

```

-- =====
-- APPENDIX B - Complete VDM++ Model -- Amalgamated - (in one file for printing)
-- =====

-- =====
-- World in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerger - spring 2010
-- =====

class World
  types

  values

  instance variables
    env : [SorterEnviroment] := nil;
    loader : [ItemLoader] := nil;

    -- Test files that contains test scenarios
    --- of items to be feeded on inductions
    testfile1 : seq1 of char := "scenario1.txt";
    testfile2 : seq1 of char := "scenario2.txt";
    testfile3 : seq1 of char := "scenario3.txt";
    testfile4 : seq1 of char := "scenario4.txt";
    testfile5 : seq1 of char := "scenario5.txt";

    testfiles : seq of seq1 of char := [testfile1,
                                         testfile2,
                                         testfile3,
                                         testfile4,
                                         testfile5];

  operations

-- World constructor
public World: () ==> World
World() ==
(
);

-- Prints configuration and result of tray allocation model simulation
public PrintSimulationResult: () ==> ()
PrintSimulationResult() ==
(
  -- Prints configuration of simulation
  IO`print("-----\n");
  IO`print("Simulation completed for sorter configuration\n");
  IO`print("-----\n");
  IO`print("Specified throughput [items/hour]: " ^
          String`NatToStr(SorterEnviroment`Throughput) ^ "\n");
  IO`print("Sorter speed [mm/sec]: " ^
          String`NatToStr(SorterEnviroment`Speed) ^ "\n");
  IO`print("Item max size [mm]: " ^
          String`NatToStr(Item`ItemMaxSize) ^ "\n");
  IO`print("Item min size [mm]: " ^
          String`NatToStr(Item`ItemMinSize) ^ "\n");
  IO`print("Tray size [mm]: " ^
          String`NatToStr(Tray`TraySize) ^ "\n");
  IO`print("Number of trays : " ^
          String`NatToStr(TrayAllocator`NumOfTrays) ^ "\n");
  IO`print("Number of inductions : " ^
          String`NatToStr(TrayAllocator`NumOfInductions) ^ "\n");
  IO`print("Induction rate : " ^
          String`NatToStr(InductionController`InductionRate) ^ "\n");
  IO`print("Induction separation [trays]: " ^

```

```

TrayAllocatorModel.vdmpp

String`NatToStr(TrayAllocator`InductionSeperation) ^ "\n");
IO`print("-----\n");

-- Prints result of simulation
let r : TrayAllocator`ThroughputResult = env.sc.allocator.GetThroughput()
in
(
  IO`print("Number of trays with items      : " ^
    String`NatToStr(r.traysWithItemOnSorter) ^ "\n");
  IO`print("Two tray items on sorter      : " ^
    String`NatToStr(r.twoTrayItemsOnSorter) ^ "\n");
  IO`print("Number of tray steps      : " ^
    String`NatToStr(r.traySteps) ^ "\n");
  IO`print("Number of inducted items      : " ^
    String`NatToStr(r.inductedItems) ^ "\n");
  IO`print("Calculated throughput[items/hour]: " ^
    String`NatToStr(floor(r.calcThroughput)) ^ "\n");
);

IO`print("-----\n");
if env.sc.allocator.IsSorterFull() = true
then
  IO`print("      ****      Sorter is full      *****\n")
else
  IO`print("      ****      Sorter is not full      *****\n");
IO`print("-----\n");
);

public Run: () ==> ()
Run() ==
(
  -- Performs model testing for each scenarios specified in test file
  for all test in set {1,...,len testfiles}
  do
  (
    env := new SorterEnviroment();
    loader := new ItemLoader(testfiles(test));
    env.AssignItemLoader(loader);

    IO`print("-----\n");
    IO`print("Tray allocation model test #" ^
      String`NatToStr(test)^ ": " ^ testfiles(test) ^ "\n");
    IO`print("-----\n");

    -- Performs simulation based on time steps
    for all t in set {0,...,loader.GetNumTimeSteps()}
    do
      env.TimeStep(t);

    PrintSimulationResult();
  );
);

functions

sync

--thread

traces

end World

