```
1  ---------- MODULE CC ----------
   TLA+ specification of Causal Consistency variants, including CC, CM, and CCv.
   See the paper "On Verifying Causal Consistency" (POPL'2017).
8  EXTENDS Naturals, Sequences, FiniteSets, Functions, FiniteSetsExt,
9          RelationUtils, TLC, PartialOrderExt

11 Key ≜ Range("abcdefghijklmnopqrstuvwxyz")  We assume single-character keys.
12 Val ≜ Nat      We assume values from Nat.
13 InitVal ≜ 0    We follow the convention in POPL'2017.
14 Oid ≜ Nat      We assume operation identifiers from Nat.

16 Operation ≜ [type : {"read", "write"}, key : Key, val : Val, oid : Oid]
17 R(k, v, oid) ≜ [type ↦ "read", key ↦ k, val ↦ v, oid ↦ oid]
18 W(k, v, oid) ≜ [type ↦ "write", key ↦ k, val ↦ v, oid ↦ oid]

20 Session ≜ Seq(Operation)  A session s ∈ Session is a sequence of operations.
21 History ≜ SUBSET Session  A history h ∈ History is a set of sessions.
22 ----------
   Utility operators for operations.
26 Ops(h) ≜  Return the set of all operations in history h ∈ History.
27     UNION {Range(s) : s ∈ h}

29 ReadOps(h) ≜  Return the set of all read operations in history h ∈ History.
30     {op ∈ Ops(h) : op.type = "read"}

32 ReadOpsOnKey(h, k) ≜  Return the set of all read operations on key k ∈ Key in history h ∈ History.
33     {op ∈ Ops(h) : op.type = "read" ∧ op.key = k}

35 WriteOps(h) ≜  Return the set of all write operations in history h ∈ History.
36     {op ∈ Ops(h) : op.type = "write"}

38 WriteOpsOnKey(h, k) ≜  Return the set of all write operations on key k ∈ Key in history h ∈ History
39     {op ∈ Ops(h) : op.type = "write" ∧ op.key = k}
40 ----------
   Well-formedness of history h ∈ History:
   - TODO: type invariants
   - uniqueness of oids
47 WellFormed(h) ≜
48   ∧ h ∈ History
49     ∧ LET ops ≜ Ops(h)
50           nops ≜ Cardinality(ops)
51           oids ≜ {o.oid : o ∈ ops}
52       IN  ∧ ∀ op ∈ ops :  Type invariants
53             ∨ op.type = "write"
54             ∨ op.type = "read"
55           ∧ nops = Cardinality(oids)  Uniqueness of oids
```

```
56                 ∧ nops = ReduceSet(LAMBDA s, x : Len(s) + x, h, 0)
57 ├──────────────────────────────────────────────────────────────────────┤
     Auxiliary definitions for the axioms used in the definitions of causal consistency
61   The program order of h ∈ History is a union of total orders among operations in the same session
62 PO(h) ≜ UNION {Seq2Rel(s) : s ∈ h}

64   The set of operations that preceed o ∈ Operation in program order in history h ∈ History
65 StrictPOPast(h, o) ≜ InverseImage(PO(h), o)
66 POPast(h, o) ≜ StrictPOPast(h, o) ∪ {o}  Original definition in paper, including itself


69   The set of operations that preceed o ∈ Operation in causal order co
70 StrictCausalPast(co, o) ≜ InverseImage(co, o)
71 CausalPast(co, o) ≜ StrictCausalPast(co, o) ∪ {o}  Original definition in paper, including itself

73   The restriction of causal order co to the operations in the causal past of operation o ∈ Operation
74 StrictCausalHist(co, o) ≜ co | StrictCausalPast(co, o)
75 CausalHist(co, o) ≜ co | CausalPast(co, o)  Original definition in paper

77   The restriction of arbitration arb to the operations in the causal past of operation o ∈ Operation
78 StrictCausalArb(co, arb, o) ≜ arb | StrictCausalPast(co, o)
79 CausalArb(co, arb, o) ≜ arb | CausalPast(co, o)  Original definition in paper
80 ├──────────────────────────────────────────────────────────────────────┤
     Axioms used in the defintions of causal consistency
84 RWRegSemantics(seq, o) ≜  Is o ∈ Operation legal when it is appended to seq
85     IF o.type = "write" THEN TRUE
86      ELSE LET wseq ≜ SelectSeq(seq, LAMBDA op : op.type = "write" ∧ op.key = o.key)
87           IN  IF wseq = ⟨⟩ THEN o.val = InitVal
88               ELSE o.val = wseq[Len(wseq)].val

90 PreSeq(seq, o) ≜  All of the operations before o in sequence seq
91     LET so ≜ Seq2Rel(seq)
92     IN  SelectSeq(seq, LAMBDA op : ⟨op, o⟩ ∈ so)

94 RWRegSemanticsPOPast(seq, popast) ≜  Is ∀ o ∈ popast legal
95      ∧ ∀ o ∈ popast :
96          LET preSeq ≜ PreSeq(seq, o)
97          IN  RWRegSemantics(preSeq, o)

99 AxCausalValue(co, o) ≜
100    LET seqs ≜ AllLinearExtensions(StrictCausalHist(co, o), StrictCausalPast(co, o))
101    IN  TRUE ∈ {RWRegSemantics(seq, o) : seq ∈ seqs}  TODO: shortcut implementation of anyTrue for efficiency

103 AxCausalSeq(h, co, o) ≜
104    LET popast ≜ POPast(h, o)
105        seqs ≜ AllLinearExtensions(CausalHist(co, o), CausalPast(co, o))
106    IN  TRUE ∈ {RWRegSemanticsPOPast(seq, popast) : seq ∈ seqs}
```

```
AxCausalArb(co, arb, o) ≜
    LET seq ≜ AnyLinearExtension(StrictCausalArb(co, arb, o), StrictCausalPast(co, o))  it is unique
    IN  RWRegSemantics(seq, o)

 Directory to store files recording strict partial order relations
POFilePath ≜ "D:\\Education\\Programs\\Python\\EnumeratePO\\POFile\\"

 A set of all subset of the Cartesian Product of ops × ops,
 each of which represent a strict partial order(irreflexive and transitive)
StrictPartialOrderSubset(ops) ≜
    PartialOrderSubset(ops, POFilePath)
```

