Refinement of the Parallel CD_x

—

Technical Report

F. Zeyda, A. Cavalcanti, A. Wellings, J. Woodcock and K. Wei

December 3, 2012

Contents

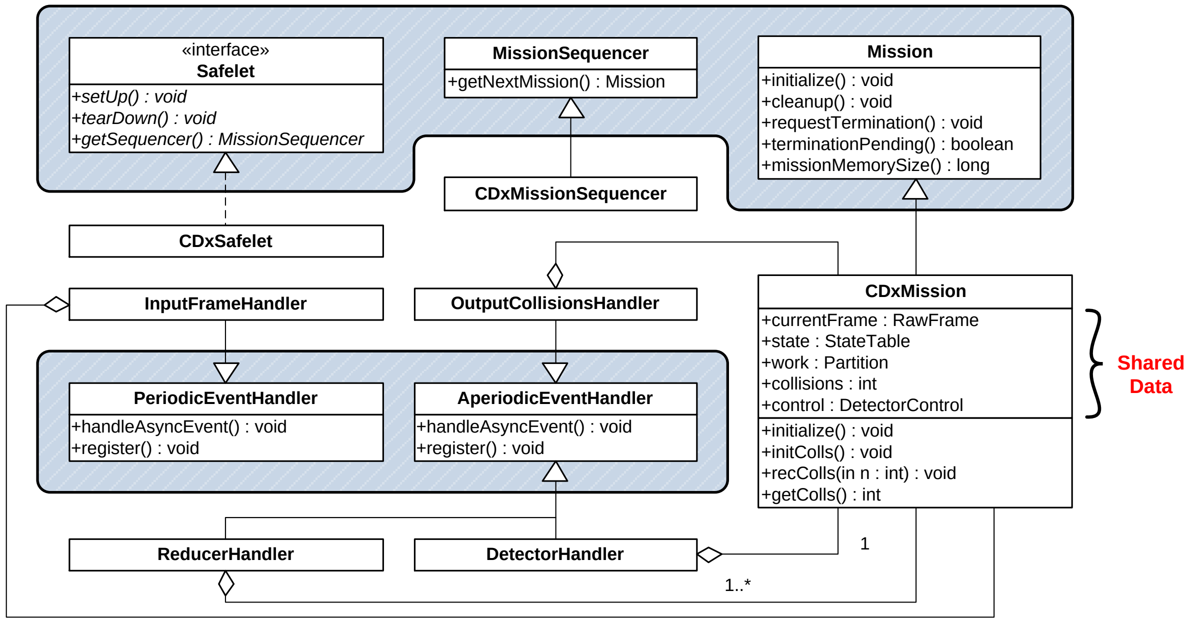

Figure 1: UML diagram for the concurrent CD_x program

1 Introduction

The purpose of the CD_x is to detect potential collisions of aircraft located by a radar device. We take the program discussed in [8] as a basis for the definition of our requirements. It uses a cyclic executive, and embeds the assumption that the radar collects (and buffers) a frame of aircraft positions that becomes available for input periodically. In each iteration, the CD_x: (1) reads a frame; (2) carries out a voxel-hashing step that maps aircraft to voxels; (3) checks for collisions in each voxel; and (4) records and reports the number of detected collisions. Unlike [8], we allow aircraft to enter or leave the radar frame.

Since the majority of the computation burden is in the checking for collisions in step (3), we propose a version of the CD_x where this task is parallelised. As a result, we obtain an SCJ program that illustrates the features of SCJ Level 1. Our aim with the concurrent CD_x is, most of all, to provide a genuine and more representative Level 1 application. Due to the novelty of the SCJ paradigm and technology, such applications are still difficult to come by in the public domain. On the other hand, even though we are not specifying a particular radar system, concurrent collision detection is a reasonable target to improve the performance of such an application. The program code is available via `http://www.cs.york.ac.uk/circus/hijac/`.

A voxel is a volumetric element; all voxels together subdivide the entire space. The voxels in the CD_x superimpose a coarse 2-dimensional grid on the x-y plane with the height of a voxel extending along the entire z-axis. Thus, the altitude of aircraft is abstracted away. This reduces the number of necessary collision tests: after mapping aircraft to the voxels that are intersected by their interpolated trajectories, it is sufficient to test for possible collisions within each voxel. Details of the algorithm can be found in [8].

The concurrent CD_x consists of a single mission that instantiates seven handlers. Figure 1 presents a UML class diagram that illustrates the design. The classes shaded are part of the SCJ API. The classes `CDxSafelet`, `CDxMissionSequencer` and `CDxMission` implement the safelet, the mission sequencer, and the mission. The behaviour of the `setUp()` and `tearDown()` methods of `CDxSafelet` is void, and `getSequencer()` simply returns an instance of `CDxMissionSequencer`. Likewise, `getNextMission()` returns an instance of `CDxMission` when called for the first time. Since the mission does not terminate, `getNextMission()` is not called again.

In the mission execution, first the `initialize()` method of `CDxMission` is called. It creates the handler objects and shared data in mission memory. The handler classes are `InputFrameHandler`, "OutputCollisions-

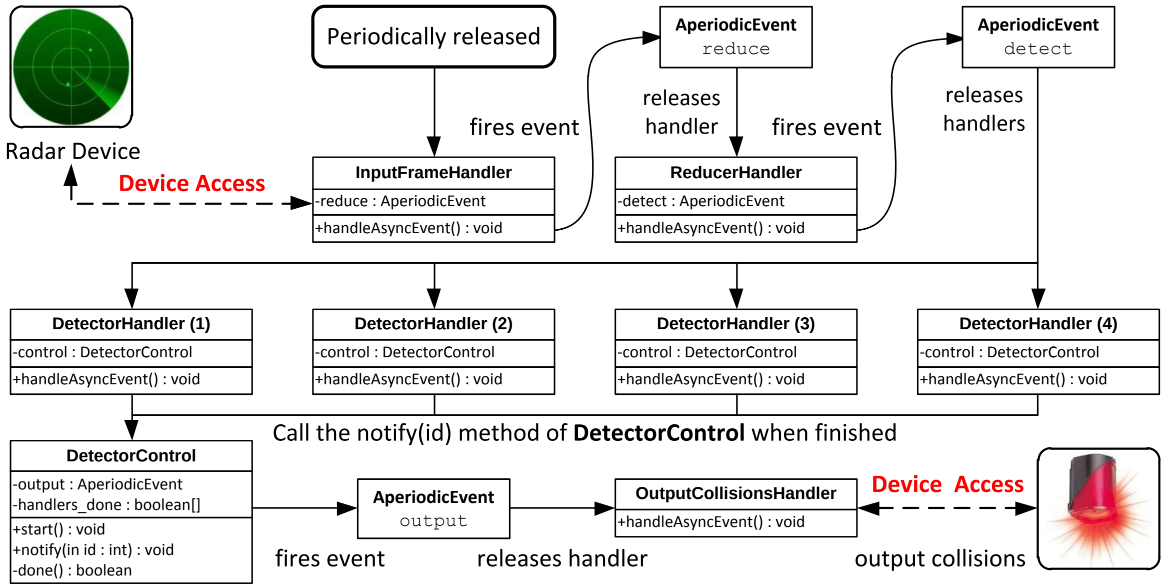

Figure 2: Parallel CD_x control flow

Handler", `ReducerHandler`, and `DetectorHandler`. We choose to create four instances of `DetectorHandler`, possibly corresponding to a scenario in which we have four processors. The refinement in the remainder of the report can, however, proceed without significant changes in the presence of a different number of instances. A more general design for the CD_x could allow the configuration of the number of instances; our program, however, is enough to illustrate the main aspects of our technique.

The shared data is held by public fields of `CDxMission`. The `currentFrame` and `state` fields record the current and previous frame of aircraft positions; recording previous positions is important for calculating their predicted motions. As we divide and distribute the computational work, `work` holds the partitions of voxels to be checked by each of the detection handlers, and `collisions` is used to accumulate the result of the detection. Another shared object `control` plays a crucial part in orchestrating the execution of handlers.

Figure 2 summarises the control mechanism of the SCJ application. The three software events, `reduce`, `detect` and `output`, are used to control execution of the handlers. The program design ensures that the handlers effectively execute sequentially in each cycle, apart from the four instances of `DetectorHandler`, which carry out their work concurrently.

The `InputFrameHandler` is the only periodic handler. It is released at the beginning of each cycle to interact with the hardware to read the frame into `currentFrame` and update `state` accordingly. Afterwards, it releases the `ReducerHandler`, via the `reduce` software event, to carry out the voxel-based reduction step. This handler also partitions and distributes the work among the detector handlers by populating `work`. Once this is done, it concurrently releases all `DetectorHandler` instances by firing the `detect` event. These handlers carry out the actual detection work and store their result in `collisions`. The mechanism for releasing `OutputCollisionsHandler`, which outputs the number of collisions to an external device, uses the shared object `control`. Its class type `DetectorControl` provides a method `notify(int id)`, which is called by the detector handlers at the end of each release. It fires the event `output` when all detection work is done. This illustrates that sharing may occur not only to exchange data between handlers, but also in the design of execution control, and our refinement strategy will have to cater for this.

Our program highlights various features of the SCJ mission framework: the subdivision of a mission into handlers, the control of handlers via software events, and the sharing of data for both data communication and control purposes. The verification of this program not only has to address functional correctness, but also must show that the flow of activities in Figure 2 can be executed within the duration of a cycle.

2 Preliminaries

In this section, we present preliminary definitions of types, operators and functions that are used later on in the models.

2.1 Extensions

We have already introduced a nondeterministic **wait** statement with the following semantics.

$$\mathbf{wait}\, S \;=_{\mathrm{df}}\; \bigsqcap t : S \bullet \mathbf{wait}\, t$$

It turns out to be useful to have an alternative construct that allows us to refer to the actual time waited via a bound identifier.

$$\mathbf{wait}\, t : S \bullet A(t) \;=_{\mathrm{df}}\; \bigsqcap t : S \bullet \mathbf{wait}\, t\, ;\; A(t)$$

With this, A can obtain information regarding the delay resulting from the **wait**; this is used in a few places in specifying the models, in particular for the E anchor.

In addition to the above, we introduce several further extensions to the SCJ*Circus* language.

1. A generic *Array* class to model one-dimensional Java arrays of a given type.
2. Methods to get and set the elements of an array as well as obtain its size:
   - $getA(index : int) : T$
   - $setA(index : int, value : T)$
   - $length() : int$
3. Support for simple `for` loops. This is via the action construct $\mathbf{for}\; i = n_0 \;\mathbf{to}\; n_1 \bullet A(i)$.
4. Support for the creation of software events. For this we have the **newEvent** construct.
5. Software events are fired using the **fire** construct.

2.2 Reals

We postulate the existence of a type $\mathbb{R}$ for real numbers.

$$\begin{array}{|l} \mathbb{R} : \mathbb{P}\,\mathbb{A} \\ \hline \mathbb{Z} \subset \mathbb{R} \end{array}$$

By introducing $\mathbb{R}$ as a subset of $\mathbb{A}$ (arithmos), we can immediately reuse all arithmetic and relational operators on numbers. Formally, we have to elaborate the semantics of those operators for elements of $\mathbb{R}$. Here, however, we content ourselves that this can be done in principle, rather than providing an axiomatisation of the reals. Such an axiomatisation has been developed in Z, for instance, in [1] and is illustrated by the ProofPower-Z theorem prover [7]. Real numbers are require in the sequel to define the vector schema type used to characterise positions and motions of aircrafts in 3-dimensional space.

2.3 Vectors

Vectors are used to represent the positions and motions of an aircraft.

$$\begin{array}{|l} \textit{Vector} \\ \hline x : \mathbb{R} \\ y : \mathbb{R} \\ z : \mathbb{R} \\ \hline \end{array}$$

We characterise vectors by a schema binding (record) with three real components, x, y and z.

Construction The following function constructs a vector from scratch.

$$\begin{array}{|l} MkVector : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow Vector \\ \hline \forall x, y, z : \mathbb{R} \bullet MkVector(x, y, z) = \langle\!| x == x, y == y, z == z |\!\rangle \end{array}$$

Zero and unit vector These are defined by explicitly giving the underlying coordinates.

$$\begin{aligned} ZeroV &== MkVector(0, 0, 0) \\ UnitV &== MkVector(1, 1, 1) \end{aligned}$$

Addition and subtraction Addition and subtraction are defined component-wise.

$\textbf{function}\ 30\ \textbf{leftassoc}\ (\_ +_V \_)$

$$\begin{array}{|l} \_ +_V \_ : Vector \times Vector \rightarrow Vector \\ \hline \forall v_1, v_2 : Vector \bullet v_1 +_V v_2 = \langle\!| x == v_1.x + v_2.x, y == v_1.y + v_2.y, z == v_1.z + v_2.z |\!\rangle \end{array}$$

$\textbf{function}\ 30\ \textbf{leftassoc}\ (\_ -_V \_)$

$$\begin{array}{|l} \_ -_V \_ : Vector \times Vector \rightarrow Vector \\ \hline \forall v_1, v_2 : Vector \bullet v_1 -_V v_2 = \langle\!| x == v_1.x - v_2.x, y == v_1.y - v_2.y, z == v_1.z - v_2.z |\!\rangle \end{array}$$

Scalar product The scalar product multiplies a vector with a real number.

$\textbf{function}\ 40\ \textbf{leftassoc}\ (\_ *_V \_)$

$$\begin{array}{|l} \_ *_V \_ : \mathbb{R} \times Vector \rightarrow Vector \\ \hline \forall r : \mathbb{R};\ v : Vector \bullet r *_V v = \langle\!| x == r * v.x, y == r * v.y, z == r * v.z |\!\rangle \end{array}$$

Dot product The dot (or inner) product multiplies two vectors.

$\textbf{function}\ 50\ \textbf{leftassoc}\ (\_ \cdot_V \_)$

$$\begin{array}{|l} \_ \cdot_V \_ : Vector \times Vector \rightarrow \mathbb{R} \\ \hline \forall v_1, v_2 : Vector \bullet v_1 \cdot_V v_2 = (v_1.x * v_2.x) + (v_1.y * v_2.y) + (v_1.z * v_2.z) \end{array}$$

We subsequently use the dot product to introduce the length of a vector.

Square of a vector The square multiplies a vector with itself.

$\textbf{function}\ (\_^2)$

$$\begin{array}{|l} \_^2 : Vector \rightarrow \mathbb{R} \\ \hline \forall v : Vector \bullet v^2 = v \cdot_V v \end{array}$$

Length of a vector We use the common definition $\mid v \mid = \sqrt{v^2}$.

$\textbf{function}\ (\mid \_ \mid)$

$$\begin{array}{|l} \mid \_ \mid : Vector \rightarrow \mathbb{R} \\ \hline \forall v : Vector \bullet \mid v \mid * \mid v \mid = v^2 \end{array}$$

2.4 Trajectories

Trajectories are modelled by a pair consisting of a position and a motion vector.

$$\mathit{Trajectory} == \mathit{Vector} \times \mathit{Vector}$$

The points on a trajectory t are given by the formula $t.1 +_V x *_V t.2$ where x ranges over the interval $[0; 1]$.

Distance of Trajectories The notion of distance between trajectories is introduced below.

$$\mathit{distance} : \mathit{Trajectory} \times \mathit{Trajectory} \to \mathbb{R}$$

$$\begin{array}{l} \forall t_1, t_2 : \mathit{Trajectory} \bullet \mathit{distance}(t_1, t_2) = \\ \quad \left(\begin{array}{l} \mu\, d : \mathbb{R} \mid \\ \quad \exists x : \mathbb{R} \mid 0 \leq x \leq 1 \bullet d = \lvert (t_2.1 +_V x *_V t_2.2) -_V (t_1.1 +_V x *_V t_1.2) \rvert \wedge \\ \quad \forall x : \mathbb{R} \mid 0 \leq x \leq 1 \bullet d \leq \lvert (t_2.1 +_V x *_V t_2.2) -_V (t_1.1 +_V x *_V t_1.2) \rvert \end{array} \right) \end{array}$$

We note that this is not the minimal distance between 'any' two points on each trajectory. The CD_x makes the simplifying assumption that aircrafts move at constant speed, so the result is an approximation of the actual minimal distance. In terms of the computation, we calculate the smallest distance between two points that simultaneously traverse each trajectory.

Collisions The threshold distance between two trajectories to flag a collision is given by the constant below. We leave its precise value implicit.

$$\mathit{THRESHOLD} : \mathbb{R}$$

$$0 \leq \mathit{THRESHOLD}$$

The relation below determines whether two trajectories collide.

relation $(\mathit{collide}\,\_)$

$$\mathit{collide}\,\_ : \mathbb{P}\,(\mathit{Trajectory} \times \mathit{Trajectory})$$

$$\forall t_1, t_2 : \mathit{Trajectory} \bullet \mathit{collide}(t_1, t_2) \Leftrightarrow \mathit{distance}(t_1, t_2) \leq \mathit{THRESHOLD}$$

2.5 Miscellaneous

The following operation calculates the sum of all elements in a sequence.

$$\Sigma : (\mathrm{seq}\,\mathbb{R}) \to \mathbb{R}$$

$$\forall s : \mathrm{seq}\,\mathbb{R} \bullet \Sigma\, s = \mathbf{if}\ s = \langle\rangle\ \mathbf{then}\ 0\ \mathbf{else}\ \mathit{head}(s) + \Sigma\,(\mathit{tail}(s))$$

3 Anchor A

In this section, we present the abstract model of the parallel CD_x.

3.1 Aircrafts

The type *Aircraft* represents aircrafts as they may enter the radar.

$$\mathit{Aircraft} == \mathrm{seq}_1\, \mathit{byte}$$

We identify aircrafts by their call sign which consists of a non-empty sequence of bytes.

3.2 Frames

A frame records the positions all of aircrafts in a radar frame. It is modelled by virtue of a (finite) partial function that maps aircrafts to vectors representing their positions in airspace. The number of aircrafts in a radar frame is restricted by a constant $MAX\_AIRCRAFTS$.

$$MAX\_AIRCRAFT : \mathbb{N}_1$$

$$Frame == \{f : Aircraft \twoheadrightarrow Vector \mid \# f \leq MAX\_AIRCRAFT\}$$

The domain of a function implicitly determines the aircrafts that are currently in view of the radar.

3.3 Collision Sets

We introduce a utility function that calculates the collision set for a frame of aircraft positions and motions. This is the set of all colliding aircrafts.

$$CalcCollisionSet : (Frame \times Frame) \rightarrow \mathbb{F}\,(Aircraft \times Aircraft)$$

$$\begin{aligned} &\forall\, posns, motions : Frame \mid \operatorname{dom} posns = \operatorname{dom} motions \bullet \\ &\quad CalcCollisionSet(posns, motions) = \\ &\qquad \left\{ \begin{array}{l} a_1, a_2 : Aircraft \mid a_1 \in \operatorname{dom} posns \wedge a_2 \in \operatorname{dom} posns \wedge \\ \quad collide((posns\ a_1, motions\ a_1), (posns\ a_2, motions\ a_2)) \end{array} \right\} \end{aligned}$$

By definition, this set is symmetric: $(a_1, a_2) \in colldide(t_1, t_2) \Leftrightarrow (a_2, a_1) \in colldide(t_1, t_2)$.

3.4 Channels

We require two channels for external interactions: one channel $next\_frame$ to input the next radar frame and another channel $output\_collisions$ to output the number of collisions at the end of the cycle.

channel $next\_frame : Frame$

channel $output\_collisions : \mathbb{N}$

We note that the value communicated by $output\_collisions$ is an upper bound for the exact number of collisions modulo symmetry. (Symmetry means (a_1, a_2) and (a_2, a_1) are not viewed as separate collisions.) This is due to efficiency and the voxel algorithm we use. That is, voxel hashing in certain cases may record colliding aircraft pairs in more than one voxel, namely if both aircrafts are close to the boundary between those voxels. The original CD_x manually removes such duplicates but we decided not to do so in our parallel implementation to simplify the program (another handler would be required for the removal step).

3.5 Constants

The following three constants specify the duration of a detection cycle ($FRAME\_PERIOD$) as well as deadlines for the input (INT_DL) and output (OUT_DL) communications.

$$\begin{aligned} &FRAME\_PERIOD : TIME \\ &INP\_DL : TIME \\ &OUT\_DL : TIME \\ \hline &INP\_DL + OUT\_DL \leq FRAME\_PERIOD \end{aligned}$$

We leave the precise values of the constants implicit. The paper [8] specifies them in terms of frames per seconds rather than periods, but this is just a technicality as we can convert between these measurements.

3.6 System

Below is the process for the behavioural requirements of the parallel CD_x.

$$
\begin{array}{l}
\textbf{process } \mathit{ABReqsCDx} \mathrel{\widehat=} \textbf{begin}\\
\quad \textbf{state } \mathit{AStateCDx}\\
\qquad \mathit{posns} : \mathit{Frame}\\
\qquad \mathit{motions} : \mathit{Frame}\\
\qquad \hline
\qquad \operatorname{dom} \mathit{posns} = \operatorname{dom} \mathit{motions}\\
\\
\quad \mathit{Init}\\
\qquad \mathit{AStateCDx}'\\
\qquad \hline
\qquad \mathit{posns}' = \varnothing \land \mathit{motions}' = \varnothing\\
\\
\quad \mathit{RecordFrame}\\
\qquad \Delta \mathit{AStateCDx}\\
\qquad \mathit{frame?} : \mathit{Frame}\\
\qquad \hline
\qquad \mathit{posns}' = \mathit{frame?}\\
\qquad \mathit{motions}' = \{a : \operatorname{dom} \mathit{posns}' \bullet a \mapsto \textbf{if } a \in \operatorname{dom} \mathit{posns} \textbf{ then } (\mathit{posns}'\ a) -_V (\mathit{posns}\ a) \textbf{ else } \mathit{ZeroV}\}\\
\\
\quad \mathit{CalcCollisions}\\
\qquad \Xi \mathit{AStateCDx}\\
\qquad \mathit{colls!} : \mathbb{N}\\
\qquad \hline
\qquad \exists\, \mathit{collset} : \mathbb{F}\,(\mathit{Aircraft} \times \mathit{Aircraft}) \mid \mathit{collset} = \mathit{CalcCollisionSet}(\mathit{posns}, \mathit{motions}) \bullet\\
\qquad\qquad (\#\, \mathit{collset} = 0 \land \mathit{colls!} = 0) \lor (\#\, \mathit{collset} > 0 \land \mathit{colls!} \geq (\#\, \mathit{collset}) \operatorname{div} 2)\\
\\
\quad \mathit{BReq1} \mathrel{\widehat=} \mathit{next\_frame} ? \mathit{frame} \longrightarrow\\
\qquad \left(\begin{array}{l} \mathit{RecordFrame};\\ \textbf{var } \mathit{colls} : \mathbb{N} \bullet \mathit{CalcCollisions} \ ; \ \ \mathit{output\_collisions} ! \mathit{colls} \longrightarrow \mathit{BReq1} \end{array}\right)\\
\quad \bullet\ \mathit{Init} \ ; \ \ \mathit{BReq1}\\
\textbf{end}
\end{array}
$$

The process for the timing requirements of the parallel CD_x is as follows.

$$
\begin{array}{l}
\textbf{process } \mathit{ATReqsCDx} \mathrel{\widehat=} \textbf{begin}\\
\mathit{TReq1} \mathrel{\widehat=} (\mathit{TReqCycle} \blacktriangleright \mathit{FRAME\_PERIOD} \interleave \textbf{wait } \mathit{FRAME\_PERIOD}) \ ; \ \ \mathit{TReq1}\\
\mathit{TReqCycle} \mathrel{\widehat=}\\
\quad \left(\begin{array}{l} \left(\begin{array}{l} \mathit{next\_frame} ? \mathit{frame} \mathbin{@} t \longrightarrow\\ \textbf{wait } 0..(\mathit{FRAME\_PERIOD} - t - \mathit{OUT\_DL}) \end{array}\right) \blacktriangleleft \mathit{INP\_DL};\\ (\mathit{output\_collisions} ? c \longrightarrow \textbf{skip}) \blacktriangleleft \mathit{OUT\_DL} \end{array}\right)\\
\bullet\ \mathit{TReq1}\\
\textbf{end}
\end{array}
$$

The requirements of the abstract system are specified by the **system** process below.

$$\textbf{system } \mathit{CDx} \mathrel{\widehat=} \mathit{ABReqsCDx} \lbrack\!\lbrack \{\!\lvert \mathit{next\_frame}, \mathit{output\_collisions} \rvert\!\} \rbrack\!\rbrack \mathit{ATReqsCDx}$$

We have one behavioural requirement (*BReq*1) and one timing requirement (*TReq*1). The behavioural requirement specifies the result of the collision detection. It is defined by a recursion that inputs the next frame via a synchronisation on $next\_frame$, updates the process state while calculating the new aircraft motions ($RecordFrame$), computes the collisions and deposits them in the local variable $colls$ ($CalcCollisions$), and outputs them on the channel $output\_collisions$. Any number greater than the precise number of collisions may be output unless there are no collisions (in that case 0 must be output). Since the collision set is symmetric, the actual number of collisions is obtained by dividing the cardinality of the set by 2. The calculation of the collision set makes use of the $CalcCollisionSet$ function defined earlier on in Section 3.2.

The time-wise behaviour in each cycle is captured by the local action $TReqCycle$. It is used in defining the overall timing requirement $TReq1$. Its specification states that $TReqCycle$ is executed once in each cycle and has to terminate within the period ($\ldots \blacktriangleright FRAME\_PERIOD$). The interleaving with **wait** $FRAME\_PERIOD$ ensures that moreover we do not terminate before the period expires.

The specification of $TReqCycle$ imposes a deadline, too, (INP_DL), namely on the input communication on $next\_frame$. This is an imposition on the environment to make the next frame available within a certain period of time once the program is ready to accept it. We record in t the time it took to communicate the next frame from when the communication was first offered. The subsequent nondeterministic **wait** provides freedom to the implementation to use up to $FRAME\_PERIOD - t - OUT\_DL$ units of time to calculate the collisions and then output the result on the $output\_collisions$ channel. The environment has to accept the output within OUT_DL time units from when it is offered by the program. This can prevent a situation in which the environment delays the communication on $output\_collisions$ right to the end of the cycle, and the infrastructure may thus not have time to initiate the next cycle.

We note that using **wait** $FRAME\_PERIOD - t - OUT\_DL$ rather than **wait** $FRAME\_PERIOD - t$ is a modelling decision to ensure that the environment is potentially given not less that OUT_DL time units to accept the output, and that an implementation cannot restrict this allowance.

The CDx system process yields the specification of the entire system. This is a parallelism between the two requirement processes $ABReqsCDx$ and $ATReqsCDx$. The processes synchronise on both external channels $next\_frame$ and $output\_collisions$.

Field	Type	Location	Access Mode	Memory Area
simulator	Simulator	CDxMission	shared	immortal
frameBuffer	FrameBuffer	Simulation	shared	immortal
currentFrame	RawFrame	CDxMission	shared	mission
state	StateTable	CDxMission	shared	mission
voxel_map	HashMap	ReducerHandler	local	per release
work	Partition	CDxMission	shared	mission
collisions	int	CDxMission	shared	mission
control	DetectorControl	CDxMission	shared	mission
factories	PersistentData	ReducerHandler	local	mission

Table 1: Analysis of relevant shared and local data in the parallel CD_x.

4 Anchor O

In this section, we discuss the construction of the O anchor. This is done in three refinement phases as explained in [6]. They are namely CS (concrete state), SD (shared data) and EL (elimination). We first examine the shared data in the CD_x program and then proceed with the refinement model(s) for each phase.

4.1 Shared Data

To guide the data refinements in this step, Table 1 summarises the relevant shared and local fields of the parallel CD_x SCJ program. All data resides in either mission or immortal memory except for `voxel_map`, which is local to `ReducerHandler`; we have to consider `voxel_map` in the O anchor data refinements though because other shared data depend on its presence in the model; this is in particular to formulate a suitable retrieve relation for `work`.

Below we give a brief explanation of the purpose of each variable.

- The `simulator` and `frameBuffer` objects are part of the simulation. The `FrameBuffer` class provides the mechanism for reading the next radar frame from the hardware. In the original CD_x, `frameBuffer` was located in a class `ImmortalEntry` whose name we changed to `Simulator`. The `frameBuffer` object is not directly represented as part of the model since we abstract from the details of the mechanism that interact with the hardware, as well as the storage to buffer device data for radar frames.

- The `currentFrame` variable corresponds to the *posns* component of the abstract model. Its type `RawFrame` records this data by virtue of various arrays of primitive types: `int[]`, `byte[]` and `float[]`. These hold the call signs and positions of the aircrafts.

- The `state` variable of type `StateTable` holds the previous positions of aircrafts. It thus does not encode the *motions* vectors directly, but we can construct them from the content of `currentFrame` and `state`. Unlike `RawFrame`, `StateTable` records the positions by way of a (customised) `CHashMap`. It also manages the allocation of `Vector3d` objects for positions as to avoid memory leaks.

- The `voxel_map` field is local to `ReducerHandler` and thus not shared. It records the result of the voxel hashing operation and is needed to specify essential properties of the algorithm as well as the retrieve relation for `work` (which is shared). It thus becomes relevant to the O anchor.

- The `work` variable of type `Partition` is used to divide and record the computational work assigned to each detection handler. The `Partition` class provides some methods that facilitate this. This object is shared between the `DetectorHandler` classes as well as `ReducerHandler` who initialises it.

- The `collisions` variable accumulates the number of collisions detected by the parallel detection handlers. It is concurrently accessed by them via synchronised methods to avoid data races.
- The `control` variable holds an object of type `DetectorControl` which is used to orchestrate the execution of the detector handlers and the output handler.
- The `factories` variable resides in mission memory despite being local. It is an artifacts of our program design to pre-allocate shared objects. This is important to avoid dynamic allocations in mission memory while the mission executes. We ignore it in the O anchor as it is introduced during algorithmic refinement.

The above analysis yields the following correspondence between abstract model variables and concrete program variables. The *posns* and *motions* state components of *ABReqsCDx* are exactly represented by the `currentFrame` and `state` variables in the program (CS phase). The other variables refer to shared and local data that has to be introduced as part of the SD phase of the O anchor, apart from the simulation-related classes and `factories`. The `control` object is also not considered in the O anchor as it will be introduced later on in the design, namely in the E anchor when refining the control behaviour.

We have omitted the shared objects for SCJ events in Table 1; this is because they are not considered as data objects. We next examine the refinements that introduce the aforementioned class objects for shared data into the model. For the CS and EL phases, we have a single (data) refinement, whereas for the SD phase we carry out the refinement in two incremental steps. Apart from the finalising EL model, none of the models discard existing state components but merely extend the state of the previous process. To disambiguate the names of state components inside the retrieve relations, we use subscripts.

4.2 Phase **CS**

In the first phase of the O anchor, we data-refine the abstract model variables *posns* and *motions* into their concrete representations in the program. This is via the shared objects *currentFrame* and *state* of class type *RawFrame* and *StateTable*. They are used to record the current and previous positions of aircrafts.

Refining State

The state of the refining process is given by the following state schema.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\mathit{OCSStateCDx}} \\
\mathit{posns}_1 : \mathit{Frame} \\
\mathit{motions}_1 : \mathit{Frame} \\
\mathit{currentFrame}_1 : \mathit{RawFrame} \\
\mathit{state}_1 : \mathit{StateTable} \\
\hline
\end{array}
$$

The class types *RawFrame* and *StateTable* are specified in Appendix A.1 and A.2. We note that we retain the abstract state components as auxiliary variables following Morgan's approach [9].

Retrieve Relation

The retrieve relation is given by the following schema that associates abstract and concrete states.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{OCSRetrCDx}} \\
\mathit{AStateCDx} \\
\mathit{OCSStateCDx} \\
\hline
\mathit{posns}_1 = \mathit{posns} \land \mathit{motions}_1 = \mathit{motions} \\
\mathit{currentFrame}_1 \neq \mathbf{null} \land \mathit{state}_1 \neq \mathbf{null} \\
\mathit{posns} = \left(\begin{array}{l}
\lambda\, a : \mathit{Aircraft} \mid \mathit{currentFrame}_1 \,.\, \mathit{find}(a) \neq -1 \bullet \\
\quad \left(\begin{array}{l}
\mathbf{let}\ i == \mathit{currentFrame}_1 \,.\, \mathit{find}(a) \bullet \\
\quad \mathit{MkVector} \left(\begin{array}{l}
\mathit{currentFrame}_1 \,.\, \mathit{positions} \,.\, \mathit{getA}(3 * i), \\
\mathit{currentFrame}_1 \,.\, \mathit{positions} \,.\, \mathit{getA}(3 * i + 1), \\
\mathit{currentFrame}_1 \,.\, \mathit{positions} \,.\, \mathit{getA}(3 * i + 2)
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\mathit{motions} = \left(\begin{array}{l}
\lambda\, a : \mathit{Aircraft} \mid \mathit{currentFrame}_1 \,.\, \mathit{find}(a) \neq -1 \bullet \\
\quad \left(\begin{array}{l}
\mathbf{let}\ \mathit{prev} == \mathit{state}_1 \,.\, \mathit{position\_map} \,.\, \mathit{get}(\mathit{MkCallSign}(a)) \bullet \\
\quad \mathbf{if}\ \mathit{prev} \neq \mathbf{null} \\
\quad \mathbf{then}\ \mathit{posns}(a) -_V \mathit{MkVector}(\mathit{prev} \,.\, x, \mathit{prev} \,.\, y, \mathit{prev} \,.\, z) \\
\quad \mathbf{else}\ \mathit{ZeroV}
\end{array}\right)
\end{array}\right) \\
\hline
\end{array}
$$

For reasons of definedness, we require $\mathit{currentFrame}_1$ and state_1 not to be **null**. We observe that the abstract state is expressed as a function of the concrete state. This enables a calculations approach to derive the refining data operations. Here, we do not fully simplify the calculated refining operations though.

Refining Process

The process for the first data refinement of the Anchor O is given below. We target the refinement of $ABReqsCDx$, that is the abstract behavioural requirements. The new state invariant and refined data operations have been calculated; as mentioned before, they are not fully simplified yet.

$\mathbf{process}\ OCSBReqsCDx \mathrel{\widehat=} \mathbf{begin}$

$$
\begin{array}{|l}
\textbf{state}\ OCSStateCDx \\
\hline
posns_1 : Frame \\
motions_1 : Frame \\
currentFrame_1 : RawFrame \\
state_1 : StateTable \\
\hline
\operatorname{dom} posns_1 = \operatorname{dom} motions_1 \\
currentFrame_1 \neq \mathbf{null} \land state_1 \neq \mathbf{null} \\
posns_1 = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_1 \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ i == currentFrame_1 \,.\, find(a) \bullet \\ \quad MkVector \left(\begin{array}{l} currentFrame_1 \,.\, positions \,.\, getA(3 * i), \\ currentFrame_1 \,.\, positions \,.\, getA(3 * i + 1), \\ currentFrame_1 \,.\, positions \,.\, getA(3 * i + 2) \end{array}\right) \end{array}\right) \end{array}\right) \\
motions_1 = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_1 \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ prev == state_1 \,.\, position\_map \,.\, get(MkCallSign(a)) \bullet \\ \quad \mathbf{if}\ prev \neq \mathbf{null} \\ \quad \mathbf{then}\ posns_1(a) -_V MkVector(prev \,.\, x, prev \,.\, y, prev \,.\, z) \\ \quad \mathbf{else}\ ZeroV \end{array}\right) \end{array}\right)
\end{array}
$$

$$
\begin{array}{|l}
Init \\
\hline
OCSStateCDx\,' \\
\hline
posns_1' = \varnothing \land motions_1' = \varnothing \\
currentFrame_1' = \mathbf{new}\ RawFrame \land state_1' = \mathbf{new}\ StateTable
\end{array}
$$

$$
\begin{array}{|l}
RecordFrame \\
\hline
\Delta OCSStateCDx \\
frame? : Frame \\
\hline
posns_1' = frame? \\
motions_1' = \{ a : \operatorname{dom} posns_1' \bullet a \mapsto \mathbf{if}\ a \in \operatorname{dom} posns_1\ \mathbf{then}\ (posns_1'\ a) -_V (posns_1\ a)\ \mathbf{else}\ ZeroV \} \\
posns_1 = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_1 \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ i == currentFrame_1 \,.\, find(a) \bullet \\ \quad MkVector \left(\begin{array}{l} currentFrame_1 \,.\, positions \,.\, getA(3 * i), \\ currentFrame_1 \,.\, positions \,.\, getA(3 * i + 1), \\ currentFrame_1 \,.\, positions \,.\, getA(3 * i + 2) \end{array}\right) \end{array}\right) \end{array}\right) \\
posns_1' = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_1' \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ i == currentFrame_1' \,.\, find(a) \bullet \\ \quad MkVector \left(\begin{array}{l} currentFrame_1' \,.\, positions \,.\, getA(3 * i), \\ currentFrame_1' \,.\, positions \,.\, getA(3 * i + 1), \\ currentFrame_1' \,.\, positions \,.\, getA(3 * i + 2) \end{array}\right) \end{array}\right) \end{array}\right) \\
motions_1' = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_1' \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ prev == state_1' \,.\, position\_map \,.\, get(MkCallSign(a)) \bullet \\ \quad \mathbf{if}\ prev \neq \mathbf{null} \\ \quad \mathbf{then}\ posns_1'(a) -_V MkVector(prev \,.\, x, prev \,.\, y, prev \,.\, z) \\ \quad \mathbf{else}\ ZeroV \end{array}\right) \end{array}\right)
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{CalcCollisions} \\
\hline
\Xi OCSStateCDx \\
colls! : \mathbb{N} \\
\hline
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns_1, motions_1) \bullet \\
\quad (\#\, collset = 0 \land colls! = 0) \lor (\#\, collset > 0 \land colls! \geq (\#\, collset) \operatorname{div} 2) \\
posns_1 = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_1 \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ i == currentFrame_1 \,.\, find(a) \bullet \\ \qquad MkVector \left(\begin{array}{l} currentFrame_1 \,.\, positions \,.\, getA(3 * i), \\ currentFrame_1 \,.\, positions \,.\, getA(3 * i + 1), \\ currentFrame_1 \,.\, positions \,.\, getA(3 * i + 2) \end{array} \right) \end{array} \right) \end{array} \right) \\
motions_1 = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_1 \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ prev == state_1 \,.\, position\_map \,.\, get(MkCallSign(a)) \bullet \\ \quad \mathbf{if}\ prev \neq \mathbf{null} \\ \quad \mathbf{then}\ posns_1(a) -_V MkVector(prev \,.\, x, prev \,.\, y, prev \,.\, z) \\ \quad \mathbf{else}\ ZeroV \end{array} \right) \end{array} \right) \\
\hline
\end{array}
$$

$$
\begin{array}{l}
BReq1 \mathrel{\widehat=} next\_frame\,?\,frame \longrightarrow \\
\quad \left(\begin{array}{l} RecordFrame; \\ \mathbf{var}\ colls : \mathbb{N} \bullet CalcCollisions \,;\ output\_collisions\,!\,colls \longrightarrow BReq1 \end{array} \right)
\end{array}
$$

$\bullet$ *Init* ; *BReq*1

end

The local action *BReq*1 and the main action remain exactly as in $ABReqsCDx$. Simulation laws in *Circus* [4] establish that this yields a valid process refinement of $ABReqsCDx$. A detailed proof of this is omitted but not difficult. Regarding the data operations, we expect that further refinement later on in the AR phase of the E anchor transforms them into executable code, so simplification of the refined data operations can (an probably should) be postponed in this anchor. Automatic tools can in principle assist the simplification and enable the developer to take full advantage of the calculational approach. This single refinement concludes the CS phase. We next turn to the SD phase where shared data is introduced.