```

```
-- =====
```

# TrayAllocatorModel.vdmpp

```
-- SorterEnvironment in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerger - spring 2010
-- =====

class SorterEnviroment
  types

  values
    public Speed      : nat = 2000; -- Sorter speed mm/sec
    public Throughput : nat = 10000; -- Required items/hour

  instance variables
    public sc : SC;
    public inductionGroup : seq of InductionController := [];
    itemId : nat := 0;
    itemLoader : [ItemLoader] := nil;

  operations

  -- SorterEnviroment constructor
  public SorterEnviroment: () ==> SorterEnviroment
  SorterEnviroment() ==
  (
    sc := new SC(self);
  );

  -- Assigning item loader to SorterEnviroment
  public AssignItemLoader: (ItemLoader) ==> ()
  AssignItemLoader(il) ==
  (
    itemLoader := il;
  );

  -- Assigning induction group to SorterEnviroment
  public AssignInductionGroup: seq of InductionController ==> ()
  AssignInductionGroup(ig) ==
  (
    inductionGroup := ig;
  );

  -- Used by traces in TestSernarios
  public FeedItemOnInduction: nat * Item ==> ()
  FeedItemOnInduction(ic, item) ==
    inductionGroup(ic).FeedItem(item);

  -- Called by world each time sorter ring moves one tray step
  public TimeStep: nat ==> ()
  TimeStep(t) ==
  (
    for all i in set {1,...,TrayAllocator`NumOfInductions}
    do
    (
      -- Check for item to feed induction at time step
      let size = itemLoader.GetItemAtTimeStep(t, i)
      in
      if (size > 0)
      then
      (
        itemId := itemId + 1;
        inductionGroup(i).FeedItem(new Item(size, itemId));
      );
    );
  );
```

# TrayAllocatorModel.vdmpp

```

-- Enviroment simulate sorter moved one tray step
sc.TrayStep(t mod TrayAllocator`NumOfTrays + 1, <Empty>);

-- Performs tray step for each induction
for all i in set {1,...,TrayAllocator`NumOfInductions}
do
    inductionGroup(i).TrayStep();

);

functions

sync

--thread

traces

end SorterEnviroment

-- =====
-- ItemLoader in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerger - spring 2010
-- =====

class ItemLoader
    types

        inline = nat * nat * nat;
        InputTP = int * seq of inline;

    values

    instance variables
        -- Not working in Overture version 0.1.9
        io : IO := new IO();

        -- Test with mix of 1 and 2 tray items
        inlines : seq of inline := [mk_(0,1,100),
        -- mk_(timeStep, icid, itemSize)
                                mk_(0,2,800),
                                mk_(0,3,200),
                                mk_(2,1,200),
                                mk_(2,2,400),
                                mk_(2,3,700),
                                mk_(4,1,800),
                                mk_(4,2,300),
                                mk_(4,3,400),
                                mk_(6,1,600),
                                mk_(6,2,400),
                                mk_(6,3,300),
                                mk_(8,1,900),
                                mk_(8,2,300),
                                mk_(8,3,200),
                                mk_(10,1,500),
                                mk_(10,2,300),
                                mk_(10,3,200)
                                ];

        numTimeSteps : nat := 21;

    operations

