

```

1: (* traceschedule.sml
2: *
3: *   Trace-scheduling basic blocks.
4: *   This code is based on Andrew Appel's book "Modern Compiler
5: *   Implementation in ML".
6: *
7: *   Copyright (c) 2005 by Matthias Blume (blume@tti-c.org)
8: *)
9: structure TraceSchedule : sig
10:
11:   val schedule : BBTREE.cluster -> TraceTree.entrytrace
12:
13: end = struct
14:
15:   structure L = Label
16:   structure M = L.Map
17:   structure B = BBTREE
18:   structure T = TraceTree
19:
20:   type label = Label.label
21:
22:   fun bug m = ErrorMsg.impossible ("TraceSchedule: " ^ m)
23:
24:   fun schedule { entryblocks, labelblocks } =
25:     let fun adde ((vl, (l, b)), m) = M.insert (m, l, (SOME vl, b))
26:         fun addl ((l, b), m) = M.insert (m, l, (NONE, b))
27:         val table = foldl addl (foldl adde M.empty entryblocks) labelblocks
28:
29:         fun findjt (table, l) =
30:           case M.find (table, l) of
31:             NONE => NONE
32:           | SOME (NONE, b) => SOME b
33:           | SOME (SOME _, _) => bug "(c)jump to entry point"
34:
35:         fun mktrace (table, (l, b), q) =
36:           let val table = #1 (M.remove (table, l))
37:               fun build (B.JUMP l') =
38:                 (case findjt (table, l') of
39:                   NONE => T.JUMP (l', startnew (table, l' :: q))
40:                 | SOME b' => T.LABEL (mktrace (table, (l', b'), q)))
41:               | build (B.TCALL (e, el)) = T.TCALL (e, el, startnew (table, q))
42:               | build (B.RETURN e) = T.RETURN (e, startnew (table, q))
43:               | build (B.CJUMP (r, e1, e2, t, f)) =
44:                 (case findjt (table, f) of
45:                   NONE =>
46:                     (case findjt (table, t) of
47:                       NONE =>
48:                         T.CJUMP (r, e1, e2, t,
49:                               (Label.new NONE,
50:                                T.JUMP (f, startnew (table, q))))
51:                     | SOME b' =>
52:                         T.CJUMP (TreeOps.notRel r, e1, e2, f,
53:                               mktrace (table, (t, b'), t :: q)))
54:                   | SOME b' =>
55:                     T.CJUMP (r, e1, e2, t,
56:                               mktrace (table, (f, b'), f :: q)))
57:               | build (B.MOVE (le, e, b')) = T.MOVE (le, e, build b')
58:               | build (B.CALL (lel, e, el, b')) = T.CALL (lel, e, el, build b')
59:               | build (B.DOEXP (e, b')) = T.DOEXP (e, build b')
60:               | build (B.DOCALL (e, el, b')) = T.DOCALL (e, el, build b')
61:               | build (B.GCTEST (e, b')) = T.GCTEST (e, build b')
62:               | build (B.ALLOCWRITE (e, b')) = T.ALLOCWRITE (e, build b')
63:           in (l, build b)
64:           end
65:
66:   and startnew (_, []) = T.END
67:     | startnew (table, l :: rest) =
68:       case M.find (table, l) of
69:         SOME (SOME vl, b) => T.ENTRY (vl, mktrace (table, (l, b), rest))
70:       | SOME (NONE, b) => T.JTARGET (mktrace (table, (l, b), rest))
71:       | NONE => startnew (table, rest)
72:   in case entryblocks of
73:     (vl, eb) :: ebs => (vl, mktrace (table, eb, map (#1 o #2) ebs))
74:   | _ => bug "no entry block"
75:   end
76: end

```