4.3 Phase SD

This phase is divided into two incremental refinements to leverage the proof effort.

- The first refinement introduces the shared variable *collisions* whose purpose is to hold the detected number of collisions after each detection cycle.
- The second refinement introduces the shared variable *work*, concurrently with the local variable *voxel_map*. Whereas the *work* component divides the computational work, *voxel_map* determines the result of the voxel hashing operation and is required to specify the retrieve relation for *work*.

We present the process model for each refinement step.

Refinement 1

In the first refinement of the SD phase, we introduce the shared *collisions* variable. It holds the result of the collision detection at the end of each cycle.

Refining State

The state of the refining process is given by the following state schema.

$$
\begin{array}{|l}
\hline
\textit{OSD1StateCDx} \\
\hline
posns_2 : Frame \\
motions_2 : Frame \\
currentFrame_2 : RawFrame \\
state_2 : StateTable \\
collisions_2 : int \\
\hline
\end{array}
$$

Again, all state components of the previous model are retained. Subscripts are used for disambiguation.

Retrieve Relation

The retrieve relation of the first refinement of SD is specified below.

$$
\begin{array}{|l}
\hline
\textit{OSD1RetrCDx} \\
\hline
OCSStateCDx \\
OSD1StateCDx \\
\hline
posns_2 = posns_1 \land motions_2 = motions_1 \\
currentFrame_2 = currentFrame_1 \land state_2 = state_1 \\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns_2, motions_2) \bullet \\
\quad (\#\, collset = 0 \land collisions_2 = 0) \lor (\#\, collset > 0 \land collisions_2 \geq (\#\, collset) \operatorname{div} 2) \\
\hline
\end{array}
$$

We constrain the value of $collisions_2$ similar to $colls!$ in the $CalcCollisions$ action. This reflects the intention of refining $CalcCollisions$ by an assignment $colls := collisions_2$.

Refining Process

The process for the first refinement of the SD phase is presented in the sequel.

$\mathbf{process}\ OSD1BReqsCDx \mathrel{\widehat=} \mathbf{begin}$

$$
\begin{array}{|l}
\hline
\mathbf{state}\ \textit{OSD1StateCDx} \\
\hline
posns_2 : Frame \\
motions_2 : Frame \\
currentFrame_2 : RawFrame \\
state_2 : StateTable \\
collisions_2 : int \\
\hline
\operatorname{dom} posns_2 = \operatorname{dom} motions_2 \\
currentFrame_2 \neq \mathbf{null} \land state_2 \neq \mathbf{null} \\
posns_2 = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_2 \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ i == currentFrame_2 \,.\, find(a) \bullet \\ \quad MkVector \left(\begin{array}{l} currentFrame_2 \,.\, positions \,.\, getA(3 * i), \\ currentFrame_2 \,.\, positions \,.\, getA(3 * i + 1), \\ currentFrame_2 \,.\, positions \,.\, getA(3 * i + 2) \end{array} \right) \end{array} \right) \end{array} \right) \\
motions_2 = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_2 \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ prev == state_2 \,.\, position\_map \,.\, get(MkCallSign(a)) \bullet \\ \quad \mathbf{if}\ prev \neq \mathbf{null} \\ \quad \mathbf{then}\ posns_2(a) -_V MkVector(prev \,.\, x, prev \,.\, y, prev \,.\, z) \\ \quad \mathbf{else}\ ZeroV \end{array} \right) \end{array} \right) \\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns_2, motions_2) \bullet \\
\quad (\#\, collset = 0 \land collisions_2 = 0) \lor (\#\, collset > 0 \land collisions_2 \geq (\#\, collset) \operatorname{div} 2) \\
\hline
\end{array}
$$

Init

$$OSD1StateCDx\,'$$

$$\begin{array}{l}
posns_2' = \varnothing \land motions_2' = \varnothing \\
currentFrame_2' = \mathbf{new}\ RawFrame \land state_2' = \mathbf{new}\ StateTable \\
collisions_2' = 0
\end{array}$$

RecordFrame

$$\begin{array}{l}
\Delta OSD1StateCDx \\
frame? : Frame
\end{array}$$

$$\begin{array}{l}
posns_2' = frame? \\
motions_2' = \{a : \operatorname{dom} posns_2' \bullet a \mapsto \mathbf{if}\ a \in \operatorname{dom} posns_2\ \mathbf{then}\ (posns_2'\ a) -_V (posns_2\ a)\ \mathbf{else}\ ZeroV\} \\
posns_2 = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_2 \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ i == currentFrame_2 \,.\, find(a) \bullet \\ \quad MkVector \left(\begin{array}{l} currentFrame_2 \,.\, positions \,.\, getA(3 * i), \\ currentFrame_2 \,.\, positions \,.\, getA(3 * i + 1), \\ currentFrame_2 \,.\, positions \,.\, getA(3 * i + 2) \end{array}\right) \end{array}\right) \end{array}\right) \\
posns_2' = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_2' \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ i == currentFrame_2' \,.\, find(a) \bullet \\ \quad MkVector \left(\begin{array}{l} currentFrame_2' \,.\, positions \,.\, getA(3 * i), \\ currentFrame_2' \,.\, positions \,.\, getA(3 * i + 1), \\ currentFrame_2' \,.\, positions \,.\, getA(3 * i + 2) \end{array}\right) \end{array}\right) \end{array}\right) \\
motions_2' = \left(\begin{array}{l} \lambda\, a : Aircraft \mid currentFrame_2' \,.\, find(a) \neq -1 \bullet \\ \quad \left(\begin{array}{l} \mathbf{let}\ prev == state_2' \,.\, position_map \,.\, get(MkCallSign(a)) \bullet \\ \quad \mathbf{if}\ prev \neq \mathbf{null} \\ \quad \mathbf{then}\ posns_2'(a) -_V MkVector(prev \,.\, x, prev \,.\, y, prev \,.\, z) \\ \quad \mathbf{else}\ ZeroV \end{array}\right) \end{array}\right) \\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns_2', motions_2') \bullet \\
\quad (\#\, collset = 0 \land collisions_2' = 0) \lor (\#\, collset > 0 \land collisions_2' \geq (\#\, collset) \operatorname{div} 2)
\end{array}$$

CalcCollisions

$$\begin{array}{l}
\Xi OSD1StateCDx \\
colls! : \mathbb{N}
\end{array}$$

$$colls! = collisions_2$$

$$\begin{array}{l}
BReq1 \mathrel{\widehat=} next_frame\,?\,frame \longrightarrow \\
\quad \left(\begin{array}{l} RecordFrame; \\ \mathbf{var}\ colls : \mathbb{N} \bullet CalcCollisions\ ;\ \ output_collisions\,!\,colls \longrightarrow BReq1 \end{array}\right)
\end{array}$$

$\bullet$ *Init* ; *BReq1*

end

As a result of this first refinement stage of SD, *RecordFrame* in the refined process not only records the radar frame and updates the previous aircraft positions but also calculates the collisions. Contrary, the *CalcCollisions* operation now simply returns the value of the shared variable $collisions_2$ rather than performing any calculation as it was the case before.

Refinement 2

The second refinement of SD introduces the shared variable *work* of type *Partition*. It is a shared object between the handlers and used to distribute the computational work determined by *voxel_map*. We hence concurrently introduce the local variable *voxel_map*, too. The latter records aircraft positions and motions in a *HashMap* object that maps *Vector2d* to *List*[*Motion*] objects. In the program, this corresponds to the voxel hashing performed by `ReducerHandler` of which `voxel_map` is a local variable. The type *Vector2d* is used to index the voxel space. The *Motion* class records the call sign, current position, and previous position of an aircraft, and *List* models a standard (Java) list. The class specifications for *Partition*, *Motion*, *Vector2d*, *HashMap*, and *List* can all be found in the Appendix A.

Refining State

The state of the refining process is given by the following state schema.

$$
\begin{array}{l}
\textit{OSD2StateCDx} \\
\hline
\textit{posns}_3 : \textit{Frame} \\
\textit{motions}_3 : \textit{Frame} \\
\textit{currentFrame}_3 : \textit{RawFrame} \\
\textit{state}_3 : \textit{StateTable} \\
\textit{work}_3 : \textit{Partition} \\
\textit{collisions}_3 : \textit{int} \\
\hline
\end{array}
$$

Again, all state components of the previous model are retained. Subscripts are used for disambiguation. We note that *voxel_map* has not been added as a state component.

Retrieve Relation

The retrieve relation of the second refinement relates *work* and *voxel_map*.

$$
\begin{array}{l}
\textit{OSD2RetrCDx} \\
\hline
\textit{OSD1StateCDx} \\
\textit{OSD2StateCDx} \\
\hline
\textit{posns}_3 = \textit{posns}_2 \land \textit{motions}_3 = \textit{motions}_2 \\
\textit{currentFrame}_3 = \textit{currentFrame}_2 \land \textit{state}_3 = \textit{state}_2 \\
\textit{work}_3 \neq \mathbf{null} \\
\exists\, \textit{voxel\_map} : \textit{HashMap}[\textit{Vector2d}, \textit{List}[\textit{Motion}]] \mid \textit{voxel\_map} \neq \mathbf{null} \bullet \\
\left(\begin{array}{l}
\forall\, a_1, a_2 : \textit{Aircraft} \mid \{a_1, a_2\} \subseteq \operatorname{dom} \textit{posns}_3 \bullet \\
\quad (a_1, a_2) \in \textit{CalcCollisionSet}(\textit{posns}_3, \textit{motions}_3) \Rightarrow \\
\qquad \left(\begin{array}{l}
\exists\, l : \textit{List}[\textit{Motion}] \mid l \in \textit{voxel\_map} \,.\, \textit{values}() \,.\, \textit{elems}() \bullet \\
\quad \textit{MkMotion}(a_1, \textit{posns}_3\, a_1 -_V \textit{motions}_3\, a_1, \textit{posns}_3\, a_1) \in l \,.\, \textit{elems}() \land \\
\quad \textit{MkMotion}(a_2, \textit{posns}_3\, a_2 -_V \textit{motions}_3\, a_2, \textit{posns}_3\, a_2) \in l \,.\, \textit{elems}()
\end{array}\right) \\
\textit{voxel\_map} \,.\, \textit{values}() \,.\, \textit{elems}() = \bigcup \{ i : 1 \,..\, 4 \bullet \textit{work}_3 \,.\, \textit{getDetectorWork}(i) \,.\, \textit{elems}() \}
\end{array}\right) \\
\textit{collisions}_3 = \textit{collisions}_2 \\
\hline
\end{array}
$$

As before, *voxel_map* and *work* must not be **null** to avoid undefinedness issues. We observe *voxel_map* has been introduced as a local variable rather than a state component of the process. Above we furthermore make use of an auxiliary function *MkMotion*, loosely specified below.

$$
\textit{MkMotion} : \textit{Aircraft} \times \textit{Vector} \times \textit{Vector} \rightarrow \textit{Motion}
$$

It yields a *Motion* object for an *Aircraft* and its previous and current position. It corresponds to the constructor of the `Motion` class in the SCJ program of the parallel CD_x. We recall that the logical method *elems*()

returns the elements of a *List* object as a set. The results of the method call $work_3 \,.\, getDetectorWork(i)$ determines the voxels to be checked by detector i; it is of type $List[List[Motion]]$. The method call $voxel\_map \,.\, values()$ returns the list of values in the hash table (it is also of type $List[List[Motion]]$).

Refining Process

The process for the second refinement of the SD phase is presented below.

process $OSD2BReqsCDx \mathrel{\widehat=}$ **begin**

state $OSD2StateCDx$

$$
\begin{array}{l}
posns_3 : Frame \\
motions_3 : Frame \\
currentFrame_3 : RawFrame \\
state_3 : StateTable \\
work_3 : Partition \\
collisions_3 : int
\end{array}
$$

$$
\begin{array}{l}
\operatorname{dom} posns_3 = \operatorname{dom} motions_3 \\
currentFrame_3 \neq \mathbf{null} \land state_3 \neq \mathbf{null} \land work_3 \neq \mathbf{null} \\
\exists\, voxel\_map : HashMap[Vector2d, List[Motion]] \mid voxel\_map \neq \mathbf{null} \bullet \\
\left(\begin{array}{l}
posns_3 = \left(\begin{array}{l}
\lambda\, a : Aircraft \mid currentFrame_3 \,.\, find(a) \neq -1 \bullet \\
\quad \left(\begin{array}{l}
\mathbf{let}\ i == currentFrame_3 \,.\, find(a) \bullet \\
\quad MkVector \left(\begin{array}{l}
currentFrame_3 \,.\, positions \,.\, getA(3 * i), \\
currentFrame_3 \,.\, positions \,.\, getA(3 * i + 1), \\
currentFrame_3 \,.\, positions \,.\, getA(3 * i + 2)
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
motions_3 = \left(\begin{array}{l}
\lambda\, a : Aircraft \mid currentFrame_3 \,.\, find(a) \neq -1 \bullet \\
\quad \left(\begin{array}{l}
\mathbf{let}\ prev == state_3 \,.\, position\_map \,.\, get(MkCallSign(a)) \bullet \\
\quad \mathbf{if}\ prev \neq \mathbf{null} \\
\quad \mathbf{then}\ posns_3(a) -_V MkVector(prev \,.\, x, prev \,.\, y, prev \,.\, z) \\
\quad \mathbf{else}\ ZeroV
\end{array}\right)
\end{array}\right) \\
\left(\begin{array}{l}
\forall\, a_1, a_2 : Aircraft \mid \{a_1, a_2\} \subseteq \operatorname{dom} posns_3 \bullet \\
\quad (a_1, a_2) \in CalcCollisionSet(posns_3, motions_3) \Rightarrow \\
\quad\quad \left(\begin{array}{l}
\exists\, l : List[Motion] \mid l \in voxel\_map \,.\, values() \,.\, elems() \bullet \\
\quad MkMotion(a_1, posns_3\ a_1 -_V motions_3\ a_1, posns_3\ a_1) \in l \,.\, elems() \land \\
\quad MkMotion(a_2, posns_3\ a_2 -_V motions_3\ a_2, posns_3\ a_2) \in l \,.\, elems()
\end{array}\right)
\end{array}\right) \\
voxel\_map \,.\, values() \,.\, elems() = \bigcup \{ i : 1 \,..\, 4 \bullet work_3 \,.\, getDetectorWork(i) \,.\, elems() \} \\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns_3, motions_3) \bullet \\
\quad (\#\, collset = 0 \land collisions_3 = 0) \lor (\#\, collset > 0 \land collisions_3 \geq (\#\, collset) \operatorname{div} 2)
\end{array}\right)
\end{array}
$$

Init

$$
OSD2StateCDx\,'
$$

$$
\begin{array}{l}
posns_3' = \varnothing \land motions_3' = \varnothing \\
currentFrame_3' = \mathbf{new}\ RawFrame \\
state_3' = \mathbf{new}\ StateTable \\
work_3' = \mathbf{new}\ Partition(4) \\
collisions_3' = 0
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{RecordFrame}} \\
\hline
\Delta OSD2StateCDx \\
frame? : Frame \\
\hline
posns_3' = frame? \\
motions_3' = \{a : \mathrm{dom}\, posns_3' \bullet a \mapsto \mathbf{if}\ a \in \mathrm{dom}\, posns_3\ \mathbf{then}\ (posns_3'\ a) -_V (posns_3\ a)\ \mathbf{else}\ ZeroV\} \\
\exists\, voxel\_map : HashMap[Vector2d, List[Motion]] \mid voxel\_map \neq \mathbf{null} \bullet \\
\left(\begin{array}{l}
posns_3 = \left(\begin{array}{l}
\lambda\, a : Aircraft \mid currentFrame_3 \,.\, find(a) \neq -1 \bullet \\
\quad \left(\begin{array}{l}
\mathbf{let}\ i == currentFrame_3 \,.\, find(a) \bullet \\
\qquad MkVector \left(\begin{array}{l}
currentFrame_3 \,.\, positions \,.\, getA(3 * i), \\
currentFrame_3 \,.\, positions \,.\, getA(3 * i + 1), \\
currentFrame_3 \,.\, positions \,.\, getA(3 * i + 2)
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
posns_3' = \left(\begin{array}{l}
\lambda\, a : Aircraft \mid currentFrame_3' \,.\, find(a) \neq -1 \bullet \\
\quad \left(\begin{array}{l}
\mathbf{let}\ i == currentFrame_3' \,.\, find(a) \bullet \\
\qquad MkVector \left(\begin{array}{l}
currentFrame_3' \,.\, positions \,.\, getA(3 * i), \\
currentFrame_3' \,.\, positions \,.\, getA(3 * i + 1), \\
currentFrame_3' \,.\, positions \,.\, getA(3 * i + 2)
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
motions_3' = \left(\begin{array}{l}
\lambda\, a : Aircraft \mid currentFrame_3' \,.\, find(a) \neq -1 \bullet \\
\quad \left(\begin{array}{l}
\mathbf{let}\ prev == state_3' \,.\, position\_map \,.\, get(MkCallSign(a)) \bullet \\
\quad \mathbf{if}\ prev \neq \mathbf{null} \\
\quad \mathbf{then}\ posns_3'(a) -_V MkVector(prev \,.\, x, prev \,.\, y, prev \,.\, z) \\
\quad \mathbf{else}\ ZeroV
\end{array}\right)
\end{array}\right) \\
\left(\begin{array}{l}
\forall\, a_1, a_2 : Aircraft \mid \{a_1, a_2\} \subseteq \mathrm{dom}\, posns_3' \bullet \\
\quad (a_1, a_2) \in CalcCollisionSet(posns_3', motions_3') \Rightarrow \\
\qquad \left(\begin{array}{l}
\exists\, l : List[Motion] \mid l \in voxel\_map \,.\, values() \,.\, elems() \bullet \\
\quad MkMotion(a_1, posns_3'\ a_1 -_V motions_3'\ a_1, posns_3'\ a_1) \in l \,.\, elems() \wedge \\
\quad MkMotion(a_2, posns_3'\ a_2 -_V motions_3'\ a_2, posns_3'\ a_2) \in l \,.\, elems()
\end{array}\right)
\end{array}\right) \\
voxel\_map \,.\, values() \,.\, elems() = \bigcup \{i : 1 \,..\, 4 \bullet work_3' \,.\, getDetectorWork(i) \,.\, elems()\} \\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns_3', motions_3') \bullet \\
\quad (\#\, collset = 0 \wedge collisions_3' = 0) \vee (\#\, collset > 0 \wedge collisions_3' \geq (\#\, collset)\ \mathrm{div}\ 2)
\end{array}\right) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{CalcCollisions}} \\
\hline
\Xi OSD2StateCDx \\
colls! : \mathbb{N} \\
\hline
colls! = collisions_3 \\
\hline
\end{array}
$$

$$
\begin{array}{l}
BReq1 \mathrel{\widehat{=}} next\_frame\,?\,frame \longrightarrow \\
\quad \left(\begin{array}{l}
RecordFrame; \\
\mathbf{var}\ colls : \mathbb{N} \bullet CalcCollisions\ ;\ \ output\_collisions\,!\,colls \longrightarrow BReq1
\end{array}\right)
\end{array}
$$

$\bullet\ Init\ ;\ \ BReq1$

end

With the above refinement we have introduced all shared data. In addition to recording the frame, updating the motions and calculation collisions, *RecordFrame* here also constructs the voxel map and carries out the calculation for dividing the computational work between the detector handlers by setting *work*.

This concludes the models for SD and in order to finalise the O anchor, the subsequent and last phase eliminates the auxiliary model variables $posns_3$ and $motions_3$.

4.4 Phase EL

In the EL phase we remove the model variables $posns_3$ and $motions_3$ which so far have been kept as auxiliary variables to facilitate the formulation of retrieve relations and operation refinements. This is achieved by turning them into local constants within the state schema and data operations.

Not to clutter up the models, we introduce the following global functions F and G.

$$
\begin{array}{|l}
F : RawFrame \rightarrow Frame \\
\hline
\forall\, currentFrame : RawFrame \bullet \\
\quad F(currentFrame) = \\
\qquad \left(\begin{array}{l}
\lambda\, a : Aircraft \mid currentFrame\,.\,find(a) \neq -1 \bullet \\
\quad \left(\begin{array}{l}
\mathbf{let}\; i == currentFrame\,.\,find(a) \bullet \\
\quad MkVector \left(\begin{array}{l}
currentFrame\,.\,positions\,.\,getA(3 * i), \\
currentFrame\,.\,positions\,.\,getA(3 * i + 1), \\
currentFrame\,.\,positions\,.\,getA(3 * i + 2)
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{|l}
G : RawFrame \times StateTable \rightarrow Frame \\
\hline
\forall\, currentFrame : RawFrame;\; state : StateTable \bullet \\
\quad G(currentFrame, state) = \\
\qquad \left(\begin{array}{l}
\lambda\, a : Aircraft \mid currentFrame\,.\,find(a) \neq -1 \bullet \\
\quad \left(\begin{array}{l}
\mathbf{let}\; prev == state\,.\,position\_map\,.\,get(MkCallSign(a)) \bullet \\
\quad \mathbf{if}\; prev \neq \mathbf{null} \\
\quad \mathbf{then}\; F(currentFrame)(a) -_V MkVector(prev\,.\,x, prev\,.\,y, prev\,.\,z) \\
\quad \mathbf{else}\; ZeroV
\end{array}\right)
\end{array}\right)
\end{array}
$$

They calculate the abstract state components *posns* and *motions* from the concrete state components *currentFrame* and *state*. The functions are introduced solely to simplify the presentation of the models.

Refining Process

The process for the result of the EL stage is presented below.

process $OBReqsCDx \mathrel{\widehat=}$ **begin**

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathbf{state}\; OStateCDx} \\
\hline
currentFrame : RawFrame \\
state : StateTable \\
work : Partition \\
collisions : int \\
\hline
currentFrame \neq \mathbf{null} \land state \neq \mathbf{null} \land work \neq \mathbf{null} \\
\exists\, voxel\_map : HashMap[Vector2d, List[Motion]] \mid voxel\_map \neq \mathbf{null} \bullet \\
\exists\, posns : Frame;\; motions : Frame \mid \operatorname{dom} posns = \operatorname{dom} motions \bullet \\
\left(\begin{array}{l}
posns = F(currentFrame) \land motions = G(currentFrame, state) \land \\
\left(\begin{array}{l}
\forall\, a_1, a_2 : Aircraft \mid \{a_1, a_2\} \subseteq \operatorname{dom} posns \bullet \\
\quad (a_1, a_2) \in CalcCollisionSet(posns, motions) \Rightarrow \\
\qquad \left(\begin{array}{l}
\exists\, l : List[Motion] \mid l \in voxel\_map\,.\,values()\,.\,elems() \bullet \\
\quad MkMotion(a_1, posns\; a_1 -_V motions\; a_1, posns\; a_1) \in l\,.\,elems() \land \\
\quad MkMotion(a_2, posns\; a_2 -_V motions\; a_2, posns\; a_2) \in l\,.\,elems()
\end{array}\right)
\end{array}\right) \land \\
voxel\_map\,.\,values()\,.\,elems() = \bigcup\{i : 1\,..\,4 \bullet work\,.\,getDetectorWork(i)\,.\,elems()\} \land \\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns, motions) \bullet \\
\quad (\#\, collset = 0 \land collisions = 0) \lor (\#\, collset > 0 \land collisions \geq (\#\, collset) \operatorname{div} 2)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Init}} \\
\hline
\mathit{OStateCDx}' \\
\hline
\mathit{currentFrame}' = \mathbf{new}\ \mathit{RawFrame} \\
\mathit{state}' = \mathbf{new}\ \mathit{StateTable} \\
\mathit{work}' = \mathbf{new}\ \mathit{Partition}(4) \\
\mathit{collisions}' = 0 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{RecordFrame}} \\
\hline
\Delta \mathit{OStateCDx} \\
\mathit{frame}? : \mathit{Frame} \\
\hline
\exists \mathit{posns}, \mathit{motions} : \mathit{Frame};\ \mathit{posns}', \mathit{motions}' : \mathit{Frame} \mid \\
\quad \mathrm{dom}\, \mathit{posns} = \mathrm{dom}\, \mathit{motions} \land \mathrm{dom}\, \mathit{posns}' = \mathrm{dom}\, \mathit{motions}' \bullet \\
\exists \mathit{voxel\_map} : \mathit{HashMap}[\mathit{Vector2d}, \mathit{List}[\mathit{Motion}]] \mid \mathit{voxel\_map} \neq \mathbf{null} \bullet \\
\left(\begin{array}{l}
\mathit{posns} = F(\mathit{currentFrame}) \land \mathit{motions} = G(\mathit{currentFrame}, \mathit{state}) \land \\
\mathit{posns}' = F(\mathit{currentFrame}') \land \mathit{motions}' = G(\mathit{currentFrame}', \mathit{state}') \land \\
\mathit{posns}' = \mathit{frame}? \land \\
\mathit{motions}' = \\
\quad \{ a : \mathrm{dom}\, \mathit{posns}' \bullet a \mapsto \mathbf{if}\ a \in \mathrm{dom}\, \mathit{posns}\ \mathbf{then}\ (\mathit{posns}'\ a) -_V (\mathit{posns}\ a)\ \mathbf{else}\ \mathit{ZeroV} \} \land \\
\left(\begin{array}{l}
\forall a_1, a_2 : \mathit{Aircraft} \mid \{a_1, a_2\} \subseteq \mathrm{dom}\, \mathit{posns}' \bullet \\
\quad (a_1, a_2) \in \mathit{CalcCollisionSet}(\mathit{posns}', \mathit{motions}') \Rightarrow \\
\qquad \left(\begin{array}{l}
\exists l : \mathit{List}[\mathit{Motion}] \mid l \in \mathit{voxel\_map} . \mathit{values}() . \mathit{elems}() \bullet \\
\quad \mathit{MkMotion}(a_1, \mathit{posns}'\ a_1 -_V \mathit{motions}'\ a_1, \mathit{posns}'\ a_1) \in l . \mathit{elems}() \land \\
\quad \mathit{MkMotion}(a_2, \mathit{posns}'\ a_2 -_V \mathit{motions}'\ a_2, \mathit{posns}'\ a_2) \in l . \mathit{elems}()
\end{array}\right)
\end{array}\right) \land \\
\mathit{voxel\_map} . \mathit{values}() . \mathit{elems}() = \bigcup \{ i : 1 \,..\, 4 \bullet \mathit{work}' . \mathit{getDetectorWork}(i) . \mathit{elems}() \} \land \\
\exists \mathit{collset} : \mathbb{F}\,(\mathit{Aircraft} \times \mathit{Aircraft}) \mid \mathit{collset} = \mathit{CalcCollisionSet}(\mathit{posns}', \mathit{motions}') \bullet \\
\quad (\#\, \mathit{collset} = 0 \land \mathit{collisions}' = 0) \lor (\#\, \mathit{collset} > 0 \land \mathit{collisions}' \geq (\#\, \mathit{collset})\ \mathrm{div}\ 2)
\end{array}\right) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{CalcCollisions}} \\
\hline
\Xi \mathit{OStateCDx} \\
\mathit{colls}! : \mathbb{N} \\
\hline
\mathit{colls}! = \mathit{collisions} \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\mathit{BReq1} \mathrel{\widehat=} \mathit{next\_frame}\,?\,\mathit{frame} \longrightarrow \\
\quad \left(\begin{array}{l}
\mathit{RecordFrame}; \\
\mathbf{var}\ \mathit{colls} : \mathbb{N} \bullet \mathit{CalcCollisions}\ ;\ \ \mathit{output\_collisions}\,!\,\mathit{colls} \longrightarrow \mathit{BReq1}
\end{array}\right)
\end{array}
$$

$\bullet$ *Init* ; *BReq*1

end

With this process, we can now specify the top-level O anchor process.

$$\mathbf{system}\ \mathit{OCDx} \mathrel{\widehat=} \mathit{OBReqsCDx} \llbracket \{\!|\, \mathit{next\_frame}, \mathit{output\_collisions} \,|\!\} \rrbracket\ \mathit{ATReqsCDx}$$

Because the data refinements only affect the process for the behavioural requirements, the process for the timing requirements is the same as in the a abstract model. The simplification, decomposition and algorithmic refinement of the data operations *RecordFrame* is still due. This is an issue for the subsequent E anchor, which we discussed in detail in the next section.

5 Anchor E

In this section, we present the models for the E anchor. This anchor consists of the following five phases.

1. CP (collapse parallelism)
2. MS (mission architecture)
3. HS (handler architecture)
4. SH (encapsulate shared data)
5. AR (algorithmic refinement)

In comparison to the serial line example in [6], we subdivided the MH phase (missions and handlers) into two separate phase, one for the missions (MS) and one for the handlers (HS). Moreover the order of the SH phase and former MH phase has been reversed. The reason for this is that the latter determines the design that gives rise to atomic data operations emerging during SH. In the CD_x, this is, in particular, access to the shared *collisions* variable and also the refinement of a barrier mechanism that emerges during Stage 5 of MH. Clearly, we cannot refine those atomic operations before the design is in place, and for this reason SH has to be postponed until the missions and handlers design has fully emerged.

The MS, HS and SH phases are additionally subdivided into a number of logical stages which are explained as we go along. This revealed a set of refinement patterns which, along with the respective laws, are presented in detail too; a cumulative list of all elementary laws and high-level laws is included in Appendix B. The E anchor is overall the most challenging and interesting of the models and at the heart of the refinement approach; it also posses a few significant challenges for automation.

5.1 Phase CP

The process after performing the collapsing of parallelism is given below.

system *ECPCDx* $\widehat{=}$ **begin**

state *ECPStateCDx*

$$\begin{array}{l}
\mathit{currentFrame} : \mathit{RawFrame} \\
\mathit{state} : \mathit{StateTable} \\
\mathit{work} : \mathit{Partition} \\
\mathit{collisions} : \mathit{int}
\end{array}$$

$$\begin{array}{l}
\mathit{currentFrame} \neq \mathbf{null} \land \mathit{state} \neq \mathbf{null} \land \mathit{work} \neq \mathbf{null} \\
\exists\, \mathit{voxel_map} : \mathit{HashMap}[\mathit{Vector2d}, \mathit{List}[\mathit{Motion}]] \mid \mathit{voxel_map} \neq \mathbf{null} \bullet \\
\exists\, \mathit{posns} : \mathit{Frame};\ \mathit{motions} : \mathit{Frame} \mid \mathrm{dom}\, \mathit{posns} = \mathrm{dom}\, \mathit{motions} \bullet \\
\left(\begin{array}{l}
\mathit{posns} = F(\mathit{currentFrame}) \land \mathit{motions} = G(\mathit{currentFrame}, \mathit{state}) \land \\
\left(\begin{array}{l}
\forall\, a_1, a_2 : \mathit{Aircraft} \mid \{a_1, a_2\} \subseteq \mathrm{dom}\, \mathit{posns} \bullet \\
\quad (a_1, a_2) \in \mathit{CalcCollisionSet}(\mathit{posns}, \mathit{motions}) \Rightarrow \\
\qquad \left(\begin{array}{l}
\exists\, l : \mathit{List}[\mathit{Motion}] \mid l \in \mathit{voxel_map} \,.\, \mathit{values}() \,.\, \mathit{elems}() \bullet \\
\quad \mathit{MkMotion}(a_1, \mathit{posns}\ a_1 -_V \mathit{motions}\ a_1, \mathit{posns}\ a_1) \in l \,.\, \mathit{elems}() \land \\
\quad \mathit{MkMotion}(a_2, \mathit{posns}\ a_2 -_V \mathit{motions}\ a_2, \mathit{posns}\ a_2) \in l \,.\, \mathit{elems}()
\end{array}\right)
\end{array}\right) \land \\
\mathit{voxel_map} \,.\, \mathit{values}() \,.\, \mathit{elems}() = \bigcup \{ i : 1 \,..\, 4 \bullet \mathit{work} \,.\, \mathit{getDetectorWork}(i) \,.\, \mathit{elems}() \} \land \\
\exists\, \mathit{collset} : \mathbb{F}\,(\mathit{Aircraft} \times \mathit{Aircraft}) \mid \mathit{collset} = \mathit{CalcCollisionSet}(\mathit{posns}, \mathit{motions}) \bullet \\
\quad (\#\, \mathit{collset} = 0 \land \mathit{collisions} = 0) \lor (\#\, \mathit{collset} > 0 \land \mathit{collisions} \geq (\#\, \mathit{collset})\ \mathrm{div}\ 2)
\end{array}\right)
\end{array}$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Init}} \\
\hline
ECPStateCDx' \\
\hline
currentFrame' = \mathbf{new}\ RawFrame \\
state' = \mathbf{new}\ StateTable \\
work' = \mathbf{new}\ Partition(4) \\
collisions' = 0 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{RecordFrame}} \\
\hline
\Delta ECPStateCDx \\
frame? : Frame \\
\hline
\exists\, posns, motions : Frame;\ posns', motions' : Frame \mid \\
\quad \mathrm{dom}\, posns = \mathrm{dom}\, motions \land \mathrm{dom}\, posns' = \mathrm{dom}\, motions' \bullet \\
\exists\, voxel\_map : HashMap[Vector2d, List[Motion]] \mid voxel\_map \neq \mathbf{null} \bullet \\
\left(\begin{array}{l}
posns = F(currentFrame) \land motions = G(currentFrame, state) \land \\
posns' = F(currentFrame') \land motions' = G(currentFrame', state') \land \\
posns' = frame? \land \\
motions' = \\
\quad \{a : \mathrm{dom}\, posns' \bullet a \mapsto \mathbf{if}\ a \in \mathrm{dom}\, posns\ \mathbf{then}\ (posns'\ a) -_V (posns\ a)\ \mathbf{else}\ ZeroV\} \land \\
\left(\begin{array}{l}
\forall\, a_1, a_2 : Aircraft \mid \{a_1, a_2\} \subseteq \mathrm{dom}\, posns' \bullet \\
\quad (a_1, a_2) \in CalcCollisionSet(posns', motions') \Rightarrow \\
\qquad \left(\begin{array}{l}
\exists\, l : List[Motion] \mid l \in voxel\_map\,.\,values()\,.\,elems() \bullet \\
\quad MkMotion(a_1, posns'\ a_1 -_V motions'\ a_1, posns'\ a_1) \in l\,.\,elems() \land \\
\quad MkMotion(a_2, posns'\ a_2 -_V motions'\ a_2, posns'\ a_2) \in l\,.\,elems()
\end{array}\right)
\end{array}\right) \land \\
voxel\_map\,.\,values()\,.\,elems() = \bigcup\{i : 1\,..\,4 \bullet work'\,.\,getDetectorWork(i)\,.\,elems()\} \land \\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns', motions') \bullet \\
\quad (\#\, collset = 0 \land collisions' = 0) \lor (\#\, collset > 0 \land collisions' \geq (\#\, collset)\ \mathrm{div}\ 2)
\end{array}\right) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{CalcCollisions}} \\
\hline
\Xi ECPStateCDx \\
colls! : \mathbb{N} \\
\hline
colls! = collisions \\
\hline
\end{array}
$$

$StartCycle \mathrel{\widehat{=}} (next\_frame\,?\,frame\ \mathtt{@}\ t_1 \longrightarrow (RecordFrame\ ;\ CalcStep(t_1))) \blacktriangleleft INP\_DL$

$CalcStep \mathrel{\widehat{=}} \mathbf{val}\ t : TIME \bullet \mathbf{wait}\ w : 0\,..\,(FRAME\_PERIOD - OUT\_DL - t) \bullet$
$\quad \mathbf{var}\ colls : int \bullet CalcCollisions\ ;\ OutputStep(t, w, colls)$

$OutputStep \mathrel{\widehat{=}} \mathbf{val}\ t_1 : TIME;\ \mathbf{val}\ w : TIME;\ \mathbf{val}\ colls : int \bullet$
$$\left(\begin{array}{l} output\_collisions\,!\,colls\ \mathtt{@}\ t_2 \longrightarrow \\ \quad \mathbf{wait}\ FRAME\_PERIOD - (t_1 + w + t_2) \end{array}\right) \blacktriangleleft OUT\_DL\ ;\ StartCycle$$

$\bullet\ Init\ ;\ StartCycle$

end

The definitions of the process state as well as the *Init*, *RecordFrame* and *CalcCollisions* data operations are exactly as in *OBReqsCDx*. The parallelisms with *ATReqsCDx*, however, has been collapsed, giving rise to the new actions *StartCycle*, *CalcStep* and *OutputStep*. The necessary refinement steps and laws are not further discussed here, however the overall refinement procedure ought to be automatable.

