IR Lowering

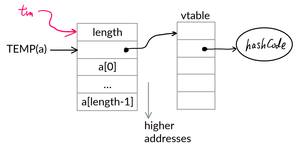
Expressions	Statements
CONST(n)	MOVE(e_{dst}, e)
TEMP(t)	~~EXP(e)~~
OP(e_1, e_2)	~~SEQ($\vec{s}$)~~
MEM(e)	JUMP(ℓ)
~~CALL(e_f, $\vec{e}$)~~	~~CJUMP(e, ℓ_t, ℓ_f)~~
NAME(ℓ)	LABEL(ℓ)
~~ESEQ($\vec{s}$, e)~~	RETURN(e)
	CALL(e_f, $\vec{e}$)

Recap
- TIR
- Expressions and statements have mutually recursive syntax
- Translating Joos expressions and statements to IR:
  E⟦e⟧ = e' and S⟦s⟧ = s': mutually recursive functions

CJUMP(e, ℓ_t, ℓ_f)

- Flatten ESEQ and SEQ to a single top-level SEQ
- expressions don't have side effects
- ≤1 side effect in each statement
- no two-way jumps

E⟦e1[e2]⟧ = ESEQ(MOVE(t_a, E⟦e1⟧);
CJUMP(EQ(t_a, 0), ℓ_{null}, $\ell_{notnull}$); // null check
$\ell_{notnull}$:
CALL(NAME(__exception))
ℓ_{err}:
$\ell_{notnull}$:
MOVE(t_i, E⟦e2⟧);
CJUMP(LTU(t_i, MEM(t_a − 4)), $\ell_{inbound}$, ℓ_{err}); // bounds check
$\ell_{inbound}$:
MEM(t_a + t_i×4 + 4))



E⟦ $e_1[e_2] = e_3$ ⟧ =
  MOVE(t_a, E⟦ e_1 ⟧) ... null check
  MOVE(t_i, E⟦ e_2 ⟧) ... bounds check
  MOVE(t_e, E⟦ e_3 ⟧) .. assignability check (A6)
  MOVE(MEM(t_a + t_i x4+4), t_e)
  t_e

E⟦ new int[e] ⟧ =
  MOVE(t_n, E⟦e⟧) ... size wrong check
  MOVE(t_m, CALL(__malloc, 4×t_n + 8))
  MOVE(MEM(t_m), t_n)
  MOVE(MEM(t_m + 4) , ...)
  ... loop to zero-initialize array
  t_m + 4

Lower TIR to Canonical-form IR.

Joos AST ⟶ TIR ⟶ --- IR

L⟦ S ⟧ = s_1 ; s_2 ; ··· ; s_n
L⟦ e ⟧ = s_1 ; s_2 ; ··· ; s_n | e'
(side effects | pure)

L⟦ l: ⟧ = l:
L⟦ s_1 ; s_2 ; ··· ; s_n ⟧ = L⟦s_1⟧ ; L⟦s_2⟧ ; ··· ; L⟦s_n⟧

$$\frac{L⟦e⟧ = \vec{s} \mid e'}{L⟦ \text{JUMP}(e) ⟧ = \vec{s} ; \text{JUMP}(e')}$$

$$\frac{L⟦e⟧ = \vec{s} \mid e'}{L⟦ \text{EXP}(e) ⟧ = \vec{s}}$$

$$\frac{L⟦e⟧ = \vec{s} \mid e'}{L⟦ \text{RETURN}(e) ⟧ = \vec{s} ; \text{RETURN}(e')}$$

