Basic Blocks and Traces

Lecture 8

Canonical Trees

```
signature CANON =
sig

  val linearize : Tree.stm -> Tree.stm list

  val basicBlocks : Tree.stm list ->
                      (Tree.stm list list * Tree.label)

  val traceSchedule : Tree.stm list list * Tree.label ->
                        Tree.stm list

end (* signature CANON *)
```

Canonical Trees

```
signature CANON =
sig

  val linearize : Tree.stm -> Tree.stm list
      (* From an arbitrary Tree statement, produce a list of
         canonical trees satisfying the following properties:
          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)

  val traceSchedule : Tree.stm list list * Tree.label ->
                          Tree.stm list

end (* signature CANON *)
```

Basic Blocks

```
signature CANON =
sig

  val linearize : Tree.stm -> Tree.stm list

  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

end (* signature CANON *)
```

Traces

```
signature CANON =
sig

  val linearize : Tree.stm -> Tree.stm list

  val basicBlocks : Tree.stm list ->
                      (Tree.stm list list * Tree.label)

  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 (* signature CANON *)
```

Canonical Trees

Canonical trees are those that:

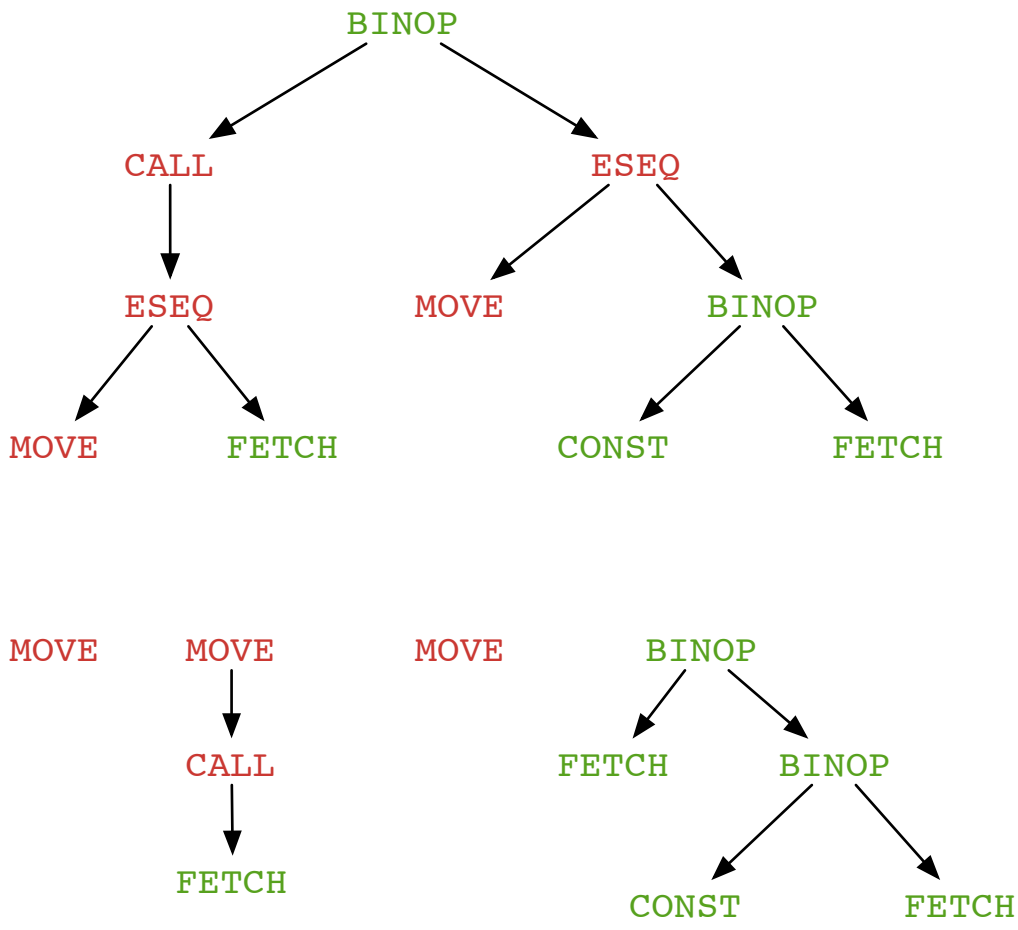
1. Have no SEQ or ESEQ subterms

2. CALLs appear only as components of stms, not as subexpressions, i.e. a CALL node has parent of the form EXP(_) or MOVE(TEMP(t),_)

The idea is to separate out statements with side-effects from pure expressions. This allows freedom to change the order of evaluation in expressions and simplifies the interaction between expression evaluation (function calls in particular), and side-effects like assignment.

Linearization pulls stms and function calls to the top and front, linked with SEQ and ESEQ. Then the SEQ and ESEQ chain can be simplified to a list of canonical trees.

Canonical Tree Transformation



Canonical Tree Transforms

A number of term transformations are used to rearrange expressions into canonical form (Figure 8.1). E.g.:

ESEQ(*s1*,ESEQ(*s2*,*e*)) ==> ESEQ(SEQ(*s1*,*s2*),*e*)

BINOP(*op*,(ESEQ(*s*,*e1*),*e2*) ==>
ESEQ(*s*,(BINOP(*op*,*e1*,*e2*))

BINOP(*op*,*e1*,(ESEQ(*s*,*e2*)) ==>
ESEQ(MOVE(TEMP t_{new}, *e1*),
ESEQ(*s*,(BINOP(*op*,FETCH(TEMP t_{new}),*e2*))

BINOP(*op*,*e1*,(ESEQ(*s*,*e2*)) ==> ESEQ(s,BINOP(*op*,*e1*,*e2*))

if *s* and *e1* *commute* (i.e. are noninterfering, the effects performed by *s* will not change the value computed by *e1*)