5.2 Phase MH

In this section, we give a full and meticulous account of the refinement steps carried out during the MH phase. The detailed work on the parallel CD_x models revealed a finer subdivision of refinements during this phase, namely into six stages. The journal paper [6] explains the purpose of each stage in more detail.

Stages of the MH phase

1. Definition of cycle timings.
2. Decomposition of data operations that are implemented across different missions and handlers.
3. Distribution of time budgets.
4. Transformation of sequential data operations into parallel handler actions.
5. Transformation of parallel data operations into parallel handler actions.
6. Extraction of the missions and handlers.

In the remainder of the section, we discuss the elementary refinement steps for each stage.

5.2.1 Stage 1

In Stage 1 we introduce a cyclic design that embeds the overall timing requirements. Part of this is to introduce an interleaving with a **wait** statement and distribute deadlines in order to localise them to their corresponding prefixes *as much as this is possible*. In our case study, we specifically introduce an interleaving with **wait** $FRAME\_PERIOD$ and distribute deadlines on the communications on $next\_frame$ and $output\_collisions$. This stage involves a sequence of low-level refinements using distribution laws for deadlines and extraction laws for interleaving with a wait; we identify them as we go along. For readability, we occasionally highlight mathematical text in colour to emphasise which part of an action or process has been affected by a transformation.

Flatten Local Actions

We first flatten the three local actions $StartCycle$, $CalcStep$, $OutputStep$ into a single local actions which corresponds to the behaviour of the system and hence will be called $System$. The flattening facilitates the application of subsequent refinement laws; it is justified by the copy rule.

system $CDxE\_MH1 \mathrel{\widehat=}$ **begin**

...

$$System \mathrel{\widehat=} \left(\begin{array}{l} next\_frame\,?\,frame\,@\,t_1 \longrightarrow \\ \quad \left(\begin{array}{l} RecordFrame; \\ \mathbf{wait}\ w : 0\,..\,(FRAME\_PERIOD - OUT\_DL - t_1) \bullet \\ \mathbf{var}\ colls : int \bullet CalcCollisions; \\ \quad \left(\begin{array}{l} output\_collisions\,!\,colls\,@\,t_2 \longrightarrow \\ \mathbf{wait}\ FRAME\_PERIOD - (t_1 + w + t_2) \end{array}\right) \blacktriangleleft OUT\_DL; \\ \quad System \end{array}\right) \end{array}\right) \blacktriangleleft INP\_DL$$

$\bullet\ Init\ ;\ System$

end

We note that the state and data operations of the process are omitted as they are similar to the ones of $ECPCDx$, the result of the CP phase. In what follows, we merely focus on the refinement of the $System$ action and ignore the rest of the process. Process refinement is established by monotonicity laws, as usual.

Narrow Time Budgets

An objective of the refinement in this stage is to remove reference to the locally bound variables t_1 and w in order to remove the **wait** block and time prefix, and subsequently distribute the deadline on $next\_frame$. The respective subsequent refinement is facilitated by narrowing the time budget for the computation in each cycle. Here, in particular, we narrow the time budget determined by the

$$\mathbf{wait}\ w : 0\,..\,(FRAME\_PERIOD - OUT\_DL - t_1) \bullet A(t_1, w)$$

statement. The following two laws enable the respective action refinement.

***Circus* Time Law 1 (narrow-time-budget-1)**

$$\mathbf{wait}\ t_1\,..\,t_2 \sqsubseteq \mathbf{wait}\ t_1'\,..\,t_2' \ \ \textbf{provided}\ \ t_1 \leq t_1'\ \ \textbf{and}\ \ t_2' \leq t_2$$

***Circus* Time Law 2 (narrow-time-budget-2)**

$$\mathbf{wait}\ w : t_1\,..\,t_2 \bullet A \sqsubseteq \mathbf{wait}\ w : t_1'\,..\,t_2' \bullet A\ \ \textbf{provided}\ \ t_1 \leq t_1'\ \ \textbf{and}\ \ t_2' \leq t_2$$

The first law applies to simple nondeterministic waits whereas the second applies to wait blocks.

We thus perform the following refinement.

$$
\begin{array}{l}
\quad \mathit{System} \\
\sqsubseteq \text{“application of the law } \mathsf{narrow\text{-}time\text{-}budget\text{-}2} \text{ using } t_1 \leq \mathit{INP\_DL}\text{”} \\
\quad \left(\begin{array}{l}
\mathit{next\_frame}\,?\,\mathit{frame}\,@\,t_1 \longrightarrow \\
\quad \left(\begin{array}{l}
\mathit{RecordFrame}; \\
\mathbf{wait}\; w : 0\,..\,(\mathit{FRAME\_PERIOD} - \mathit{OUT\_DL} - \mathit{INP\_DL}) \bullet \\
\mathbf{var}\; \mathit{colls} : \mathit{int} \bullet \mathit{CalcCollisions}; \\
\quad \left(\begin{array}{l}
\mathit{output\_collisions}\,!\,\mathit{colls}\,@\,t_2 \longrightarrow \\
\mathbf{wait}\; \mathit{FRAME\_PERIOD} - (t_1 + w + t_2)
\end{array}\right) \blacktriangleleft \mathit{OUT\_DL}; \\
\quad \mathit{System}
\end{array}\right)
\end{array}\right) \blacktriangleleft \mathit{INP\_DL}
\end{array}
$$

To apply the law, we require a local (contextual) assumption $t_1 \leq \mathit{INP\_DL}$. Suitable opening and closing rules for the window inference mechanism that realises action refinement introduce assumptions like the above. To give an example, constructs of the form $(c\,@\,t \longrightarrow A(t)) \blacktriangleleft d$ are expected to have special opening rules that introduce the contextual assumption $t \leq d$ when shifting the focus to $A(t)$. We see that the introduction of contextual assumptions during refinement is mostly a technical issue; we will not further discuss it here but point to the literature on window inference [] and mechanised *Circus* refinement [11].

Introduce Interleaving for Cycle Time

In this step, the aim is to replace the inner $\mathbf{wait}\; \mathit{FRAME\_PERIOD} - (t_1 + w + t_2)$ with an outer interleaving with $\mathbf{wait}\; \mathit{FRAME\_PERIOD}$. This interleaving was already present in the abstract $\mathit{ATReqsCDx}$ process, however, has been removed during the CP phase. To achieve this we require the following law.

Circus **Time Law 3 (time-prefix-elim)**

$$(c\,@\,t \longrightarrow \mathbf{wait}\; t_1 - t) \blacktriangleleft d \equiv ((c \longrightarrow \mathbf{skip}) \blacktriangleleft d) \interleave \mathbf{wait}\; t_1 \quad \mathbf{provided} \quad d \leq t_1$$

It yields the refinement given next.

$$
\begin{array}{l}
\ldots \equiv \text{“application of the law } \mathsf{time\text{-}prefix\text{-}elim} \text{ using } \mathit{OUT\_DL} \leq \mathit{FRAME\_PERIOD} - t_1 - w\text{”} \\
\quad \left(\begin{array}{l}
\mathit{next\_frame}\,?\,\mathit{frame}\,@\,t_1 \longrightarrow \\
\quad \left(\begin{array}{l}
\mathit{RecordFrame}; \\
\mathbf{wait}\; w : 0\,..\,(\mathit{FRAME\_PERIOD} - \mathit{INP\_DL} - \mathit{OUT\_DL}) \bullet \\
\mathbf{var}\; \mathit{colls} : \mathit{int} \bullet \mathit{CalcCollisions}; \\
\quad \left(\begin{array}{l}
(\mathit{output\_collisions}\,!\,\mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT\_DL} \\
\interleave \mathbf{wait}\; \mathit{FRAME\_PERIOD} - (t_1 + w)
\end{array}\right); \\
\quad \mathit{System}
\end{array}\right)
\end{array}\right) \blacktriangleleft \mathit{INP\_DL}
\end{array}
$$

The proviso $\mathit{OUT\_DL} \leq \mathit{FRAME\_PERIOD} - t_1 - w$ is discharged by the contextual assumptions

(1) $t_1 \leq \mathit{INP\_DL}$ and

(2) $w \leq \mathit{FRAME\_PERIOD} - \mathit{INP\_DL} - \mathit{OUT\_DL}$

using elementary laws of linear arithmetics. This relies on another opening rule for window inference that applies to actions of the form $\mathbf{wait}\; w : t_0\,..\,t_1 \bullet A(w)$ and introduces the local assumption $t_0 \leq w \leq t_1$.

Extract Interleaving for Cycle Time

We next extract the interleaving with $\mathbf{wait}\; \mathit{FRAME\_PERIOD} - (t_1 + w)$ to the outer level. This requires a number of extraction laws for interleaving with a basic $\mathbf{wait}\; t$ statement. Below we present them.

Circus **Time Law 4 (extract-inter-wait-seq)**

$Op\ ;\ (A \interleave \mathbf{wait}\ t) \equiv (Op\ ;\ A) \interleave \mathbf{wait}\ t$

provided Op is a data operation and $wrtV(Op) \cap FV(t) = \varnothing$

Circus **Time Law 5 (extract-inter-wait-var)**

$\mathbf{var}\ x : T \bullet (A \interleave \mathbf{wait}\ t) \equiv (\mathbf{var}\ x : T \bullet A) \interleave \mathbf{wait}\ t$

provided $x \notin FV(t)$

Circus **Time Law 6 (extract-inter-wait-waitblock)**

$\mathbf{wait}\ w : t_1 \mathinner{..} t_2 \bullet (A(w) \interleave \mathbf{wait}\ t - w) \equiv (\mathbf{wait}\ w : t_1 \mathinner{..} t_2 \bullet A(w)) \interleave \mathbf{wait}$

provided $t_2 \leq t$

Circus **Time Law 7 (extract-inter-wait-prefix)**

$(c\ @\ t \longrightarrow (A(t) \interleave \mathbf{wait}\ (t_1 - t))) \blacktriangleleft d \equiv ((c\ @\ t \longrightarrow A(t)) \blacktriangleleft d) \interleave \mathbf{wait}\ t_1$

provided $d \leq t_1$

The laws allow us to proceed with the refinement as follows.

$\ldots \equiv$ "application of extraction laws for interleaving"

$$\left(\begin{array}{l}\left(\begin{array}{l} next\_frame\ ?\ frame\ @\ t_1 \longrightarrow \\ \quad \left(\begin{array}{l} RecordFrame; \\ \mathbf{wait}\ w : 0 \mathinner{..} (FRAME\_PERIOD - INP\_DL - OUT\_DL) \bullet \\ \mathbf{var}\ colls : int \bullet CalcCollisions; \\ \quad (output\_collisions\ !\ colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL; \\ \quad System \end{array}\right) \end{array}\right) \blacktriangleleft INP\_DL \\ \color{red}{\interleave \mathbf{wait}\ FRAME\_PERIOD} \end{array}\right)$$

We omitted the detailed refinement steps for this transformation; they are straight-forward.

Remove Unused Time Variables

We observe that the local constants t_1 and w, introduced by the time prefix and **wait** block, are not referenced anymore. The two laws below justify their removal.

Circus **Time Law 8 (remove-unused-time-prefix)**

$c\ @\ t \longrightarrow A \equiv c \longrightarrow A$ **provided** $t \notin FV(A)$

Circus **Time Law 9 (remove-unused-wait-block)**

$\mathbf{wait}\ w : T \bullet A \equiv \mathbf{wait}\ T\ ;\ A$ **provided** $w \notin FV(A)$

Application of the above laws yields the following simplified action.

$\ldots \equiv$ "application of the laws remove-unused-time-prefix and remove-unused-wait-block"

$$\left(\begin{array}{l}\left(\begin{array}{l} next\_frame\ ?\ frame \longrightarrow \\ \quad \left(\begin{array}{l} RecordFrame; \\ \mathbf{wait}\ 0 \mathinner{..} (FRAME\_PERIOD - INP\_DL - OUT\_DL) \bullet \\ \mathbf{var}\ colls : int \bullet CalcCollisions; \\ \quad (output\_collisions\ !\ colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL; \\ \quad System \end{array}\right) \end{array}\right) \blacktriangleleft INP\_DL \\ \interleave \mathbf{wait}\ FRAME\_PERIOD \end{array}\right)$$

The removal of the locally bound t_1 and w was essential in order to distribute the outer synchronisation deadline on $next\_frame$. This is done in the last sub-step of Stage 1.

Distribute Deadlines

The primary law we use in this sub-step is

Circus **Time Law 10 (distr-sync-deadline-seq)**

$$(c \longrightarrow (A_1 \ ; \ A_2)) \blacktriangleleft d \equiv ((c \longrightarrow A_1) \blacktriangleleft d) \ ; \ A_2$$

We also require a basic law that distributes a prefix over a sequence.

Circus **Law 1 (distr-prefix-seq)**

$$c \longrightarrow (A_1 \ ; \ A_2) \equiv (c \longrightarrow A_1) \ ; \ A_2$$

The application of the two laws in sequence yields

$\ldots \equiv$ "application of the law distr-prefix-seq"

$$\left(\begin{array}{l} \left(\begin{array}{l} (next\_frame \, ? \, frame \longrightarrow RecordFrame); \\ \mathbf{wait}\ 0 \mathrel{..} (FRAME\_PERIOD - INP\_DL - OUT\_DL); \\ \mathbf{var}\ colls : int \bullet CalcCollisions; \\ \quad (output\_collisions \, ! \, colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL; \\ \quad System \end{array}\right) \blacktriangleleft INP\_DL \\ \interleave \mathbf{wait}\ FRAME\_PERIOD \end{array}\right)$$

$\ldots \equiv$ "application of the law distr-sync-deadline-seq"

$$\left(\begin{array}{l} \left(\begin{array}{l} (next\_frame \, ? \, frame \longrightarrow RecordFrame) \blacktriangleleft INP\_DL; \\ \mathbf{wait}\ 0 \mathrel{..} (FRAME\_PERIOD - INP\_DL - OUT\_DL); \\ \mathbf{var}\ colls : int \bullet CalcCollisions; \\ \quad (output\_collisions \, ! \, colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL; \\ \quad System \end{array}\right) \\ \interleave \mathbf{wait}\ FRAME\_PERIOD \end{array}\right)$$

In a final sub-step, we extract the sequence with *System* from the inner block. This uses associativity of sequential composition as well as a basic law to extract the sequence with *System* from the variable declaration. These laws shall not be presented here but can be found in Appendix B.1.

$\ldots \sqsubseteq$ "application of associativity and distribution laws to extract sequence with *System*"

$$\left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} (next\_frame \, ? \, frame \longrightarrow RecordFrame) \blacktriangleleft INP\_DL; \\ \mathbf{wait}\ 0 \mathrel{..} (FRAME\_PERIOD - INP\_DL - OUT\_DL); \\ \mathbf{var}\ colls : int \bullet CalcCollisions; \\ \quad (output\_collisions \, ! \, colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL \end{array}\right) \\ \interleave \mathbf{wait}\ FRAME\_PERIOD \end{array}\right); \\ System \end{array}\right)$$

The above action concludes Stage 1 of the MH phase. All deadlines have been localised to the corresponding synchronisations. Besides, we have narrowed the time budget and introduced the cycle time. We notice, however, that the time budget is still captures by a single **wait**; it is decomposed later on in Stage 3.

5.2.2 Stage 2

In this stage we decompose the data operations to match them to the design of missions and handlers. Here, this is, in particular, the *RecordFrame* data operation. The decomposition is performed in two separate refinements which we present in the sequel.

Refinement 1

The first refinement decomposes the *RecordFrame* into three sequential data operations. Specifically, they are *StoreFrame*, *ReduceAndPartitionWork* and *DetectCollisions*.

system *CDxE_MH2A* $\widehat{=}$ **begin**

state *CDxMH2AState* == *ECPStateCDx*

Init

$CDxMH2AState'$

$$
\begin{array}{l}
currentFrame' = \mathbf{new}\ RawFrame\\
state' = \mathbf{new}\ StateTable\\
voxel\_map' = \mathbf{new}\ HashMap[Vector2d, List[Motion]]\\
work' = \mathbf{new}\ Partition(4)\\
collisions' = 0
\end{array}
$$

StoreFrame

$\Delta CDxMH2AState$
$frame? : Frame$

$$
\begin{array}{l}
\exists\, posns, posns' : Frame;\ motions, motions' : Frame \mid\\
\quad \mathrm{dom}\, posns = \mathrm{dom}\, motions \land \mathrm{dom}\, posns' = \mathrm{dom}\, motions' \bullet\\
\left(\begin{array}{l}
posns' = frame? \land\\
motions' =\\
\quad \{a : \mathrm{dom}\, posns' \bullet a \mapsto \mathbf{if}\ a \in \mathrm{dom}\, posns\ \mathbf{then}\ (posns'\ a) -_V (posns\ a)\ \mathbf{else}\ ZeroV\} \land\\
posns = F(currentFrame) \land motions = G(currentFrame, state) \land\\
posns' = F(currentFrame') \land motions' = G(currentFrame', state')
\end{array}\right)
\end{array}
$$

ReduceAndPartitionWork

$\Delta CDxMH2AState$

$$
\begin{array}{l}
currentFrame' = currentFrame \land state' = state\\
\exists\, posns : Frame;\ motions : Frame \mid \mathrm{dom}\, posns = \mathrm{dom}\, motions \bullet\\
\left(\begin{array}{l}
posns = F(currentFrame) \land motions = G(currentFrame, state) \land\\
\left(\begin{array}{l}
\forall\, a_1, a_2 : Aircraft \mid \{a_1, a_2\} \subseteq \mathrm{dom}\, posns \bullet\\
\quad (a_1, a_2) \in CalcCollisionSet(posns, motions) \Rightarrow\\
\qquad \left(\begin{array}{l}
\exists\, l : List[Motion] \mid l \in voxel\_map' . values() . elems() \bullet\\
\quad MkMotion(a_1, posns\ a_1 -_V motions\ a_1, posns\ a_1) \in l . elems() \land\\
\quad MkMotion(a_2, posns\ a_2 -_V motions\ a_2, posns\ a_2) \in l . elems()
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{array}
$$

DetectCollisions

$\Delta CDxMH2AState$

$$
\begin{array}{l}
currentFrame' = currentFrame \land state' = state \land voxel\_map' = voxel\_map \land work' = work\\
\exists\, posns : Frame;\ motions : Frame \mid \mathrm{dom}\, posns = \mathrm{dom}\, motions \bullet\\
\left(\begin{array}{l}
posns = F(currentFrame) \land motions = G(currentFrame, state) \land\\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns, motions) \bullet\\
\quad (\#\, collset = 0 \land collisions' = 0) \lor (\#\, collset > 0 \land collisions' \geq (\#\, collset)\ \mathrm{div}\ 2)
\end{array}\right)
\end{array}
$$

Data Operation	**Handler**	**Instances**	**Execution**
StoreFrame	`InputFrameHandler`	1	sequential
ReduceAndPartitionWork	`ReducerHandler`	1	sequential
DetectCollisions	`DetectorHandler`	4	parallel

Table 2: Mapping of data operations in $CDxE\_MH1A$ to handlers in the program.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{CalcCollisions}} \\
\hline
\Xi\, CDxMH2AState \\
colls! : \mathbb{N} \\
\hline
\exists\, posns : Frame;\ motions : Frame \mid \operatorname{dom} posns = \operatorname{dom} motions \bullet \\
\left(\begin{array}{l}
posns = F(currentFrame) \land motions = G(currentFrame, state) \land \\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns, motions) \bullet \\
\quad (\#\, collset = 0 \land colls! = 0) \lor (\#\, collset > 0 \land colls! \geq (\#\, collset) \operatorname{div} 2)
\end{array}\right) \\
\hline
\end{array}
$$

$$
\begin{array}{l}
System \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
(next\_frame\,?\,frame \longrightarrow StoreFrame) \blacktriangleleft INP\_DL; \\
ReduceAndPartitionWork; \\
DetectCollisions; \\
\mathbf{wait}\ 0 \mathinner{..} (FRAME\_PERIOD - INP\_DL - OUT\_DL); \\
\mathbf{var}\ colls : int \bullet CalcCollisions; \\
\quad (output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL
\end{array}\right) \\
\interleave \mathbf{wait}\ FRAME\_PERIOD
\end{array}\right); \\
System
\end{array}\right)
\end{array}
$$

$\bullet$ *Init* ; *System*

end

We observe that the *RecordFrame* data operation has been removed from the model: its behaviour is now realised by the sequence of *StoreFrame*, *ReduceAndPartitionWork* and *DetectCollisions* in *StartCycle*. The decomposition of *RecordFrame* is necessary for the subsequent design that splits its behaviour between the handlers of the mission. Table 2 summarises which handler(s) of the application caters for which data operation in the model. We note that *DetectCollisions* is implemented by four handler instances.

In addition to refining data operations, we also carry out a minor action refinement of the *System* action in the above process. After decomposition, the following fragment emerges in the *System* action.

$$
\left(\begin{array}{l}
next\_frame\,?\,frame \longrightarrow \\
\quad \left(\begin{array}{l}
StoreFrame; \\
ReduceAndPartitionWork; \\
DetectCollisions
\end{array}\right)
\end{array}\right) \blacktriangleleft INP\_DL
$$

The action refinement binds the prefix to the *StoreFrame* data operation and distributes the deadline through the sequence. This is similar to the refinement in the last sub-step of Stage 1, using exactly the same laws.

$\ldots \equiv$ "application of the laws distr-prefix-seq and distr-sync-deadline-seq"

$$
\left(\begin{array}{l}
(next\_frame\,?\,frame \longrightarrow StoreFrame) \blacktriangleleft INP\_DL; \\
ReduceAndPartitionWork; \\
DetectCollisions
\end{array}\right)
$$

In general, the decomposition of *RecordFrame* essentially extracts conjuncts of the schema predicate that modify the value of specific variables. In principle, this can be automated, subject to guidance by the user.

Refinement 2

The second refinement further decomposes the *DetectCollisions* data operation into a conjunction. This is desirable because the behaviour of this operation will later be implemented by parallel handlers.

$$\textbf{system}\ \mathit{CDxE\_MH2B} \mathrel{\widehat{=}} \textbf{begin}$$

$$\textbf{state}\ \mathit{CDxMH2BState} == \mathit{ECPStateCDx}$$

Init

$$\mathit{CDxMH2BState}'$$

$$\begin{array}{l}
\mathit{currentFrame}' = \textbf{new}\ \mathit{RawFrame} \\
\mathit{state}' = \textbf{new}\ \mathit{StateTable} \\
\mathit{voxel_map}' = \textbf{new}\ \mathit{HashMap}[\mathit{Vector2d}, \mathit{List}[\mathit{Motion}]] \\
\mathit{work}' = \textbf{new}\ \mathit{Partition}(4) \\
\mathit{collisions}' = 0
\end{array}$$

StoreFrame

$$\begin{array}{l}
\Delta \mathit{CDxMH2BState} \\
\mathit{frame}? : \mathit{Frame}
\end{array}$$

$$\begin{array}{l}
\exists\, \mathit{posns}, \mathit{posns}' : \mathit{Frame};\ \mathit{motions}, \mathit{motions}' : \mathit{Frame} \mid \\
\quad \operatorname{dom} \mathit{posns} = \operatorname{dom} \mathit{motions} \land \operatorname{dom} \mathit{posns}' = \operatorname{dom} \mathit{motions}' \bullet \\
\left(\begin{array}{l}
\mathit{posns}' = \mathit{frame}? \land \\
\mathit{motions}' = \\
\quad \{a : \operatorname{dom} \mathit{posns}' \bullet a \mapsto \textbf{if}\ a \in \operatorname{dom} \mathit{posns}\ \textbf{then}\ (\mathit{posns}'\ a) -_V (\mathit{posns}\ a)\ \textbf{else}\ \mathit{ZeroV}\} \land \\
\mathit{posns} = F(\mathit{currentFrame}) \land \mathit{motions} = G(\mathit{currentFrame}, \mathit{state}) \land \\
\mathit{posns}' = F(\mathit{currentFrame}') \land \mathit{motions}' = G(\mathit{currentFrame}', \mathit{state}')
\end{array}\right)
\end{array}$$

ReduceAndPartitionWork

$$\Delta \mathit{CDxMH2BState}$$

$$\begin{array}{l}
\mathit{currentFrame}' = \mathit{currentFrame} \land \mathit{state}' = \mathit{state} \\
\exists\, \mathit{posns} : \mathit{Frame};\ \mathit{motions} : \mathit{Frame} \mid \operatorname{dom} \mathit{posns} = \operatorname{dom} \mathit{motions} \bullet \\
\left(\begin{array}{l}
\mathit{posns} = F(\mathit{currentFrame}) \land \mathit{motions} = G(\mathit{currentFrame}, \mathit{state}) \land \\
\left(\begin{array}{l}
\forall\, a_1, a_2 : \mathit{Aircraft} \mid \{a_1, a_2\} \subseteq \operatorname{dom} \mathit{posns} \bullet \\
\quad (a_1, a_2) \in \mathit{CalcCollisionSet}(\mathit{posns}, \mathit{motions}) \Rightarrow \\
\qquad \left(\begin{array}{l}
\exists\, l : \mathit{List}[\mathit{Motion}] \mid l \in \mathit{voxel_map}' . \mathit{values}() . \mathit{elems}() \bullet \\
\quad \mathit{MkMotion}(a_1, \mathit{posns}\ a_1 -_V \mathit{motions}\ a_1, \mathit{posns}\ a_1) \in l . \mathit{elems}() \land \\
\quad \mathit{MkMotion}(a_2, \mathit{posns}\ a_2 -_V \mathit{motions}\ a_2, \mathit{posns}\ a_2) \in l . \mathit{elems}()
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{array}$$

CalcPartCollisions

$$\begin{array}{l}
\Xi \mathit{CDxMH2BState} \\
\mathit{pcolls}! : \mathit{int} \\
i? : 1 \mathrel{..} 4
\end{array}$$

$$\begin{array}{l}
\mathit{pcolls}! = \\
\quad \# \left\{\begin{array}{l}
a_1 : \mathit{Aircraft};\ a_2 : \mathit{Aircraft} \mid \\
\quad \left(\begin{array}{l}
\exists\, l : \mathit{List}[\mathit{Motion}] \mid l \in \mathit{work} . \mathit{getDetectorWork}(i?) . \mathit{elems}() \bullet \\
\exists\, v_1, v_2 : \mathit{Vector};\ w_1, w_2 : \mathit{Vector} \bullet \\
\quad \mathit{MkMotion}(a_1, v_1, w_1) \in l . \mathit{elems}() \land \\
\quad \mathit{MkMotion}(a_2, v_2, w_2) \in l . \mathit{elems}() \land \\
\quad \mathit{collide}((v_1, w_1 -_V v_1), (v_2, w_2 -_V v_2))
\end{array}\right)
\end{array}\right\} \operatorname{div} 2
\end{array}$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad SetCollisionsFromParts \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}}\\
\Delta CDxMH2BState \\
collsbag? : \mathrm{bag}\, int \\
\hline
currentFrame' = currentFrame \land state' = state \\
voxel\_map' = voxel\_map \land work' = work \\
\exists\, s : \mathrm{seq}\, int \mid s = items\ collsbag? \bullet collisions' = \Sigma\, s \\
\hline
\end{array}
$$

$$
\begin{array}{l}
DetectCollisions \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\quad \left(\begin{array}{l}
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4)
\end{array}\right); \\
\quad SetCollisionsFromParts(\llbracket\, colls1, colls2, colls3, colls4 \,\rrbracket)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad CalcCollisions \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}}\\
\Xi CDxMH2BState \\
colls! : \mathbb{N} \\
\hline
\exists\, posns : Frame;\ motions : Frame \mid \mathrm{dom}\, posns = \mathrm{dom}\, motions \bullet \\
\left(\begin{array}{l}
posns = F(currentFrame) \land motions = G(currentFrame, state) \land \\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns, motions) \bullet \\
\quad (\#\, collset = 0 \land colls! = 0) \lor (\#\, collset > 0 \land colls! \geq (\#\, collset)\ \mathrm{div}\ 2)
\end{array}\right) \\
\hline
\end{array}
$$

$$
\begin{array}{l}
System \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\left(\begin{array}{l}
\left(\begin{array}{l}
(next\_frame\,?\,frame \longrightarrow StoreFrame) \blacktriangleleft INP\_DL; \\
ReduceAndPartitionWork; \\
DetectCollisions; \\
\mathbf{wait}\, 0\,..\,(FRAME\_PERIOD - INP\_DL - OUT\_DL); \\
\mathbf{var}\ colls : int \bullet CalcCollisions; \\
\quad (output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL
\end{array}\right) \\
\interleave \mathbf{wait}\ FRAME\_PERIOD
\end{array}\right)\right); \\
System
\end{array}\right)
\end{array}
$$

$\bullet$ *Init* ; *System*

end

Two data operations *CalcPartCollisions* and *SetCollisionsFromParts* have been introduced in this refinement. The first one calculates the collisions result for a particular partition of the subdivided work. The second one merges all partial results (this is just adding them together). The merge operation is specified in terms of a bag to emphasise that the order in which the results are computed and merged is immaterial. This will later on be exploited when parallelising *DetectCollisions* at the level of actions.

We note, however, that the behaviour of the detector handlers is not fully parallelised yet in this model. In particular, the effect of *SetCollisionsFromParts* has to be distributed into the handlers. This cannot be done at the level of data operations though due to the absence of sharing.

This model concludes Stage 2 of the MH phase. All refinement at the level of data operations is completed here and subsequent stages focus on the refinement of actions. In general, decomposition of data operations is a non-trivial design task; automation through tools may be envisaged for particular patterns.

Time Budget	Respective Data Operation
SF_{TB}	*StoreFrame*
RPW_{TB}	*ReduceAndPartitionWork*
DC_{TB}	*DetectCollisions*
CC_{TB}	*CalcCollisions*

Table 3: Time budgets introduced for the data operations.

5.2.3 Stage 3

In this stage we distribute time budgets between data operations. This involves two sub-steps.

1. Decompose nondeterministic **wait** statements for time budgets where appropriate.
2. Move decomposed time budgets to the respective data operation.

The decomposition in sub-step (1) is effectively achieved by the following two laws.

Circus **Time Law 11 (split-time-budget-1)**

$$\mathbf{wait}\ 0 \mathinner{..} t \equiv \mathbf{wait}\ 0 \mathinner{..} t_1\ ;\ \mathbf{wait}\ 0 \mathinner{..} t_2\ \ \mathbf{provided}\ \ t = t_1 + t_2$$

Circus **Time Law 12 (split-time-budget-2)**

$$\mathbf{wait}\ 0 \mathinner{..} t \sqsubseteq \mathbf{wait}\ 0 \mathinner{..} t_1\ ;\ \mathbf{wait}\ 0 \mathinner{..} t_2\ \ \mathbf{provided}\ \ t_1 + t_2 \leq t$$

In our example, we use multiple applications of the second law.

$$\ldots \sqsubseteq \text{“multiple applications of law split-time-budget-2”}$$

$$\left(\begin{array}{l}\left(\begin{array}{l}\left(\begin{array}{l}(next\_frame\,?\,frame \longrightarrow StoreFrame) \blacktriangleleft INP\_DL;\\ ReduceAndPartitionWork;\\ DetectCollisions;\\ \mathbf{wait}\ 0 \mathinner{..} SF_{TB};\\ \mathbf{wait}\ 0 \mathinner{..} RPW_{TB};\\ \mathbf{wait}\ 0 \mathinner{..} DC_{TB};\\ \mathbf{wait}\ 0 \mathinner{..} CC_{TB};\\ \mathbf{var}\ colls : int \bullet CalcCollisions;\\ \quad (output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL\end{array}\right)\\ ||| \ \mathbf{wait}\ FRAME\_PERIOD\end{array}\right);\\ System\end{array}\right)$$

Above, the axiomatic constants SF_{TB}, RPW_{TB}, DC_{TB} and CC_{TB} have been introduced to determine the time budgets for individual data operations. Table 3 summarises the relationship between these constants and the corresponding data operation of the *Circus* process.
At this point, it is in fact not necessary to precisely specify the values of the time budgets. However, to discharge the proviso of the above refinement, we require at least the following property.

$$SF_{TB} + RPW_{TB} + DC_{TB} + CC_{TB} \leq FRAME\_PERIOD - INP\_DL - OUT\_DL$$

We therefore introduce the time budgets axiomatically as follows. This assumes $TIME : \mathbb{P}\,(\mathbb{A})$.

$$\begin{array}{|l}SF_{TB} : TIME\\ RPW_{TB} : TIME\\ DC_{TB} : TIME\\ CC_{TB} : TIME\\ \hline SF_{TB} + RPW_{TB} + DC_{TB} + CC_{TB} \leq FRAME\_PERIOD - INP\_DL - OUT\_DL\end{array}$$

In sub-step (2), we move the operation-specific time budgets to the data operations they apply to.This uses associativity of sequential composition $(A_1\ ;\ A_2)\ ;\ A_3 \equiv A_1\ ;\ (A_2\ ;\ A_3)$, elementary distribution theorems for sequencing, and the following commutativity law for a time budget and data operation.

Circus **Time Law 13 (time-budget-op-comm)**

$$P(Op\ ;\ \textbf{wait}\ t_1 \mathinner{..} t_2) \equiv P(\textbf{wait}\ t_1 \mathinner{..} t_2\ ;\ Op)\ \ \textbf{provided}\ \ Op \text{ is a data operation}$$

This law is in fact non-compositional: it is a law about processes rather than actions. Hence, it only holds if the underlying action $Op\ ;\ \textbf{wait}\ t_1 \mathinner{..} t_2$ is embedded in a process P. The justification for the law comes from the structure and semantics of processes that prevents one from observing the precise time at which an (internal) state change takes place. It is proved by induction over the structure of processes.

Using multiple and symmetric applications of the previous law, we proceed to obtain

$\ldots \sqsubseteq$ "multiple applications of law time-budget-op-comm and elementary laws"

$$\left(\begin{array}{l}\left(\begin{array}{l}\left(\begin{array}{l}(next\_frame\,?\,frame \longrightarrow \textbf{wait}\ 0 \mathinner{..} SF_{TB}\ ;\ StoreFrame) \blacktriangleleft INP\_DL;\\ \textbf{wait}\ 0 \mathinner{..} RPW_{TB}\ ;\ ReduceAndPartitionWork;\\ \textbf{wait}\ 0 \mathinner{..} DC_{TB}\ ;\ DetectCollisions;\\ \textbf{var}\ colls : int \bullet \textbf{wait}\ 0 \mathinner{..} CC_{TB}\ ;\ CalcCollisions;\\ \quad (output\_collisions\,!\,colls \longrightarrow \textbf{skip}) \blacktriangleleft OUT\_DL\end{array}\right)\\ |||\ \textbf{wait}\ FRAME\_PERIOD\end{array}\right)\ ;\\ System\end{array}\right)$$

Once again, we omit the details of the elementary refinement steps (the laws used are in Appendix B.1). This concludes Stage 3 since all data operations are now equipped with an operation-specific time budget.

5.2.4 Stage 4

This stage addresses the parallelisation of sequential data operations into parallel handler actions. Generally, this takes advantage of the following two laws.

Circus **Law 2 (seq-to-par-1)**

$$A_1 \; ; \; A_2 \equiv ((A_1 \; ; \; c \longrightarrow \mathbf{skip}) \llbracket \mathit{wrtV}(A_1) \mid \{\!| c |\!\} \mid \mathit{wrtV}(A_2) \rrbracket (c \longrightarrow A_2)) \setminus \{\!| c |\!\}$$

$$\textbf{provided} \;\; \mathit{wrtV}(A_1) \cap \mathit{wrtV}(A_2) = \varnothing \;\; \textbf{and} \;\; \mathit{wrtV}(A_1) \cap \mathit{usedV}(A_2) = \varnothing \;\; \textbf{and}$$

$$c \notin \mathit{usedC}(A_1) \cup \mathit{usedC}(A_2)$$

Circus **Law 3 (seq-to-par-2)**

$$A_1 \; ; \; A_2 \equiv ((A_1 \; ; \; c \,!\, x \longrightarrow \mathbf{skip}) \llbracket \mathit{wrtV}(A_1) \mid \{\!| c |\!\} \mid \mathit{wrtV}(A_2) \rrbracket (c \,?\, x \longrightarrow A_2)) \setminus \{\!| c |\!\}$$

$$\textbf{provided} \;\; \mathit{wrtV}(A_1) \cap \mathit{wrtV}(A_2) = \varnothing \;\; \textbf{and} \;\; \mathit{wrtV}(A_1) \cap \mathit{usedV}(A_2) = \{x\} \;\; \textbf{and}$$

$$c \notin \mathit{usedC}(A_1) \cup \mathit{usedC}(A_2)$$

The first law is applicable when no shared data is calculated and passed between the sequential actions, or in other words, the first action A_1 does not write data that the second action A_2 reads. If there is such data, the second law has to be applied. We note that in seq-to-par-1, the new channel c is typeless and we can think of it purely in terms of establishing control of execution. In seq-to-par-2, the new channel c is typed according to the shared data that is passed between the sequential actions. It thus fulfils the dual purpose of exercising control and providing a means for communicating shared data through the parallelism.