Specification of *CC*

Final Version: Enumerate all possible strict partial order subsets

```
CC(h) ≜  Check whether h ∈ History satisfies CC (Causal Consistency)
    LET ops ≜ Ops(h)
    IN  ∃ co ∈ StrictPartialOrderSubset(ops) :  Optimized implementation
            ∧ Respect(co, PO(h))                AxCausal
            ∧ PrintT("co: " ∘ ToString(co))
            ∧ ∀ o ∈ ops : AxCausalValue(co, o)  AxCausalValue
```

Version 1: Following the definition of *POPL*2017

```
CC1(h) ≜  Check whether h ∈ History satisfies CC (Causal Consistency)
    LET ops ≜ Ops(h)
    IN  ∃ co ∈ SUBSET (ops × ops) :  Raw implementation: Cartesian Product
            ∧ Respect(co, PO(h))                AxCausal
            ∧ IsStrictPartialOrder(co, ops)
            ∧ PrintT("co: " ∘ ToString(co))
            ∧ ∀ o ∈ ops : AxCausalValue(co, o)  AxCausalValue
```

Specification of *CCv*

Final Version: Enumerate all possible strict partial order subsets

```
CCv(h) ≜  Check whether h ∈ History satisfies CCv (Causal Convergence)
    LET ops ≜ Ops(h)
    IN  ∃ co ∈ StrictPartialOrderSubset(ops) :  Optimized implementation
            ∧ Respect(co, PO(h))                AxCausal
            ∧ PrintT("co: " ∘ ToString(co))
            ∧ ∃ arb ∈ {Seq2Rel(le) : le ∈ AllLinearExtensions(co, ops)} :  AxArb
                ∧ ∀ o ∈ ops : AxCausalArb(co, arb, o)  AxCausalArb
                ∧ PrintT("arb: " ∘ ToString(arb))
```

Version 3: If exists, arbitration order is one of the linear exetentions of *co* on the set *ops*

```
CCv3(h) ≜  Check whether h ∈ History satisfies CCv (Causal Convergence)
    LET ops ≜ Ops(h)
    IN  ∃ co ∈ SUBSET (ops × ops) :  Raw implementation: Cartesian Product
            ∧ Respect(co, PO(h))                                   AxCausal
            ∧ IsStrictPartialOrder(co, ops)
            ∧ PrintT("co: " ∘ ToString(co))
            ∧ ∃ arb ∈ {Seq2Rel(le) : le ∈ AllLinearExtensions(co, ops)} :   AxArb
                  ∧ ∀ o ∈ ops : AxCausalArb(co, arb, o)   AxCausalArb
                  ∧ PrintT("arb: " ∘ ToString(arb))
```