```

# TrayAllocatorModel.vdmpp

```

-- Loads test scenario from file
public ItemLoader : seq1 of char ==> ItemLoader
ItemLoader(fname) ==
(
  -- Not working in Overture version 0.1.9
  def mk_(-,mk_(timeval,input)) = io.freadval[InputTP](fname)
  in
    (
      numTimeSteps := timeval;
      inlines := input
    );
);

-- Returns number of time steps to simulate
public GetNumTimeSteps : () ==> nat
GetNumTimeSteps() ==
  return numTimeSteps;

-- Returns size of item if found in test scenario
-- Returns zero if no item is found for time step and induction id
public GetItemAtTimeStep : nat * nat1 ==> nat
GetItemAtTimeStep(timeStep, icid) ==
(
  -- {size | mk_(time, id, size) in set elems inlines
  --- & time = timestep and id = icid};
  let elm = {e | e in set elems inlines & e.#1 = timeStep and e.#2 = icid}
  in
    if elm = {}
    then return 0
    else
      let {mk_(-,-,size)} = elm
      in
        return size;
);

functions

sync

--thread

traces

end ItemLoader

=====
-- SorterController in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerger - spring 2010
=====

class SC
  types

  values

  instance variables
    public allocator : TrayAllocator;

  operations

  -- SystemController constructor
  public SC: SorterEnviroment ==> SC

```

# TrayAllocatorModel.vdmpp

```

SC(e) ==
(
    allocator := new TrayAllocator(e);
);

-- Notified each time sorter-ring has moved one tray step
public TrayStep: Tray`UID * Tray`State ==> ()
TrayStep(uid, state) ==
(
    IO`print("Card reader tray id " ^ String`NatToStr(uid) ^ "\n");
    allocator.CardReader(uid, state);
);

functions

sync

--thread

traces

end SC

-- =====
-- Item in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

class Item
types
    public ItemTraySize = nat1
    inv it == it <= ItemMaxTrays; -- Limitation on how many trays an item occupies

values
    public ItemMaxSize : nat = 1500; -- Item maximum size in mm
    public ItemMinSize : nat = 100; -- Item minimum size in mm
    public ItemMaxTrays: nat = 2; -- Maimum number of trays an item occupies

instance variables
    id : nat; -- Item ID for induction
    size : nat1; -- Item size in mm
    inv size >= ItemMinSize and size <= ItemMaxSize;

    sizeOfTrays : ItemTraySize; -- Number of trays item occupies
    trays : set of Tray := {};

    -- If the item is on the sorter ring the size of trays the item occupies
    -- must be equal to number of tray associations
    -- inv let t = card trays in t > 0 => sizeOfTrays = t;

operations

-- InductionController constructor
public Item: nat1 * nat ==> Item
Item(s, i) ==
(
    size := s;
    sizeOfTrays := size div Tray`TraySize + 1;
    id := i;
);

-- Return item id
public GetId: () ==> nat

```

## TrayAllocatorModel.vdmpp

```

GetId() ==
    return id;

-- Returns the number of trays the item occupies
public GetSizeOfTrays: () ==> ItemTraySize
GetSizeOfTrays() ==
    return sizeOfTrays;

-- Return item size
public GetSize: () ==> nat
GetSize() ==
    return size;

-- Creates association between item and tray
public AssignItemToTray: Tray ==> ()
AssignItemToTray(tray) ==
    trays := trays union {tray};

-- Remove item from sorter ring - Implicit operation - not used yet !!!
public RemoveItemFromTray ()
ext wr trays : set of Tray
post trays = {};

functions

sync

--thread

traces

end Item

-- =====
-- Tray in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerger - spring 2010
-- =====

class Tray
    types
        public State = <Empty> | <Full>;
        public UID = nat
        inv u == u <= TrayAllocator`NumOfTrays;          -- Limitation on UID

    values
        public TraySize      : nat1 = 600              -- Size of any tray mm

    instance variables

        -- It is allowed for a tray to be <Full> with no item associated
        -- in this case an unknown item is detected by the card reader
        state : State := <Empty>;
        item : [Item] := nil;

        -- If an item is associated with a tray the state must be <Full>
        inv item <> nil => state = <Full>;

        id : UID;                                -- Tray UID

    operations

    -- Tray constructor
    public Tray: UID ==> Tray
    Tray(i) ==

