Canonisation

Abstract

It's useful to be able to evaluate the sub-expressions of an expression in any order. If tree expressions did not contain `ESEQ` and `CALL` nodes, then the order of evaluation would not matter.

Why `CALL` nodes are an issue?

In actual implementation, `CALL` nodes will return value in the same register (`a0` in case of RISC V). Thus in an expression like `BINOP(PLUS, CALL(...), CALL(...))`; the second call will overwrite the `a0` register before the `PLUS` can be executed.

Remedy is to do the transformation; `CALL(fun, args) -> ESEQ(MOVE(TEMP t, CALL(fun, args)), TEMP t)`

Why `ESEQ` nodes are an issue?

Clearly in case of simple `ESEQ(s, e)`, statement `s` can have direct or side effects on an expression `e`.

Remedy is as shown in below figure (basically lifting them higher and higher until they become `SEQ` nodes).

The transformation is done in three stages: First, a tree is rewritten into a list of canonical trees without `SEQ` or `ESEQ` nodes; then this list is grouped into a set of basic blocks, which contain no internal jumps or labels; then the basic blocks are ordered into a set of traces in which every `CJUMP` is immediately followed by its false label.

Signature

```
signature CANON =
sig
  val linearize : Tree.stm -> Tree.stm list
    (*
      From an arbitrary Tree statement, produce a list of cleaned
trees satisfying the following properties:
```

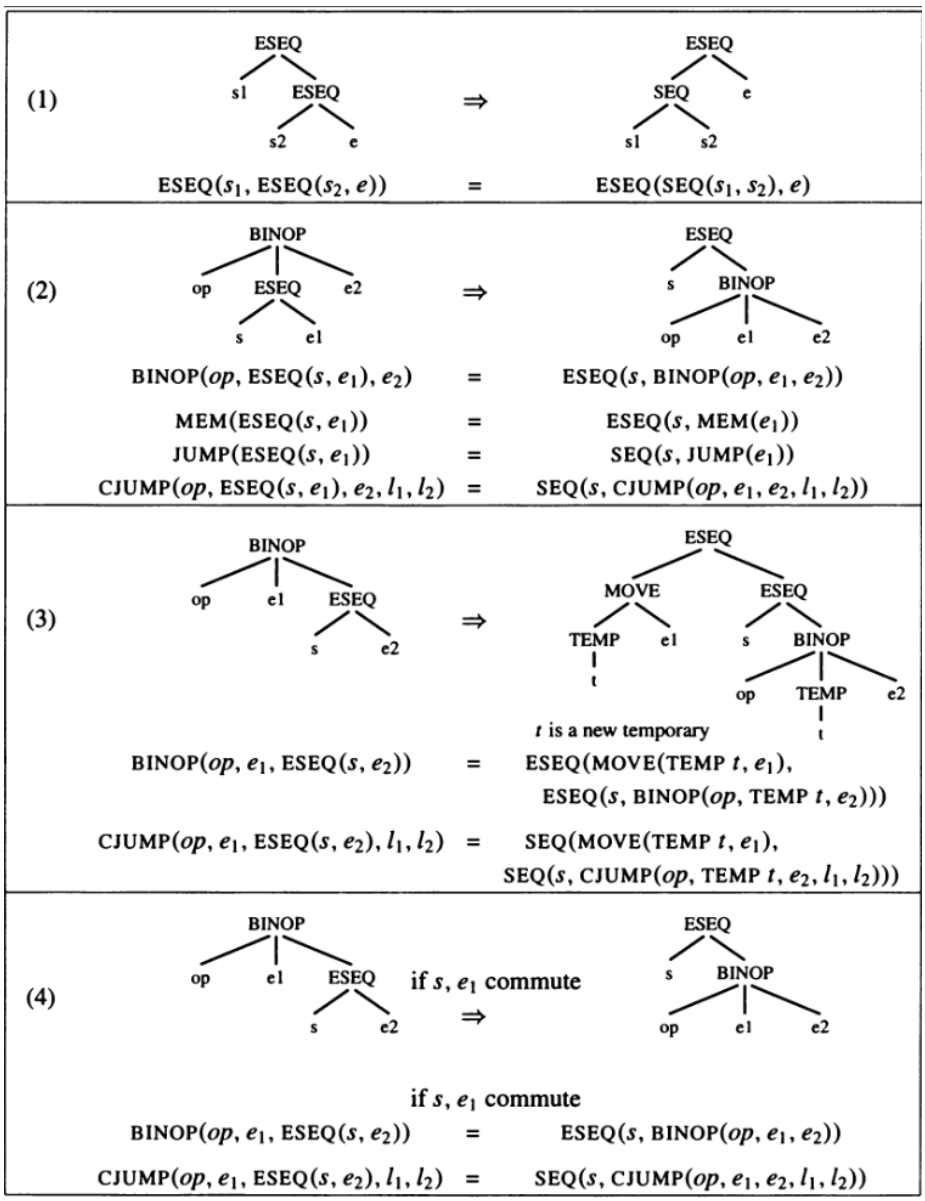

Figure 1: ESEQ Removal

```
      1.  No SEQ's or ESEQ's
      2.  The parent of every CALL is an EXP(..) or a MOVE(TEMP
t,..)
    *)

  val basicBlocks : Tree.stm list -> (Tree.stm list list *
  Tree.label)
    (*
      From a list of cleaned trees, produce a list of basic
blocks satisfying the following properties:
      1. and 2. as above;
      3.  Every block begins with a LABEL;
      4.  A LABEL appears only at the beginning of a block;
      5.  Any JUMP or CJUMP is the last stm in a block;
      6.  Every block ends with a JUMP or CJUMP;
      Also produce the "label" to which control will be passed
upon exit.
    *)

  val traceSchedule : Tree.stm list list * Tree.label -> Tree.stm
  list
    (*
      From a list of basic blocks satisfying properties 1-6,
along with an "exit" label, produce a list of stms such that:
      1. and 2. as above;
      7. Every CJUMP(_,t,f) is immediately followed by LABEL f.
The blocks are reordered to satisfy property 7; also in this
reordering as many JUMP(T.NAME(lab)) statements as possible are
eliminated by falling through into T.LABEL(lab).
    *)
end
```