Version 2: Re-arrange clauses

```
CCv2(h) ≜  Check whether h ∈ History satisfies CCv (Causal Convergence)
    LET ops ≜ Ops(h)
    IN  ∃ co ∈ SUBSET (ops × ops) :
            ∧ Respect(co, PO(h))   AxCausal
            ∧ IsStrictPartialOrder(co, ops)
            ∧ PrintT("co: " ∘ ToString(co))
            ∧ ∃ arb ∈ SUBSET (ops × ops) :   to generate; not to test
                  ∧ Respect(arb, co)                                AxArb
                  ∧ IsStrictTotalOrder(arb, ops)
                  ∧ ∀ o ∈ ops : AxCausalArb(co, arb, o)   AxCausalArb
                  ∧ PrintT("arb: " ∘ ToString(arb))
```

Version 1: Following the definition of *POPL*2017

```
CCv1(h) ≜  Check whether h ∈ History satisfies CCv (Causal Convergence)
    LET ops ≜ Ops(h)
    IN  ∃ co ∈ SUBSET (ops × ops) :
            ∧ ∃ arb ∈ SUBSET (ops × ops) :
                ∧ PrintT("co: " ∘ ToString(co))
                ∧ PrintT("arb: " ∘ ToString(arb))
                ∧ IsStrictPartialOrder(co, ops)
                ∧ IsStrictTotalOrder(arb, ops)
                ∧ Respect(co, PO(h))                AxCausal
                ∧ Respect(arb, co)                              AxArb
                ∧ ∀ o ∈ ops : AxCausalArb(co, arb, o)   AxCausalArb
```

Specification of *CM*

Final Version: Enumerate all possible strict partial order subsets

```
CM(h) ≜  Check whether h ∈ History satisfies CM (Causal Memory)
    LET ops ≜ Ops(h)
    IN  ∃ co ∈ StrictPartialOrderSubset(ops) :
            ∧ Respect(co, PO(h))                AxCausal
            ∧ ∀ o ∈ ops : AxCausalSeq(h, co, o)   AxCausalSeq
```

Version 1: Following the definition of *POPL*2017

```
CM1(h) ≜ Check whether h ∈ History satisfies CM (Causal Memory)
    LET ops ≜ Ops(h)
    IN  ∃ co ∈ SUBSET (ops × ops) :
            ∧ IsStrictPartialOrder(co, ops)
            ∧ Respect(co, PO(h))                AxCausal
            ∧ ∀ o ∈ ops : AxCausalSeq(h, co, o) AxCausalSeq
```

Auxiliary operators used in the checking algorithms: We consider only differentiated histories.

```
KeyOf(h) ≜ the set of keys read or written in h ∈ History
    {op.key : op ∈ Ops(h)}

IsDifferentiated(h) ≜ Is h ∈ History differentiated?
    ∀ k ∈ KeyOf(h) :
      LET writes ≜ WriteOpsOnKey(h, k)
      IN  ∀ w1 ∈ writes, w2 ∈ writes :
              ∧ w1.val ≠ w2.val
              ∧ w1.val ≠ InitVal
```