```

```

(
    id := i;
);

-- Return tray id
public GetId: () ==> nat
GetId() ==
    return id;

-- Returns true if tray is empty
public IsTrayEmpty: () ==> bool
IsTrayEmpty () ==
    return state = <Empty>;

-- Returns true if tray is full
public IsTrayFull: () ==> bool
IsTrayFull () ==
    return state = <Full>;

-- Returns item on tray
public GetItem: () ==> [Item]
GetItem () ==
    return item;

-- Set state of tray
public SetState: State ==> ()
SetState (s) ==
(
    if s = <Empty>
    then -- Remove item if tray is empty
        item := nil;
    state := s;
);

-- Returns state of tray ==> <empty> or <full>
public GetState: () ==> State
GetState () ==
    return state;

-- Puts an item on the tray and creates association between tray and item
public ItemOnTray: Item ==> ()
ItemOnTray (i) ==
(
    atomic -- Only needed if item is assigned before state
    (
        item := i;
        state := <Full>;
    );
    item.AssignItemToTray(self);

    IO`print("-> Item id " ^ String`NatToStr(item.GetId()) ^
        "size " ^ String`NatToStr(item.GetSize()) ^
        "on tray id " ^ String`NatToStr(id) ^ "\n");
)
pre state = <Empty> and item = nil;

functions

sync

--thread

traces

```



**end** Tray

```
-- =====
-- InductionController in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerger - spring 2010
-- =====

class InductionController
  types

  values
    public InductionRate : nat = 2; -- trays between each item

  instance variables
    priority : nat := 0;           -- priority of induction,
                                -- incremented each time wait to induct item
    id : nat1;                    -- Induction ID
    allocator : TrayAllocator;     -- TrayAllocator
    items : seq of Item := [];    -- set of items ready to be inducted
    stepCount : nat := 0;         -- Counts the number of steps between inducing items

    -- If induction is waiting there must be items in sequence
    inv priority > 0 => len items > 0;

  operations

  -- InductionController constructor
  public InductionController: TrayAllocator * nat ==> InductionController
  InductionController(a, n) ==
  (
    allocator := a;
    id := n;
  );

  -- Returns induction controller UID
  public GetId: () ==> nat
  GetId() ==
    return id;

  -- Returns priority of induction controller
  public GetPriority: () ==> nat
  GetPriority() ==
    return priority;

  -- Returns true if induction is waiting with an item
  public IsItemWaiting: () ==> bool
  IsItemWaiting() ==
    return priority > 0;

  -- Get size of waiting item in number of trays
  public GetSizeOfWaitingItem: () ==> nat
  GetSizeOfWaitingItem() ==
    if not IsItemWaiting()
    then
      return 0 -- No waiting items
    else
      let item = hd items
      in item.GetSizeOfTrays();

  -- Enviroment feeds a new item on induction
  public FeedItem: Item ==> ()
  FeedItem(i) ==
    items := items ^ [i];
```

# TrayAllocatorModel.vdmpp

```

-- Simulate sorter-ring moved one tray step
public TrayStep: () ==> ()
TrayStep() ==
(
  -- Induct next item based on InductionRate
  stepCount := stepCount + 1;
  if IsItemWaiting() or (stepCount >= InductionRate)
  then
    (
      InductNextItem();
      stepCount := 0;
    )
);

-- It any items on induction then induct next item
-- If next item could be inducted then removed it from the head of item sequence
-- If item could not be inducted then increment priority
private InductNextItem: () ==> ()
InductNextItem() ==
  let n = len items
  in
    if n > 0
    then
      let item = hd items
      in
        if allocator.InductItem(self, item)
        then
          (
            atomic -- Due to invariant
            (
              items := tl items;
              priority := 0;
            );
          )
        else
          priority := priority + 1;
          -- Increment priority wait counter

functions

sync

--thread

traces

end InductionController

-- =====
-- TrayAllocator in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerger - spring 2010
-- =====

class TrayAllocator

  types
    public ThroughputResult::
      traysWithItemOnSorter : nat
      twoTrayItemsOnSorter : nat
      traySteps : nat
      inductedItems : nat
      calcThroughput : real;

  values