In our example, we apply seq-to-par-2 three times to fully parallelise the sequence of data operations. The law seq-to-par-1 is not used, although we do require it later on in the refinement for the SH phase. The three applications of seq-to-par-2 are interleaved with auxiliary refinement steps that distribute input prefixes and extract the hiding of the new channels. Below, we highlight the focus of the action refinement in this stage.

$$\begin{array}{l}
\mathit{System} \mathrel{\widehat=} \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
(\mathit{next_frame} \,?\, \mathit{frame} \longrightarrow (\mathbf{wait}\, 0 \mathrel{..} \mathit{SF}_{TB} \; ; \; \mathit{StoreFrame})) \blacktriangleleft \mathit{INP_DL}; \\
\mathbf{wait}\, 0 \mathrel{..} \mathit{RPW}_{TB} \; ; \; \mathit{ReduceAndPartitionWork}; \\
\mathbf{wait}\, 0 \mathrel{..} \mathit{DC}_{TB} \; ; \; \mathit{DetectCollisions}; \\
\mathbf{var}\; \mathit{colls} : \mathit{int} \bullet \mathbf{wait}\, 0 \mathrel{..} \mathit{CC}_{TB} \; ; \; \mathit{CalcCollisions}; \\
(\mathit{output_collisions} \,!\, \mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT_DL}
\end{array}\right) \\
\interleave \mathbf{wait}\; \mathit{FRAME_PERIOD}
\end{array}\right) ; \\
\mathit{System}
\end{array}\right)
\end{array}$$

We proceed by refining the highlighted action as follows.

$$\left(\begin{array}{l}
(\mathit{next_frame} \,?\, \mathit{frame} \longrightarrow (\mathbf{wait}\, 0 \mathrel{..} \mathit{SF}_{TB} \; ; \; \mathit{StoreFrame})) \blacktriangleleft \mathit{INP_DL}; \\
\mathbf{wait}\, 0 \mathrel{..} \mathit{RPW}_{TB} \; ; \; \mathit{ReduceAndPartitionWork}; \\
\mathbf{wait}\, 0 \mathrel{..} \mathit{DC}_{TB} \; ; \; \mathit{DetectCollisions}; \\
\mathbf{var}\; \mathit{colls} : \mathit{int} \bullet \mathbf{wait}\, 0 \mathrel{..} \mathit{CC}_{TB} \; ; \; \mathit{CalcCollisions}; \\
(\mathit{output_collisions} \,!\, \mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT_DL}
\end{array}\right)$$

$\equiv$ "application of the law seq-to-par-2 introducing a channel *reduce* of type $\mathit{RawFrame} \times \mathit{StateTable}$"

$$\left(\begin{array}{l}
\left(\begin{array}{l}
(\mathit{next_frame} \,?\, \mathit{frame} \longrightarrow (\mathbf{wait}\, 0 \mathrel{..} \mathit{SF}_{TB} \; ; \; \mathit{StoreFrame})) \blacktriangleleft \mathit{INP_DL}; \\
\mathit{reduce} \,!\, \mathit{currentFrame} \,!\, \mathit{state} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad \llbracket \{\mathit{currentFrame}, \mathit{state}\} \mid \{\!| \mathit{reduce} |\!\} \mid \{\mathit{voxel_map}, \mathit{work}, \mathit{collisions}\} \rrbracket \\
\left(\begin{array}{l}
\mathit{reduce} \,?\, \mathit{currentFrame} \,?\, \mathit{state} \longrightarrow \\
\left(\begin{array}{l}
\mathbf{wait}\, 0 \mathrel{..} \mathit{RPW}_{TB} \; ; \; \mathit{ReduceAndPartitionWork}; \\
\mathbf{wait}\, 0 \mathrel{..} \mathit{DC}_{TB} \; ; \; \mathit{DetectCollisions}; \\
\mathbf{var}\; \mathit{colls} : \mathit{int} \bullet \mathbf{wait}\, 0 \mathrel{..} \mathit{CC}_{TB} \; ; \; \mathit{CalcCollisions}; \\
(\mathit{output_collisions} \,!\, \mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT_DL}
\end{array}\right)
\end{array}\right)
\end{array}\right) \setminus \{\!| \mathit{reduce} |\!\}$$

$\equiv$ "distribution of input prefix $reduce\,?\,currentFrame\,?\,state \longrightarrow \ldots$ using elementary laws"

$$\left(\begin{array}{l}
\left(\begin{array}{l}(next_frame\,?\,frame \longrightarrow (\mathbf{wait}\ 0\,..\,SF_{TB}\ ;\ StoreFrame)) \blacktriangleleft INP_DL;\\ reduce\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}\end{array}\right)\\
\quad \llbracket \{currentFrame, state\} \mid \{\!|\, reduce\,|\!\} \mid \{voxel_map, work, collisions\} \rrbracket\\
\left(\begin{array}{l}
\left(\begin{array}{l} reduce\,?\,currentFrame\,?\,state \longrightarrow \\ \mathbf{wait}\ 0\,..\,RPW_{TB}\ ;\ ReduceAndPartitionWork\end{array}\right);\\
\left(\begin{array}{l}\mathbf{wait}\ 0\,..\,DC_{TB}\ ;\ DetectCollisions;\\ \mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0\,..\,CC_{TB}\ ;\ CalcCollisions;\\ (output_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT_DL\end{array}\right)
\end{array}\right)
\end{array}\right) \setminus \{\!|\, reduce\,|\!\}$$

$\equiv$ "application of the law seq-to-par-2 introducing a channel $detect$ of type $Partition$"

$$\left(\begin{array}{l}
\left(\begin{array}{l}(next_frame\,?\,frame \longrightarrow (\mathbf{wait}\ 0\,..\,SF_{TB}\ ;\ StoreFrame)) \blacktriangleleft INP_DL;\\ reduce\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}\end{array}\right)\\
\quad \llbracket \{currentFrame, state\} \mid \{\!|\, reduce\,|\!\} \mid \{voxel_map, work, collisions\} \rrbracket\\
\left(\begin{array}{l}
\left(\begin{array}{l} reduce\,?\,currentFrame\,?\,state \longrightarrow \\ \mathbf{wait}\ 0\,..\,RPW_{TB}\ ;\ ReduceAndPartitionWork;\\ detect\,!\,work \longrightarrow \mathbf{skip}\end{array}\right)\\
\quad \llbracket \{voxel_map, work\} \mid \{\!|\, detect\,|\!\} \mid \{collisions\} \rrbracket\\
\left(\begin{array}{l} detect\,?\,work \longrightarrow \\
\left(\begin{array}{l}\mathbf{wait}\ 0\,..\,DC_{TB}\ ;\ DetectCollisions;\\ \mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0\,..\,CC_{TB}\ ;\ CalcCollisions;\\ (output_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT_DL\end{array}\right)\end{array}\right)
\end{array}\right) \setminus \{\!|\, detect\,|\!\}
\end{array}\right) \setminus \{\!|\, reduce\,|\!\}$$

$\equiv$ "distribution of input prefix $detect\,?\,work \longrightarrow \ldots$ using elementary laws"

$$\left(\begin{array}{l}
\left(\begin{array}{l}(next_frame\,?\,frame \longrightarrow (\mathbf{wait}\ 0\,..\,SF_{TB}\ ;\ StoreFrame)) \blacktriangleleft INP_DL;\\ reduce\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}\end{array}\right)\\
\quad \llbracket \{currentFrame, state\} \mid \{\!|\, reduce\,|\!\} \mid \{voxel_map, work, collisions\} \rrbracket\\
\left(\begin{array}{l}
\left(\begin{array}{l} reduce\,?\,currentFrame\,?\,state \longrightarrow \\ \mathbf{wait}\ 0\,..\,RPW_{TB}\ ;\ ReduceAndPartitionWork;\\ detect\,!\,work \longrightarrow \mathbf{skip}\end{array}\right)\\
\quad \llbracket \{voxel_map, work\} \mid \{\!|\, detect\,|\!\} \mid \{collisions\} \rrbracket\\
\left(\begin{array}{l}
\left(\begin{array}{l} detect\,?\,work \longrightarrow \\ \mathbf{wait}\ 0\,..\,DC_{TB}\ ;\ DetectCollisions\end{array}\right);\\
\left(\begin{array}{l}\mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0\,..\,CC_{TB}\ ;\ CalcCollisions;\\ (output_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT_DL\end{array}\right)\end{array}\right)
\end{array}\right) \setminus \{\!|\, detect\,|\!\}
\end{array}\right) \setminus \{\!|\, reduce\,|\!\}$$

$\equiv$ "extraction of hiding of the channel $detect$ using elementary laws"

$$\left(\begin{array}{l}
\left(\begin{array}{l}(next_frame\,?\,frame \longrightarrow (\mathbf{wait}\ 0\,..\,SF_{TB}\ ;\ StoreFrame)) \blacktriangleleft INP_DL;\\ reduce\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}\end{array}\right)\\
\quad \llbracket \{currentFrame, state\} \mid \{\!|\, reduce\,|\!\} \mid \{voxel_map, work, collisions\} \rrbracket\\
\left(\begin{array}{l} reduce\,?\,currentFrame\,?\,state \longrightarrow \\ \mathbf{wait}\ 0\,..\,RPW_{TB}\ ;\ ReduceAndPartitionWork;\\ detect\,!\,work \longrightarrow \mathbf{skip}\end{array}\right)\\
\quad \llbracket \{voxel_map, work\} \mid \{\!|\, detect\,|\!\} \mid \{collisions\} \rrbracket\\
\left(\begin{array}{l}
\left(\begin{array}{l} detect\,?\,work \longrightarrow \\ \mathbf{wait}\ 0\,..\,DC_{TB}\ ;\ DetectCollisions\end{array}\right);\\
\left(\begin{array}{l}\mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0\,..\,CC_{TB}\ ;\ CalcCollisions;\\ (output_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT_DL\end{array}\right)\end{array}\right)
\end{array}\right) \setminus \{\!|\, reduce, detect\,|\!\}$$

$\equiv$ "application of the law seq-to-par-2 introducing a channel $output$ of type int"

$$
\left(
\begin{array}{l}
\left(\begin{array}{l}
(\mathit{next\_frame} ? \mathit{frame} \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} \mathit{SF}_{TB} ;\ \mathit{StoreFrame})) \blacktriangleleft \mathit{INP\_DL}; \\
\mathit{reduce} ! \mathit{currentFrame} ! \mathit{state} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{currentFrame}, \mathit{state}\} \mid \{\!| \mathit{reduce} |\!\} \mid \{\mathit{voxel\_map}, \mathit{work}, \mathit{collisions}\}]\!] \\
\left(\begin{array}{l}
\mathit{reduce} ? \mathit{currentFrame} ? \mathit{state} \longrightarrow \\
\mathbf{wait}\ 0 \mathinner{..} \mathit{RPW}_{TB} ;\ \mathit{ReduceAndPartitionWork}; \\
\mathit{detect} ! \mathit{work} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{voxel\_map}, \mathit{work}\} \mid \{\!| \mathit{detect} |\!\} \mid \{\mathit{collisions}\}]\!] \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{detect} ? \mathit{work} \longrightarrow \\
\mathbf{wait}\ 0 \mathinner{..} \mathit{DC}_{TB} ;\ \mathit{DetectCollisions}; \\
\mathit{output} ! \mathit{collisions} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{collisions}\} \mid \{\!| \mathit{output} |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathit{output} ? \mathit{collisions} \longrightarrow \\
\mathbf{var}\ \mathit{colls} : \mathit{int} \bullet \mathbf{wait}\ 0 \mathinner{..} \mathit{CC}_{TB} ;\ \mathit{CalcCollisions}; \\
(\mathit{output\_collisions} ! \mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT\_DL}
\end{array}\right)
\end{array}\right) \setminus \{\!| \mathit{output} |\!\}
\end{array}
\right) \setminus \{\!| \mathit{reduce}, \mathit{detect} |\!\}
$$

$\equiv$ "extraction of hiding of the channel *output* using elementary laws"

$$
\left(
\begin{array}{l}
\left(\begin{array}{l}
(\mathit{next\_frame} ? \mathit{frame} \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} \mathit{SF}_{TB} ;\ \mathit{StoreFrame})) \blacktriangleleft \mathit{INP\_DL}; \\
\mathit{reduce} ! \mathit{currentFrame} ! \mathit{state} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{currentFrame}, \mathit{state}\} \mid \{\!| \mathit{reduce} |\!\} \mid \{\mathit{voxel\_map}, \mathit{work}, \mathit{collisions}\}]\!] \\
\left(\begin{array}{l}
\mathit{reduce} ? \mathit{currentFrame} ? \mathit{state} \longrightarrow \\
\mathbf{wait}\ 0 \mathinner{..} \mathit{RPW}_{TB} ;\ \mathit{ReduceAndPartitionWork}; \\
\mathit{detect} ! \mathit{work} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{voxel\_map}, \mathit{work}\} \mid \{\!| \mathit{detect} |\!\} \mid \{\mathit{collisions}\}]\!] \\
\left(\begin{array}{l}
\mathit{detect} ? \mathit{work} \longrightarrow \\
\mathbf{wait}\ 0 \mathinner{..} \mathit{DC}_{TB} ;\ \mathit{DetectCollisions}; \\
\mathit{output} ! \mathit{collisions} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{collisions}\} \mid \{\!| \mathit{output} |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathit{output} ? \mathit{collisions} \longrightarrow \\
\mathbf{var}\ \mathit{colls} : \mathit{int} \bullet \mathbf{wait}\ 0 \mathinner{..} \mathit{CC}_{TB} ;\ \mathit{CalcCollisions}; \\
(\mathit{output\_collisions} ! \mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT\_DL}
\end{array}\right)
\end{array}
\right) \setminus \\
\quad \{\!| \mathit{reduce}, \mathit{detect}, \mathit{output} |\!\}
$$

After inserting the refined action into the overall *System* we obtain the following result.

$$
\mathit{System} \mathrel{\widehat=} \\
\left(
\begin{array}{l}
\left(
\begin{array}{l}
\left(
\begin{array}{l}
\left(\begin{array}{l}
(\mathit{next\_frame} ? \mathit{frame} \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} \mathit{SF}_{TB} ;\ \mathit{StoreFrame})) \blacktriangleleft \mathit{INP\_DL}; \\
\mathit{reduce} ! \mathit{currentFrame} ! \mathit{state} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{currentFrame}, \mathit{state}\} \mid \{\!| \mathit{reduce} |\!\} \mid \{\mathit{voxel\_map}, \mathit{work}, \mathit{collisions}\}]\!] \\
\left(\begin{array}{l}
\mathit{reduce} ? \mathit{currentFrame} ? \mathit{state} \longrightarrow \\
\mathbf{wait}\ 0 \mathinner{..} \mathit{RPW}_{TB} ;\ \mathit{ReduceAndPartitionWork}; \\
\mathit{detect} ! \mathit{work} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{voxel\_map}, \mathit{work}\} \mid \{\!| \mathit{detect} |\!\} \mid \{\mathit{collisions}\}]\!] \\
\left(\begin{array}{l}
\mathit{detect} ? \mathit{work} \longrightarrow \\
\mathbf{wait}\ 0 \mathinner{..} \mathit{DC}_{TB} ;\ \mathit{DetectCollisions}; \\
\mathit{output} ! \mathit{collisions} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{collisions}\} \mid \{\!| \mathit{output} |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathit{output} ? \mathit{collisions} \longrightarrow \\
\mathbf{var}\ \mathit{colls} : \mathit{int} \bullet \mathbf{wait}\ 0 \mathinner{..} \mathit{CC}_{TB} ;\ \mathit{CalcCollisions}; \\
(\mathit{output\_collisions} ! \mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT\_DL}
\end{array}\right)
\end{array}
\right) \setminus \\
\quad \{\!| \mathit{reduce}, \mathit{detect}, \mathit{output} |\!\} \\
|\!|\!| \ \mathbf{wait}\ \mathit{FRAME\_PERIOD}
\end{array}
\right) ; \\
\mathit{System}
\end{array}
\right)
$$

We further extract the hiding of the new channels to the outer level.

$$
\begin{array}{l}
\ldots \equiv \text{“extraction of hiding of } \{\!| \mathit{reduce}, \mathit{detect}, \mathit{output} |\!\} \text{ using elementary laws”} \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
(\mathit{next\_frame} \,?\, \mathit{frame} \longrightarrow (\mathbf{wait}\, 0 \mathinner{\ldotp\ldotp} \mathit{SF}_{TB} \;;\; \mathit{StoreFrame})) \blacktriangleleft \mathit{INP\_DL}; \\
\mathit{reduce} \,!\, \mathit{currentFrame} \,!\, \mathit{state} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{currentFrame}, \mathit{state}\} \mid \{\!| \mathit{reduce} |\!\} \mid \{\mathit{voxel\_map}, \mathit{work}, \mathit{collisions}\}]\!] \\
\left(\begin{array}{l}
\mathit{reduce} \,?\, \mathit{currentFrame} \,?\, \mathit{state} \longrightarrow \\
\mathbf{wait}\, 0 \mathinner{\ldotp\ldotp} \mathit{RPW}_{TB} \;;\; \mathit{ReduceAndPartitionWork}; \\
\mathit{detect} \,!\, \mathit{work} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{voxel\_map}, \mathit{work}\} \mid \{\!| \mathit{detect} |\!\} \mid \{\mathit{collisions}\}]\!] \\
\left(\begin{array}{l}
\mathit{detect} \,?\, \mathit{work} \longrightarrow \\
\mathbf{wait}\, 0 \mathinner{\ldotp\ldotp} \mathit{DC}_{TB} \;;\; \mathit{DetectCollisions}; \\
\mathit{output} \,!\, \mathit{collisions} \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{\mathit{collisions}\} \mid \{\!| \mathit{output} |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathit{output} \,?\, \mathit{collisions} \longrightarrow \\
\mathbf{var}\, \mathit{colls} : \mathit{int} \bullet \mathbf{wait}\, 0 \mathinner{\ldotp\ldotp} \mathit{CC}_{TB} \;;\; \mathit{CalcCollisions}; \\
(\mathit{output\_collisions} \,!\, \mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT\_DL}
\end{array}\right)
\end{array}\right) \\
\interleave \mathbf{wait}\, \mathit{FRAME\_PERIOD}
\end{array}\right) ; \\
\mathit{System}
\end{array}\right) \setminus \\
\quad \{\!| \mathit{reduce}, \mathit{detect}, \mathit{output} |\!\}
\end{array}
$$

This concludes Stage 4 of the refinement for the MH phase. We observe that all sequential data operations have been transformed into parallel actions. Execution control and the passing of data is achieved by the new channels *reduce*, *detect* and *output*. Parallelisation is, however, not completed yet. In particular, the parallelism of detection handlers has not emerged in this stage. The application of laws follows a uniform pattern and hence automation guided by the developer should be possible in this stage.

5.2.5 Stage 5

Stage 5 deals with the transformation of parallel data operations (schema conjunctions) into parallel handler actions. In our example, this is the refinement of *DetectCollisions*, including its time budget. Below we use the copy rule to expand the definition of *DetectCollisions* in the *System* action.

The focus of the subsequent refinement steps is highlighted.

$\ldots \equiv$ "copy rule expanding *DetectCollisions*"

$$
\begin{array}{l}
System \mathrel{\widehat=} \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
(next\_frame\,?\,frame \longrightarrow (\mathbf{wait}\ 0\,..\,SF_{TB}\ ;\ \ StoreFrame)) \blacktriangleleft INP\_DL; \\
reduce\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{currentFrame, state\} \mid \{\!|\ reduce\ |\!\} \mid \{voxel\_map, work, collisions\}]\!] \\
\left(\begin{array}{l}
reduce\,?\,currentFrame\,?\,state \longrightarrow \\
\mathbf{wait}\ 0\,..\,RPW_{TB}\ ;\ \ ReduceAndPartitionWork; \\
detect\,!\,work \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{voxel\_map, work\} \mid \{\!|\ detect\ |\!\} \mid \{collisions\}]\!] \\
\left(\begin{array}{l}
detect\,?\,work \longrightarrow \mathbf{wait}\ 0\,..\,DC_{TB}; \\
\left(\begin{array}{l}
\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\qquad \left(\begin{array}{l}
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4)
\end{array}\right); \\
\qquad SetCollisionsFromParts([\![\ colls1, colls2, colls3, colls4\ ]\!]\,)
\end{array}\right); \\
output\,!\,collisions \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{collisions\} \mid \{\!|\ output\ |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
output\,?\,collisions \longrightarrow \\
\mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0\,..\,CC_{TB}\ ;\ \ CalcCollisions; \\
(output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL
\end{array}\right)
\end{array}\right) \\
|\!|\!|\ \mathbf{wait}\ FRAME\_PERIOD
\end{array}\right); \\
System
\end{array}\right) \setminus \\
\qquad \{\!|\ reduce, detect, output\ |\!\}
\end{array}
$$

We start by decomposing the time budget for *DetectCollisions* as already illustrated in Stage 3.

$$
\begin{array}{l}
\mathbf{wait}\ 0\,..\,DC_{TB}; \\
\left(\begin{array}{l}
\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\qquad \left(\begin{array}{l}
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4)
\end{array}\right); \\
\qquad SetCollisionsFromParts([\![\ colls1, colls2, colls3, colls4\ ]\!]\,)
\end{array}\right)
\end{array}
$$

$\sqsubseteq$ "application of the law split-time-budget-2"

$$
\begin{array}{l}
\mathbf{wait}\ 0\,..\,CPC_{TB}\ ;\ \ \mathbf{wait}\ 0\,..\,SC_{TB}; \\
\left(\begin{array}{l}
\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\qquad \left(\begin{array}{l}
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3) \land \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4)
\end{array}\right); \\
\qquad SetCollisionsFromParts([\![\ colls1, colls2, colls3, colls4\ ]\!]\,)
\end{array}\right)
\end{array}
$$

This assume the presence of two further constants CPC_{TB} and SC_{TB} with $CPC_{TB} + SC_{TB} \leq DC_{TB}$.
We next distribute the time budgets in order to attach them to the respective data operations. This, again,

is similar in principle to what we has already been illustrated in Stage 3. The following supplementary law facilitates distribution of time budgets into local variable declarations.

Circus **Time Law 14 (distr-wait-seq-var)**

$$\textbf{wait}\ t_1 \mathrel{..} t_2\ ;\ \textbf{var}\ x : T \bullet A \equiv \textbf{var}\ x : T \bullet (\textbf{wait}\ t_1 \mathrel{..} t_2\ ;\ A)$$
$$\textbf{provided}\ \ x \notin FV(t_1)\ \ \textbf{and}\ \ x \notin FV(t_2)$$

The resulting transformation is given below.

$\equiv$ "application of the laws distr-wait-seq-var and time-budget-op-comm"

$$\left(\begin{array}{l}\textbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \textbf{wait}\ 0 \mathrel{..} CPC_{TB};\\ \quad \left(\begin{array}{l}(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \wedge i? = 1) \wedge\\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \wedge i? = 2) \wedge\\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \wedge i? = 3) \wedge\\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \wedge i? = 4)\end{array}\right);\\ \quad \textbf{wait}\ 0 \mathrel{..} SC_{TB};\\ \quad SetCollisionsFromParts(\langle\!\langle\ colls1, colls2, colls3, colls4\ \rangle\!\rangle)\end{array}\right)$$

The next law is used to turn the schema conjunction into a parallelism of actions.

Circus **Law 4 (conj-to-par)**

$$Op_1 \wedge Op_2 \equiv Op_1 \llbracket wrtV(Op_1) \mid \varnothing \mid wrtV(Op_2) \rrbracket Op_2\ \ \textbf{provided}\ \ wrtV(Op_1) \cap wrtV(Op_2) = \varnothing$$

It is applicable since all schemas in the above conjunction write to a different variable $colls\mathtt{i}$.

$\ldots \equiv$ "multiple applications of law conj-to-par"

$$\left(\begin{array}{l}\textbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \textbf{wait}\ 0 \mathrel{..} CPC_{TB};\\ \quad \left(\begin{array}{l}(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \wedge i? = 1)\\ \quad \llbracket \{colls1\} \mid \varnothing \mid \{colls2, colls3, colls4\} \rrbracket\\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \wedge i? = 2)\\ \quad \llbracket \{colls2\} \mid \varnothing \mid \{colls3, colls4\} \rrbracket\\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \wedge i? = 3)\\ \quad \llbracket \{colls3\} \mid \varnothing \mid \{colls4\} \rrbracket\\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \wedge i? = 4)\end{array}\right);\\ \quad \textbf{wait}\ 0 \mathrel{..} SC_{TB};\\ \quad SetCollisionsFromParts(\langle\!\langle\ colls1, colls2, colls3, colls4\ \rangle\!\rangle)\end{array}\right)$$

Another distribution law enables us to move $\textbf{wait}\ 0 \mathrel{..} CPC_{TB}$ into the parallelism.

Circus **Time Law 15 (distr-wait-seq-par)**

$$\textbf{wait}\ t_1 \mathrel{..} t_2\ ;\ (Op_1 \llbracket \ldots \rrbracket Op_2) \equiv (\textbf{wait}\ t_1 \mathrel{..} t_2\ ;\ Op_1) \llbracket \ldots \rrbracket (\textbf{wait}\ t_1 \mathrel{..} t_2\ ;\ Op_2)$$
$$\textbf{provided}\ \ Op_1 \text{ and } Op_2 \text{ are data operations}$$

Intuitively, since the two data operations execute in parallel, each of them has a time budget $\textbf{wait}\ t_1 \mathrel{..} t_2$.

$\ldots \equiv$ “multiple applications of law distr-wait-seq-par”

$$\left(\begin{array}{l}
\mathbf{var}\ \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 : \mathit{int} \bullet\\
\quad \left(\begin{array}{l}
(\mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{TB} \ ; \ (\exists i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}1/\mathit{pcolls}!] \land i? = 1))\\
\quad [\![\{\mathit{colls}1\} \mid \varnothing \mid \{\mathit{colls}2, \mathit{colls}3, \mathit{colls}4\}]\!]\\
(\mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{TB} \ ; \ (\exists i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}2/\mathit{pcolls}!] \land i? = 2))\\
\quad [\![\{\mathit{colls}2\} \mid \varnothing \mid \{\mathit{colls}3, \mathit{colls}4\}]\!]\\
(\mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{TB} \ ; \ (\exists i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}3/\mathit{pcolls}!] \land i? = 3))\\
\quad [\![\{\mathit{colls}3\} \mid \varnothing \mid \{\mathit{colls}4\}]\!]\\
(\mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{TB} \ ; \ (\exists i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}4/\mathit{pcolls}!] \land i? = 4))
\end{array}\right) ;\\
\quad \mathbf{wait}\ 0 \mathrel{..} \mathit{SC}_{TB} \ ; \ \mathit{SetCollisionsFromParts}([\![\mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4]\!])
\end{array}\right)$$

Above we still have the sequence with the *SetCollisionsFromParts* data operation; it also has to be parallelised. To achieve this, we use seq-to-par-2 from Stage 4 again, this time introducing a channel *setColls* of type $\mathit{int} \times \mathit{int} \times \mathit{int} \times \mathit{int}$ to communicate all partial results computed by the four detector handlers.

$\ldots \equiv$ “application of the law seq-to-par-2”

$$\left(\begin{array}{l}
\mathbf{var}\ \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 : \mathit{int} \bullet\\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
(\mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{TB} \ ; \ (\exists i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}1/\mathit{pcolls}!] \land i? = 1))\\
\quad [\![\{\mathit{colls}1\} \mid \varnothing \mid \{\mathit{colls}2, \mathit{colls}3, \mathit{colls}4\}]\!]\\
(\mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{TB} \ ; \ (\exists i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}2/\mathit{pcolls}!] \land i? = 2))\\
\quad [\![\{\mathit{colls}2\} \mid \varnothing \mid \{\mathit{colls}3, \mathit{colls}4\}]\!]\\
(\mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{TB} \ ; \ (\exists i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}3/\mathit{pcolls}!] \land i? = 3))\\
\quad [\![\{\mathit{colls}3\} \mid \varnothing \mid \{\mathit{colls}4\}]\!]\\
(\mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{TB} \ ; \ (\exists i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}4/\mathit{pcolls}!] \land i? = 4))
\end{array}\right) ;\\
\mathit{setColls}\,!\,\mathit{colls}1\,!\,\mathit{colls}2\,!\,\mathit{colls}3\,!\,\mathit{colls}4 \longrightarrow \mathbf{skip}
\end{array}\right)\\
\quad [\![\{\mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4\} \mid \{\!| \mathit{setColls} |\!\} \mid \{\mathit{collisions}\}]\!]\\
\left(\begin{array}{l}
\mathit{setColls}\,?\,\mathit{colls}1\,?\,\mathit{colls}2\,?\,\mathit{colls}3\,?\,\mathit{colls}4 \longrightarrow\\
\mathbf{wait}\ 0 \mathrel{..} \mathit{SC}_{TB} \ ; \ \mathit{SetCollisionsFromParts}([\![\mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4]\!])
\end{array}\right)
\end{array}\right) \setminus\\
\quad \{\!| \mathit{setColls} |\!\}
\end{array}\right)$$

To eliminate the prefix that was introduced in the left branch of the parallelism, we require a specialised channel decomposition law that replaces the channel *setColls* of type $\mathit{int} \times \mathit{int} \times \mathit{int} \times \mathit{int}$ by a channel *recColls* of type *int*. The intention of the refinement is to decompose a single communication on *setColls* into an interleaving of four communications on *recColls*. We note that *setColls* is assumed to be concealed in the context where this law is applicable. A detailed investigation of the law is future work for now.

$\ldots \equiv$ "application of a specialised high-level channel decomposition law"

$$\left(\begin{array}{l}\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\ \quad \left(\left(\begin{array}{l}\left(\begin{array}{l}(\mathbf{wait}\ 0 \mathinner{..} CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1)) \\ \quad [\![\{colls1\} \mid \varnothing \mid \{colls2, colls3, colls4\}]\!] \\ (\mathbf{wait}\ 0 \mathinner{..} CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2)) \\ \quad [\![\{colls2\} \mid \varnothing \mid \{colls3, colls4\}]\!] \\ (\mathbf{wait}\ 0 \mathinner{..} CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3)) \\ \quad [\![\{colls3\} \mid \varnothing \mid \{colls4\}]\!] \\ (\mathbf{wait}\ 0 \mathinner{..} CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4))\end{array}\right) ; \\ \left(\begin{array}{l}(recColls\,!\,colls1 \longrightarrow \mathbf{skip})\ |||\\ (recColls\,!\,colls2 \longrightarrow \mathbf{skip})\ |||\\ (recColls\,!\,colls3 \longrightarrow \mathbf{skip})\ |||\\ (recColls\,!\,colls4 \longrightarrow \mathbf{skip})\end{array}\right)\end{array}\right) \\ \quad [\![\{colls1, colls2, colls3, colls4\} \mid \{\!|\ recColls\ |\!\} \mid \{collisions\}]\!] \\ \left(\begin{array}{l}\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\ \left(\begin{array}{l}(recColls\,?\,x \longrightarrow colls1 := x)\ |||\\ (recColls\,?\,x \longrightarrow colls2 := x)\ |||\\ (recColls\,?\,x \longrightarrow colls3 := x)\ |||\\ (recColls\,?\,x \longrightarrow colls4 := x)\end{array}\right) ; \\ \mathbf{wait}\ 0 \mathinner{..} SC_{TB}\ ;\ \ SetCollisionsFromParts(\langle\!|\ colls1, colls2, colls3, colls4\ |\!\rangle)\end{array}\right)\right) \setminus \\ \quad \{\!|\ recColls\ |\!\}\end{array}\right)$$

Likewise, another specialised law is subsequently used to distribute the interleaving in the left hand of the outer parallelism into the inner parallelism that computes that partial collision results.

$\ldots \equiv$ "application of a specialised high-level law for distribution of an interleaving of prefixes"

$$\left(\begin{array}{l}\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\ \quad \left(\left(\begin{array}{l}\left(\begin{array}{l}\mathbf{wait}\ 0 \mathinner{..} CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\ recColls\,!\,colls1 \longrightarrow \mathbf{skip}\end{array}\right) \\ \quad [\![\{colls1\} \mid \varnothing \mid \{colls2, colls3, colls4\}]\!] \\ \left(\begin{array}{l}\mathbf{wait}\ 0 \mathinner{..} CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\ recColls\,!\,colls2 \longrightarrow \mathbf{skip}\end{array}\right) \\ \quad [\![\{colls2\} \mid \varnothing \mid \{colls3, colls4\}]\!] \\ \left(\begin{array}{l}\mathbf{wait}\ 0 \mathinner{..} CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\ recColls\,!\,colls3 \longrightarrow \mathbf{skip}\end{array}\right) \\ \quad [\![\{colls3\} \mid \varnothing \mid \{colls4\}]\!] \\ \left(\begin{array}{l}\mathbf{wait}\ 0 \mathinner{..} CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\ recColls\,!\,colls4 \longrightarrow \mathbf{skip}\end{array}\right)\end{array}\right) \\ \quad [\![\{colls1, colls2, colls3, colls4\} \mid \{\!|\ recColls\ |\!\} \mid \{collisions\}]\!] \\ \left(\begin{array}{l}\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\ \left(\begin{array}{l}(recColls\,?\,x \longrightarrow colls1 := x)\ |||\\ (recColls\,?\,x \longrightarrow colls2 := x)\ |||\\ (recColls\,?\,x \longrightarrow colls3 := x)\ |||\\ (recColls\,?\,x \longrightarrow colls4 := x)\end{array}\right) ; \\ \mathbf{wait}\ 0 \mathinner{..} SC_{TB}\ ;\ \ SetCollisionsFromParts(\langle\!|\ colls1, colls2, colls3, colls4\ |\!\rangle)\end{array}\right)\right) \setminus \\ \quad \{\!|\ recColls\ |\!\}\end{array}\right)$$

A final high-level law sequentialises the interleaving in the right-hand branch of the outer parallelism.

$\ldots \equiv$ "application of a specialised high-level law for sequentialising prefix interleaving"

$$
\left(\begin{array}{l}
\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\qquad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\
recColls\,!\,colls1 \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad \llbracket \{colls1\} \mid \varnothing \mid \{colls2, colls3, colls4\} \rrbracket \\
\left(\begin{array}{l}
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\
recColls\,!\,colls2 \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad \llbracket \{colls2\} \mid \varnothing \mid \{colls3, colls4\} \rrbracket \\
\left(\begin{array}{l}
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\
recColls\,!\,colls3 \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad \llbracket \{colls3\} \mid \varnothing \mid \{colls4\} \rrbracket \\
\left(\begin{array}{l}
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\
recColls\,!\,colls4 \longrightarrow \mathbf{skip}
\end{array}\right)
\end{array}\right) \\
\quad \llbracket \{colls1, colls2, colls3, colls4\} \mid \{\!|\, recColls \,|\!\} \mid \{collisions\} \rrbracket \\
\left(\begin{array}{l}
\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\left(\begin{array}{l}
(recColls\,?\,x \longrightarrow colls1 := x)\ ; \\
(recColls\,?\,x \longrightarrow colls2 := x)\ ; \\
(recColls\,?\,x \longrightarrow colls3 := x)\ ; \\
(recColls\,?\,x \longrightarrow colls4 := x)
\end{array}\right); \\
\mathbf{wait}\ 0 \mathrel{..} SC_{TB}\ ;\ SetCollisionsFromParts(\llbracket\, colls1, colls2, colls3, colls4 \,\rrbracket\,)
\end{array}\right)
\end{array}\right) \setminus \\
\qquad \{\!|\, recColls \,|\!\}
\end{array}\right)
$$

This refinement is valid because the *SetCollisionsFromParts* operation is parametrised in terms of a bag and therefore is agnostic to the order in which results are communicated through the *recColls* channel. The sequentialising of the interleaving is important in order to decompose and distribute the time budget SC_{TB} between the elements of the sequence. Hence, in the next step we decompose and distribute the time budget SC_{TB} into two time budgets, RC_{TB} and $SCFP_{TB}$ where $4 * RC_{TB} + SCFP_{TB} \leq SC_{TB}$.

$\ldots \equiv$ "decomposition and distribution of $\mathbf{wait}\ 0 \mathrel{..} SC_{TB}$ using laws from Stage 3"

$$
\left(\begin{array}{l}
\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\qquad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\
recColls\,!\,colls1 \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad \llbracket \{colls1\} \mid \varnothing \mid \{colls2, colls3, colls4\} \rrbracket \\
\left(\begin{array}{l}
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\
recColls\,!\,colls2 \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad \llbracket \{colls2\} \mid \varnothing \mid \{colls3, colls4\} \rrbracket \\
\left(\begin{array}{l}
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\
recColls\,!\,colls3 \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad \llbracket \{colls3\} \mid \varnothing \mid \{colls4\} \rrbracket \\
\left(\begin{array}{l}
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\
recColls\,!\,colls4 \longrightarrow \mathbf{skip}
\end{array}\right)
\end{array}\right) \\
\quad \llbracket \{colls1, colls2, colls3, colls4\} \mid \{\!|\, recColls \,|\!\} \mid \{collisions\} \rrbracket \\
\left(\begin{array}{l}
\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\left(\begin{array}{l}
(recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} RC_{TB}\ ;\ colls1 := x))\ ; \\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} RC_{TB}\ ;\ colls2 := x))\ ; \\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} RC_{TB}\ ;\ colls3 := x))\ ; \\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} RC_{TB}\ ;\ colls4 := x))
\end{array}\right); \\
\mathbf{wait}\ 0 \mathrel{..} SCFP_{TB}\ ;\ SetCollisionsFromParts(\llbracket\, colls1, colls2, colls3, colls4 \,\rrbracket\,)
\end{array}\right)
\end{array}\right) \setminus \\
\qquad \{\!|\, recColls \,|\!\}
\end{array}\right)
$$

The above refinement is justified by the assumption $4 * RC_{TB} + SCFP_{TB} \leq SC_{TB}$ about the new constants, and also the fact that the communication on *recColls* is concealed. A detailed formulation of the law including its proof are future work. For one, it seems that it is not compositional. Comparing, for instance,

$$\begin{pmatrix} (recColls\,?\,x \longrightarrow colls1 := x)\,; \\ (recColls\,?\,x \longrightarrow colls2 := x)\,; \\ (recColls\,?\,x \longrightarrow colls3 := x)\,; \\ (recColls\,?\,x \longrightarrow colls4 := x) \end{pmatrix} \quad \text{and}$$

$$\begin{pmatrix} (recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ \ colls1 := x))\,; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ \ colls2 := x))\,; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ \ colls3 := x))\,; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ \ colls4 := x)) \end{pmatrix}$$

we observe that the second action refuses more than the first action in terms of its time-wise behaviour. Namely, the first action permits multiple *recColls* events to occur in the same instant whereas in the second action there can be a delay of up to RC_{TB} time units between them, so this is not a failures refinement. Therefore, we have to refine the above fragment in context rather than in isolation; this is future work.

We next distribute the local variable declarations into the respective parallel branches that write to the variable. A few basic laws, namely distr-var-hide, distr-var-par, remove-var and compact-write-sets-par are useful here; they are given in Appendix B.1. We shall not discuss the refinement in detail.