Auxiliary relations used in the checking algorithms

```
RF(h) ≜ the read-from relation TODO: using infix symbolic operator???
    {⟨w, r⟩ ∈ WriteOps(h) × ReadOps(h) : w.key = r.key ∧ w.val = r.val}

CO(h) ≜ the CO order defined as the transitive closure of the union of PO(h) and RF(h)
    TC(PO(h) ∪ RF(h))

CF(h) ≜ the conflict relation
    LET co ≜ CO(h)
        rf ≜ RF(h)
     reads ≜ ReadOps(h)
    writes ≜ WriteOps(h)
    IN  {⟨w1, w2⟩ ∈ writes × writes :
            ∧ w1.key = w2.key
            ∧ w1.val ≠ w2.val
            ∧ ∃ r ∈ reads : ⟨w1, r⟩ ∈ co ∧ ⟨w2, r⟩ ∈ rf}

HB(h) ≜ \* All of the happened-before relation of operation o in history h

BaseHB(h, o) ≜ CO | CasualPast(o)
    LET co ≜ CO(h)
    IN  co | CausalPast(co, o)

HBo(h, o) ≜ Happened-before relation for o, denoted HBo ⊆ O × O, to be the smallest relation such that
    LET po ≜ PO(h)
    writes ≜ WriteOps(h)
      base ≜ BaseHB(h, o)  CO | CasualPast(o) ⊆ HBo
      RECURSIVE HBoRE(_)
      HBoRE(hbo) ≜
```

```
            LET update ≜ {
                    ⟨w1, w2⟩ ∈ writes × writes :
                      ∧ w1.key = w2.key
                      ∧ w1.val ≠ w2.val
                      ∧ ∃ r2 ∈ ReadOpsOnKey(h, w2.key) :
                          ∧ r2.val = w2.val
                          ∧ ⟨w1, r2⟩ ∈ hbo
                          ∧ ∨ r2 = o
                            ∨ ⟨r2, o⟩ ∈ po
                  }
                 hbo2 ≜ update ∪ hbo
            IN   IF hbo2 = hbo
                   THEN hbo
                   ELSE HBoRE(TC(hbo2))
      IN   TC(HBoRE(base))

HB(h) ≜  All happened-before relation for o ∈ history h
    {⟨o, HBo(h, o)⟩ : o ∈ Ops(h)}

  All bad patterns defined in POPL'2017 (see Table 2 of POPL'2017)
CyclicCO(h) ≜ Cyclic(PO(h) ∪ RF(h))

WriteCOInitRead(h) ≜
  ∃ k ∈ KeyOf(h) :
    ∃ r ∈ ReadOpsOnKey(h, k), w ∈ WriteOpsOnKey(h, k) :
      ∧ ⟨w, r⟩ ∈ CO(h)  TODO: for efficiency
      ∧ r.val = InitVal

ThinAirRead(h) ≜
  ∃ k ∈ KeyOf(h) :
    ∃ r ∈ ReadOpsOnKey(h, k) :
      ∧ r.val ≠ InitVal
      ∧ ∀ w ∈ WriteOpsOnKey(h, k) : ⟨w, r⟩ ∉ RF(h)

WriteCORead(h) ≜
  ∃ k ∈ KeyOf(h) :
    ∃ w1, w2 ∈ WriteOpsOnKey(h, k), r1 ∈ ReadOpsOnKey(h, k) :
      ∧ ⟨w1, w2⟩ ∈ CO(h)
      ∧ ⟨w2, r1⟩ ∈ CO(h)  TODO: efficiency
      ∧ ⟨w1, r1⟩ ∈ RF(h)

CyclicCF(h) ≜
  Cyclic(CF(h) ∪ CO(h))

WriteHBInitRead(h) ≜
  ∃ o ∈ Ops(h) :
```

```
        LET hbo ≜ HBo(h, o)
            popast ≜ POPast(h, o)
        IN  ∃ r ∈ popast :
              ∧ r.val = InitVal
              ∧ LET writes ≜ WriteOpsOnKey(h, r.key)
                IN  ∃ w ∈ writes :
                      ⟨w, r⟩ ∈ hbo

CyclicHB(h) ≜
  ∃ o ∈ Ops(h) :
    Cyclic(HBo(h, o))


  Checking algorithms of POPL'2017 (see Table 3 of POPL'2017)
CCAlg(h) ≜  Checking algorithm for CC (Causal Consistency)
  ∧ ¬CyclicCO(h)
  ∧ ¬WriteCOInitRead(h)
  ∧ ¬ThinAirRead(h)
  ∧ ¬WriteCORead(h)

CCvAlg(h) ≜  Checking algorithm for CCv (Causal Convergence)
  ∧ ¬CyclicCO(h)
  ∧ ¬WriteCOInitRead(h)
  ∧ ¬ThinAirRead(h)
  ∧ ¬WriteCORead(h)
  ∧ ¬CyclicCF(h)

CMAlg(h) ≜  TODO: Checking algorithm for CM (Causal Memory)
  ∧ ¬CyclicCO(h)
  ∧ ¬WriteCOInitRead(h)
  ∧ ¬ThinAirRead(h)
  ∧ ¬WriteCORead(h)
  ∧ ¬WriteHBInitRead(h)
  ∧ ¬CyclicHB(h)

\* Modification Historyy
\* Last modified Tue Apr 20 13:26:56 CST 2021 by hengxin
\* Created Tue Apr 01 10:24:07 CST 2021 by hengxin
```