```

# TrayAllocatorModel.vdmpp

```

public InductionSeperation: nat = 2;
public NumOfInductions : nat = 3;
public NumOfTrays : nat = 20;
public SecInHour : nat = 3600    -- Number of seconds in an hour

instance variables
  -- Ensure sufficient number of trays
  -- on sorter ring based on inductions and separation
  inv NumOfTrays > InductionSeperation * NumOfInductions;

  countTraySteps : nat := 0;      -- Used for calculation of throughput
  countItemsInducted : nat := 0;  -- Counts the number of items inducted

  -- Induction group and invariants
  public inductionGroup : seq of InductionController := [];
  inv len inductionGroup = NumOfInductions;
  -- Induction id and inds of inductionGroup sequence must be the same
  inv forall id in set inds inductionGroup & inductionGroup(id).GetId() = id;

  -- Sorter ring and invariants
  public sorterRing : inmap Tray`UID to Tray;
  inv card dom sorterRing = NumOfTrays;
  -- Tray id and dom of sorterRing map must be the same
  inv forall id in set dom sorterRing & sorterRing(id).GetId() = id;

  -- Tray at card reader and invariants
  public trayAtCardReader : Tray`UID := 0;
  -- trayAtCardReader must be a valid tray in sorterRing
  inv trayAtCardReader > 0 => trayAtCardReader in set dom sorterRing;

  -- Allocation "strategy pattern" for one and two tray items
  oneTrayStrategy : AllocatorOneTray;
  twoTrayStrategy : AllocatorTwoTray;

operations

-- TrayAllocator constructor
public TrayAllocator: SorterEnviroment==> TrayAllocator
TrayAllocator(e) ==
(
  sorterRing := {num |-> new Tray(num) | num in set {1,...,NumOfTrays}};
  inductionGroup := [new InductionController(self, num) |
                    num in set {1,...,NumOfInductions}];
  e.AssignInductionGroup(inductionGroup);

  -- Creating strategies for allocation of one and two tray items
  oneTrayStrategy := new AllocatorOneTray(self);
  twoTrayStrategy := new AllocatorTwoTray(self);
);

-- Simulate sorter-ring moved one tray step
public CardReader: Tray`UID * Tray`State ==> ()
CardReader(uid, state) ==
(
  -- Update current tray at card reader
  trayAtCardReader := uid;

  -- Simulate change of Tray state !!!
  -- sorterRing(trayAtCardReader).SetState(state);

  -- Count the number of tray steps
  countTraySteps := countTraySteps + 1;
)
pre uid in set dom sorterRing;

```

```

-- Inducting item on sorter if empty trays and no higher induction priority
public InductItem: InductionController * Item ==> bool
InductItem(ic, item) ==
(
    dcl strategy : AllocatorStrategy;

    -- Determine the strategy to compute the allocation of trays
    let numTrays = item.GetSizeOfTrays()
    in
        cases numTrays:
            1 -> strategy := oneTrayStrategy,
            2 -> strategy := twoTrayStrategy
        end;

    -- Central part of the Tray allocation algorithm
    -- Look for inductions with higher priority
    -- before allocation of empty trays
    if strategy.InductionsWithHigherPriority(ic)
    then
        return false
    else
        let trays = strategy.AllocateTray(ic.GetId())
        in
            if trays = {}
            then
                return false
            else
                (
                    countItemsInducted := countItemsInducted + 1;
                    IO`print("*Induction id " ^
                        String`NatToStr(ic.GetId()) ^ "\n");
                    -- Assign item to trays
                    PutItemOnTrays(item, trays);
                    return true;
                )
    )
pre ic in set elems inductionGroup and item.GetSizeOfTrays() <= 2;
-- To be changed if Tray`ItemMaxTrays is increased

-- Assign item on empty trays
private PutItemOnTrays: Item * set of Tray ==> ()
PutItemOnTrays(item, trays) ==
    for all t in set trays
    do
        t.ItemOnTray(item)
pre trays <> {} and forall t in set trays & t.IsTrayEmpty();

-- Returns true if sorter is full
public IsSorterFull: () ==> bool
IsSorterFull() ==
    return forall id in set dom sorterRing & sorterRing(id).GetState() = <Full>;

-- Returns calculated throughput of soter capacity
-- for current state of sorter ring
public GetThroughput: () ==> ThroughputResult
GetThroughput() ==
    CalculateThroughput(countTraySteps, rng sorterRing, countItemsInducted);

functions

-- Calculates the current throughput based on items on sorter ring
/*
Calculation as sum of simulation time steps

