which E is not computed → places that might benefit from an insertion of a copy of E.

What is our definition of "legal"?
Baseline definition of legal:

Given a transformation $T$,

$P$ computes the same result as $P'$ (without the transformation $T$)

$\Rightarrow$ "no visible difference"

"same" = computes identical results
- also included are the exceptions thrown by $P$
- reports no new errors
- throws no additional exceptions
- terminates behaviour unchanged

(warning: I/O mapped to memory: special case)
A program $P$ may encounter an error earlier (transformed) or throw an exception earlier (if we, e.g., insert a copy of $E$).

The values stored by $P$ may be different from the values stored by $P'$ at the time error occurred (in $P'$).

[Could be a problem... $E$ may not...]

Bottom line: we may introduce a copy of $E$ into a block $B$ if and only if $E$ appears on every path from $B$ to "Exit".

(E would be computed later and any exception thrown by $E$ would have appeared later.)
A copy of $E$ can be inserted only into those paths that include (further downstream) a copy of $E$. 
Consider this example

\[ E_1 = a + b \]

\[ E_2 = \exists (E_1, t) \]

\[ E_3 = a + b \]

Q: Can we insert \( E_3 = a + b \) into \( B \)?

No!

\( B' \) postdom \( B \) and there is a path from \( B \) to Exit without \( E \)
We call this property ("expr E can be inserted") down safety.

E is downsafe at B iff E is computed along all paths from B to Exit.

(earlier papers: "E is anticipated")

Two issues to consider (for B to be not downsafe)

- E path P from B to Exit and E does not occur on this path.

- E occurs only as an operand to a T node and this T node is not downsafe.
Example of $E$ with a $\top$ node with conditions (2).

- \[ B_0 \quad E_1 = \ldots \]
- \[ B_3 \quad E_2 = \top(E_1, \perp) \]
- \[ B_5 \quad E_3 = \top(E_2, \perp) \]
- \[ B_7 \quad E = E_3 \]
- \[ E = E_3 \]
- \[ Q: \text{E downsafe in } B_1? \]
- \[ Q: \text{E downsafe in } B_2? \]

* downsafe
* not downsafe
E is downsafe (in this example) in blocks that exclude E

E is not downsafe in B4

(E occurs only as operand of \( \Phi \) node and \( \Phi \) node not downsafe)

does not count for comparison of

E is not downsafe in B3

(E occurs on all paths to Exit but on one path only as an operand in a \( \Phi \) node ...)

E is not downsafe in B1 or B2
we want to consider I nodes with 1 operands (candidates for insertion)
check if there is a "real" use of E along all paths to Exit
(real: use other than use as operand in I node)
no: (no real use): block that corresponds to I operand is not down safe.
Simple way to determine this property
- visit all nodes backward from Exit
- mark a I node as "legal" if there is a real use of E along all paths from Exit to block with I node.
  (property must hold for all sequences..."
After visiting all nodes in CFG, all \( \Phi \) nodes that are legal are marked.

A \( \Phi \) node that is marked legal and one of its operands is "1" is a candidate for insertion of \( E \) into Block B (associated with \( L \)).

Down safety is a necessary condition!

- Profitable? Next section.
What do we mean with "same results must be computed"?

- What happens to exceptions?

(Inserting a copy of E into some block B might generate an exception)

Consult language definition!

- Clearly defined
  (maybe restrictive)

- "implementation dependent"

- Forgotten
(one escape: let the user decide, add flag to compiler to force/allow some transformation)

Analyze program to determine "visible state"
- change to visible state not allowed
- additional constraints for legality of inserting a copy of E

* copy cannot be inserted if it changes visible state.

* consider delayed response (delayed response to an exception)
  - software based
  - hardware support
Assume $E$ can be inserted into $B$

... but exceptions generated by $E$ cannot be reported early

* Software-based handling

  Keep exception $E$ generated by $E$ in $B$ "pending" until $E \circ \phi_0$ would be evaluated in $B'$. $(B' \text{ exists})$

  ? How do we know when "at $\phi_0" would have been evaluated?

* Hardware support

  Special instructions/special operations for "early" (hoisted) operations/instructions
1d \ a[\text{box}] \rightarrow \mathbb{R}^3 \quad \text{normal load}

sp-1d \ a[\text{box}] \rightarrow \mathbb{R} \quad \text{special load: any exception is delayed}

delay response until value (or value computed with "loaded" value) is to be stored.
- no special store

check the results at the time compiler attempts to update memory