Intermediate Representation

Expressions $e ::=$
- CONST(n) — literal — n
- \| TEMP(t) — t
- \| MEM(e)
- \| OP(e_1, e_2) — $e_1 + e_2$
- \| CALL(e_f, e_1, ⋯, e_n)
- \| NAME(l) — l
- \| ESEQ(s, e) — do statement s and eval e — s; e

Statements $s ::=$
- MOVE(e_{dst}, e_{src}) — either MEM(e) or TEMP(t)
- \| EXP(e) — EXP(CALL(e_f, e))
- \| SEQ(s_1, s_2, ⋯, s_n) — s_1; s_2; ⋯; s_n
- \| JUMP(e)
- \| CJUMP(e, l_1, l_2)
- \| LABEL(l) — l:
- \| RETURN(e)

OP ::= ADD | SUB | MUL | DIV | MOD | AND | OR | XOR | NOT | RSHIFT | LSHIFT | ARSHIFT | EQ | NEQ | LT | LEQ | ⋯

Translating Joos to TIR

$E[\![e]\!] = e'$ (Joos → TIR) $\quad S[\![s]\!] = s' \quad F[\![f]\!] = s$

$E[\![n]\!] = n \quad E[\![\text{true}]\!] = 1 \quad E[\![\text{false}]\!] = 0$

$E[\![e_1 + e_2]\!] = \text{ADD}(E[\![e_1]\!], E[\![e_2]\!])$

$E[\![x]\!] = \text{TEMP}(x)$

$E[\![f(e_1, \cdots, e_n)]\!] = \text{CALL}(\text{NAME}(f), E[\![e_1]\!], \cdots, E[\![e_n]\!])$

$S[\![\text{int } x = e;]\!] = \text{MOVE}(x, E[\![e]\!])$

$S[\![s_1; \cdots; s_n]\!] = \text{SEQ}(S[\![s_1]\!], \cdots, S[\![s_n]\!])$

$S[\![\text{if}(e)\ s]\!] =$
```
CJUMP(E[[e]], l_t, l_f);
l_t: S[[s]];
l_f:
```

$S[\![\text{return } e]\!] = \text{RETURN}(E[\![e]\!])$

$F[\![T\ f(T_1\ x_1, \cdots, T_n\ x_n)\ \{s\}]\!] =$
```
f: MOVE(x_1, arg 0); ⋯; MOVE(x_n, arg_{n-1}); S[[s]]
```

$E[\![e_1 \,\&\&\, e_2]\!] =$
```
MOVE(t, 0);
CJUMP(E[[e_1]], l_t, l_f);
l_t: MOVE(t, E[[e_2]]);
l_f:
t
```

$S[\![\text{if}(e_1 \,\&\&\, e_2)\ s]\!] =$ CJUMP(E[[e_1 && e_2]], l_t, l_f); l_t: S[[s]]; l_f:

Better:

$S[\![\text{if}(e_1 \,\&\&\, e_2)\ s]\!] =$
```
CJUMP(E[[e_1]], l, l_f);
l: CJUMP(E[[e_2]], l_t, l_f);
l_t: S[[s]]
l_f:
```

$C[\![e, l_t, l_f]\!]$ = IR statement that jumps to l_t if e is true, l_f otherwise

$S[\![\text{if}(e)\ s]\!] =$ C[[e, l_t, l_f]]; l_t: S[[s]]; l_f:

$C[\![\text{true}, l_t, l_f]\!] = \text{JUMP}(l_t)$

$C[\![\text{false}, l_t, l_f]\!] = \text{JUMP}(l_f)$

$C[\![!e, l_t, l_f]\!] = C[\![e, l_f, l_t]\!]$

$C[\![e_1 \,\&\&\, e_2, l_t, l_f]\!] =$ CJUMP(E[[e_1]], l, l_f); l: CJUMP(E[[e_2]], l_t, l_f);

Better:

$C[\![e_1 \,\&\&\, e_2, l_t, l_f]\!] = C[\![e_1, l, l_f]\!]; \ l: C[\![e_2, l_t, l_f]\!]$

$C[\![e_1 \,\|\, e_2, l_t, l_f]\!] = C[\![e_1, l_t, l$?$]\!]; \ l: C[\![e_2, l_t, l_f]\!]$

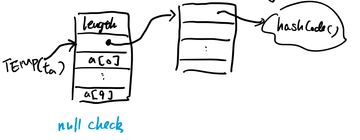
fall-back case

$C[\![e, l_t, l_f]\!] = \text{CJUMP}(E[\![e]\!], l_t, l_f)$

$S[\![\text{while}(e)\ s]\!] =$ l_w: C[[e, l_t, l_f]]; l_t: S[[s]]; JUMP(l_w); l_f:

Arrays

int[] a = new int[10];



E[[e_1[e_2]]] =
```
MOVE(t_a, E[[e_1]]);          null check
CJUMP(EQ(t_a, 0), l_notnull, l_err)
l_err: CALL(__exception)
l_notnull:
MOVE(t_i, E[[e_2]]);          bounds check
CJUMP(LTU(t_i, MEM(t_a - 4)), l_inbound, l_err)
l_inbound:
MEM(t_a + t_i x 4 + 4)
```