```

## TrayAllocatorModel.vdmpp

```

    = number of steps * Tray`TraySize/SorterEnviroment`Speed
throughput calculated as items inducted in simulation time converted to items/hour
calculate the number of items inducted = number of tray with status
equal <full> minus sum of two tray items divided by 2
*/
private CalculateThroughput: nat * set of Tray * nat-> ThroughputResult
CalculateThroughput(steps, trays, items) ==
    let runTime :real = steps * (Tray`TraySize/SorterEnviroment`Speed),
        traysWithItems = {twi | twi in set trays & twi.IsTrayFull()},
        traysWith2Items = {tw2i | tw2i in set traysWithItems & tw2i.GetItem() <> nil
                                and tw2i.GetItem().GetSizeOfTrays() = 2},
        itemsOnSorter = card traysWithItems,
        twoTrayItemsOnSorter = card traysWith2Items,
        throughput = itemsOnSorter * SecInHour/runTime
    in
        mk_ThroughputResult(itemsOnSorter, twoTrayItemsOnSorter,
                            steps, items, throughput)
pre trays <> {};

sync

--thread

traces

end TrayAllocator

-- =====
-- Allocator in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerger - spring 2010
-- (strategy pattern)
-- =====

class AllocatorStrategy

    instance variables
        protected trayAllocator : [TrayAllocator] := nil;    -- TrayAllocator

    operations

        public AllocateTray: nat ==> set of Tray
        AllocateTray (-) ==
            is subclass responsibility;

        public InductionsWithHigherPriority: InductionController ==> bool
        InductionsWithHigherPriority(ic) ==
            is subclass responsibility;

    functions

        -- Calculate current tray UID at position in front
        -- of induction based on position of card reader
        protected InductionOffset: Tray`UID * nat -> Tray`UID
        InductionOffset(trayAtCardReader, icid) ==
            ((trayAtCardReader + icid*TrayAllocator`InductionSeperation)
             mod TrayAllocator`NumOfTrays) + 1;

end AllocatorStrategy

-- =====
-- AllocatorOneTray in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerger - spring 2010
-- (strategy pattern)