$\ldots\ \equiv$ "application of the laws distr-var-hide, distr-var-par, remove-var and compact-write-sets-par"

$$\left(\begin{array}{l} \left(\begin{array}{l} \begin{pmatrix} \mathbf{var}\ colls1 : int\ \bullet \\ \mathbf{wait}\,0\,..\,CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \wedge i? = 1); \\ recColls\,!\,colls1 \longrightarrow \mathbf{skip} \end{pmatrix} \\ \quad [\![\varnothing \mid \varnothing \mid \varnothing]\!] \\ \begin{pmatrix} \mathbf{var}\ colls2 : int\ \bullet \\ \mathbf{wait}\,0\,..\,CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \wedge i? = 2); \\ recColls\,!\,colls2 \longrightarrow \mathbf{skip} \end{pmatrix} \\ \quad [\![\varnothing \mid \varnothing \mid \varnothing]\!] \\ \begin{pmatrix} \mathbf{var}\ colls3 : int\ \bullet \\ \mathbf{wait}\,0\,..\,CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \wedge i? = 3); \\ recColls\,!\,colls3 \longrightarrow \mathbf{skip} \end{pmatrix} \\ \quad [\![\varnothing \mid \varnothing \mid \varnothing]\!] \\ \begin{pmatrix} \mathbf{var}\ colls4 : int\ \bullet \\ \mathbf{wait}\,0\,..\,CPC_{TB}\ ;\ \ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \wedge i? = 4); \\ recColls\,!\,colls4 \longrightarrow \mathbf{skip} \end{pmatrix} \end{array}\right) \\ \quad [\![\varnothing \mid \{\!|\, recColls\, |\!\} \mid \{collisions\}]\!] \\ \left(\begin{array}{l} \mathbf{var}\ colls1, colls2, colls3, colls4 : int\ \bullet \\ \begin{pmatrix} (recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ \ colls1 := x))\,; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ \ colls2 := x))\,; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ \ colls3 := x))\,; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ \ colls4 := x)) \end{pmatrix}; \\ \mathbf{wait}\,0\,..\,SCFP_{TB}\ ;\ \ SetCollisionsFromParts(\langle\!\langle\, colls1, colls2, colls3, colls4\, \rangle\!\rangle\,) \end{array}\right) \end{array}\right) \setminus \{\!|\, recColls\, |\!\}$$

With the last refinement we have localised the declaration of the *colls*i into the parallel detector handlers. This transformation concludes the refinement of the action fragment of *System* that corresponds to the four detector handlers, whose parallelism has fully emerged now. The right-hand action of the top-level parallelism is a residual control fragment that later on in the SH phase is going to be refined into shared data access to the *collisions* variable by suitable atomic operations.

We now inject the refinement of the parallel detector handlers back into the *System* action.

$$
\begin{array}{l}
System \mathrel{\widehat=} \\
\quad \left(\left(\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
(next\_frame\,?\,frame \longrightarrow (\mathbf{wait}\,0\,..\,SF_{TB}\ ;\ StoreFrame)) \blacktriangleleft INP\_DL; \\
reduce\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{currentFrame, state\} \mid \{\!|\, reduce \,|\!\} \mid \{voxel\_map, work, collisions\}]\!] \\
\left(\begin{array}{l}
reduce\,?\,currentFrame\,?\,state \longrightarrow \\
\mathbf{wait}\,0\,..\,RPW_{TB}\ ;\ ReduceAndPartitionWork; \\
detect\,!\,work \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{voxel\_map, work\} \mid \{\!|\, detect \,|\!\} \mid \{collisions\}]\!] \\
\left(\begin{array}{l}
detect\,?\,work \longrightarrow \\
\left(\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ colls1 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\
recColls\,!\,colls1 \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\varnothing \mid \varnothing \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathbf{var}\ colls2 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\
recColls\,!\,colls2 \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\varnothing \mid \varnothing \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathbf{var}\ colls3 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\
recColls\,!\,colls3 \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\varnothing \mid \varnothing \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathbf{var}\ colls4 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\
recColls\,!\,colls4 \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!|\, recColls \,|\!\} \mid \{collisions\}]\!] \\
\left(\begin{array}{l}
\mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\left(\begin{array}{l}
(recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ colls1 := x))\,; \\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ colls2 := x))\,; \\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ colls3 := x))\,; \\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\ ;\ colls4 := x))
\end{array}\right); \\
\mathbf{wait}\,0\,..\,SCFP_{TB}; \\
SetCollisionsFromParts(\langle\!|\, colls1, colls2, colls3, colls4 \,|\!\rangle)
\end{array}\right)
\end{array}\right) \setminus \{\!|\, recColls \,|\!\};\right) \\
output\,!\,collisions \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{collisions\} \mid \{\!|\, output \,|\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
output\,?\,collisions \longrightarrow \\
\mathbf{var}\ colls : int \bullet \mathbf{wait}\,0\,..\,CC_{TB}\ ;\ CalcCollisions; \\
(output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL
\end{array}\right)
\end{array}\right) \\
\ |||\ \mathbf{wait}\ FRAME\_PERIOD
\end{array}\right) ; \\
System
\end{array}\right) \setminus \\
\quad\quad \{\!|\, reduce, detect, output \,|\!\}
\end{array}
$$

We observe that the resulting action does not have the desired shape yet: this is a parallelism of handlers plus possible parallel control actions. The refinement that concludes Stage 5 thus has to carry out elementary transformations to put the action (back) into this form.

Consolidation of Mission Actions

The consolidation steps here, in particular, involve extracting the hiding of *recColls* and distributing the prefixes $detect\,?\,work \longrightarrow \ldots$ and $output\_collisions\,!\,collisions \longrightarrow \mathbf{skip}$ into the parallel actions which they surround. The required laws for distributing the prefixes are presented below.

Circus **Law 9 (distr-prefix-par-1)**

$$c ? x \longrightarrow (A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2) \equiv (c ? x \longrightarrow A_1 \llbracket ns_1 \mid cs \cup \{c\} \mid ns_2 \rrbracket c ? x \longrightarrow A_2)$$

provided $c \notin usedC(A_1)$ **and** $c \notin usedC(A_2)$

Circus **Law 10 (distr-prefix-par-2)**

$$(A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2) \,;\; c ! x \longrightarrow \mathbf{skip} \equiv$$
$$(A_1 \,;\; c ? y \longrightarrow \mathbf{skip}) \llbracket ns_1 \mid cs \cup \{\!| c |\!\} \mid ns_2 \rrbracket (A_2 \,;\; c ! x \longrightarrow \mathbf{skip})$$

provided $c \notin usedC(A_1)$ **and** $c \notin usedC(A_2)$ **and** $x \notin ns_1$

Their application yields the following action.

$\ldots \equiv$ "application of the laws distr-prefix-par-1 and distr-prefix-par-2 and extraction of hiding"

$$\left(\begin{array}{l}
\left(\left(\left(\begin{array}{l}
\left(\begin{array}{l}(next_frame ? frame \longrightarrow (\mathbf{wait}\ 0 \,..\, SF_{TB} \,;\; StoreFrame)) \blacktriangleleft INP_DL; \\ reduce ! currentFrame ! state \longrightarrow \mathbf{skip}\end{array}\right) \\
\quad \llbracket \{currentFrame, state\} \mid \{\!| reduce |\!\} \mid \{voxel_map, work, collisions\} \rrbracket \\
\left(\begin{array}{l} reduce ? currentFrame ? state \longrightarrow \\ \mathbf{wait}\ 0 \,..\, RPW_{TB} \,;\; ReduceAndPartitionWork; \\ detect ! work \longrightarrow \mathbf{skip}\end{array}\right) \\
\quad \llbracket \{voxel_map, work\} \mid \{\!| detect |\!\} \mid \{collisions\} \rrbracket \\
\left(\begin{array}{l}
\left(\begin{array}{l} detect ? work \longrightarrow \mathbf{var}\ colls1 : int \bullet \mathbf{wait}\ 0 \,..\, CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\ recColls ! colls1 \longrightarrow \mathbf{skip} \,;\; output ? y \longrightarrow \mathbf{skip}\end{array}\right) \\
\quad \llbracket \varnothing \mid \{\!| detect, output |\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l} detect ? work \longrightarrow \mathbf{var}\ colls2 : int \bullet \mathbf{wait}\ 0 \,..\, CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\ recColls ! colls2 \longrightarrow \mathbf{skip} \,;\; output ? y \longrightarrow \mathbf{skip}\end{array}\right) \\
\quad \llbracket \varnothing \mid \{\!| detect, output |\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l} detect ? work \longrightarrow \mathbf{var}\ colls3 : int \bullet \mathbf{wait}\ 0 \,..\, CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\ recColls ! colls3 \longrightarrow \mathbf{skip} \,;\; output ? y \longrightarrow \mathbf{skip}\end{array}\right) \\
\quad \llbracket \varnothing \mid \{\!| detect, output |\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l} detect ? work \longrightarrow \mathbf{var}\ colls4 : int \bullet \mathbf{wait}\ 0 \,..\, CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\ recColls ! colls4 \longrightarrow \mathbf{skip} \,;\; output ? y \longrightarrow \mathbf{skip}\end{array}\right) \\
\quad \llbracket \varnothing \mid \{\!| detect, output, recColls |\!\} \mid \{collisions\} \rrbracket \\
\left(\begin{array}{l} detect ? work \longrightarrow \mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\left(\begin{array}{l} (recColls ? x \longrightarrow (\mathbf{wait}\ 0 \,..\, RC_{TB} \,;\; colls1 := x)) \,; \\ (recColls ? x \longrightarrow (\mathbf{wait}\ 0 \,..\, RC_{TB} \,;\; colls2 := x)) \,; \\ (recColls ? x \longrightarrow (\mathbf{wait}\ 0 \,..\, RC_{TB} \,;\; colls3 := x)) \,; \\ (recColls ? x \longrightarrow (\mathbf{wait}\ 0 \,..\, RC_{TB} \,;\; colls4 := x))\end{array}\right); \\
\mathbf{wait}\ 0 \,..\, SCFP_{TB}; \\
SetCollisionsFromParts(\llbracket colls1, colls2, colls3, colls4 \rrbracket); \\
output ! collisions \longrightarrow \mathbf{skip}\end{array}\right)
\end{array}\right) \\
\quad \llbracket \{collisions\} \mid \{\!| output |\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l} output ? collisions \longrightarrow \\ \mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0 \,..\, CC_{TB} \,;\; CalcCollisions; \\ (output_collisions ! colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT_DL\end{array}\right)
\end{array}\right) \\
\quad \interleave \mathbf{wait}\ FRAME_PERIOD
\right)\right) ; \\
System
\end{array}\right) \setminus$$

$$\{\!| reduce, detect, output, recColls |\!\}$$

We lastly reorder the parallelism and adjust write sets in order to isolated the handler actions and the control fragment into separate parallel branches.

$\ldots \equiv$ "reordering parallel actions and adjusting write sets"

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
(next\_frame ? frame \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} SF_{TB} ;\ StoreFrame)) \blacktriangleleft INP\_DL; \\
reduce ! currentFrame ! state \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{ currentFrame, state \} \mid \{\!| reduce |\!\} \mid \{ voxel\_map, work \}]\!] \\
\left(\begin{array}{l}
reduce ? currentFrame ? state \longrightarrow \\
\mathbf{wait}\ 0 \mathrel{..} RPW_{TB} ;\ ReduceAndPartitionWork; \\
detect ! work \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\{ voxel\_map, work \} \mid \{\!| detect |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
detect ? work \longrightarrow \mathbf{var}\ colls1 : int \bullet \mathbf{wait}\ 0 \mathrel{..} CPC_{TB}; \\
(\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\
recColls ! colls1 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
detect ? work \longrightarrow \mathbf{var}\ colls2 : int \bullet \mathbf{wait}\ 0 \mathrel{..} CPC_{TB}; \\
(\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\
recColls ! colls2 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
detect ? work \longrightarrow \mathbf{var}\ colls3 : int \bullet \mathbf{wait}\ 0 \mathrel{..} CPC_{TB}; \\
(\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\
recColls ! colls3 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
detect ? work \longrightarrow \mathbf{var}\ colls4 : int \bullet \mathbf{wait}\ 0 \mathrel{..} CPC_{TB}; \\
(\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\
recColls ! colls4 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip}
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| output |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
output ? collisions \longrightarrow \\
\mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0 \mathrel{..} CC_{TB} ;\ CalcCollisions; \\
(output\_collisions ! colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL
\end{array}\right)
\end{array}\right) \\
\quad [\![\{ currentFrame, state, voxel\_map, work \} \mid \\
\qquad \{\!| detect, output, recColls |\!\} \mid \{ collisions \}]\!] \\
\left(\begin{array}{l}
detect ? work \longrightarrow \mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\left(\begin{array}{l}
(recColls ? x \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} RC_{TB} ;\ colls1 := x)) ; \\
(recColls ? x \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} RC_{TB} ;\ colls2 := x)) ; \\
(recColls ? x \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} RC_{TB} ;\ colls3 := x)) ; \\
(recColls ? x \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} RC_{TB} ;\ colls4 := x))
\end{array}\right) ; \\
\mathbf{wait}\ 0 \mathrel{..} SCFP_{TB}; \\
SetCollisionsFromParts([\![colls1, colls2, colls3, colls4]\!]); \\
output ! collisions \longrightarrow \mathbf{skip}
\end{array}\right)
\end{array}\right) \\
|\!|\!|\ \mathbf{wait}\ FRAME\_PERIOD
\end{array}\right) ; \\
System
\end{array}\right) \setminus \\
\quad \{\!| reduce, detect, output, recColls |\!\}
$$

The above refinement of *System* concludes Stage 5 of the MH phase. The parallelism of handler actions has fully emerged now. Whereas Stage 4 seems to provide good opportunities for automation, Stage 5, in comparison, appears to be more challenging in that respect. Automation may be envisage through the application of high-level refinement patterns that encapsulate particular structures, such as the shape of *DetectCollisions*. Subsequent elementary refinements could possibly be tackled by refinement tactics [10].

5.2.6 Stage 6

In Stage 6 we extract the mission and handler actions. The only remaining transformation required for this is to distribute the top-level recursion in the *System* action into the parallel branches that correspond to handler actions and control fragments. Lock-step progress per cycle is ensured by a new channel *sync*.

Distribute Recursion into Parallel Actions

The result of distributing the top-level recursion in *System* is given below.

$\ldots \equiv$ "distribution law for recursion into parallel actions"

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mu X \bullet (next\_frame\,?\,frame \longrightarrow (\mathbf{wait}\,0\,..\,SF_{TB}\,;\ StoreFrame)) \blacktriangleleft INP\_DL;\\
reduce\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\{currentFrame, state\} \mid \{\!| reduce, sync |\!\} \mid \{voxel\_map, work\}]\!]\\
\left(\begin{array}{l}
\mu X \bullet reduce\,?\,currentFrame\,?\,state \longrightarrow\\
\mathbf{wait}\,0\,..\,RPW_{TB}\,;\ ReduceAndPartitionWork;\\
detect\,!\,work \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\{voxel\_map, work\} \mid \{\!| detect, sync |\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect\,?\,work \longrightarrow \mathbf{var}\ colls1 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB};\\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1);\\
recColls\,!\,colls1 \longrightarrow \mathbf{skip}\,;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect\,?\,work \longrightarrow \mathbf{var}\ colls2 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB};\\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2);\\
recColls\,!\,colls2 \longrightarrow \mathbf{skip}\,;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect\,?\,work \longrightarrow \mathbf{var}\ colls3 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB};\\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3);\\
recColls\,!\,colls3 \longrightarrow \mathbf{skip}\,;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect\,?\,work \longrightarrow \mathbf{var}\ colls4 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB};\\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4);\\
recColls\,!\,colls4 \longrightarrow \mathbf{skip}\,;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!| output, sync |\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet output\,?\,collisions \longrightarrow\\
\mathbf{var}\ colls : int \bullet \mathbf{wait}\,0\,..\,CC_{TB}\,;\ CalcCollisions;\\
(output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL\,;\ sync \longrightarrow X
\end{array}\right)
\end{array}\right)\\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid\\
\qquad \{\!| detect, output, recColls, sync |\!\} \mid \{collisions\}]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect\,?\,work \longrightarrow \mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet\\
\left(\begin{array}{l}
(recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\,;\ colls1 := x))\,;\\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\,;\ colls2 := x))\,;\\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\,;\ colls3 := x))\,;\\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\,;\ colls4 := x))
\end{array}\right);\\
\mathbf{wait}\,0\,..\,SCFP_{TB};\\
SetCollisionsFromParts([\![colls1, colls2, colls3, colls4]\!]);\\
output\,!\,collisions \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X
\end{array}\right)
\end{array}\right)\\
\quad [\![\{currentFrame, state, voxel\_map, work, collisions\} \mid \{\!| sync |\!\} \mid \varnothing]\!]\\
(\mu X \bullet \mathbf{wait}\ FRAME\_PERIOD\,;\ sync \longrightarrow X)
\end{array}\right) \setminus\\
\quad \{\!| reduce, detect, output, recColls, sync |\!\}
$$

A binary version of the distribution law required above is given in the sequel; it was already presented as rec-sync in [3]. We note that we use a generalised version of this law that deals with n parallel branches.

Circus **Law 11 (lockstep-intro)**

$$(\mu X \bullet (A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2) \; ; \; X) \equiv$$
$$\left(\begin{array}{l}(\mu X \bullet A_1 \; ; \; sync \longrightarrow X) \\ \qquad \llbracket ns_1 \mid cs \cup \{\!| Sync |\!\} \mid ns_2 \rrbracket \\ (\mu X \bullet A_2 \; ; \; sync \longrightarrow X)\end{array}\right) \setminus \{\!| sync |\!\}$$

provided $sync \notin usedC(A_1) \cup usedC(A_2)$ **and** $wrtV(A_1) \cap usedV(A_2) = \varnothing$ **and** $wrtV(A_2) \cap usedV(A_1) = \varnothing$

We omit the presentation of the generalised law, but it is straight forward.

Introduce Handler Actions

Local actions are now introduced for the seven handlers that have emerged.

$$InputFrameHandler \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mu X \bullet (next\_frame ? frame \longrightarrow (\mathbf{wait}\, 0 \,..\, SF_{TB} \; ; \; StoreFrame)) \blacktriangleleft INP\_DL; \\ reduce ! currentFrame ! state \longrightarrow \mathbf{skip} \; ; \; sync \longrightarrow X\end{array}\right)$$

$$ReducerHandler \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mu X \bullet reduce ? currentFrame ? state \longrightarrow \\ \mathbf{wait}\, 0 \,..\, RPW_{TB} \; ; \; ReduceAndPartitionWork; \\ detect ! work \longrightarrow \mathbf{skip} \; ; \; sync \longrightarrow X\end{array}\right)$$

$$DetectorHandler1 \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mu X \bullet detect ? work \longrightarrow \mathbf{var}\; colls1 : int \bullet \mathbf{wait}\, 0 \,..\, CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\ recColls ! colls1 \longrightarrow \mathbf{skip} \; ; \; output ? y \longrightarrow \mathbf{skip} \; ; \; sync \longrightarrow X\end{array}\right)$$

$$DetectorHandler2 \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mu X \bullet detect ? work \longrightarrow \mathbf{var}\; colls2 : int \bullet \mathbf{wait}\, 0 \,..\, CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\ recColls ! colls2 \longrightarrow \mathbf{skip} \; ; \; output ? y \longrightarrow \mathbf{skip} \; ; \; sync \longrightarrow X\end{array}\right)$$

$$DetectorHandler3 \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mu X \bullet detect ? work \longrightarrow \mathbf{var}\; colls3 : int \bullet \mathbf{wait}\, 0 \,..\, CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\ recColls ! colls3 \longrightarrow \mathbf{skip} \; ; \; output ? y \longrightarrow \mathbf{skip} \; ; \; sync \longrightarrow X\end{array}\right)$$

$$DetectorHandler4 \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mu X \bullet detect ? work \longrightarrow \mathbf{var}\; colls4 : int \bullet \mathbf{wait}\, 0 \,..\, CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\ recColls ! colls4 \longrightarrow \mathbf{skip} \; ; \; output ? y \longrightarrow \mathbf{skip} \; ; \; sync \longrightarrow X\end{array}\right)$$

$$OutputCollisionsHandler \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mu X \bullet output ? collisions \longrightarrow \\ \mathbf{var}\; colls : int \bullet \mathbf{wait}\, 0 \,..\, CC_{TB} \; ; \; CalcCollisions; \\ (output\_collisions ! colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL \; ; \; sync \longrightarrow X\end{array}\right)$$

We also introduce a local action for the parallel fragment that controls handler execution.

$$
\begin{array}{l}
\mathit{InteractionHandlers} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{detect} \mathbin{?} \mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 : \mathit{int} \bullet \\
\left(\begin{array}{l}
(\mathit{recColls} \mathbin{?} x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ \mathit{colls}1 := x))\ ; \\
(\mathit{recColls} \mathbin{?} x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ \mathit{colls}2 := x))\ ; \\
(\mathit{recColls} \mathbin{?} x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ \mathit{colls}3 := x))\ ; \\
(\mathit{recColls} \mathbin{?} x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ \mathit{colls}4 := x))
\end{array}\right) ; \\
\mathbf{wait}\ 0 \mathinner{..} \mathit{SCFP}_{TB}; \\
\mathit{SetCollisionsFromParts}(\langle\!\langle\, \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 \,\rangle\!\rangle); \\
\mathit{output} \mathbin{!} \mathit{collisions} \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

Lastly, the cycle time is captured by a further local action.

$$\mathit{Cycle} \mathrel{\widehat=} \left(\mu X \bullet \mathbf{wait}\ \mathit{FRAME\_PERIOD}\ ;\ \ \mathit{sync} \longrightarrow X\right)$$

With this, the *System* action is written in the following manner to bring out the mission structure.

$$
\begin{array}{l}
\mathit{System} \mathrel{\widehat=} \\
\quad \left(\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{InputFrameHandler} \\
\quad [\![\{\mathit{currentFrame}, \mathit{state}\} \mid \{\!|\, \mathit{reduce}, \mathit{sync} \,|\!\} \mid \{\mathit{voxel\_map}, \mathit{work}\}]\!] \\
\mathit{ReducerHandler} \\
\quad [\![\{\mathit{voxel\_map}, \mathit{work}\} \mid \{\!|\, \mathit{detect}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\
\mathit{DetectorHandler}1 \\
\quad [\![\varnothing \mid \{\!|\, \mathit{detect}, \mathit{output}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\
\mathit{DetectorHandler}2 \\
\quad [\![\varnothing \mid \{\!|\, \mathit{detect}, \mathit{output}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\
\mathit{DetectorHandler}3 \\
\quad [\![\varnothing \mid \{\!|\, \mathit{detect}, \mathit{output}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\
\mathit{DetectorHandler}4 \\
\quad [\![\varnothing \mid \{\!|\, \mathit{output}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\
\mathit{OutputCollisionsHandler}
\end{array}\right) \\
\quad [\![\{\mathit{currentFrame}, \mathit{state}, \mathit{voxel\_map}, \mathit{work}\} \mid \\
\qquad \{\!|\, \mathit{detect}, \mathit{output}, \mathit{recColls}, \mathit{sync} \,|\!\} \mid \{\mathit{collisions}\}]\!] \\
\mathit{InteractionHandlers}
\end{array}\right) \\
\quad [\![\{\mathit{currentFrame}, \mathit{state}, \mathit{voxel\_map}, \mathit{work}, \mathit{collisions}\} \mid \{\!|\, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\
\mathit{Cycle}
\right) \setminus \\
\qquad \{\!|\, \mathit{reduce}, \mathit{detect}, \mathit{output}, \mathit{recColls}, \mathit{sync} \,|\!\}
\end{array}
$$

It exhibits the desired shape of a parallelism of handler actions, including two auxiliary actions: one that controls handler execution and another one for the cycle time. Auxiliary control actions are expected at this point, and their elimination is an issue for SH rather than MH. The *Cycle* action is logically attributed to *InputFrameHandler* and will eventually be collapsed with it; for the time being, however, we keep it as a separate parallel branch in order to facilitate the subsequent refinement in the SH phase.

To conclude the account on the MH models, we present the complete process that results from this phase. For this, we have to declare several channels that have been introduced during the course of refinement.

$$\mathbf{channel}\ \mathit{reduce} : \mathit{RawFrame} \times \mathit{StateTable}$$

$$\mathbf{channel}\ \mathit{detect} : \mathit{Partition}$$

$$\mathbf{channel}\ \mathit{output}, \mathit{recColls} : \mathit{int}$$

$$\mathbf{channel}\ \mathit{sync}$$

Whereas *reduce*, *detect*, *output* and *sync* are specification channels, *recColls* later becomes a method channel.

5.2.7 Process

The complete process for the MH phase is presented below. Its state and data operations are in fact the same as those of $CDxE\_MH2B$, the result of Stage 2.

system $CDxE\_MH \mathrel{\widehat{=}}$ **begin**

state $CDxMHState == ECPStateCDx$

$$
\begin{array}{l}
\textit{Init} \\
\hline
CDxMHState' \\
\hline
currentFrame' = \mathbf{new}\ RawFrame \\
state' = \mathbf{new}\ StateTable \\
voxel\_map' = \mathbf{new}\ HashMap[Vector2d, List[Motion]] \\
work' = \mathbf{new}\ Partition(4) \\
collisions' = 0 \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{StoreFrame} \\
\hline
\Delta CDxMHState \\
frame? : Frame \\
\hline
\exists\, posns, posns' : Frame;\ motions, motions' : Frame \mid \\
\quad \mathrm{dom}\, posns = \mathrm{dom}\, motions \land \mathrm{dom}\, posns' = \mathrm{dom}\, motions' \bullet \\
\left(\begin{array}{l}
posns' = frame? \land \\
motions' = \\
\quad \{a : \mathrm{dom}\, posns' \bullet a \mapsto \mathbf{if}\ a \in \mathrm{dom}\, posns\ \mathbf{then}\ (posns'\ a) -_V (posns\ a)\ \mathbf{else}\ ZeroV\} \land \\
posns = F(currentFrame) \land motions = G(currentFrame, state) \land \\
posns' = F(currentFrame') \land motions' = G(currentFrame', state')
\end{array}\right) \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{ReduceAndPartitionWork} \\
\hline
\Delta CDxMHState \\
\hline
currentFrame' = currentFrame \land state' = state \\
\exists\, posns : Frame;\ motions : Frame \mid \mathrm{dom}\, posns = \mathrm{dom}\, motions \bullet \\
\left(\begin{array}{l}
posns = F(currentFrame) \land motions = G(currentFrame, state) \land \\
\left(\begin{array}{l}
\forall\, a_1, a_2 : Aircraft \mid \{a_1, a_2\} \subseteq \mathrm{dom}\, posns \bullet \\
\quad (a_1, a_2) \in CalcCollisionSet(posns, motions) \Rightarrow \\
\qquad \left(\begin{array}{l}
\exists\, l : List[Motion] \mid l \in voxel\_map' . values() . elems() \bullet \\
\quad MkMotion(a_1, posns\ a_1 -_V motions\ a_1, posns\ a_1) \in l . elems() \land \\
\quad MkMotion(a_2, posns\ a_2 -_V motions\ a_2, posns\ a_2) \in l . elems()
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{CalcPartCollisions} \\
\hline
\Xi CDxMHState \\
pcolls! : int \\
i? : 1 \mathinner{\ldotp\ldotp} 4 \\
\hline
pcolls! = \\
\quad \# \left\{\begin{array}{l}
a_1 : Aircraft;\ a_2 : Aircraft \mid \\
\quad \left(\begin{array}{l}
\exists\, l : List[Motion] \mid l \in work . getDetectorWork(i?). elems() \bullet \\
\exists\, v_1, v_2 : Vector;\ w_1, w_2 : Vector \bullet \\
\quad MkMotion(a_1, v_1, w_1) \in l . elems() \land \\
\quad MkMotion(a_2, v_2, w_2) \in l . elems() \land \\
\quad collide((v_1, w_1 -_V v_1), (v_2, w_2 -_V v_2))
\end{array}\right)
\end{array}\right\} \mathrm{div}\ 2 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{SetCollisionsFromParts}} \\
\hline
\Delta \mathit{CDxMHState} \\
\mathit{collsbag}? : \mathrm{bag}\, \mathit{int} \\
\hline
\mathit{currentFrame}' = \mathit{currentFrame} \land \mathit{state}' = \mathit{state} \\
\mathit{voxel\_map}' = \mathit{voxel\_map} \land \mathit{work}' = \mathit{work} \\
\exists\, s : \mathrm{seq}\, \mathit{int} \mid s = \mathit{items}\, \mathit{collsbag}? \bullet \mathit{collisions}' = \Sigma\, s \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{CalcCollisions}} \\
\hline
\Xi \mathit{CDxMHState} \\
\mathit{colls}! : \mathbb{N} \\
\hline
\exists\, \mathit{posns} : \mathit{Frame};\ \mathit{motions} : \mathit{Frame} \mid \mathrm{dom}\, \mathit{posns} = \mathrm{dom}\, \mathit{motions} \bullet \\
\left(\begin{array}{l}
\mathit{posns} = F(\mathit{currentFrame}) \land \mathit{motions} = G(\mathit{currentFrame}, \mathit{state}) \land \\
\exists\, \mathit{collset} : \mathbb{F}\,(\mathit{Aircraft} \times \mathit{Aircraft}) \mid \mathit{collset} = \mathit{CalcCollisionSet}(\mathit{posns}, \mathit{motions}) \bullet \\
\quad (\#\, \mathit{collset} = 0 \land \mathit{colls}! = 0) \lor (\#\, \mathit{collset} > 0 \land \mathit{colls}! \geq (\#\, \mathit{collset})\ \mathrm{div}\ 2)
\end{array}\right) \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\mathit{InputFrameHandler} \;\widehat{=} \\
\quad \left(\begin{array}{l}
\mu X \bullet (\mathit{next\_frame}\,?\,\mathit{frame} \longrightarrow (\mathbf{wait}\, 0 \mathrel{..} \mathit{SF}_{TB}\ ;\ \mathit{StoreFrame})) \blacktriangleleft \mathit{INP\_DL}; \\
\mathit{reduce}\,!\,\mathit{currentFrame}\,!\,\mathit{state} \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{ReducerHandler} \;\widehat{=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{reduce}\,?\,\mathit{currentFrame}\,?\,\mathit{state} \longrightarrow \\
\mathbf{wait}\, 0 \mathrel{..} \mathit{RPW}_{TB}\ ;\ \mathit{ReduceAndPartitionWork}; \\
\mathit{detect}\,!\,\mathit{work} \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{DetectorHandler}1 \;\widehat{=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{detect}\,?\,\mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls}1 : \mathit{int} \bullet \mathbf{wait}\, 0 \mathrel{..} \mathit{CPC}_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}1/\mathit{pcolls}!] \land i? = 1); \\
\mathit{recColls}\,!\,\mathit{colls}1 \longrightarrow \mathbf{skip}\ ;\ \mathit{output}\,?\,y \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{DetectorHandler}2 \;\widehat{=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{detect}\,?\,\mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls}2 : \mathit{int} \bullet \mathbf{wait}\, 0 \mathrel{..} \mathit{CPC}_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}2/\mathit{pcolls}!] \land i? = 2); \\
\mathit{recColls}\,!\,\mathit{colls}2 \longrightarrow \mathbf{skip}\ ;\ \mathit{output}\,?\,y \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{DetectorHandler}3 \;\widehat{=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{detect}\,?\,\mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls}3 : \mathit{int} \bullet \mathbf{wait}\, 0 \mathrel{..} \mathit{CPC}_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}3/\mathit{pcolls}!] \land i? = 3); \\
\mathit{recColls}\,!\,\mathit{colls}3 \longrightarrow \mathbf{skip}\ ;\ \mathit{output}\,?\,y \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{DetectorHandler}4 \;\widehat{=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{detect}\,?\,\mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls}4 : \mathit{int} \bullet \mathbf{wait}\, 0 \mathrel{..} \mathit{CPC}_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}4/\mathit{pcolls}!] \land i? = 4); \\
\mathit{recColls}\,!\,\mathit{colls}4 \longrightarrow \mathbf{skip}\ ;\ \mathit{output}\,?\,y \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{OutputCollisionsHandler} \;\widehat{=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{output}\,?\,\mathit{collisions} \longrightarrow \\
\mathbf{var}\ \mathit{colls} : \mathit{int} \bullet \mathbf{wait}\, 0 \mathrel{..} \mathit{CC}_{TB}\ ;\ \mathit{CalcCollisions}; \\
(\mathit{output\_collisions}\,!\,\mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT\_DL}\ ;\ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
InteractionHandlers \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet detect\,?\,work \longrightarrow \mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\
\left(\begin{array}{l}
(recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ colls1 := x))\ ; \\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ colls2 := x))\ ; \\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ colls3 := x))\ ; \\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ colls4 := x))
\end{array}\right); \\
\mathbf{wait}\ 0\,..\,SCFP_{TB}; \\
SetCollisionsFromParts(\langle\!\langle\, colls1, colls2, colls3, colls4 \,\rangle\!\rangle\,); \\
output\,!\,collisions \longrightarrow \mathbf{skip}\ ;\ \ sync \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
Cycle \mathrel{\widehat=} \big(\mu X \bullet \mathbf{wait}\ FRAME\_PERIOD\ ;\ \ sync \longrightarrow X\big)
$$

$$
\begin{array}{l}
System \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
InputFrameHandler \\
\quad \llbracket \{currentFrame, state\} \mid \{\!|\, reduce, sync \,|\!\} \mid \{voxel\_map, work\} \rrbracket \\
(ReducerHandler \\
\quad \llbracket \{voxel\_map, work\} \mid \{\!|\, detect, sync \,|\!\} \mid \varnothing \rrbracket \\
(DetectorHandler1 \\
\quad \llbracket \varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing \rrbracket \\
(DetectorHandler2 \\
\quad \llbracket \varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing \rrbracket \\
(DetectorHandler3 \\
\quad \llbracket \varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing \rrbracket \\
(DetectorHandler4 \\
\quad \llbracket \varnothing \mid \{\!|\, output, sync \,|\!\} \mid \varnothing \rrbracket \\
OutputCollisionsHandler)))))
\end{array}\right) \\
\quad \llbracket \{currentFrame, state, voxel\_map, work\} \mid \\
\qquad \{\!|\, detect, output, recColls, sync \,|\!\} \mid \{collisions\} \rrbracket \\
InteractionHandlers
\end{array}\right) \\
\quad \llbracket \{currentFrame, state, voxel\_map, work, collisions\} \mid \{\!|\, sync \,|\!\} \mid \varnothing \rrbracket \\
Cycle
\end{array}\right) \setminus \\
\qquad \{\!|\, reduce, detect, output, recColls, sync \,|\!\}
\end{array}
$$

$\bullet$ $Init\ ;\ \ System$

end

Parts of the process that could not be parsed due to limitations of the *Circus* parser in CZT are highlighted.

5.3 Phase SH

The SH phase is subdivided into four stages.

1. Encapsulate shared data of sequential handlers.
2. Encapsulate shared data of concurrent handlers.
3. Introduce data to realise control mechanisms.
4. Collect specification of the memory area data.

This subdivision refines the account in [5]. In this section, we discuss the refinement steps in each stage separately. For this, we require specialised high-level laws which are presented in Section 5.3.1, where we call them 'patterns'. We thus have Pattern 1 and Pattern 2 being used in Stage 1, Pattern 3 being used in Stage 2, and Pattern 4 being used in Stage 3. The refinement patterns are expected to be useful in other case studies too, and apply to action shapes emerging in Stage 4 and Stage 5 of the MH phase.

5.3.1 Patterns

In this section, we present several high-level patterns that are used in the refinements carried out during the SH phase. We also examine the proof of some of those patterns, or at least sketch out a proof strategy.

Pattern 1

The first pattern targets data passing between sequentially executed handlers. It is used in Stage 1 of the SH phase. The action structure we refine results from the application of the law seq-to-par-2 which is typically applied during Stage 4 of the MH phase. We recaptured the shape of this action below.

$$\left(\begin{array}{l}(\mu X \bullet A_1 \,;\; c\,!\,x \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)\\ \quad [\![ns_1 \mid cs \mid ns_2]\!]\\ (\mu X \bullet c\,?\,x \longrightarrow A_2 \,;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, c \,|\!\}$$

where $\{\!|\, c, sync \,|\!\} \subseteq cs \land c \notin usedC(A_1) \cup usedC(A_2)$

Our target for its refinement is the following action.

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1 \,;\; c_1\,!\,x \longrightarrow \mathbf{skip} \,;\; c_3 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)\\ \quad [\![ns_1 \mid (cs \setminus \{\!|\, c \,|\!\}) \cup \{\!|\, c_3 \,|\!\} \mid ns_2]\!]\\ (\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2\,?\,x \longrightarrow A_2 \,;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, c_3 \,|\!\}\\ \quad [\![ns_1 \cup ns_2 \mid \{\!|\, c_1, c_2 \,|\!\} \mid \varnothing]\!]\\ \left(\begin{array}{l}\mathbf{var}\; v : T \bullet\\ \quad \mu X \bullet \left(\begin{array}{l}(c_1\,?\,x \longrightarrow v := x) \,\Box\\ (c_2\,!\,v \longrightarrow \mathbf{skip})\end{array}\right) ;\; X\end{array}\right)\end{array}\right) \setminus \{\!|\, c_1, c_2 \,|\!\}$$

where c_1, c_2 and c_3 are fresh channels

Initially, the channel c fulfils a dual purpose of controlling execution and passing data between the parallel actions. These concerns are disentangled by the refinement: data sharing is realised by the typed channels c_1 and c_2, and the typeless channel c_3 establishes the (sequential) flow of execution. It is possible to deal with the control issue separately; for this, we merely aim for the following intermediate refinement.