L⟦ n ⟧ = • | n
L⟦ t ⟧ = • | t
L⟦ l ⟧ = • | l

$$\frac{L⟦e⟧ = \vec{s} \mid e'}{L⟦ \text{MEM}(e) ⟧ = \vec{s} \mid \text{MEM}(e')} \qquad \frac{L⟦s⟧ = \vec{s'} \quad L⟦e⟧ = \vec{s''} \mid e'}{L⟦ \text{ESEQ}(s, e) ⟧ = \vec{s'}; \vec{s''} \mid e'}$$

$$\frac{L⟦e_1⟧ = \vec{s_1} \mid e_1' \quad L⟦e_2⟧ = \vec{s_2} \mid e_2'}{L⟦ \text{OP}(e_1, e_2) ⟧ = \vec{s_1} ; \vec{s_2} \mid \text{OP}(e_1', e_2')}$$

When does s_2 have side effects that affect e_1' ?
- s_2 writes into temp read by e_1' — easy to detect
  t_3 = t_9 ?
- s_2 ---------- mem ---------- undecidable
  MEM(e_3) = MEM(e_4) ?

$$\frac{L⟦e_1⟧ = \vec{s_1} \mid e_1' \quad L⟦e_2⟧ = \vec{s_2} \mid e_2'}{L⟦ \text{OP}(e_1, e_2) ⟧ = \vec{s_1} ; \text{MOVE}(t, e_1') ; \vec{s_2} \mid \text{OP}(t, e_2')}$$

$$\frac{\forall i \;\; L⟦e_i⟧ = \vec{s_i} \mid e_i'}{L⟦ \text{CALL}(e_f, \vec{e}_n) ⟧ = \vec{s_f} ; \text{MOVE}(t_0, e_0') ; \vec{s_1} ; \text{MOVE}(t_1, e_1') ; \vdots ; \vec{s_n} ; \text{MOVE}(t_n, e_n') ; \text{CALL}(t_0, t_1, \cdots, t_n) \mid \text{TEMP(RET)}}$$
(eax in x86)

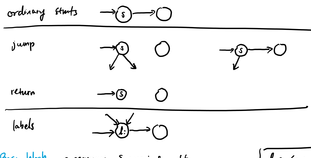
$$\frac{L⟦e_2⟧ = \vec{s_1} \mid e_2'}{L⟦ \text{MOVE}(t, e_2) ⟧ = \vec{s_2} ; \text{MOVE}(t, e_2')}$$

$$\frac{L⟦e_1⟧ = \vec{s_1} \mid e_1' \quad L⟦e_2⟧ = \vec{s_2} \mid e_2'}{L⟦ \text{MOVE}(\text{MEM}(e_1), e_2) ⟧ = \vec{s_1} ; \text{MOVE}(t, e_1') ; \vec{s_2} ; \text{MOVE}(\text{MEM}(t), e_2')}$$

Eliminate two-way jumps.

CJUMP(e, l_t, l_f) ⟶ CJUMP(e, l_t) ;
JUMP(l_f) ;

Idea: reorder statements so that l_f is the next statement.



Basic block: a sequence s_1 ; ··· ; s_n s.t.
- all but s_1 and s_n are ordinary
- s_1 can be a label statement
- s_n can be ordinary or jump / return

l: s_2 ; ⋮ s_{n-1} ; s_n / jump / return

Maximal basic block.

Example.

l_0: CJUMP(e, l_2, l_3)
l_1: MOVE(x, y)
l_2: MOVE(x, y+2)
JUMP(l_1)
l_3: CALL(f)
RETURN

CFG.
Vertices = maximal BBs.
Edges = control flow.

- delete jumps to next BB
- invert cjumps as needed, drop orgs
- append jumps to BB as needed
- delete unneeded labels

NOT(e) l_3
l_0: CJUMP(~~e~~, ~~l_2~~, ~~l_3~~)
l_2: MOVE(x, y+2)
~~JUMP(l_1)~~
~~l_1~~: MOVE(x, y)
JUMP(l_2)
l_3: CALL(f)
RETURN

Trace: a CFG path of distinct BBs.

Reorder (Greedy)
1. Init all BBs as "unmarked"
2. Repeat until all BBs are marked.
   - Choose an unmarked block
   - Mark a maximal unmarked trace
   - Append blocks in that trace to output code
     (while simplifying / fixing Jumps & CJumps)