

```

1  |----- MODULE OT -----|
  | Specification of OT (Operational Transformation) functions. It consists of the basic OT functions
  | for two operations and more general ones involving operation sequences.
7  | EXTENDS Op
  |-----|
8  |
  | OT (Operational Transformation) functions.
  | Naming convention: I for “Ins” and D for “Del”.
  |
  | The left “Ins” lins transformed against the right “Ins” rins.
18 XformII(lins, rins)  $\triangleq$ 
19   IF lins.pos < rins.pos
20   THEN lins
21   ELSE IF lins.pos > rins.pos
22         THEN [lins EXCEPT !.pos = @ + 1]
23         ELSE IF lins.ch = rins.ch
24               THEN Nop
25               ELSE IF lins.pr > rins.pr
26                     THEN [lins EXCEPT !.pos = @ + 1]
27                     ELSE lins
  |
  | The left “Ins” lins transformed against the right “Del” rdel.
32 XformID(ins, del)  $\triangleq$ 
33   IF ins.pos ≤ del.pos
34   THEN ins
35   ELSE [ins EXCEPT !.pos = @ - 1]
  |
  | The left “Del” ldel transformed against the right “Ins” rins.
40 XformDI(del, ins)  $\triangleq$ 
41   IF del.pos < ins.pos
42   THEN del
43   ELSE [del EXCEPT !.pos = @ + 1]
  |
  | The left “Del” ldel transformed against the right “Del” rdel.
48 XformDD(ldel, rdel)  $\triangleq$ 
49   IF ldel.pos < rdel.pos
50   THEN ldel
51   ELSE IF ldel.pos > rdel.pos
52         THEN [ldel EXCEPT !.pos = @ - 1]
53         ELSE Nop
54  |-----|
  | Transform the left operation lop against the right operation rop with appropriate OT function.
59 Xform(lop, rop)  $\triangleq$ 
60   CASE lop = Nop ∨ rop = Nop → lop
61   □ lop.type = “Ins” ∧ rop.type = “Ins” → XformII(lop, rop)

```

```

62      □  $lop.type = \text{"Ins"} \wedge rop.type = \text{"Del"} \rightarrow XformID(lop, rop)$ 
63      □  $lop.type = \text{"Del"} \wedge rop.type = \text{"Ins"} \rightarrow XformDI(lop, rop)$ 
64      □  $lop.type = \text{"Del"} \wedge rop.type = \text{"Del"} \rightarrow XformDD(lop, rop)$ 

```

Generalized *OT* functions on operation sequences.

Iteratively/recursively transforms the operation *op* against an operation sequence *ops*.

```

74  RECURSIVE  $XformOpOps(-, -)$ 
75   $XformOpOps(op, ops) \triangleq$ 
76    IF  $ops = \langle \rangle$ 
77    THEN  $op$ 
78    ELSE  $XformOpOps(Xform(op, Head(ops)), Tail(ops))$ 

```

Iteratively/recursively transforms the operation *op* against an operation sequence *ops*. Different from *XformOpOps*, *XformOpOpsX* maintains the intermediate transformed operation

```

86  RECURSIVE  $XformOpOpsX(-, -)$ 
87   $XformOpOpsX(op, ops) \triangleq$ 
88    IF  $ops = \langle \rangle$ 
89    THEN  $\langle op \rangle$ 
90    ELSE  $\langle op \rangle \circ XformOpOpsX(Xform(op, Head(ops)), Tail(ops))$ 

```

Iteratively/recursively transforms the operation sequence *ops* against an operation *op*.

```

96   $XformOpsOp(ops, op) \triangleq$ 
97    LET  $opX \triangleq XformOpOpsX(op, ops)$ 
98    IN  $[i \in 1 \dots Len(ops) \mapsto Xform(ops[i], opX[i])]$ 

```

The *CP1* (Convergence) Property.

```

103  $CP1 \triangleq \forall l \in List, op1 \in Op, op2 \in Op :$ 
104    $ApplyOps(\langle op1, Xform(op2, op1) \rangle, l) = ApplyOps(\langle op2, Xform(op1, op2) \rangle, l)$ 

```

```

\ * Modification History
\ * Last modified Wed Jul 04 15:16:53 CST 2018 by hengxin
\ * Created Sun Jun 24 15:57:48 CST 2018 by hengxin

```