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1 \,;\; c_1\,!\,x \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)\\ \quad [\![ns_1 \mid cs \setminus \{\!|\, c \,|\!\} \mid ns_2]\!]\\ (\mu X \bullet c_2\,?\,x \longrightarrow A_2 \,;\; sync \longrightarrow X)\end{array}\right)\\ \quad [\![ns_1 \cup ns_2 \mid \{\!|\, c_1, c_2, sync \,|\!\} \mid \varnothing]\!]\\ \left(\begin{array}{l}\mathbf{var}\; v : T \bullet\\ \quad \mu X \bullet \left(\begin{array}{l}(c_1\,?\,x \longrightarrow v := x) \,\Box\\ (c_2\,!\,v \longrightarrow \mathbf{skip})\end{array}\right) ;\; X\end{array}\right)\\ \quad [\![\varnothing \mid \{\!|\, c_1, c_2 \,|\!\} \mid \varnothing]\!]\\ (\mu X \bullet c_1\,?\,y \longrightarrow c_2\,?\,y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, c_1, c_2 \,|\!\}$$

provided c_1 and c_2 are fresh channels

We then, however, require further refinement steps that introduce the control mechanism via a new channel. Thus here, the pattern breaks down into two steps. The advantage of this approach is that we can account for different strategies for designing control. The disadvantage is that it is not clear (yet) to what extent the refinement of control can be automated, and how much guidance by the user is required for automation.

In Pattern 1, there seems moreover not a notable design space for realising the control aspect; hence, we opt for the solution that uses a single law that already embeds the control mechanism via the synchronisation channel c_3. Intuitively, we can think of this channel in terms of (abstractly) modelling a software event.

High-level Law 1 (seq-share-1)

$$\left(\begin{array}{l}(\mu X \bullet A_1 \;;\; c\,!\,x \longrightarrow \textbf{skip} \;;\; sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs \mid ns_2]\!] \\ (\mu X \bullet c\,?\,x \longrightarrow A_2 \;;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, c \,|\!\}$$

$$\equiv$$

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1 \;;\; c_1\,!\,x \longrightarrow \textbf{skip} \;;\; c_3 \longrightarrow \textbf{skip} \;;\; sync \longrightarrow X) \\ \quad [\![ns_1 \mid (cs \setminus \{\!|\, c \,|\!\}) \cup \{\!|\, c_3 \,|\!\} \mid ns_2]\!] \\ (\mu X \bullet c_3 \longrightarrow \textbf{skip} \;;\; c_2\,?\,x \longrightarrow A_2 \;;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, c_3 \,|\!\} \\ \quad [\![ns_1 \cup ns_2 \mid \{\!|\, c_1, c_2 \,|\!\} \mid \varnothing]\!] \\ \left(\begin{array}{l}\textbf{var}\; v : T \bullet \\ \qquad \mu X \bullet \left(\begin{array}{l}(c_1\,?\,x \longrightarrow v := x) \;\Box \\ (c_2\,!\,v \longrightarrow \textbf{skip})\end{array}\right) \;;\; X\end{array}\right)\end{array}\right) \setminus \{\!|\, c_1, c_2 \,|\!\}$$

provided $\{\!|\, c, sync \,|\!\} \subseteq cs \wedge c \notin usedC(A_1) \cup usedC(A_2)$ **and** c_1, c_2 and c_3 are fresh channels

The right-hand action of the resulting parallelism contributes directly to the *MArea* action. It is worth to examine the proof of this law in more detail as it reveals some common and recurring themes.

Proof of Pattern 1

The proof is done by transforming the left-hand action of the law into the right-hand action using (mostly) elementary refinement laws. We start with the left-hand side of the law.

$$\left(\begin{array}{l}(\mu X \bullet A_1 \;;\; c\,!\,x \longrightarrow \textbf{skip} \;;\; sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs \mid ns_2]\!] \\ (\mu X \bullet c\,?\,x \longrightarrow A_2 \;;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, c \,|\!\}$$

where $\{\!|\, c, sync \,|\!\} \subseteq cs \wedge c \notin usedC(A_1) \cup usedC(A_2)$

The first step replaces the channel c by two channels c_1 and c_2. We require a specialised law for this.

Circus **Law 12 (replace-sync-chan-seq)**

$$\left(\begin{array}{l}(\mu X \bullet A_1 \;;\; c\,!\,x \longrightarrow \textbf{skip} \;;\; sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs \mid ns_2]\!] \\ (\mu X \bullet c\,?\,x \longrightarrow A_2 \;;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, c \,|\!\}$$

$$\equiv$$

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1 \;;\; c_1\,!\,x \longrightarrow \textbf{skip} \;;\; sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs \setminus \{\!|\, c \,|\!\} \mid ns_2]\!] \\ (\mu X \bullet c_2\,?\,x \longrightarrow A_2 \;;\; sync \longrightarrow X)\end{array}\right) \\ \quad [\![ns_1 \cup ns_2 \mid \{\!|\, c_1, c_2, sync \,|\!\} \mid \varnothing]\!] \\ (\mu X \bullet c_1\,?\,x \longrightarrow c_2\,!\,x \longrightarrow \textbf{skip} \;;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, c_1, c_2 \,|\!\}$$

provided $\{\!|\, c, sync \,|\!\} \subseteq cs \wedge c \notin usedC(A_1) \cup usedC(A_2)$ **and** c_1 and c_2 are fresh channels

Strictly, synchronisation on *sync* is not necessary in the right-hand parallel action. However, including it turns out to simplify subsequent refinement steps. In particular, when the right-hand control fragment is later on decomposed and collapsed with other actions the presence of *sync* is useful.

After applying the law we, obtain the following result.

$$\ldots \equiv \text{"application of the law replace-sync-chan-seq"}$$

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1 \,;\; c_1 \,!\, x \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs' \mid ns_2]\!] \\ (\mu X \bullet c_2 \,?\, x \longrightarrow A_2 \,;\; sync \longrightarrow X)\end{array}\right) \\ \quad [\![ns_1 \cup ns_2 \mid \{\!|\, c_1, c_2, sync \,|\!\} \mid \varnothing]\!] \\ (\mu X \bullet c_1 \,?\, x \longrightarrow c_2 \,!\, x \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, c_1, c_2 \,|\!\}$$

where $cs' = cs \setminus \{\!|\, c \,|\!\}$ **and** c_1 and c_2 are fresh channels

We observe that the left-hand action of the outer parallelism already has the correct shape for the *intermediate* target. We therefore focus on the right-hand action. Basically, we want to bring it into a form that resembles the desired shape for the *MArea* action. The strategy for this is to introduce a parallelism in which one action becomes the significant part of *MArea* and the other action a residual part that exercises control.

Before performing this parallelisation, we encapsulate the shared data in a local variable by virtue of four basic laws var-intro, extract-var-prefix, extract-var-seq and extract-var-rec listed in Appendix B.1. This permits the following refinement steps, applied to the left-hand action of the parallelism.

$$\mu X \bullet c_1 \,?\, x \longrightarrow c_2 \,!\, x \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X$$

$\equiv$ "application of the law var-intro"

$$\mu X \bullet c_1 \,?\, x \longrightarrow (\mathbf{var}\, v : T \bullet v := x \,;\; c_2 \,!\, v \longrightarrow \mathbf{skip}) \,;\; sync \longrightarrow X$$

$\equiv$ "application of the law extract-var-prefix"

$$\mu X \bullet (\mathbf{var}\, v : T \bullet c_1 \,?\, x \longrightarrow v := x \,;\; c_2 \,!\, v \longrightarrow \mathbf{skip}) \,;\; sync \longrightarrow X)$$

$\equiv$ "application of the law extract-var-seq"

$$\mu X \bullet (\mathbf{var}\, v : T \bullet c_1 \,?\, x \longrightarrow v := x \,;\; c_2 \,!\, v \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)$$

$\equiv$ "application of the law extract-var-rec"

$$\mathbf{var}\, v : T \bullet \mu X \bullet (c_1 \,?\, x \longrightarrow v := x \,;\; c_2 \,!\, v \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)$$

We next refine the body of the recursion introducing the aforementioned parallelism. The slightly more specific laws required for this are listed below; others are included in Appendix B.1.

The following three laws are variations of distributing a prefix into a parallelism (Law **A.24** in [4]).

Circus **Law 17 (distr-prefix-par-3)**

$$c \longrightarrow \mathbf{skip} \,;\; (\mathbf{skip} \,[\![ns_1 \mid cs \mid ns_2]\!]\, A) \equiv \mathbf{skip} \,[\![ns_1 \mid cs \mid ns_2]\!]\, (c \longrightarrow A)$$

provided $c \notin cs$

Circus **Law 18 (distr-prefix-par-4)**

$$c \,!\, x \longrightarrow \mathbf{skip} \,;\; (A_1 \,[\![ns_1 \mid cs \mid ns_2]\!]\, A_2) \equiv (c \,!\, x \longrightarrow \mathbf{skip} \,;\; A_1) \,[\![ns_1 \mid cs \mid ns_2]\!]\, (c \,?\, y \longrightarrow A_2)$$

provided $c \in cs$ **and** y is not free in A_2

Circus **Law 19 (distr-prefix-par-5)**

$$c \,?\, x \longrightarrow v := x \,;\; (A_1 \,[\![ns_1 \mid cs \mid ns_2]\!]\, A_2) \equiv$$
$$(c \,?\, x \longrightarrow v := x \,;\; A_1) \,[\![ns_1 \cup \{v\} \mid cs \mid ns_2]\!]\, (c \,?\, y \longrightarrow A_2)$$

provided $c \in cs$ **and** v and y are not free in A_2

The following law is important for the steps that establish the recursive shape of $MArea$.

Circus **Law 20 (extchoice-par-intro)**

$$((c \longrightarrow A_1)\ ;\ A_2) \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket (c \longrightarrow A_3)\ \equiv$$
$$(((c \longrightarrow A_1) \mathbin{\Box} (c_1 \longrightarrow B_1) \mathbin{\Box} \ldots \mathbin{\Box} (c_n \longrightarrow B_n))\ ;\ A_2) \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket (c \longrightarrow A_3)$$

provided $c \in cs$ **and** c is distinct from all c_i (the B_i can be chosen arbitrarily)

With the above laws we can proceed with the proof as follows.

$$\mathbf{var}\ v : T \bullet \mu X \bullet (c_1 ? x \longrightarrow v := x\ ;\ c_2 ! v \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X)$$

$\equiv$ "application of the laws seq-skip-left-intro and distr-prefix-seq"

$$\mathbf{var}\ v : T \bullet \mu X \bullet (c_1 ? x \longrightarrow v := x\ ;\ c_2 ! v \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow \mathbf{skip})\ ;\ X$$

$\equiv$ "application of the laws seq-skip-left-intro and par-skip-intro"

$$\mathbf{var}\ v : T \bullet \mu X \bullet \begin{pmatrix} c_1 ? x \longrightarrow v := x\ ;\ c_2 ! v \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow \mathbf{skip}; \\ (\mathbf{skip} \llbracket \varnothing \mid \varnothing \mid \varnothing \rrbracket \mathbf{skip}) \end{pmatrix}\ ;\ X$$

$\equiv$ "application of the law extend-sync-par"

$$\mathbf{var}\ v : T \bullet \mu X \bullet \begin{pmatrix} c_1 ? x \longrightarrow v := x\ ;\ c_2 ! v \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow \mathbf{skip}; \\ (\mathbf{skip} \llbracket \varnothing \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket \mathbf{skip}) \end{pmatrix}\ ;\ X$$

$\equiv$ "application of the law distr-prefix-par-3"

$$\mathbf{var}\ v : T \bullet \mu X \bullet \begin{pmatrix} c_1 ? x \longrightarrow v := x\ ;\ c_2 ! v \longrightarrow \mathbf{skip}; \\ \begin{pmatrix} \mathbf{skip} \\ \quad \llbracket \varnothing \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket \\ sync \longrightarrow \mathbf{skip} \end{pmatrix} \end{pmatrix}\ ;\ X$$

$\equiv$ "application of the law distr-prefix-par-4 and eliminating sequence with **skip**"

$$\mathbf{var}\ v : T \bullet \mu X \bullet \begin{pmatrix} c_1 ? x \longrightarrow v := x; \\ \begin{pmatrix} (c_2 ! v \longrightarrow \mathbf{skip}) \\ \quad \llbracket \varnothing \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket \\ (c_2 ? y \longrightarrow sync \longrightarrow \mathbf{skip}) \end{pmatrix} \end{pmatrix}\ ;\ X$$

$\equiv$ "application of the laws extchoice-par-intro and extchoice-comm"

$$\mathbf{var}\ v : T \bullet \mu X \bullet \begin{pmatrix} c_1 ? x \longrightarrow v := x; \\ \begin{pmatrix} (c_1 ? x \longrightarrow v := x \mathbin{\Box} c_2 ! v \longrightarrow \mathbf{skip}) \\ \quad \llbracket \varnothing \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket \\ (c_2 ? y \longrightarrow sync \longrightarrow \mathbf{skip}) \end{pmatrix} \end{pmatrix}\ ;\ X$$

$\equiv$ "application of the law distr-prefix-par-5"

$$\mathbf{var}\ v : T \bullet \mu X \bullet \begin{pmatrix} \begin{pmatrix} \begin{pmatrix} (c_1 ? x \longrightarrow v := x); \\ (c_1 ? x \longrightarrow v := x \mathbin{\Box} c_2 ! v \longrightarrow \mathbf{skip}) \end{pmatrix} \\ \quad \llbracket \{v\} \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket \\ (c_1 ? y \longrightarrow c_2 ? y \longrightarrow sync \longrightarrow \mathbf{skip}) \end{pmatrix} \end{pmatrix}\ ;\ X$$

$\equiv$ "application of the laws extchoice-par-intro and extchoice-comm"

$$\mathbf{var}\ v : T \bullet \mu X \bullet \begin{pmatrix} \begin{pmatrix} (c_1 ? x \longrightarrow v := x \mathbin{\Box} c_2 ! v \longrightarrow \mathbf{skip}); \\ (c_1 ? x \longrightarrow v := x \mathbin{\Box} c_2 ! v \longrightarrow \mathbf{skip}) \end{pmatrix} \\ \quad \llbracket \{v\} \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket \\ (c_1 ? y \longrightarrow c_2 ? y \longrightarrow sync \longrightarrow \mathbf{skip}) \end{pmatrix}\ ;\ X$$

We now use a specialised law to distribute the recursion into the parallelism. This law is somewhat similar to rec-sync in [2], however lock-step progress is achieve by a synchronisation at the start of the recursion rather than the end. We first recapture rec-sync in [2], used later on too, and name it distr-rec-par-1.

Circus **Law 25 (distr-rec-par-1)**

$$
\begin{array}{l}
\mu X \bullet (A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2) \,;\; c \longrightarrow X \;\equiv \\
(\mu X \bullet A_1 \,;\; c \longrightarrow X) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (\mu X \bullet A_2 \,;\; c \longrightarrow X) \\
\textbf{provided } c \in cs \;\textbf{ and }\; c \notin usedC(A_1) \cup usedC(A_2) \;\textbf{ and} \\
wrtV(A_1) \cap usedV(A_2) = \varnothing \;\textbf{ and }\; wrtV(A_2) \cap usedV(A_1)
\end{array}
$$

We next give the alternative version that we require for the transformation in the sequel.

Circus **Law 26 (distr-rec-par-2)**

$$
\begin{array}{l}
\mu X \bullet (((c_1 \longrightarrow A_1 \,\Box\, c_2 \longrightarrow A_2) \,;\; A_3) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (c_1 \longrightarrow A_4)) \,;\; X \;\equiv \\
(\mu X \bullet (c_1 \longrightarrow A_1 \,\Box\, c_2 \longrightarrow A_2) \,;\; A_3 \,;\; X) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (\mu X \bullet c_1 \longrightarrow A_4 \longrightarrow X) \\
\textbf{provided } \{c_1, c_2\} \subseteq cs \;\textbf{ and }\; c_1 \notin usedC(A_i) \text{ for all } i \in \{1,2,3,4\} \;\textbf{ and} \\
(wrtV(A_1) \cup wrtV(A_2) \cup wrtV(A_3)) \cap usedV(A_4) = \varnothing \;\textbf{ and} \\
wrtV(A_4) \cap (usedV(A_1) \cup usedV(A_2) \cup usedV(A_3)) = \varnothing
\end{array}
$$

The external choice in the left-hand branch of the parallel action is another elaboration we require to use the law. It is not a problem as it is a Hobson's choice in the context of the right-hand parallel action.

We thus obtain the following refinement.

$\equiv$ "application of the law distr-rec-par-2" Problem: *What about the provisos $c_1 \notin usedC(A_3)$?*

$$
\mathbf{var}\; v : T \bullet \left(\begin{array}{l}
\left(\mu X \bullet \left(\begin{array}{l}(c_1\,?\,x \longrightarrow v := x \,\Box\, c_2\,!\,v \longrightarrow \mathbf{skip}); \\ (c_1\,?\,x \longrightarrow v := x \,\Box\, c_2\,!\,v \longrightarrow \mathbf{skip})\end{array}\right) ;\; X\right) \\
\quad \llbracket \{v\} \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket \\
(\mu X \bullet c_1\,?\,y \longrightarrow c_2\,?\,y \longrightarrow sync \longrightarrow X)
\end{array}\right)
$$

The application of extchoice-par-intro now reveals its purpose of putting the left-hand action of the parallelism into a form $\mu X \bullet A \,;\; A \,;\; X$. The following law simplifies it eliminating repeated actions A.

Circus **Law 27 (elim-repeated-seq-rec)**

$$
\mu X \bullet A \,;\; A \,;\; \ldots \,;\; A \,;\; X \;\equiv\; \mu X \bullet A \,;\; X
$$

We thus obtain the action below in the next refinement step.

$\equiv$ "application of the law elim-repeated-seq-rec"

$$
\mathbf{var}\; v : T \bullet \left(\begin{array}{l}
\left(\mu X \bullet \left(\begin{array}{l}(c_1\,?\,x \longrightarrow v := x) \,\Box \\ (c_2\,!\,v \longrightarrow \mathbf{skip})\end{array}\right) ;\; X\right) \\
\quad \llbracket \{v\} \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket \\
(\mu X \bullet \; c_1\,?\,y \longrightarrow c_2\,?\,y \longrightarrow sync \longrightarrow X)
\end{array}\right)
$$

The only remaining task is to distribute the local variable block into the left-hand action of the parallelism. This is achieved by the basic laws distr-var-par and var-elim given in Appendix B.1. This produces the two parallel fragments of the *intermediate* result presented at the beginning of the section.

$\equiv$ "application of the laws distr-var-par and var-elim and adjusting write sets of the parallelism"

$$\left(\begin{array}{l}\left(\begin{array}{l}\mathbf{var}\ v : T \bullet \\ \quad \mu X \bullet \left(\begin{array}{l}(c_1 ? x \longrightarrow v := x)\ \Box \\ (c_2 ! v \longrightarrow \mathbf{skip})\end{array}\right) ;\ X\end{array}\right) \\ \quad [\![\varnothing \mid \{\!|\ c_1, c_2\ |\!\} \mid \varnothing]\!] \\ \left(\mu X \bullet\ c_1 ? y \longrightarrow c_2 ? y \longrightarrow sync \longrightarrow X\right)\end{array}\right)$$

The left-hand action of the parallelism now has the desired shape for $MArea$. The right-hand action encapsulates control of execution, however, without any concerns for shared data. This part needs to be further refined by introducing a basic channel to establish the necessary control between the two handlers. Therefore, we ignore the left-hand action for now and continue refining the right-hand action in combination with the parallelism of handlers. The proof tactic as is follows.

1. Introduce a fresh hidden channel c_3 in the control branch and extend its scope.
2. Decompose the control branch into a parallelism of smaller fragments.
3. Match and collapse these parallel fragments suitably with the handlers.

These steps are fairly straight-forward and do not require specialised laws, apart from step laws to introduce and collapse parallelism. More importantly, they reveal a general strategy for eliminating control fragments.

Below we recapture the current (intermediate) result of the refinement steps so far.

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1 ;\ c_1 ! x \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs' \mid ns_2]\!] \\ (\mu X \bullet c_2 ? x \longrightarrow A_2 ;\ sync \longrightarrow X)\end{array}\right) \\ \quad [\![ns_1 \cup ns_2 \mid \{\!|\ c_1, c_2, sync\ |\!\} \mid \varnothing]\!] \\ \left(\begin{array}{l}\mathbf{var}\ v : T \bullet \\ \quad \mu X \bullet \left(\begin{array}{l}(c_1 ? x \longrightarrow v := x)\ \Box \\ (c_2 ! v \longrightarrow \mathbf{skip})\end{array}\right) ;\ X\end{array}\right) \\ \quad [\![\varnothing \mid \{\!|\ c_1, c_2\ |\!\} \mid \varnothing]\!] \\ \left(\mu X \bullet\ c_1 ? y \longrightarrow c_2 ? y \longrightarrow sync \longrightarrow X\right)\end{array}\right) \setminus \{\!|\ c_1, c_2\ |\!\}$$

$$\begin{array}{l}\mathbf{where}\ \{\!|\ c, sync\ |\!\} \subseteq cs \land c \notin usedC(A_1) \cup usedC(A_2)\ \ \mathbf{and} \\ cs' = cs \setminus \{\!|\ c\ |\!\}\ \ \mathbf{and}\ \ c_1 \text{ and } c_2 \text{ are fresh channels}\end{array}$$

The focus for the remaining part of the proof is the fragment below which we extract from this action.

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1 ;\ c_1 ! x \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs' \mid ns_2]\!] \\ (\mu X \bullet c_2 ? x \longrightarrow A_2 ;\ sync \longrightarrow X)\end{array}\right) \\ \quad [\![ns_1 \cup ns_2 \mid \{\!|\ c_1, c_2, sync\ |\!\} \mid \varnothing]\!] \\ \left(\mu X \bullet\ c_1 ? y \longrightarrow c_2 ? y \longrightarrow sync \longrightarrow X\right)\end{array}\right)$$

Thus, we ignore the middle action which already converged into the desirable shape for $MArea$.

We first introduce the typeless control channel c_3.

$\ldots \equiv$ "introduction of a hidden communication on a new channel c_3 and extracting its hiding"

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1\ ;\ c_1 ! x \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs' \mid ns_2]\!] \\ (\mu X \bullet c_2 ? x \longrightarrow A_2\ ;\ sync \longrightarrow X)\end{array}\right) \\ \quad [\![ns_1 \cup ns_2 \mid \{\!| c_1, c_2, sync \mid \varnothing]\!] \\ \left(\mu X \bullet\ c_1 ? y \longrightarrow c_3 \longrightarrow c_2 ? y \longrightarrow sync \longrightarrow X\right)\end{array}\right) \setminus \{\!| c_3 |\!\}$$

The derivation might use the following law to introduce the prefix at the right place.

Circus **Law 29 (hidden-sync-intro)**

$$A \equiv (c \longrightarrow A) \setminus \{\!| c |\!\}\ \ \mathbf{provided}\ \ c \notin usedC(A)$$

The remaining steps merely extract the concealment of c_3 from the prefixes, recursion and parallelism using the elementary laws extract-hide-prefix, extract-hide-rec and extract-hide-par-right in Appendix B.1.

We once again use (custom) step laws to introduce a parallelism inside the right-hand recursion with the objective of splitting the recursion into a parallelism of two recursions of which each one is collapsed with one of the handlers. Below we just give the result after introducing the parallelism.

$\sqsubseteq$ "introduction of parallelism using (custom) parallel step laws"

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1\ ;\ c_1 ! x \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs' \mid ns_2]\!] \\ (\mu X \bullet c_2 ? x \longrightarrow A_2\ ;\ sync \longrightarrow X)\end{array}\right) \\ \quad [\![ns_1 \cup ns_2 \mid \{\!| c_1, c_2, sync |\!\} \mid \varnothing]\!] \\ \left(\mu X \bullet \left(\begin{array}{l}(c_1 ? y \longrightarrow \mathbf{skip}\ ;\ c_3 \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X) \\ \quad [\![\varnothing \mid \{\!| c_3 |\!\} \mid \varnothing]\!] \\ (c_3 \longrightarrow \mathbf{skip}\ ;\ c_2 ? y \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X)\end{array}\right)\right)\end{array}\right) \setminus \{\!| c_3 |\!\}$$

$\sqsubseteq$ "application of the law distr-rec-par-1 to distribution the recursion into the parallelism"

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1\ ;\ c_1 ! x \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs' \mid ns_2]\!] \\ (\mu X \bullet c_2 ? x \longrightarrow A_2\ ;\ sync \longrightarrow X)\end{array}\right) \\ \quad [\![ns_1 \cup ns_2 \mid \{\!| c_1, c_2, sync |\!\} \mid \varnothing]\!] \\ \left(\begin{array}{l}(\mu X \bullet c_1 ? y \longrightarrow \mathbf{skip}\ ;\ c_3 \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X) \\ \quad [\![\varnothing \mid \{\!| c_3 |\!\} \mid \varnothing]\!] \\ (\mu X \bullet c_3 \longrightarrow \mathbf{skip}\ ;\ c_2 ? y \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X)\end{array}\right)\end{array}\right) \setminus \{\!| c_3 |\!\}$$

$\sqsubseteq$ "reordering parallel actions matching them with a control fragment"

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1\ ;\ c_1 ! x \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X) \\ \quad [\![ns_1 \mid \{\!| c_1, sync |\!\} \mid \varnothing]\!] \\ (\mu X \bullet c_1 ? y \longrightarrow \mathbf{skip}\ ;\ c_3 \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X)\end{array}\right) \\ \quad [\![ns_1 \mid cs' \cup \{\!| c_3 |\!\} \mid ns_2]\!] \\ \left(\begin{array}{l}(\mu X \bullet c_2 ? x \longrightarrow A_2\ ;\ sync \longrightarrow X) \\ \quad [\![ns_2 \mid \{\!| c_2, sync |\!\} \mid \varnothing]\!] \\ (\mu X \bullet c_3 \longrightarrow \mathbf{skip}\ ;\ c_2 ? y \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X)\end{array}\right)\end{array}\right) \setminus \{\!| c_3 |\!\}$$

$\sqsubseteq$ "extracting recursions from both parallelisms (symmetric law of distr-rec-par-1)"

$$\left(\begin{array}{l}\left(\mu X \bullet \left(\begin{array}{l}(A_1\ ;\ \ c_1 \,!\, x \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X)\\ \qquad [\![ns_1 \mid \{\!|\, c_1, \mathit{sync} \,|\!\} \mid \varnothing]\!]\\ (c_1 \,?\, y \longrightarrow \mathbf{skip}\ ;\ \ c_3 \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X)\end{array}\right)\right)\\ \quad [\![ns_1 \mid cs' \cup \{\!|\, c_3 \,|\!\} \mid ns_2]\!]\\ \left(\mu X \bullet \left(\begin{array}{l}(c_2 \,?\, x \longrightarrow A_2\ ;\ \ \mathit{sync} \longrightarrow X)\\ \qquad [\![ns_2 \mid \{\!|\, c_2, \mathit{sync} \,|\!\} \mid \varnothing]\!]\\ (c_3 \longrightarrow \mathbf{skip}\ ;\ \ c_2 \,?\, y \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X)\end{array}\right)\right)\end{array}\right) \setminus \{\!|\, c_3 \,|\!\}$$

$\sqsubseteq$ "collapsing of parallel actions using step laws; this exploits that c_1, c_2 and c_3 are fresh"

$$\left(\begin{array}{l}\big(\mu X \bullet A_1\ ;\ \ c_1 \,!\, x \longrightarrow \mathbf{skip}\ ;\ \ c_3 \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X\big)\\ \quad [\![ns_1 \mid cs' \cup \{\!|\, c_3 \,|\!\} \mid ns_2]\!]\\ \big(\mu X \bullet c_3 \longrightarrow \mathbf{skip}\ ;\ \ c_2 \,?\, x \longrightarrow A_2\ ;\ \ \mathit{sync} \longrightarrow X\big)\end{array}\right) \setminus \{\!|\, c_3 \,|\!\}$$

Injecting the result back into the context of the refined action.

$$\left(\begin{array}{l}\left(\begin{array}{l}\big(\mu X \bullet A_1\ ;\ \ c_1 \,!\, x \longrightarrow \mathbf{skip}\ ;\ \ c_3 \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X\big)\\ \quad [\![ns_1 \mid cs' \cup \{\!|\, c_3 \,|\!\} \mid ns_2]\!]\\ \big(\mu X \bullet c_3 \longrightarrow \mathbf{skip}\ ;\ \ c_2 \,?\, x \longrightarrow A_2\ ;\ \ \mathit{sync} \longrightarrow X\big)\end{array}\right) \setminus \{\!|\, c_3 \,|\!\}\\ \quad [\![ns_1 \cup ns_2 \mid \{\!|\, c_1, c_2 \,|\!\} \mid \varnothing]\!]\\ \left(\begin{array}{l}\mathbf{var}\ v : T \bullet\\ \quad \mu X \bullet \left(\begin{array}{l}(c_1 \,?\, x \longrightarrow v := x)\ \Box\\ (c_2 \,!\, v \longrightarrow \mathbf{skip})\end{array}\right)\ ;\ \ X\end{array}\right)\end{array}\right) \setminus \{\!|\, c_1, c_2 \,|\!\}$$

$$\mathbf{where}\ cs' = cs \setminus \{\!|\, c \,|\!\}\ \ \mathbf{and}\ \ c_1 \text{ and } c_2 \text{ are fresh channels}$$

This is exactly the right-hand side of the law and thus concludes the proof. We next look at a generalisation of this law that furthermore turns out to be required in our case study.

Pattern 2

In our case study, we may anticipate to apply Pattern 1 three times, mirroring three applications of seq-to-par-2 in Stage 4 of the MH phase. It turns out though that seq-share-1 is too specific to be applied in all three cases where data is passed between sequential handlers. This is due to inhomogeneities introduced in Stage 5, more specifically during the merge sub-step concluding that stage. We consider the fragment

$$\begin{array}{l}
\left(\begin{array}{l}\mu X \bullet reduce\,?\,currentFrame\,?\,state \longrightarrow \\ \mathbf{wait}\,0\,..\,RPW_{TB}\,;\;\; ReduceAndPartitionWork; \\ detect\,!\,work \longrightarrow \mathbf{skip}\,;\;\; sync \longrightarrow X\end{array}\right) \\
\quad \llbracket \{voxel_map, work\} \mid \{\!|\, sync, detect \,|\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}\mu X \bullet detect\,?\,work \longrightarrow \mathbf{var}\; colls1 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\ recColls\,!\,colls1 \longrightarrow \mathbf{skip}\,;\;\; output\,?\,y \longrightarrow \mathbf{skip}\,;\;\; sync \longrightarrow X\end{array}\right) \\
\quad \llbracket \varnothing \mid \{\!|\, sync, detect, output \,|\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}\mu X \bullet detect\,?\,work \longrightarrow \mathbf{var}\; colls2 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\ recColls\,!\,colls2 \longrightarrow \mathbf{skip}\,;\;\; output\,?\,y \longrightarrow \mathbf{skip}\,;\;\; sync \longrightarrow X\end{array}\right) \\
\quad \llbracket \varnothing \mid \{\!|\, sync, detect, output \,|\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}\mu X \bullet detect\,?\,work \longrightarrow \mathbf{var}\; colls3 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\ recColls\,!\,colls3 \longrightarrow \mathbf{skip}\,;\;\; output\,?\,y \longrightarrow \mathbf{skip}\,;\;\; sync \longrightarrow X\end{array}\right) \\
\quad \llbracket \varnothing \mid \{\!|\, sync, detect, output \,|\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}\mu X \bullet detect\,?\,work \longrightarrow \mathbf{var}\; colls4 : int \bullet \mathbf{wait}\,0\,..\,CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\ recColls\,!\,colls4 \longrightarrow \mathbf{skip}\,;\;\; output\,?\,y \longrightarrow \mathbf{skip}\,;\;\; sync \longrightarrow X\end{array}\right)
\end{array}$$

from the *System* action. The data here is transmitted through the *detect* channel, however, with *four* synchronising actions at the receiving end. Also the synchronisations on *output* within the detection handlers are an issue. The sender in this case is the control action *InteractionHandlers*, omitted above.

This highlights the need for further laws to introduce sharing in sequential handler actions. A generalised version of seq-share-2 accounts for a possible parallelism of handlers concurrently reading the data.

High-level Law 2 (seq-share-2)

$$\left(\begin{array}{l}
(\mu X \bullet A_1\,;\;\; c\,!\,x \longrightarrow \mathbf{skip}\,;\;\; sync \longrightarrow X) \\
\quad \llbracket ns_1 \mid cs_1 \mid ns_2 \cup ns_3 \cup \ldots \cup ns_n \rrbracket \\
(\mu X \bullet c\,?\,x \longrightarrow A_2\,;\;\; sync \longrightarrow X) \\
\quad \llbracket ns_2 \mid cs_2 \mid ns_3 \cup ns_4 \cup \ldots \cup ns_n \rrbracket \\
(\mu X \bullet c\,?\,x \longrightarrow A_3\,;\;\; sync \longrightarrow X) \\
\quad \ldots \\
\quad \llbracket ns_{n-1} \mid cs_{n-1} \mid ns_n \rrbracket \\
(\mu X \bullet c\,?\,x \longrightarrow A_n\,;\;\; sync \longrightarrow X)
\end{array}\right) \setminus \{\!|\, c \,|\!\}$$

$$\equiv$$

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
(\mu X \bullet A_1 \,;\; c_1 ! x \longrightarrow \mathbf{skip} \,;\; c_3 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad [\![ns_1 \mid (cs_1 \setminus \{\!| c |\!\}) \cup \{\!| c_3 |\!\} \mid ns_2 \cup ns_3 \cup \ldots \cup ns_n]\!] \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2 ? x \longrightarrow A_2 \,;\; sync \longrightarrow X) \\
\quad [\![ns_2 \mid (cs_2 \setminus \{\!| c |\!\}) \cup \{\!| c_3 |\!\} \mid ns_3 \cup ns_4 \cup \ldots \cup ns_n]\!] \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2 ? x \longrightarrow A_3 \,;\; sync \longrightarrow X) \\
\quad \ldots \\
\quad [\![ns_{n-1} \mid (cs_{n-1} \setminus \{\!| c |\!\}) \cup \{\!| c_3 |\!\} \mid ns_n]\!] \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2 ? x \longrightarrow A_n \,;\; sync \longrightarrow X)
\end{array}\right) \setminus \{\!| c_3 |\!\} \\
\quad [\![ns_1 \cup ns_2 \cup \ldots \cup ns_n \mid \{\!| c_1, c_2 |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathbf{var}\; v : T \bullet \\
\quad \mu X \bullet \left(\begin{array}{l} (c_1 ? x \longrightarrow v := x) \;\Box \\ (c_2 ! v \longrightarrow \mathbf{skip}) \end{array}\right) \,;\; X
\end{array}\right)
\end{array}\right) \setminus \{\!| c_1, c_2 |\!\}
$$

provided $\{\!| c, sync |\!\} \subseteq cs_i \wedge c \notin usedC(A_i)$ for $i \in 1 \,..\, n$ **and** c_1, c_2 and c_3 are fresh channels

A detailed proof of this law is omitted but is expected to be very similar to the one for seq-share-1, up to the point where we match and collapse the control fragment with the handlers actions.

Below, we recapture the control fragment that is collapsed in the proof of the law seq-share-1.

$$
\left(\begin{array}{l}
(\mu X \bullet c_1 ? y \longrightarrow \mathbf{skip} \,;\; c_3 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad [\![\varnothing \mid \{\!| c_3 |\!\} \mid \varnothing]\!] \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2 ? y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)
\end{array}\right)
$$

In the proof of seq-share-2, we require an additional step that duplicates the right-hand action using idempotency of parallel composition. The respective law is.

Circus **Law 34 (idem-par)**

$$A \;\equiv\; (A \,[\![\varnothing \mid usedC(A) \mid \varnothing]\!]\, A) \;\; \textbf{provided} \;\; wrtV(A) = \varnothing \;\; \textbf{and} \;\; A \text{ is deterministic}$$

Using the law, we refine the control fragment before matching and collapsing it with the handlers.

$\ldots \;\equiv\;$ "multiple applications of the law idem-par"

$$
\left(\begin{array}{l}
(\mu X \bullet c_1 ? y \longrightarrow \mathbf{skip} \,;\; c_3 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad [\![\varnothing \mid \{\!| c_3 |\!\} \mid \varnothing]\!] \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2 ? y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad [\![\varnothing \mid \{\!| c_2, c_3, sync |\!\} \mid \varnothing]\!] \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2 ? y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \ldots \\
\quad [\![\varnothing \mid \{\!| c_2, c_3, sync |\!\} \mid \varnothing]\!] \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2 ? y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)
\end{array}\right)
$$

Hereafter we proceed in essence in the same way as in the proof of seq-share-1, reordering the parallelisms to match each control action with a handler action and use step laws to collapse those parallel actions.

Pattern 3

The third pattern targets the refinement of the control action *InteractionHandlers*. This will become an issue for Stage 2 of the SH phase where we encapsulate shared data that is concurrently accessed.

$$
\begin{array}{l}
\mathit{InteractionHandlers} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{detect}\,?\,\mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 : \mathit{int} \bullet \\
\left(\begin{array}{l}
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\mathrel{..}RC_{TB}\ ;\ \ \mathit{colls}1 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\mathrel{..}RC_{TB}\ ;\ \ \mathit{colls}2 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\mathrel{..}RC_{TB}\ ;\ \ \mathit{colls}3 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\mathrel{..}RC_{TB}\ ;\ \ \mathit{colls}4 := x))
\end{array}\right); \\
\mathbf{wait}\ 0\mathrel{..}\mathit{SCFP}_{TB}; \\
\mathit{SetCollisionsFromParts}([\![\, \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 \,]\!]); \\
\mathit{output}\,!\,\mathit{collisions} \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

This action emerges as a residual control fragment during the refinement in Stage 5 of the MH phase. Rather than defining a law that applies to actions of the above shape, we formulate a more general law that is more likely to be reusable. It besides abstracts from the actual number of parallel handlers.

