

```

Initialize(consume gcTid:Tid,
            linear mutatorTids:[int]bool) {
...
    async call GarbageCollect(gcTid);
}

```

```

Eq(linear tid:Tid, x:idx, y:idx)
returns (eq:bool) {...}
...
assert ...
eq := rootAbs[x] == rootAbs[y];

```

```

Alloc(consume tid_in:Tid, y:idx)
returns(linear tid:Tid) {
    call tid := TestRootScanBarrier(tid_in);
    call UpdateMutatorPhase(tid);
    var ptr:int, absPtr:obj := AllocRaw(tid, y);
}

```

```

// y = x.f
ReadField(linear tid:Tid, x:idx, f:fld, y:idx) {
    call ReadFieldRaw(tid, x, f, y);
}
...
assert mutatorTidWhole(tid)
&& fieldIndex(f)
&& rootAddr(x) && tidOwens(tid, x)
&& rootAddr(y) && tidOwens(tid, y)
&& memAddrAbs(rootAbs[x]);
rootAbs[y] := memAbs[rootAbs[x]][f];

```

```

// x.f = y
WriteField(linear tid:Tid, x:idx, f:fld, y:idx) {
    call WriteBarrier(tid, y);
    call WriteFieldRaw(tid, x, f, y);
}
...
assert mutatorTidWhole(tid)
&& fieldIndex(f)
&& rootAddr(x) && tidOwens(tid, x)
&& rootAddr(y) && tidOwens(tid, y)
&& memAddrAbs(rootAbs[x]);
memAbs[rootAbs[x]][f] := rootAbs[y];

```

```

assert mutatorTidWhole(tid_in)
&& rootAddr(y) && tidOwens(tid, y);
var o:obj;
assume (memAddrAbs(o) && !allocSet[o]);
allocSet[o] := true;
rootAbs[y] := o;
memAbs[o] := ...initial fields...;
tid := tid_in;

```

phase 6  
interface

```

GarbageCollect(linear tid:Tid) {
    while (true) {
        call WaitForMutators(tid, Handshake(tid));
        call Mark(tid);
        call WaitForMutators(tid, Handshake(tid));
        call Sweep(tid);
        call Handshake(tid);
    }
}

```

phase 6  
internals

```

Mark(linear tid:Tid) {
    call ResetSweepPtr(tid);
    while (true) {
        if (ScanRoots(tid)) { return; }
        call MarkAllGrays(tid);
    }
}

```

```

Sweep(linear tid:Tid) { ...
    for (var i:int := memLo; i < memHi; i++) {
        call SweepOneObject(tid);
    }
}

```

```

MarkAllGrays(linear tid:Tid) {
    while (true) {
        var isEmpty:bool, node:int := GraySetChoose(tid);
        if (isEmpty) { break; }
        for (var f:int := 0; f < numFields; f := f + 1) {
            var child:int := ReadFieldC(tid, node, f);
            if (memAddr(child)) {
                call GraySetInsert(tid, node, child);
            }
        }
        call GraySetRemove(tid, node);
    }
}

```

```

ScanRoots(linear tid:Tid) returns (done:bool) {
    call CollectorRootScanBarrierStart(tid);
    call CollectorRootScanBarrierWait(tid);
    for (var i:int := 0; i < numRoots; i++) {
        var obj:int := ReadRootInRootScanBarrier(tid, i);
        if (memAddr(obj)) {
            call GraySetInsertIfWhite(tid, obj);
        }
    }
    call done := IsGraySetEmpty(tid);
    call CollectorRootScanBarrierEnd(tid);
}
...
assert tid == GcTid;
Color := ...;
done := (forall v:int :: memAddr(v) ==>
        Color[v] != GRAY);

```

```

WriteBarrier(linear tid:Tid, y:idx) {
    var rootVal:int := ReadRoot(tid, y);
    if (memAddr(rootVal)) {
        if (ReadMutatorPhase(tid) == MARK) {
            call GraySetInsertIfWhite(tid, rootVal);
        }
    }
}

```

```

assert mutatorTidWhole(tid)
&& rootAddr(y) && tidOwens(tid, y);
if ( memAddr(root[y])
    && Color[root[y]] == WHITE
    && MarkPhase(mutatorPhase[tid])) {
    Color[val] := GRAY;
}

```

```

WriteFieldRaw(linear tid:Tid, x:idx, f:fld, y:idx) {
    var valx:int := ReadRoot(tid, x);
    var valy:int := ReadRoot(tid, y);
    call WriteFieldGeneral(tid, valx, f, valy);
}

```

```

assert mutatorTidWhole(tid)
&& rootAddr(x) && tidOwens(tid, x)
&& rootAddr(y) && tidOwens(tid, y)
&& fieldIndex(f)
&& memAddr(root[x])
&& memAddrAbs(rootAbs[x]);
memAbs[rootAbs[x]][f] := rootAbs[y];
mem[root[x]][f] := root[y];

```

phase 5