

```

1: (* lambda2anf.sml
2: *
3: *   Conversion from straight Lambda to ANF.
4: *
5: *   Copyright (c) 2005 by Matthias Blume (blume@tti-c.org)
6: *)
7: structure LambdaToANF : sig
8:
9:   val convert : Lambda.function -> ANF.function
10:
11: end = struct
12:
13:   structure L = Lambda
14:   structure A = ANF
15:
16:   fun convert f =
17:     let fun tmpvar NONE = LVar.new "tmp"
18:         | tmpvar (SOME v) = LVar.clone v
19:     fun ijump (f, x) = A.JUMP (Purity.Impure, (A.VAR f, [x]))
20:     fun exp (L.VALUE v, _, k) = k v
21:         | exp (L.LET (v, e, b), v0, k) =
22:           exp (e, SOME v, fn x =>
23:             A.BIND (v, x, exp (b, v0, k)))
24:         | exp (L.FIX (fl, b), v0, k) =
25:           A.FIX (map function fl, exp (b, v0, k))
26:         | exp (L.ARITH (aop, e1, e2), v0, k) =
27:           let val v = tmpvar v0
28:             in exp (e1, NONE, fn x1 =>
29:               exp (e2, NONE, fn x2 =>
30:                 A.ARITH (aop, x1, x2, v, k (A.VAR v))))
31:           end
32:         | exp (L.RECORD (m, e1), v0, k) =
33:           let val v = tmpvar v0
34:             in explist (e1, fn x1 =>
35:               A.RECORD (m, x1, v, k (A.VAR v)))
36:           end
37:         | exp (L.SELECT (e, i, m), v0, k) =
38:           let val v = tmpvar v0
39:             in exp (e, NONE, fn x =>
40:               A.SELECT (x, i, m, v, k (A.VAR v)))
41:           end
42:         | exp (L.UPDATE (e1, i, e2), _, k) =
43:           exp (e1, NONE, fn x1 =>
44:             exp (e2, NONE, fn x2 =>
45:               A.UPDATE (x1, i, x2, k (A.INT 0))))
46:         | exp (L.CMP (cop, e1, e2, et, ef), v0, k) =
47:           let val f = tmpvar NONE
48:             val v = tmpvar v0
49:             val et' = exp (et, v0, fn x => ijump (f, x))
50:             val ef' = exp (ef, v0, fn x => ijump (f, x))
51:             in exp (e1, NONE, fn x1 =>
52:               exp (e2, NONE, fn x2 =>
53:                 A.FIX ([{ f = (f, [v], k (A.VAR v)) }, inl = false ],
54:                   A.CMP (cop, x1, x2, et', ef'))))
55:           end
56:         | exp (L.APP (ta, e, el), v0, k) =
57:           let val v = tmpvar v0
58:             in exp (e, NONE, fn x =>
59:               explist (el, fn x1 =>
60:                 A.CALL (ta, [v], (x, x1), k (A.VAR v))))
61:           end
62:
63:   and exp' (L.LET (v, e, b)) =
64:     exp (e, SOME v, fn x => A.BIND (v, x, exp' b))
65:   | exp' (L.FIX (fl, b)) =

```

```
66:           A.FIX (map function f1, exp' b)
67:       | exp' (L.CMP (cop, e1, e2, et, ef)) =
68:           exp (e1, NONE, fn x1 =>
69:               exp (e2, NONE, fn x2 =>
70:                   A.CMP (cop, x1, x2, exp' et, exp' ef)))
71:       | exp' (L.APP (ta, e, el)) =
72:           exp (e, NONE, fn x =>
73:               explist (el, fn x1 =>
74:                   A.JUMP (ta, (x, x1))))
75:       | exp' e = exp (e, NONE, fn x => A.VALUES [x])
76:
77:   and explist ([], k) = k []
78:   | explist (e :: el, k) =
79:       exp (e, NONE, fn x => explist (el, fn x1 => k (x :: x1)))
80:
81:   and function (f, vl, b, inl) = { f = (f, vl, exp' b), inl = inl }
82: in function f
83: end
84: end
```