High-level Law 3 (par-share)

$$
\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ v : T \bullet \\
\qquad \left(\begin{array}{l}
\mu X \bullet\ \mathit{start} \longrightarrow \mathbf{wait}\ 0\mathrel{..}\mathit{Init}_{TB}\ ;\ \ \mathit{InitOp}; \\
\mathbf{var}\ x_1, x_2, \ldots, x_n : T \bullet \\
\left(\begin{array}{l}
(\mathit{record}\,?\,x \longrightarrow (\mathbf{wait}\ 0\mathrel{..}RC_{TB}\ ;\ \ x_1 := x)); \\
(\mathit{record}\,?\,x \longrightarrow (\mathbf{wait}\ 0\mathrel{..}RC_{TB}\ ;\ \ x_2 := x)); \\
\qquad \ldots \\
(\mathit{record}\,?\,x \longrightarrow (\mathbf{wait}\ 0\mathrel{..}RC_{TB}\ ;\ \ x_n := x)); \\
\mathbf{wait}\ 0\mathrel{..}\mathit{Merge}_{TB}\ ;\ \ \mathit{MergeOp}([\![\, x_1, x_2, \ldots, x_n \,]\!]); \\
\mathit{output}\,!\,v \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\quad \sqsubseteq \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ v : T \bullet \\
\qquad \mu X \bullet \left(\begin{array}{l}
\mathit{init} \longrightarrow (\mathbf{wait}\ 0\mathrel{..}\mathit{Init}_{TB}\ ;\ \ \mathit{InitOp})\ \Box \\
\mathit{record}\,?\,x \longrightarrow (\mathbf{wait}\ 0\mathrel{..}RC_{TB}\ ;\ \ \mathit{MergeOp}([\![\, x \,]\!]))\ \Box \\
\mathit{output}\,!\,v \longrightarrow \mathbf{skip}
\end{array}\right);\ \ X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!|\, \mathit{init}, \mathit{record}, \mathit{output} \,|\!\} \mid \varnothing]\!] \\
\left(\mu X \bullet \mathit{init} \longrightarrow \mathit{start} \longrightarrow \left(\begin{array}{l}
(\mathit{record}\,?\,y \longrightarrow \mathbf{skip})\ |\!|\!| \\
(\mathit{record}\,?\,y \longrightarrow \mathbf{skip})\ |\!|\!| \\
\qquad \ldots \\
(\mathit{record}\,?\,y \longrightarrow \mathbf{skip})
\end{array}\right);\ \ \mathit{output}\,?\,y \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X\right)
\end{array}\right) \setminus \\
\quad \{\!|\, \mathit{init} \,|\!\}
\end{array}
$$

provided *InitOp* and *MergeOp* are data operations **and**

$\mathit{wrtV}(\mathit{InitOp}) = \{v\} = \mathit{wrtV}(\mathit{MergeOp})$ **and** $\mathit{MergeOp}(b_1 \uplus b_2) = \mathit{MergeOp}(b_1)\ ;\ \mathit{MergeOp}(b_2)$

We observe that the time budget $\mathbf{wait}\ 0\mathrel{..}\mathit{Merge}_{TB}$ is removed by the law, assuming that the time budget $\mathbf{wait}\ 0\mathrel{..}RC_{TB}$ already subsumes the time require for the merge operation. This models a program design in which the merge is done incrementally with each call of a method that records a partial result.

The law assumes that the merge operation ($\mathit{SetCollisionsFromParts}$ in the CD_x) is can be expressed in terms of a sequence $\mathit{InitOp} \mathbin{;} \mathit{MergeOp}$. For the CD_x refinement, the respective decomposition is as follows.

$$\begin{array}{l}
\textit{InitOp} \\
\hline
\Delta \mathit{CDxMHState} \\
\hline
\mathit{currentFrame}' = \mathit{currentFrame} \land \mathit{state}' = \mathit{state} \land \mathit{voxel_map}' = \mathit{voxel_map} \land \mathit{work}' = \mathit{work} \\
\mathit{collisions}' = 0 \\
\hline
\end{array}$$

$$\begin{array}{l}
\textit{MergeOp} \\
\hline
\Delta \mathit{CDxMHState} \\
\mathit{collsbag}? : \mathrm{bag}\ \mathit{int} \\
\hline
\mathit{currentFrame}' = \mathit{currentFrame} \land \mathit{state}' = \mathit{state} \land \mathit{voxel_map}' = \mathit{voxel_map} \land \mathit{work}' = \mathit{work} \\
\exists\, s : \mathrm{seq}\ \mathit{int} \mid s = \mathit{items}\ \mathit{collsbag}? \bullet \mathit{collisions}' = \mathit{collisions} + \Sigma\, s \\
\hline
\end{array}$$

The necessary proof is to show that $\mathit{SetCollisionsFromParts}(cb) = \mathit{InitOp}; \mathit{MergeOp}(cb)$. This is not difficult by eliminating the schema sequence using the one-point rule. The reason we require manual decomposition prior to applying the law is that it seems not possible to derive the initialisation and step-wise merge operation automatically, for instance, from $\mathit{SetCollisionsFromParts}$. Thus, this transformation has to be done by the user but in practical terms this should in most cases not be difficult.

The left-hand action of the result of the law contributes to MArea. Channels may be renamed and further decomposed during the AR phase into Call and Ret pairs to correspond to methods in the program. The right-hand action is a control fragment that needs to be decomposed and distributed into the parallelism of handlers. Again, there is an issue of control versus sharing. The law in this case does not attempt to commit to a particular control mechanism but merely designs access to the shared data. The strategy for eliminating the control action is exactly as illustrated in the proof of the law seq-share-1.

The proof of par-share shall not be discussed in detail here. This is future work for the time being but one may expect similar themes to emerge as in the proofs of the previous laws for refinement patterns.

Discussion

The difference between seq-share-1 / seq-share-2 and par-share is that the control action is implicitly eliminated in the former laws whereas in the latter law it persists. We could potentially define a more high-level version of par-share that takes the context of the refined action into account and aggregates the elimination of the residual parallelism. But on the other hand, this might restrict applicability of the law in the general case.

The important conclusion we draw is that for modularity, we require a general tactic, preferably automated, to eliminate control actions. There seem three obvious approaches for this.

1. Define sufficiently high-level laws that do not give rise to such actions in the first place.

2. Have specialised high-level laws that eliminate them in the context of parallel handler actions.

3. Have a generic strategy that eliminates them, possibly modulo guidance by the user.

In terms of automation, approach (1) and (2) seem most promising. We also see that (2) improves modularity in comparison to (1). In terms of genericity, (3) seems more powerful than the other approaches. I cannot see a clear strategy for approach (3) yet though and there are various open issues, for instance, with regards to the information that the user has to provide during refinement. This is a challenge for future work.

Below we present a corresponding law for approach (2) to eliminate the control action that emerges from application of the law par-share. It applies in the context of $n+1$ parallel handlers.

High-level Law 4 (par-share-control)

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
(\mu X \bullet A \,;\; start \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns \mid \lbag start, sync \rbag \mid \varnothing \rrbracket \\
(\mu X \bullet start \longrightarrow \mathbf{var}\; v : T \bullet A_1 \,;\; record \,!\, v \longrightarrow \mathbf{skip} \,;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket \varnothing \mid \lbag start, output, sync \rbag \mid \varnothing \rrbracket \\
(\mu X \bullet start \longrightarrow \mathbf{var}\; v : T \bullet A_2 \,;\; record \,!\, v \longrightarrow \mathbf{skip} \,;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \ldots \\
\quad \llbracket \varnothing \mid \lbag start, output, sync \rbag \mid \varnothing \rrbracket \\
(\mu X \bullet start \longrightarrow \mathbf{var}\; v : T \bullet A_n \,;\; record \,!\, v \longrightarrow \mathbf{skip} \,;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)
\end{array}\right) \\
\quad \llbracket ns \mid \lbag start, record, output, sync \rbag \mid \varnothing \rrbracket \\
\left(\mu X \bullet init \longrightarrow start \longrightarrow \left(\begin{array}{l}
(record \,?\, y \longrightarrow \mathbf{skip}) \interleave \\
(record \,?\, y \longrightarrow \mathbf{skip}) \interleave \\
\quad \ldots \\
(record \,?\, y \longrightarrow \mathbf{skip})
\end{array}\right) ;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X \right)
\end{array}\right)
$$

$$\sqsubseteq$$

$$
\left(\begin{array}{l}
(\mu X \bullet A \,;\; init \longrightarrow \mathbf{skip} \,;\; start \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns \mid \lbag start, sync \rbag \mid \varnothing \rrbracket \\
(\mu X \bullet start \longrightarrow \mathbf{var}\; v : T \bullet A_1 \,;\; record \,!\, v \longrightarrow \mathbf{skip} \,;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns \mid \lbag start, output, sync \rbag \mid \varnothing \rrbracket \\
(\mu X \bullet start \longrightarrow \mathbf{var}\; v : T \bullet A_2 \,;\; record \,!\, v \longrightarrow \mathbf{skip} \,;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \ldots \\
\quad \llbracket ns \mid \lbag start, output, sync \rbag \mid \varnothing \rrbracket \\
(\mu X \bullet start \longrightarrow \mathbf{var}\; v : T \bullet A_n \,;\; record \,!\, v \longrightarrow \mathbf{skip} \,;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)
\end{array}\right)
$$

$$
\begin{array}{l}
\mathbf{provided}\;\; \{start, sync\} \cap usedC(A) = \varnothing \;\; \mathbf{and} \\
\quad \{start, record, output, sync\} \cap usedC(A_i) = \varnothing \text{ for all } i : 1 \,..\, n
\end{array}
$$

We observe that the parallel fragment is almost entirely absorbed, apart from a prefix $init \longrightarrow \mathbf{skip}$ that we highlighted above. The proof of the law decomposes the control action as illustrated below.

$$
\left(\begin{array}{l}
(\mu X \bullet init \longrightarrow start \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket \varnothing \mid \lbag start, sync \rbag \mid \varnothing \rrbracket \\
(\mu X \bullet start \longrightarrow record \,?\, y \longrightarrow output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket \varnothing \mid \lbag start, output, sync \rbag \mid \varnothing \rrbracket \\
(\mu X \bullet start \longrightarrow record \,?\, y \longrightarrow output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \ldots \\
\quad \llbracket \varnothing \mid \lbag start, output, sync \rbag \mid \varnothing \rrbracket \\
(\mu X \bullet start \longrightarrow record \,?\, y \longrightarrow output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)
\end{array}\right)
$$

The smaller control fragments that emerge are collapsed, as before, with the handler actions. All of this is justified using step laws. The prefixes $sync \longrightarrow \mathbf{skip}$ reveal their use in facilitating distribution of recursions.

Pattern 4

This pattern is needed for the refinement in Stage 3 for our case study. Unlike the previous three patterns, which are geared towards the encapsulation of shared data that is already somewhat explicit in the model, Pattern 4 addresses the refinement of a control mechanism. The pattern effectively refines a synchronisation barrier by a mechanism that makes use of shared data. It is capture by two laws: the first law sync-barrier-elim eliminates the synchronisation barrier by virtue of a control action, and the second law sync-barrier-control replaces the control action by an action that introduces new shared data and becomes part of the *MArea* action. This factorisation fosters modularisation of the refinement strategy; whereas sync-barrier-elim is universally applied, we may envisage different designs that eliminate the control fragment.

Below we present the first law to remove the barrier by virtue of a control action.

High-level Law 5 (sync-barrier-elim)

$$\begin{pmatrix} (\mu X \bullet start \longrightarrow A_1 \,;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs_1 \mid ns_2 \cup \ldots \cup ns_n]\!] \\ (\mu X \bullet start \longrightarrow A_2 \,;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad [\![ns_2 \mid cs_2 \mid ns_3 \cup \ldots \cup ns_n]\!] \\ \qquad \ldots \\ \quad [\![ns_{n-1} \mid cs_{n-1} \mid ns_n]\!] \\ (\mu X \bullet start \longrightarrow A_n \,;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \end{pmatrix}$$

$$\equiv$$

$$\left(\begin{array}{l} \begin{pmatrix} (\mu X \bullet start \longrightarrow A_1 \,;\; notify\,!\,1 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs_1 \setminus \{\!|\, done \,|\!\} \mid ns_2 \cup \ldots \cup ns_n]\!] \\ (\mu X \bullet start \longrightarrow A_2 \,;\; notify\,!\,2 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad [\![ns_2 \mid cs_2 \setminus \{\!|\, done \,|\!\} \mid ns_3 \cup \ldots \cup ns_n]\!] \\ \qquad \ldots \\ \quad [\![ns_{n-1} \mid cs_{n-1} \setminus \{\!|\, done \,|\!\} \mid ns_n]\!] \\ (\mu X \bullet start \longrightarrow A_n \,;\; notify\,!\,n \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \end{pmatrix} \\ \quad [\![ns_1 \cup \ldots \cup ns_n \mid \{\!|\, start, notify, sync \,|\!\} \mid \varnothing]\!] \\ \left(\mu X \bullet start \longrightarrow \begin{pmatrix} (notify\,!\,1 \longrightarrow \mathbf{skip}) \,|||\, \\ (notify\,!\,2 \longrightarrow \mathbf{skip}) \,|||\, \\ \ldots \\ (notify\,!\,n \longrightarrow \mathbf{skip}) \end{pmatrix} ;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X \right) \end{array}\right) \setminus \{\!|\, notify \,|\!\}$$

provided $\{\!|\, start, done, sync \,|\!\} \subseteq cs_i \wedge \{start, done, sync\} \cap usedC(A_i) = \varnothing$ for all $i : 1 \,..\, n$

and *notify* is a fresh channel of type $\mathbb{N}$

Proof of the Law

The proof of this law can be simplified by proceeding backwards (from the right-hand action) rather then forwards (from the left-hand action). The essential idea is once again to decompose the control action and collapse it with the handlers. The decomposition of the control fragment is sketched below.

$$\left(\mu X \bullet start \longrightarrow \begin{pmatrix} (notify\,!\,1 \longrightarrow \mathbf{skip}) \,|||\, \\ (notify\,!\,2 \longrightarrow \mathbf{skip}) \,|||\, \\ \ldots \\ (notify\,!\,n \longrightarrow \mathbf{skip}) \end{pmatrix} ;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X \right)$$

$\equiv$ "distribution of recursion and application of suitable parallel step laws"

$$\begin{pmatrix} (\mu X \bullet \mathit{start} \longrightarrow \mathit{notify}\,!\,1 \longrightarrow \mathit{done}\ ;\ \mathit{sync} \longrightarrow X) \\ \quad [\![\varnothing \mid \{\!|\, \mathit{start}, \mathit{done}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\ (\mu X \bullet \mathit{start} \longrightarrow \mathit{notify}\,!\,2 \longrightarrow \mathit{done}\ ;\ \mathit{sync} \longrightarrow X) \\ \quad \ldots \\ \quad [\![\varnothing \mid \{\!|\, \mathit{start}, \mathit{done}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\ (\mu X \bullet \mathit{start} \longrightarrow \mathit{notify}\,!\,n \longrightarrow \mathit{done}\ ;\ \mathit{sync} \longrightarrow X) \end{pmatrix}$$

We omit a detailed account of the derivation. After the decomposed fragments are collapsed with the handler actions, the channel *notify* is subsequently removed since none of the parallel actions synchronise on it.

The second law is used to transform the control action that arises from sync-barrier-elim into a program design that uses new shared data.

High-level Law 6 (sync-barrier-control)

$$\left(\mu X \bullet \mathit{start} \longrightarrow \begin{pmatrix} (\mathit{notify}\,!\,1 \longrightarrow \mathbf{skip}) \mid\mid\mid \\ (\mathit{notify}\,!\,2 \longrightarrow \mathbf{skip}) \mid\mid\mid \\ \ldots \\ (\mathit{notify}\,!\,n \longrightarrow \mathbf{skip}) \end{pmatrix}\ ;\ \mathit{done} \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X \right)$$

$$\equiv$$

$$\left(\begin{array}{l} \begin{pmatrix} \big(\mu X \bullet \mathit{reset} \longrightarrow \mathit{start} \longrightarrow X\ ;\ \mathit{sync} \longrightarrow X\big) \\ \quad [\![\varnothing \mid \{\!|\, \mathit{start}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\ \big(\mu X \bullet \mathit{start} \longrightarrow \mathit{notify}\,!\,1 \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X\big) \\ \quad [\![\varnothing \mid \{\!|\, \mathit{start}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\ \big(\mu X \bullet \mathit{start} \longrightarrow \mathit{notify}\,!\,2 \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X\big) \\ \quad [\![\varnothing \mid \{\!|\, \mathit{start}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\ \quad \ldots \\ \quad [\![\varnothing \mid \{\!|\, \mathit{start}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\ \big(\mu X \bullet \mathit{start} \longrightarrow \mathit{notify}\,!\,n \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X\big) \end{pmatrix} \\ \quad [\![\varnothing \mid \{\!|\, \mathit{reset}, \mathit{notify}, \mathit{sync} \,|\!\} \mid \varnothing]\!] \\ \left(\begin{array}{l} \mathbf{var}\ \mathit{active} : \mathbb{P}\,(1\,..\,n) \bullet \\ \quad \mu X \bullet \left(\begin{array}{l} (\mathit{reset} \longrightarrow \mathit{active} := 1\,..\,n) \\ \quad \Box \\ (\mathit{notify}\,?\,x \longrightarrow \begin{pmatrix} \mathit{active} := \mathit{active} \setminus \{x\}; \\ \mathbf{if}\ \mathit{active} = \varnothing \longrightarrow \mathit{done} \longrightarrow \mathbf{skip} \\ [\!]\ \neg\ \mathit{active} = \varnothing \longrightarrow \mathbf{skip} \\ \mathbf{fi} \end{pmatrix}) \end{array}\right)\ ;\ X \end{array}\right) \end{array}\right) \setminus \{\!|\, \mathit{reset} \,|\!\}$$

We note that apart from the right hand parallel branch encapsulating the shared data, we also have a left hand parallel branch that contains a parallelism of smaller control fragments. These fragments will have to be decomposed in the refinement strategy and give rise to another synchronisation on the *start* channel that initialises the *active* variable prior to starting execution of the parallel handlers. The fragments are very simple though and it should be possible to distribute them (mostly) automatically.

Finally, we consider a third law that combines the previous two laws. In this law, we implicitly already collapsed four of the five smaller control fragments emerging from the application of sync-barrier-control.

High-level Law 7 (sync-barrier-design)

$$
\begin{array}{l}
\left(\begin{array}{l}
(\mu X \bullet start \longrightarrow A_1 \,;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns_1 \mid cs_1 \mid ns_2 \cup \ldots \cup ns_n \rrbracket \\
(\mu X \bullet start \longrightarrow A_2 \,;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns_2 \mid cs_2 \mid ns_3 \cup \ldots \cup ns_n \rrbracket \\
\qquad \ldots \\
\quad \llbracket ns_{n-1} \mid cs_{n-1} \mid ns_n \rrbracket \\
(\mu X \bullet start \longrightarrow A_n \,;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)
\end{array}\right) \\
\quad \sqsubseteq \\
\left(\begin{array}{l}
(\mu X \bullet reset \longrightarrow start \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket \varnothing \mid \{\!| reset, start, sync |\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}
(\mu X \bullet start \longrightarrow A_1 \,;\; notify\,!\,1 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns_1 \mid cs_1 \setminus \{\!| done |\!\} \mid ns_2 \cup \ldots \cup ns_n \rrbracket \\
(\mu X \bullet start \longrightarrow A_2 \,;\; notify\,!\,2 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns_2 \mid cs_2 \setminus \{\!| done |\!\} \mid ns_3 \cup \ldots \cup ns_n \rrbracket \\
\qquad \ldots \\
\quad \llbracket ns_{n-1} \mid cs_{n-1} \setminus \{\!| done |\!\} \mid ns_n \rrbracket \\
(\mu X \bullet start \longrightarrow A_n \,;\; notify\,!\,n \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)
\end{array}\right) \\
\quad \llbracket ns_1 \cup \ldots \cup ns_n \mid \{\!| start, notify, sync |\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}
\mathbf{var}\ active : \mathbb{P}\,(1 \mathrel{..} n) \bullet \\
\quad \mu X \bullet \left(\begin{array}{l}
(reset \longrightarrow active := 1 \mathrel{..} n) \\
\quad \Box \\
(notify\,?\,x \longrightarrow \left(\begin{array}{l}
active := active \setminus \{x\}; \\
\mathbf{if}\ active = \varnothing \longrightarrow done \longrightarrow \mathbf{skip} \\
[\!]\ \neg\ active = \varnothing \longrightarrow \mathbf{skip} \\
\mathbf{fi}
\end{array}\right))
\end{array}\right) \,;\; X
\end{array}\right)
\end{array}\right) \setminus \{\!| reset, notify |\!\}
\end{array}
$$

provided $\{\!| reset, start, done, sync |\!\} \subseteq cs_i \land \{reset, start, done, sync\} \cap usedC(A_i) = \varnothing$ for all $i : 1 \mathrel{..} n$
and $notify$ is a fresh channel of type $\mathbb{N}$

This law is less modular but more useful in terms of automation. The residual refinement effort consists of distributing the simple control fragment highlighted above. Intuitively, this corresponds to calling an initialisation method on the shared state. We note that one may envisage designs that do not require an initialisation (*notify* could cater for this too). For such designs the law sync-barrier-elim would still be useful, though sync-barrier-design is too specific to be applicable. This highlights a general trade-off between modularity and reuse and automation. It is an important insight and lesson learned in this case study.

We note that even in the design law sync-barrier-design, additional refinement is still required during the AR phase to data refine the *active* component into a class object and to turn the channels *start* and *notify* into method call channel pairs. Otherwise, we have now all ingredients to tackle the refinement stages for the SH phase in our example. We examine them in detail in the remainder of the section.

5.3.2 Stage 1

Our main objective in Stage 1 to Stage 3 is to encapsulate shared data and provide means for accessing it. In doing so, we tease out the *MArea* action, and Step 4 transforms it into the precise shape to match the program design. The essence of the refinement steps is mostly the application of the high-level patterns that have been presented in Section 5.3.1. Our starting point is the *System* action resulting from MH.

$$
\begin{array}{l}
System \mathrel{\widehat=} \\
\quad \left(\left(\left(\begin{array}{l}
\left(\begin{array}{l} \mu X \bullet (next\_frame ? frame \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} SF_{TB} ;\ StoreFrame)) \blacktriangleleft INP\_DL; \\ reduce ! currentFrame ! state \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X \end{array} \right) \\
\quad [\![\{ currentFrame, state \} \mid \{\!|\ reduce, sync\ |\!\} \mid \{ voxel\_map, work \}]\!] \\
\left(\begin{array}{l} \mu X \bullet reduce ? currentFrame ? state \longrightarrow \\ \mathbf{wait}\ 0 \mathrel{..} RPW_{TB} ;\ ReduceAndPartitionWork; \\ detect ! work \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X \end{array} \right) \\
\quad [\![\{ voxel\_map, work \} \mid \{\!|\ detect, sync\ |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l} \mu X \bullet detect ? work \longrightarrow \mathbf{var}\ colls1 : int \bullet \mathbf{wait}\ 0 \mathrel{..} CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\ recColls ! colls1 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X \end{array} \right) \\
\quad [\![\varnothing \mid \{\!|\ detect, output, sync\ |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l} \mu X \bullet detect ? work \longrightarrow \mathbf{var}\ colls2 : int \bullet \mathbf{wait}\ 0 \mathrel{..} CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\ recColls ! colls2 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X \end{array} \right) \\
\quad [\![\varnothing \mid \{\!|\ detect, output, sync\ |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l} \mu X \bullet detect ? work \longrightarrow \mathbf{var}\ colls3 : int \bullet \mathbf{wait}\ 0 \mathrel{..} CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\ recColls ! colls3 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X \end{array} \right) \\
\quad [\![\varnothing \mid \{\!|\ detect, output, sync\ |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l} \mu X \bullet detect ? work \longrightarrow \mathbf{var}\ colls4 : int \bullet \mathbf{wait}\ 0 \mathrel{..} CPC_{TB}; \\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\ recColls ! colls4 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X \end{array} \right) \\
\quad [\![\varnothing \mid \{\!|\ output, sync\ |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l} \mu X \bullet output ? collisions \longrightarrow \\ \mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0 \mathrel{..} CC_{TB} ;\ CalcCollisions; \\ (output\_collisions ! colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL ;\ sync \longrightarrow X \end{array} \right)
\end{array} \right) \\
\quad [\![\{ currentFrame, state, voxel\_map, work \} \mid \\
\qquad \{\!|\ detect, output, recColls, sync\ |\!\} \mid \{ collisions \}]\!] \\
InteractionHandlers
\end{array} \right) \\
\quad [\![\{ currentFrame, state, voxel\_map, work, collisions \} \mid \{\!|\ sync\ |\!\} \mid \varnothing]\!] \\
Cycle
\end{array} \right) \setminus \\
\quad \{\!|\ reduce, detect, output, recColls, sync\ |\!\}
\end{array}
$$

We have flattened the calls to local actions for handlers using the copy rule, though for brevity we keep the calls to *InteractionHandlers* and *Cycle* until we require to expand their definitions in Stage 2.

The refinement in this stage applies the laws seq-share-1 and seq-share-2 to the synchronisations on *reduce* and *detect*, respectively. Above, we have highlighted the target for the first application of seq-share-1. This requires some reordering of parallel actions. After applying the law, we also extract channel hiding and isolate the action that contributes to *MArea* into a separate branch of the top-level parallelism. These supplementary steps can be tedious on the paper but do not pose a challenge to automation.

$\equiv$ "reordering parallel actions and distributing the hiding of the *reduce* channel"

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mu X \bullet (next\_frame \,?\, frame \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} SF_{TB} \ ; \ StoreFrame)) \blacktriangleleft INP\_DL; \\
reduce \,!\, currentFrame \,!\, state \longrightarrow \mathbf{skip} \ ; \ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\{currentFrame, state\} \mid \{\!|\, reduce, sync \,|\!\} \mid \{voxel\_map, work\}]\!] \\
\left(\begin{array}{l}
\mu X \bullet reduce \,?\, currentFrame \,?\, state \longrightarrow \\
\mathbf{wait}\ 0 \mathrel{..} RPW_{TB} \ ; \ ReduceAndPartitionWork; \\
detect \,!\, work \longrightarrow \mathbf{skip} \ ; \ sync \longrightarrow X
\end{array}\right)
\end{array}\right) \\
\quad \setminus \{\!|\, reduce \,|\!\} \\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \{\!|\, detect, sync \,|\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \,?\, work \longrightarrow \mathbf{var}\ colls1 : int \bullet \mathbf{wait}\ 0 \mathrel{..} CPC_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\
recColls \,!\, colls1 \longrightarrow \mathbf{skip} \ ; \ output \,?\, y \longrightarrow \mathbf{skip} \ ; \ sync \longrightarrow X
\end{array}\right) \\
\quad \ldots
\end{array}\right) \\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \\
\qquad \{\!|\, detect, output, recColls, sync \,|\!\} \mid \{collisions\}]\!] \\
InteractionHandlers
\end{array}\right) \\
\quad [\![\{currentFrame, state, voxel\_map, work, collisions\} \mid \{\!|\, sync \,|\!\} \mid \varnothing \,]\!] \ Cycle
\end{array}\right) \setminus \\
\quad \{\!|\, detect, output, recColls, sync \,|\!\}
$$

$\equiv$ "application of the law seq-share-1 introducing a new typeless channel *reduce*"

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mu X \bullet (next\_frame \,?\, frame \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} SF_{TB} \ ; \ StoreFrame)) \blacktriangleleft INP\_DL; \\
setFrameState \,!\, currentFrame \,!\, state \longrightarrow \mathbf{skip} \ ; \ reduce \longrightarrow \mathbf{skip} \ ; \ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\{currentFrame, state\} \mid \{\!|\, reduce, sync \,|\!\} \mid \{voxel\_map, work\}]\!] \\
\left(\begin{array}{l}
\mu X \bullet reduce \longrightarrow \mathbf{skip} \ ; \ getFrameState \,?\, currentFrame \,?\, state \longrightarrow \\
\mathbf{wait}\ 0 \mathrel{..} RPW_{TB} \ ; \ ReduceAndPartitionWork; \\
detect \,!\, work \longrightarrow \mathbf{skip} \ ; \ sync \longrightarrow X
\end{array}\right)
\end{array}\right) \\
\quad \setminus \{\!|\, reduce \,|\!\} \\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \{\!|\, setFrameState, getFrameState \,|\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathbf{var}\ currentFrame : RawFrame \bullet \\
\mathbf{var}\ state : StateTable \bullet \\
\quad \mu X \bullet \left(\begin{array}{l}
(setFrameState \,?\, v_1 \,?\, v_2 \longrightarrow currentFrame, state := v_1, v_2) \ \Box \\
(getFrameState \,!\, currentFrame \,!\, state \longrightarrow \mathbf{skip})
\end{array}\right) ; \ X
\end{array}\right)
\end{array}\right) \\
\quad \setminus \{\!|\, setFrameState, getFrameState \,|\!\} \\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \{\!|\, detect, sync \,|\!\} \mid \varnothing]\!] \\
\quad \ldots
\end{array}\right) \\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \\
\qquad \{\!|\, detect, output, recColls, sync \,|\!\} \mid \{collisions\}]\!] \\
InteractionHandlers
\end{array}\right) \\
\quad [\![\{currentFrame, state, voxel\_map, work, collisions\} \mid \{\!|\, sync \,|\!\} \mid \varnothing \,]\!] \ Cycle
\end{array}\right) \setminus \\
\quad \{\!|\, detect, output, recColls, sync \,|\!\}
$$

We note that the fresh channel *reduce*, highlighted in blue, is different from the former existing channel *reduce*, highlighted in red: whereas *reduce* is typeless, *reduce* is of type $RawFrame \times StateTable$. To emphasise the part of the action that contributes to $MArea$ as well as the underlying channels for shared data access, we use a green highlight. Next, we reorder the parallelism once again to isolate the sharing action.

$\equiv$ "reordering of parallelism and extraction of hiding using suitable laws"

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mu X \bullet (next\_frame \,?\, frame \longrightarrow (\mathbf{wait}\, 0 \ldots SF_{TB} \,;\; StoreFrame)) \blacktriangleleft INP\_DL; \\
setFrameState \,!\, currentFrame \,!\, state \longrightarrow \mathbf{skip} \,;\; reduce \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X
\end{array}\right) \\
\quad [\![\{currentFrame, state\} \mid \{\!| reduce, sync |\!\} \mid \{voxel\_map, work\}]\!] \\
\left(\begin{array}{l}
\mu X \bullet reduce \longrightarrow \mathbf{skip} \,;\; getFrameState \,?\, currentFrame \,?\, state \longrightarrow \\
\mathbf{wait}\, 0 \ldots RPW_{TB} \,;\; ReduceAndPartitionWork; \\
detect \,!\, work \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X
\end{array}\right) \\
\quad [\![\{voxel\_map, work\} \mid \{\!| detect, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \,?\, work \longrightarrow \mathbf{var}\; colls1 : int \bullet \mathbf{wait}\, 0 \ldots CPC_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\
recColls \,!\, colls1 \longrightarrow \mathbf{skip} \,;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \,?\, work \longrightarrow \mathbf{var}\; colls2 : int \bullet \mathbf{wait}\, 0 \ldots CPC_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\
recColls \,!\, colls2 \longrightarrow \mathbf{skip} \,;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \,?\, work \longrightarrow \mathbf{var}\; colls3 : int \bullet \mathbf{wait}\, 0 \ldots CPC_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\
recColls \,!\, colls3 \longrightarrow \mathbf{skip} \,;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \,?\, work \longrightarrow \mathbf{var}\; colls4 : int \bullet \mathbf{wait}\, 0 \ldots CPC_{TB}; \\
(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\
recColls \,!\, colls4 \longrightarrow \mathbf{skip} \,;\; output \,?\, y \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet output \,?\, collisions \longrightarrow \\
\mathbf{var}\; colls : int \bullet \mathbf{wait}\, 0 \ldots CC_{TB} \,;\; CalcCollisions; \\
(output\_collisions \,!\, colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL \,;\; sync \longrightarrow X
\end{array}\right)
\end{array}\right) \\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \\
\qquad \{\!| detect, output, recColls, sync |\!\} \mid \{collisions\}]\!] \\
InteractionHandlers
\end{array}\right) \\
\quad [\![\{currentFrame, state, voxel\_map, work, collisions\} \mid \{\!| sync |\!\} \mid \varnothing]\!] \; Cycle
\end{array}\right) \\
\quad [\![\{currentFrame, state, voxel\_map, work, collisions\} \mid \{\!| setFrameState, getFrameState |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathbf{var}\; currentFrame : RawFrame \bullet \\
\mathbf{var}\; state : StateTable \bullet \\
\quad \mu X \bullet \left(\begin{array}{l}
(setFrameState \,?\, v_1 \,?\, v_2 \longrightarrow currentFrame, state := v_1, v_2) \,\Box \\
(getFrameState \,!\, currentFrame \,!\, state \longrightarrow \mathbf{skip})
\end{array}\right) ;\; X
\end{array}\right)
\end{array}\right) \setminus \\
\quad \{\!| reduce, detect, output, recColls, setFrameState, getFrameState, sync |\!\}
$$

This concludes the application of Pattern 1. We proceed exactly in the same way in order to encapsulate the data passed through the *detect* channel. The only difference is that we have to apply seq-share-2 rather than seq-share-1, that is the law for Pattern 2. We omit the low-level steps and just give the result here.

$\equiv$ "application of the law seq-share-2 including pre- and post-processing transformations"

$$
\left(
\begin{array}{l}
\left(
\begin{array}{l}
\left(
\begin{array}{l}
\left(
\begin{array}{l}
\left(\begin{array}{l}
\mu X \bullet (next\_frame ? frame \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} SF_{TB} ;\ StoreFrame)) \blacktriangleleft INP\_DL;\\
setFrameState ! currentFrame ! state \longrightarrow \mathbf{skip} ;\ reduce \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\{ currentFrame, state \} \mid \{\!|\ reduce, sync\ |\!\} \mid \{ voxel\_map, work \}]\!]\\
\left(\begin{array}{l}
\mu X \bullet reduce \longrightarrow \mathbf{skip} ;\ getFrameState ? currentFrame ? state \longrightarrow\\
\mathbf{wait}\ 0 \mathrel{..} RPW_{TB} ;\ ReduceAndPartitionWork;\\
setWork ! work \longrightarrow \mathbf{skip} ;\ detect \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\{ voxel\_map, work \} \mid \{\!|\ detect, sync\ |\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls1 : int \bullet\\
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB} ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1);\\
recColls ! colls1 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\ detect, output, sync\ |\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls2 : int \bullet\\
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB} ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2);\\
recColls ! colls2 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\ detect, output, sync\ |\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls3 : int \bullet\\
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB} ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3);\\
recColls ! colls3 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\ detect, output, sync\ |\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls4 : int \bullet\\
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB} ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4);\\
recColls ! colls4 \longrightarrow \mathbf{skip} ;\ output ? y \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\ output, sync\ |\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet output ? collisions \longrightarrow\\
\mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0 \mathrel{..} CC_{TB} ;\ CalcCollisions;\\
(output\_collisions ! colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL ;\ sync \longrightarrow X
\end{array}\right)
\end{array}
\right)\\
\quad [\![\{ currentFrame, state, voxel\_map, work \} \mid\\
\qquad \{\!|\ detect, output, recColls, sync\ |\!\} \mid \{ collisions \}]\!]\\
InteractionHandlers
\end{array}
\right)\\
\quad [\![\{ currentFrame, state, voxel\_map, work, collisions \} \mid \{\!|\ sync\ |\!\} \mid \varnothing\ ]\!]\ Cycle
\end{array}
\right)\\
\quad [\![\{ currentFrame, state, voxel\_map, work, collisions \} \mid\\
\qquad \{\!|\ setFrameState, getFrameState, setWork, getWork\ |\!\} \mid \varnothing]\!]\\
\left(
\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ currentFrame : RawFrame \bullet\\
\mathbf{var}\ state : StateTable \bullet\\
\quad \mu X \bullet \left(\begin{array}{l}
(setFrameState ? v_1 ? v_2 \longrightarrow currentFrame, state := v_1, v_2)\ \Box\\
(getFrameState ! currentFrame ! state \longrightarrow \mathbf{skip})
\end{array}\right) ;\ X
\end{array}\right) \interleave\\
\left(\begin{array}{l}
\mathbf{var}\ work : Partition \bullet\\
\quad \mu X \bullet \left(\begin{array}{l}
(setWork ? v \longrightarrow work := v)\ \Box\\
(getWork ! work \longrightarrow \mathbf{skip})
\end{array}\right) ;\ X
\end{array}\right)
\end{array}
\right)
\end{array}
\right) \setminus
$$

$$\{\!|\ reduce, detect, output, recColls, setFrameState, getFrameState, getWork, setWork, sync\ |\!\}$$

We note that applying the law for Pattern 2, we have to take into account that *InteractionHandlers* also synchronises on the *detect* channel although it is not interested in the value communicated. Hence, for

InteractionHandler we obtain the following action as a result of the previous refinement pattern.

$$
\begin{array}{l}
\mathit{InteractionHandlers} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{detect} \longrightarrow \mathit{getWork}\,?\,y \longrightarrow \\
\mathbf{var}\ \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 : \mathit{int} \bullet \\
\left(\begin{array}{l}
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ \mathit{colls}1 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ \mathit{colls}2 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ \mathit{colls}3 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ \mathit{colls}4 := x))
\end{array}\right); \\
\mathbf{wait}\ 0\,..\,\mathit{SCFP}_{TB}; \\
\mathit{SetCollisionsFromParts}(\llbracket\, \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 \,\rrbracket\,); \\
\mathit{output}\,!\,\mathit{collisions} \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

Clearly, the value y is not used in the action after the prefix $\mathit{getWork}\,?\,y \longrightarrow A$ above. Since the channel *getWork* is concealed and *InteractionHandlers* only synchronises with the *MArea* fragment on this channel, we can use a noncompositional rule to remove this communication altogether. Intuitively, this is justified by *getWork* never being blocked and not having an affect on the action's behaviour.