```

```

-- =====

class AllocatorOneTray is subclass of AllocatorStrategy

  operations

    -- AllocatorOneTray constructor
    public AllocatorOneTray: TrayAllocator==> AllocatorOneTray
    AllocatorOneTray(ta) ==
    (
      trayAllocator := ta;
    );

    -- Allocates tray if empty at induction offset
    public AllocateTray: nat ==> set of Tray
    AllocateTray (icid) ==
      def posTray = InductionOffset(trayAllocator.trayAtCardReader, icid)
      in
        if trayAllocator.sorterRing(posTray).IsTrayEmpty()
        then return {trayAllocator.sorterRing(posTray)}
        else return {}
    pre icid in set inds trayAllocator.inductionGroup;

    -- Returns true if higher priority inductions in induction group
    public InductionsWithHigherPriority: InductionController ==> bool
    InductionsWithHigherPriority(ic) ==
      return exists i in set
        elems trayAllocator.inductionGroup(1,...,len trayAllocator.inductionGroup)
          & i.GetId() <> ic.GetId()
          and i.GetPriority() > ic.GetPriority()
      -- Looking at induction in front of this ic
      -- causes starvation of the first induction in group
      -- return exists i in set
      --   elems inductionGroup(ic.GetId()+1,...,len inductionGroup)
      --   & i.GetPriority() > ic.GetPriority()
    pre ic in set elems trayAllocator.inductionGroup;

end AllocatorOneTray

-- =====
-- AllocatorTwoTray in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- (strategy pattern)
-- =====

class AllocatorTwoTray is subclass of AllocatorStrategy

  operations

    -- AllocatorTwoTray constructor
    public AllocatorTwoTray: TrayAllocator==> AllocatorTwoTray
    AllocatorTwoTray(ta) ==
    (
      trayAllocator := ta;
    );

    -- Allocates trays if empty at induction offset and offset + 1
    public AllocateTray: nat ==> set of Tray
    AllocateTray (icid) ==
      let posTray = InductionOffset(trayAllocator.trayAtCardReader, icid),
          posTrayNext = if (posTray - 1) = 0
                        then TrayAllocator.NumOfTrays
                        else posTray - 1

```

```

    in
        if trayAllocator.sorterRing(posTray).IsTrayEmpty() and -- Tray at ic
            trayAllocator.sorterRing(posTrayNext).IsTrayEmpty() -- Tray at ic-1
        then return {trayAllocator.sorterRing(posTray),
                    trayAllocator.sorterRing(posTrayNext)}
        else return {}
    pre icid in set inds trayAllocator.inductionGroup;

-- Returns true if higher priority inductions in induction group
public InductionsWithHigherPriority: InductionController ==> bool
InductionsWithHigherPriority(ic) ==
    return exists i in set
        elems trayAllocator.inductionGroup(1,...,len trayAllocator.inductionGroup)
            & i.GetId() <> ic.GetId()
            and i.GetPriority() > ic.GetPriority()
-- Waiting with items of same size (two tray items)
--         and i.GetSizeOfWaitingItem() = ic.GetSizeOfWaitingItem()
-- Looking at induction in front of this ic
-- causes starvation of the first induction in group
-- return exists i in set
--         elems inductionGroup(ic.GetId()+1,...,len inductionGroup)
--         & i.GetPriority() > ic.GetPriority()
    pre ic in set elems trayAllocator.inductionGroup;

end AllocatorTwoTray

-- =====
-- String helper class for converting numbers
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

class String
    types

    values

    instance variables
        static numeric : seq1 of char := "0123456789";

    operations

    static public NatToStr: nat ==> seq1 of char
    NatToStr (val) ==
    (
        decl string : seq1 of char := " ";
        decl x1 : nat := val;
        decl x2 : nat;

        if val = 0 then string := "0";

        while x1 > 0 do
            (
                x2 := (x1 mod 10) + 1;
                string := [numeric(x2)] ^ string;
                x1 := x1 div 10;
            );

        return string;
    );

    functions

end String

```

```
-- =====
-- TestTraces in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

class TestTraces
  types

  values

  instance variables
    env : SorterEnviroment := new SorterEnviroment();

    testfile1 : seq1 of char := "scenario1.txt";
    loader1 : ItemLoader := new ItemLoader(testfile1);

    testfile2 : seq1 of char := "scenario2.txt";
    loader2 : ItemLoader := new ItemLoader(testfile2);
    tests : set of ItemLoader := {loader1, loader2};

  operations

  functions

  sync

  --thread

  traces

  -- To run TestSenarios - IO`print has to be commented out
  TestSenario1: (
    let loader in set tests
    in
    (
      env.AssignItemLoader(loader);
      let step in set {1,...,loader.GetNumTimeSteps()}
      in
        env.TimeStep(step)
      )
    );

end TestTraces
```