$\ldots\ \equiv$ "Specialised noncompositional rule to remove channel communication"

$$
\left(\begin{array}{l}
\mu X \bullet \mathit{detect} \longrightarrow \\
\mathbf{var}\ \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 : \mathit{int} \bullet \\
\left(\begin{array}{l}
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ \mathit{colls}1 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ \mathit{colls}2 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ \mathit{colls}3 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ \mathit{colls}4 := x))
\end{array}\right); \\
\mathbf{wait}\ 0\,..\,\mathit{SCFP}_{TB}; \\
\mathit{SetCollisionsFromParts}(\llbracket\, \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 \,\rrbracket\,); \\
\mathit{output}\,!\,\mathit{collisions} \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X
\end{array}\right)
$$

Here, we shall not examine this rule further but merely identify the need for it. Alternatively, we could provide a more specialised version of the seq-share-2 law; such, however, may not be as reusable as the present version of the law. To compare and evaluate both options further investigation is necessary.

At this point, one may be tempted to apply Pattern 2 yet another time in order to replace the typed *output* channel. This indeed is possible, module a minor alteration of the law, but it does not produce the desired program design. More specifically, end up with a shared variable *collisions* that is accessed via get and set operations, modelled by the channels *getColls* and *setColls*, for instance. This results in a different design where *InteractionHandlers* collects the partial results but carries out the update to *collisions* in a single atomic operation. We, however, want to carry out this update incrementally and concurrently during the detection phase. For this reason, Stage 1 is completed here and we look at the refinement of *InteractionHandlers* in Stage 2 using a different Pattern 3.

5.3.3 Stage 2

In Stage 2 of the SH phase, we apply Pattern 3. This consists of applying the law par-share presented earlier on in Section 5.3.1. The law cannot be applied immediately; we first have to bring the action *InteractionHandlers*, which is the target for the law application, into a form that matches the left-hand action of the law. After application of the law, more work has to be done to eliminate residual parallel actions by distributing them into the handlers. This gives rise to localising the synchronisation that initialises the shared *collsisions* variable prior to the parallel detection phase. We discuss each sub-step in detail.

Transformation of Application Target

We first recall the current definition of *InteractionHandlers* to which we aim to apply the par-share law.

$$
\begin{array}{l}
\mathit{InteractionHandlers} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{detect} \longrightarrow \\
\mathbf{var}\ \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 : \mathit{int} \bullet \\
\left(\begin{array}{l}
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,\mathit{RC}_{TB}\ ;\ \ \mathit{colls}1 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,\mathit{RC}_{TB}\ ;\ \ \mathit{colls}2 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,\mathit{RC}_{TB}\ ;\ \ \mathit{colls}3 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,\mathit{RC}_{TB}\ ;\ \ \mathit{colls}4 := x))
\end{array} \right) ; \\
\mathbf{wait}\,0\,..\,\mathit{SCFP}_{TB}; \\
\mathit{SetCollisionsFromParts}(\lbrakk\, \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 \,\rbrakk); \\
\mathit{output}\,!\,\mathit{collisions} \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X
\end{array} \right)
\end{array}
$$

The left-hand side of the par-share law is recaptured below.

$$
\left(\begin{array}{l}
\mathbf{var}\ v : T \bullet \\
\qquad \left(\begin{array}{l}
\mu X \bullet\ \mathit{start} \longrightarrow \mathbf{wait}\,0\,..\,\mathit{Init}_{TB}\ ;\ \ \mathit{InitOp}; \\
\mathbf{var}\ x_1, x_2, \ldots, x_n : T \bullet \\
\left(\begin{array}{l}
(\mathit{record}\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,\mathit{RC}_{TB}\ ;\ \ x_1 := x)); \\
(\mathit{record}\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,\mathit{RC}_{TB}\ ;\ \ x_2 := x)); \\
\quad \ldots \\
(\mathit{record}\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,\mathit{RC}_{TB}\ ;\ \ x_n := x)); \\
\mathbf{wait}\,0\,..\,\mathit{Merge}_{TB}\ ;\ \ \mathit{MergeOp}(\lbrakk\, x_1, x_2, \ldots, x_n \,\rbrakk); \\
\mathit{output}\,!\,v \longrightarrow \mathbf{skip}\ ;\ \ \mathit{sync} \longrightarrow X
\end{array} \right)
\end{array} \right)
\end{array} \right)
$$

Most notable, *SetCollisionsFromParts* has to be decomposed. In Section 5.3.1 (Pattern 3), we have already illustrated the decomposition of *SetCollisionsFromParts* by way of the following two schema operations.

$$
\begin{array}{|l}
\hline
\mathit{InitColls} \\
\hline
\Delta\,\mathit{CDxMHState} \\
\hline
\mathit{currentFrame}' = \mathit{currentFrame} \land \mathit{state}' = \mathit{state} \land \mathit{voxel\_map}' = \mathit{voxel\_map} \land \mathit{work}' = \mathit{work} \\
\mathit{collisions}' = 0 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\mathit{RecColls} \\
\hline
\Delta\,\mathit{CDxMHState} \\
\mathit{collsbag}? : \mathrm{bag}\ \mathit{int} \\
\hline
\mathit{currentFrame}' = \mathit{currentFrame} \land \mathit{state}' = \mathit{state} \land \mathit{voxel\_map}' = \mathit{voxel\_map} \land \mathit{work}' = \mathit{work} \\
\exists\, s : \mathrm{seq}\ \mathit{int} \mid s = \mathit{items}\ \mathit{collsbag}? \bullet \mathit{collisions}' = \mathit{collisions} + \Sigma\, s \\
\hline
\end{array}
$$

We have changed their names here into *InitColls* and *RecColls*. We have that $\mathit{SetCollisionsFromParts}(cb) = \mathit{InitColls}\ ;\ \ \mathit{RecColls}(cb)$. We also have the following algorithmic refinements used later on.

$$
\mathit{InitColls} \sqsubseteq \mathit{collisions} := 0 \quad \text{and} \quad \mathit{RecColls}(\lbrakk\, \mathit{colls} \,\rbrakk) \sqsubseteq \mathit{collision} := \mathit{collisions} + \mathit{colls}
$$

The refinement of *InteractionHandlers* thus yields.

$\ldots \equiv$ "decomposition of *SetCollisionsFromParts* into an initialisation and merge operation"

$$InteractionHandlers \mathrel{\widehat=} \left(\begin{array}{l} \mu X \bullet detect \longrightarrow \\ \mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \\ \left(\begin{array}{l} (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls1 := x))\ ; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls2 := x))\ ; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls3 := x))\ ; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls4 := x)) \end{array}\right); \\ \mathbf{wait}\ 0 \mathinner{..} SCFP_{TB}; \\ InitColls\ ;\ \ RecColls([\![\ colls1, colls2, colls3, colls4\ ]\!]\ ); \\ output\,!\,collisions \longrightarrow \mathbf{skip}\ ;\ \ sync \longrightarrow X \end{array}\right)$$

$\ldots \equiv$ "multiple applications of the law seq-op-comm to move *InitColls* through the sequence"

$$InteractionHandlers \mathrel{\widehat=} \left(\begin{array}{l} \mu X \bullet detect \longrightarrow \\ \mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet InitColls; \\ \left(\begin{array}{l} (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls1 := x))\ ; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls2 := x))\ ; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls3 := x))\ ; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls4 := x)) \end{array}\right); \\ \mathbf{wait}\ 0 \mathinner{..} SCFP_{TB}; \\ RecColls([\![\ colls1, colls2, colls3, colls4\ ]\!]\ ); \\ output\,!\,collisions \longrightarrow \mathbf{skip}\ ;\ \ sync \longrightarrow X \end{array}\right)$$

$\ldots \equiv$ "introduction of spurious **wait** statement using the law zero-wait-intro"

$$InteractionHandlers \mathrel{\widehat=} \left(\begin{array}{l} \mu X \bullet detect \longrightarrow \\ \mathbf{var}\ colls1, colls2, colls3, colls4 : int \bullet \mathbf{wait}\ 0\ ;\ \ InitColls; \\ \left(\begin{array}{l} (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls1 := x))\ ; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls2 := x))\ ; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls3 := x))\ ; \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB}\ ;\ \ colls4 := x)) \end{array}\right); \\ \mathbf{wait}\ 0 \mathinner{..} SCFP_{TB}; \\ RecColls([\![\ colls1, colls2, colls3, colls4\ ]\!]\ ); \\ output\,!\,collisions \longrightarrow \mathbf{skip}\ ;\ \ sync \longrightarrow X \end{array}\right)$$

Above we use a special commutativity law for data operations in the second sub-step.

***Circus* Law 35 (seq-op-comm)**

$$A\ ;\ \ Op \equiv Op\ ;\ \ A$$

$$\mathbf{provided}\ usedV(Op) \cup wrtV(A) = \varnothing\ \ \mathbf{and}\ \ usedV(A) \cup wrtV(Op) = \varnothing$$

We also require a law to introduce a spurious **wait** 0 statement in order to align the shape of the action to match the left-hand side of the law. The respective law is zero-wait-intro, included in Appendix B.1.

We observe that the action *InteractionHandlers* now has the correct shape to apply the par-share law. Altogether the most challenging task in the above sub-steps is the decomposition of the *SetCollisionsFromParts* operation. In practical terms, it may be possible to define tactics that target particular schema operation shapes. The above sub-steps rely on the fact that *InitColls* is instantaneous (does not consume time). Dealing with cases where this assumption is not given is part of our future work.

Application of Pattern 3

The application of Pattern 3 is entailed by the following refinement.

$$
\begin{array}{l}
\mathit{InteractionHandlers} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \mathit{detect} \longrightarrow \\
\mathbf{var}\ \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 : \mathit{int} \bullet \mathbf{wait}\ 0\ ;\ \mathit{InitColls}; \\
\left(\begin{array}{l}
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \mathit{colls}1 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \mathit{colls}2 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \mathit{colls}3 := x))\ ; \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \mathit{colls}4 := x))
\end{array}\right)\ ; \\
\mathbf{wait}\ 0\,..\,\mathit{SCFP}_{TB}; \\
\mathit{RecColls}([\![\, \mathit{colls}1, \mathit{colls}2, \mathit{colls}3, \mathit{colls}4 \,]\!]\,); \\
\mathit{output}\,!\,\mathit{collisions} \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}
$$

$\ldots\ \equiv$ "application of the law par-share"

$$
\left(\begin{array}{l}
\left(\mu X \bullet \mathit{initColls} \longrightarrow \mathit{detect} \longrightarrow \left(\begin{array}{l}
(\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip})\ |||\\
(\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip})\ |||\\
(\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip})\ |||\\
(\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip})
\end{array}\right)\ ;\ \mathit{output}\,?\,y \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X\right) \\
\quad [\![\varnothing \mid \{\!|\ \mathit{initColls}, \mathit{recColls}, \mathit{output}\ |\!\} \mid \varnothing]\!] \\
\left(\mu X \bullet \left(\begin{array}{l}
(\mathit{initColls} \longrightarrow (\mathbf{wait}\ 0\ ;\ \mathit{InitColls}))\ \Box \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \mathit{RecColls}([\![\, x \,]\!]\,)))\ \Box \\
(\mathit{output}\,!\,\mathit{collisions} \longrightarrow \mathbf{skip})
\end{array}\right)\ ;\ X\right)
\end{array}\right) \\
\setminus \{\!|\ \mathit{initColls}\ |\!\}
$$

This introduces a new channel *initColls* which corresponds to the method call that initialises *collisions*.

Although this might be an issue for the AR phase, we carry out further refinement that transforms *InitColls* as well as the call $\mathit{RecColls}([\![\, x \,]\!]\,)$ into simple assignments, as previously suggested.

$\ldots\ \equiv$ "algorithmic refinement of *InitColls* and $\mathit{RecColls}([\![\, x \,]\!]\,)$ and elimination of spurious **wait**"

$$
\left(\begin{array}{l}
\left(\mu X \bullet \mathit{initColls} \longrightarrow \mathit{detect} \longrightarrow \left(\begin{array}{l}
(\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip})\ |||\\
(\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip})\ |||\\
(\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip})\ |||\\
(\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip})
\end{array}\right)\ ;\ \mathit{output}\,?\,y \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X\right) \\
\quad [\![\varnothing \mid \{\!|\ \mathit{initColls}, \mathit{recColls}, \mathit{output}\ |\!\} \mid \varnothing]\!] \\
\left(\mu X \bullet \left(\begin{array}{l}
(\mathit{initColls} \longrightarrow \mathit{collisions} := 0)\ \Box \\
(\mathit{recColls}\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ \mathit{collisions} := \mathit{collisions} + x))\ \Box \\
(\mathit{output}\,!\,\mathit{collisions} \longrightarrow \mathbf{skip})
\end{array}\right)\ ;\ X\right)
\end{array}\right) \\
\setminus \{\!|\ \mathit{initColls}\ |\!\}
$$

The underlying *MArea* fragment, highlighted as before in green, now has the desired shape in the program. The only remaining issue is the decomposition and distribution of the left-hand control action.

Elimination of Parallel Control Action

In order to eliminate the control action that emerged from the application of the law par-share, we may envisage two possible approaches. First, we may carry out manual elementary refinement steps that achieve the decomposition and distribution of the resulting smaller parallel fragments. Or otherwise, we may use a law that already entails the collapsing of (most of) the parallelism in the context of the handlers that record the results. The first approach is sketched by the refinement below that decomposes the control fragment.

$$\left(\mu X \bullet \mathit{initColls} \longrightarrow \mathit{detect} \longrightarrow \begin{pmatrix} (\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip}) \mathbin{|||} \\ (\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip}) \mathbin{|||} \\ (\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip}) \mathbin{|||} \\ (\mathit{recColls}\,?\,y \longrightarrow \mathbf{skip}) \end{pmatrix} ;\ \mathit{output}\,?\,y \longrightarrow \mathbf{skip} ;\ \mathit{sync} \longrightarrow X\right)$$

$\ldots \equiv$ "application of the law distr-rec-par-1 and elementary parallel step laws for decomposition"

$$\begin{pmatrix} \big(\mu X \bullet \mathit{initColls} \longrightarrow \mathit{detect} ;\ \mathit{sync} \longrightarrow X\big) \\ \quad [\![\varnothing \mid \{\!|\, \mathit{detect} \,|\!\} \mid \varnothing]\!] \\ \big(\mu X \bullet \mathit{detect} \longrightarrow \mathit{recColls}\,?\,y \longrightarrow \mathit{output}\,?\,y \longrightarrow \mathbf{skip} ;\ \mathit{sync} \longrightarrow X\big) \\ \quad [\![\varnothing \mid \{\!|\, \mathit{detect}, \mathit{output} \,|\!\} \mid \varnothing]\!] \\ \big(\mu X \bullet \mathit{detect} \longrightarrow \mathit{recColls}\,?\,y \longrightarrow \mathit{output}\,?\,y \longrightarrow \mathbf{skip} ;\ \mathit{sync} \longrightarrow X\big) \\ \quad [\![\varnothing \mid \{\!|\, \mathit{detect}, \mathit{output} \,|\!\} \mid \varnothing]\!] \\ \big(\mu X \bullet \mathit{detect} \longrightarrow \mathit{recColls}\,?\,y \longrightarrow \mathit{output}\,?\,y \longrightarrow \mathbf{skip} ;\ \mathit{sync} \longrightarrow X\big) \\ \quad [\![\varnothing \mid \{\!|\, \mathit{detect}, \mathit{output} \,|\!\} \mid \varnothing]\!] \\ \big(\mu X \bullet \mathit{detect} \longrightarrow \mathit{recColls}\,?\,y \longrightarrow \mathit{output}\,?\,y \longrightarrow \mathbf{skip} ;\ \mathit{sync} \longrightarrow X\big) \end{pmatrix}$$

The five smaller fragments are now collapsed with suitable handlers. Whereas the first action is collapsed with the reducer handler, the remaining four actions are collapsed with the four detection handlers. Only the first action leaves a trace, namely a prefix $\mathit{initColls} \longrightarrow \mathbf{skip}$; the other actions are simply absorbed. Thus, we could encompass the collapsing of the other four fragments directly into a law that refines the control fragment that arises from the application of the par-share law *in the context* of the detection handlers. This is exactly along the lines of the modularisation in Pattern 4 via a sharing, control and design law.

Below we present the result after injecting the refinement of *InteractionHandlers* and subsequently distributing the residual parallel control action (we omit the detailed steps for decomposition and collapsing).

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}\mu X \bullet (\mathit{next\_frame} ? \mathit{frame} \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} \mathit{SF}_{\mathit{TB}} \,;\ \mathit{StoreFrame})) \blacktriangleleft \mathit{INP\_DL};\\ \mathit{setFrameState} ! \mathit{currentFrame} ! \mathit{state} \longrightarrow \mathbf{skip} \,;\ \mathit{reduce} \longrightarrow \mathbf{skip} \,;\ \mathit{sync} \longrightarrow X\end{array}\right)\\
\quad \llbracket \{ \mathit{currentFrame}, \mathit{state} \} \mid \{\!\mid \mathit{reduce}, \mathit{sync} \mid\!\} \mid \{ \mathit{voxel\_map}, \mathit{work} \} \rrbracket\\
\left(\begin{array}{l}\mu X \bullet \mathit{reduce} \longrightarrow \mathbf{skip} \,;\ \mathit{getFrameState} ? \mathit{currentFrame} ? \mathit{state} \longrightarrow\\ \mathbf{wait}\ 0 \mathrel{..} \mathit{RPW}_{\mathit{TB}} \,;\ \mathit{ReduceAndPartitionWork};\\ \mathit{setWork} ! \mathit{work} \longrightarrow \mathbf{skip} \,;\ \mathit{initColls} \longrightarrow \mathbf{skip} \,;\ \mathit{detect} \longrightarrow \mathbf{skip} \,;\ \mathit{sync} \longrightarrow X\end{array}\right)\\
\quad \llbracket \{ \mathit{voxel\_map}, \mathit{work} \} \mid \{\!\mid \mathit{detect}, \mathit{sync} \mid\!\} \mid \varnothing \rrbracket\\
\left(\begin{array}{l}\mu X \bullet \mathit{detect} \longrightarrow \mathbf{skip} \,;\ \mathit{getWork} ? \mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls1} : \mathit{int} \bullet\\ \mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{\mathit{TB}} \,;\ (\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls1}/\mathit{pcolls}!] \land i? = 1);\\ \mathit{recColls} ! \mathit{colls1} \longrightarrow \mathbf{skip} \,;\ \mathit{output} ? y \longrightarrow \mathbf{skip} \,;\ \mathit{sync} \longrightarrow X\end{array}\right)\\
\quad \llbracket \varnothing \mid \{\!\mid \mathit{detect}, \mathit{output}, \mathit{sync} \mid\!\} \mid \varnothing \rrbracket\\
\left(\begin{array}{l}\mu X \bullet \mathit{detect} \longrightarrow \mathbf{skip} \,;\ \mathit{getWork} ? \mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls2} : \mathit{int} \bullet\\ \mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{\mathit{TB}} \,;\ (\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls2}/\mathit{pcolls}!] \land i? = 2);\\ \mathit{recColls} ! \mathit{colls2} \longrightarrow \mathbf{skip} \,;\ \mathit{output} ? y \longrightarrow \mathbf{skip} \,;\ \mathit{sync} \longrightarrow X\end{array}\right)\\
\quad \llbracket \varnothing \mid \{\!\mid \mathit{detect}, \mathit{output}, \mathit{sync} \mid\!\} \mid \varnothing \rrbracket\\
\left(\begin{array}{l}\mu X \bullet \mathit{detect} \longrightarrow \mathbf{skip} \,;\ \mathit{getWork} ? \mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls3} : \mathit{int} \bullet\\ \mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{\mathit{TB}} \,;\ (\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls3}/\mathit{pcolls}!] \land i? = 3);\\ \mathit{recColls} ! \mathit{colls3} \longrightarrow \mathbf{skip} \,;\ \mathit{output} ? y \longrightarrow \mathbf{skip} \,;\ \mathit{sync} \longrightarrow X\end{array}\right)\\
\quad \llbracket \varnothing \mid \{\!\mid \mathit{detect}, \mathit{output}, \mathit{sync} \mid\!\} \mid \varnothing \rrbracket\\
\left(\begin{array}{l}\mu X \bullet \mathit{detect} \longrightarrow \mathbf{skip} \,;\ \mathit{getWork} ? \mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls4} : \mathit{int} \bullet\\ \mathbf{wait}\ 0 \mathrel{..} \mathit{CPC}_{\mathit{TB}} \,;\ (\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls4}/\mathit{pcolls}!] \land i? = 4);\\ \mathit{recColls} ! \mathit{colls4} \longrightarrow \mathbf{skip} \,;\ \mathit{output} ? y \longrightarrow \mathbf{skip} \,;\ \mathit{sync} \longrightarrow X\end{array}\right)\\
\quad \llbracket \varnothing \mid \{\!\mid \mathit{output}, \mathit{sync} \mid\!\} \mid \varnothing \rrbracket\\
\left(\begin{array}{l}\mu X \bullet \mathit{output} ? \mathit{collisions} \longrightarrow\\ \mathbf{var}\ \mathit{colls} : \mathit{int} \bullet \mathbf{wait}\ 0 \mathrel{..} \mathit{CC}_{\mathit{TB}} \,;\ \mathit{CalcCollisions};\\ (\mathit{output\_collisions} ! \mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT\_DL} \,;\ \mathit{sync} \longrightarrow X\end{array}\right)
\end{array}\right)\\
\quad \llbracket \{ \mathit{currentFrame}, \mathit{state}, \mathit{voxel\_map}, \mathit{work} \} \mid \{\!\mid \mathit{sync} \mid\!\} \mid \varnothing \rrbracket\ \mathit{Cycle}
\end{array}\right)\\
\quad \llbracket \{ \mathit{currentFrame}, \mathit{state}, \mathit{voxel\_map}, \mathit{work} \} \mid \{\!\mid \mathit{setFrameState}, \mathit{getFrameState},\\
\qquad \mathit{setWork}, \mathit{getWork}, \mathit{initColls}, \mathit{recColls}, \mathit{output} \mid\!\} \mid \{ \mathit{collisions} \} \rrbracket\\
\left(\begin{array}{l}
\left(\begin{array}{l}\mathbf{var}\ \mathit{currentFrame} : \mathit{RawFrame} \bullet\\ \mathbf{var}\ \mathit{state} : \mathit{StateTable} \bullet\\ \quad \mu X \bullet \left(\begin{array}{l}(\mathit{setFrameState} ? v_1 ? v_2 \longrightarrow \mathit{currentFrame}, \mathit{state} := v_1, v_2) \mathrel{\Box}\\ (\mathit{getFrameState} ! \mathit{currentFrame} ! \mathit{state} \longrightarrow \mathbf{skip})\end{array}\right) ;\ X\end{array}\right) \interleave\\
\left(\begin{array}{l}\mathbf{var}\ \mathit{work} : \mathit{Partition} \bullet\\ \quad \mu X \bullet \left(\begin{array}{l}(\mathit{setWork} ? v \longrightarrow \mathit{work} := v) \mathrel{\Box}\\ (\mathit{getWork} ! \mathit{work} \longrightarrow \mathbf{skip})\end{array}\right) ;\ X\end{array}\right) \interleave\\
\left(\mu X \bullet \left(\begin{array}{l}(\mathit{initColls} \longrightarrow \mathit{collisions} := 0) \mathrel{\Box}\\ (\mathit{recColls} ? x \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} \mathit{RC}_{\mathit{TB}} \,;\ \mathit{collisions} := \mathit{collisions} + x)) \mathrel{\Box}\\ (\mathit{output} ! \mathit{collisions} \longrightarrow \mathbf{skip})\end{array}\right) ;\ X\right)
\end{array}\right)
\end{array}\right) \setminus
$$

$$
\{\!\mid \mathit{reduce}, \mathit{detect}, \mathit{output}, \mathit{setFrameState}, \mathit{getFrameState}, \mathit{getWork}, \mathit{setWork}, \mathit{initColls}, \mathit{recColls}, \mathit{sync} \mid\!\}
$$

The highlighted communication has been inserted into the reducer handler during the collapsing of one of the control fragments. We also observe that right-hand action now encapsulates the shared data for the *collisions* variable. The variable declaration, however, has not been localised yet by the law.

$\ldots \equiv$ "application of elementary distribution laws to localised variable declarations"

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mu X \bullet (next\_frame\,?\,frame \longrightarrow (\mathbf{wait}\ 0\,..\,SF_{TB}\ ;\ StoreFrame)) \blacktriangleleft INP\_DL;\\
setFrameState\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}\ ;\ reduce \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\{currentFrame, state\} \mid \{\!|\, reduce, sync \,|\!\} \mid \{voxel\_map, work\}]\!]\\
\left(\begin{array}{l}
\mu X \bullet reduce \longrightarrow \mathbf{skip}\ ;\ getFrameState\,?\,currentFrame\,?\,state \longrightarrow\\
\mathbf{wait}\ 0\,..\,RPW_{TB}\ ;\ ReduceAndPartitionWork;\\
setWork\,!\,work \longrightarrow \mathbf{skip}\ ;\ initColls \longrightarrow \mathbf{skip}\ ;\ detect \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\{voxel\_map, work\} \mid \{\!|\, detect, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls1 : int \bullet\\
\mathbf{wait}\ 0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \wedge i? = 1);\\
recColls\,!\,colls1 \longrightarrow \mathbf{skip}\ ;\ output\,?\,y \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls2 : int \bullet\\
\mathbf{wait}\ 0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \wedge i? = 2);\\
recColls\,!\,colls2 \longrightarrow \mathbf{skip}\ ;\ output\,?\,y \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls3 : int \bullet\\
\mathbf{wait}\ 0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \wedge i? = 3);\\
recColls\,!\,colls3 \longrightarrow \mathbf{skip}\ ;\ output\,?\,y \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls4 : int \bullet\\
\mathbf{wait}\ 0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \wedge i? = 4);\\
recColls\,!\,colls4 \longrightarrow \mathbf{skip}\ ;\ output\,?\,y \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet output\,?\,collisions \longrightarrow\\
\mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0\,..\,CC_{TB}\ ;\ CalcCollisions;\\
(output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL\ ;\ sync \longrightarrow X
\end{array}\right)
\end{array}\right)\\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \{\!|\, sync \,|\!\} \mid \varnothing]\!]\ Cycle
\end{array}\right)\\
[\![\{currentFrame, state, voxel\_map, work\} \mid\\
\quad \{\!|\, setFrameState, getFrameState, setWork, getWork, initColls, recColls, output \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ currentFrame : RawFrame \bullet\\
\mathbf{var}\ state : StateTable \bullet\\
\quad \mu X \bullet \left(\begin{array}{l}
(setFrameState\,?\,v_1\,?\,v_2 \longrightarrow currentFrame, state := v_1, v_2) \Box\\
(getFrameState\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip})
\end{array}\right);\ X
\end{array}\right) \interleave\\
\left(\begin{array}{l}
\mathbf{var}\ work : Partition \bullet\\
\quad \mu X \bullet \left(\begin{array}{l}
(setWork\,?\,v \longrightarrow work := v) \Box\\
(getWork\,!\,work \longrightarrow \mathbf{skip})
\end{array}\right);\ X
\end{array}\right) \interleave\\
\left(\begin{array}{l}
\mathbf{var}\ collisions : int \bullet\\
\quad \mu X \bullet \left(\begin{array}{l}
(initColls \longrightarrow collisions := 0) \Box\\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ collisions := collisions + x)) \Box\\
(output\,!\,collisions \longrightarrow \mathbf{skip})
\end{array}\right);\ X
\end{array}\right)
\end{array}\right)
\end{array}\right) \setminus
$$

$$\{\!|\, reduce, detect, output, setFrameState, getFrameState, getWork, setWork, initColls, recColls, sync \,|\!\}$$

This concludes Stage 2 of the SH phase. The refinement strategy is more involved here, requiring several auxiliary steps before and after application of the respective high-level law. Whereas the application of Pattern 1 and Pattern 2 generally lend themselves fairly well for automation, some of the steps during the application of Pattern 3 are expected to require more guidance by and refinement effort by the developer.

The next stage targets the synchronisations on the channel *output*. This on one hand releases the output handler but also acts as a barrier for the detection handlers.

5.3.4 Stage 3

Stage 3 of SH introduces shared data to refine control mechanisms that still may exist in the model. In the example, this is the the barrier-like synchronisation on $output\,?\,y \longrightarrow \mathbf{skip}$ within the detection handlers. As in the previous stage, it turns out that we cannot apply the respective law barrier-sync-design immediately but have to peform some pre-processing to transform the *System* action into the right shape. Below we recapture the current parallelism of handlers.

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mu X \bullet (next\_frame\,?\,frame \longrightarrow (\mathbf{wait}\,0\,..\,SF_{TB}\;;\;StoreFrame)) \blacktriangleleft INP\_DL;\\
setFrameState\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}\;;\;reduce \longrightarrow \mathbf{skip}\;;\;sync \longrightarrow X
\end{array}\right)\\
\quad [\![\{currentFrame, state\} \mid \{\!|\, reduce, sync \,|\!\} \mid \{voxel\_map, work\}]\!]\\
\left(\begin{array}{l}
\mu X \bullet reduce \longrightarrow \mathbf{skip}\;;\;getFrameState\,?\,currentFrame\,?\,state \longrightarrow\\
\mathbf{wait}\,0\,..\,RPW_{TB}\;;\;ReduceAndPartitionWork;\\
setWork\,!\,work \longrightarrow \mathbf{skip}\;;\;initColls \longrightarrow \mathbf{skip}\;;\;detect \longrightarrow \mathbf{skip}\;;\;sync \longrightarrow X
\end{array}\right)\\
\quad [\![\{voxel\_map, work\} \mid \{\!|\, detect, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\;;\;getWork\,?\,work \longrightarrow \mathbf{var}\;colls1 : int \bullet\\
\mathbf{wait}\,0\,..\,CPC_{TB}\;;\;(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1);\\
recColls\,!\,colls1 \longrightarrow \mathbf{skip}\;;\;output\,?\,y \longrightarrow \mathbf{skip}\;;\;sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\;;\;getWork\,?\,work \longrightarrow \mathbf{var}\;colls2 : int \bullet\\
\mathbf{wait}\,0\,..\,CPC_{TB}\;;\;(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2);\\
recColls\,!\,colls2 \longrightarrow \mathbf{skip}\;;\;output\,?\,y \longrightarrow \mathbf{skip}\;;\;sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\;;\;getWork\,?\,work \longrightarrow \mathbf{var}\;colls3 : int \bullet\\
\mathbf{wait}\,0\,..\,CPC_{TB}\;;\;(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3);\\
recColls\,!\,colls3 \longrightarrow \mathbf{skip}\;;\;output\,?\,y \longrightarrow \mathbf{skip}\;;\;sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\;;\;getWork\,?\,work \longrightarrow \mathbf{var}\;colls4 : int \bullet\\
\mathbf{wait}\,0\,..\,CPC_{TB}\;;\;(\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4);\\
recColls\,!\,colls4 \longrightarrow \mathbf{skip}\;;\;output\,?\,y \longrightarrow \mathbf{skip}\;;\;sync \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet output\,?\,collisions \longrightarrow\\
\mathbf{var}\;colls : int \bullet \mathbf{wait}\,0\,..\,CC_{TB}\;;\;CalcCollisions;\\
(output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL\;;\;sync \longrightarrow X
\end{array}\right)
\end{array}\right)\\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \{\!|\, sync \,|\!\} \mid \varnothing\,]\!]\; Cycle
\end{array}\right)\\
[\![\{currentFrame, state, voxel\_map, work\} \mid\\
\quad \{\!|\, setFrameState, getFrameState, setWork, getWork, initColls, recColls, output \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\ldots \mid\mid\mid\\
\left(\begin{array}{l}
\mathbf{var}\;collisions : int \bullet\\
\quad \mu X \bullet \left(\begin{array}{l}
(initColls \longrightarrow collisions := 0)\;\Box\\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\;;\;collisions := collisions + x))\;\Box\\
(output\,!\,collisions \longrightarrow \mathbf{skip})
\end{array}\right)\;;\;X
\end{array}\right)
\end{array}\right)
\end{array}\right) \setminus
$$

$$\{\!|\, reduce, detect, output, setFrameState, getFrameState, getWork, setWork, initColls, recColls, sync \,|\!\}$$

All synchronisations on the *output* channel are highlighted in red. This channel, like *reduce* and *detect* in Stage 1, fulfils a dual purpose of communicating the number of detected collisions to the output handler as well as acting as a synchronisation barrier for the detector handlers and output handler to ensure that the collisions are only communicated once all four detector handlers have committed their results. Hence, *output* controls when the output handler is release.

Thought the scenario is somewhat similar to the one for Pattern 1, here we cannot simply apply the seq-share-1 law in order to introduce the control channel for the software event that releases the output handler. For one this is because the data communicated through *output* has already been encapsulated in Stage 2. We therefore use a different strategy outlined below.

1. Introduce a new typeless channel *output* to replace the originial channel *output* of type *int* in all places where we are merely interested in the control aspect and rename *output* into *getColls*. The new channel isolates the control aspect where *getColls* provides the means for accessing the data.

2. Eliminate the residual parallel fragment results from the above introduction.

3. Use the synch-barrier-design law to refine the barrier synchronisation mechanism.

To proceed with (1) we require a specialised channel replacement law. Possibly, this law can be specified in a more general manner, for instance, by some inductively-defined substitution procedure. Essentially, it replaces occurrences of prefixes of the form $output\,?\,x \longrightarrow A$ by simple prefixes $outout \longrightarrow A$ where x is not used in A and otherwise by $getColls\,?\,x \longrightarrow A$. This replacement is justified by two facts.

1. The original channel *output* is concealed in the system action.

2. Inclusion of an additional parallel control fragment:

$$\left(\mu X \bullet output \longrightarrow getColls\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X\right)$$

The control fragment is expected to be eliminated in the usual way. Here, this is collapsing it with the handler that outputs the collisions. The local refinement for this transformation is given below.

$$\left(\begin{array}{l}\left(\begin{array}{l}\mu X \bullet getColls\,?\,collisions \longrightarrow \\ \mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0\,..\,CC_{TB}\,;\ \ CalcCollisions; \\ (output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL\,;\ sync \longrightarrow X\end{array}\right) \\ \quad [\![\varnothing \mid \{\!|\ getColls\ |\!\} \mid \varnothing]\!] \\ \left(\mu X \bullet output \longrightarrow getColls\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X\right)\end{array}\right)$$

$\sqsubseteq$ "distributing recursion in the parallel actions and collapsing the parallelism using step laws"

$$\left(\begin{array}{l}\mu X \bullet output \longrightarrow getColls\,?\,collisions \longrightarrow \\ \mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0\,..\,CC_{TB}\,;\ \ CalcCollisions; \\ (output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL\,;\ sync \longrightarrow X\end{array}\right)$$

The collapsing of parallelism introduces the prefix $output \longrightarrow \ldots$ ' into the output handler and thereby establishes the desired design in which *output* acts as a software event that releases this handler.

The above refinement, as noted, rests on a channel replacement law that, however, will not be discussed in more detail here; instead, we just present the *System* action that we expect to result from its application.

$\sqsubseteq$ "specialised channel replacement law and distributing residual parallel control action"

$$
\left(\begin{array}{l}
\left(\left(\left(\begin{array}{l}
\left(\begin{array}{l}\mu X \bullet (next\_frame\,?\,frame \longrightarrow (\mathbf{wait}\ 0\,..\,SF_{TB}\ ;\ StoreFrame)) \blacktriangleleft INP\_DL;\\ setFrameState\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}\ ;\ reduce \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X\end{array}\right)\\
\quad [\![\{currentFrame, state\} \mid \{\!|\, reduce, sync \,|\!\} \mid \{voxel\_map, work\}]\!]\\
\left(\begin{array}{l}\mu X \bullet reduce \longrightarrow \mathbf{skip}\ ;\ getFrameState\,?\,currentFrame\,?\,state \longrightarrow\\ \mathbf{wait}\ 0\,..\,RPW_{TB}\ ;\ ReduceAndPartitionWork;\\ setWork\,!\,work \longrightarrow \mathbf{skip}\ ;\ initColls \longrightarrow \mathbf{skip}\ ;\ detect \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X\end{array}\right)\\
\quad [\![\{voxel\_map, work\} \mid \{\!|\, detect, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls1 : int \bullet\\ \mathbf{wait}\ 0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \wedge i? = 1);\\ recColls\,!\,colls1 \longrightarrow \mathbf{skip}\ ;\ output \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls2 : int \bullet\\ \mathbf{wait}\ 0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \wedge i? = 2);\\ recColls\,!\,colls2 \longrightarrow \mathbf{skip}\ ;\ output \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls3 : int \bullet\\ \mathbf{wait}\ 0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \wedge i? = 3);\\ recColls\,!\,colls3 \longrightarrow \mathbf{skip}\ ;\ output \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, detect, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls4 : int \bullet\\ \mathbf{wait}\ 0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \wedge i? = 4);\\ recColls\,!\,colls4 \longrightarrow \mathbf{skip}\ ;\ output \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, output, sync \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}\mu X \bullet output \longrightarrow getColls\,?\,collisions \longrightarrow\\ \mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0\,..\,CC_{TB}\ ;\ CalcCollisions;\\ (output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL\ ;\ sync \longrightarrow X\end{array}\right)
\end{array}\right)\right.\right.\\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \{\!|\, sync \,|\!\} \mid \varnothing\,]\!]\ Cycle \Bigg)\\
[\![\{currentFrame, state, voxel\_map, work\} \mid\\
\quad \{\!|\, [set/get]FrameState, [set/get]Work, initColls, recColls, getColls, output \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\left(\begin{array}{l}\mathbf{var}\ currentFrame : RawFrame \bullet\\ \mathbf{var}\ state : StateTable \bullet\\ \quad \mu X \bullet \left(\begin{array}{l}(setFrameState\,?\,v_1\,?\,v_2 \longrightarrow currentFrame, state := v_1, v_2)\ \Box\\ (getFrameState\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip})\end{array}\right) ;\ X\end{array}\right) |||\\
\left(\begin{array}{l}\mathbf{var}\ work : Partition \bullet\\ \quad \mu X \bullet \left(\begin{array}{l}(setWork\,?\,v \longrightarrow work := v)\ \Box\\ (getWork\,!\,work \longrightarrow \mathbf{skip})\end{array}\right) ;\ X\end{array}\right) |||\\
\left(\begin{array}{l}\mathbf{var}\ collisions : int \bullet\\ \quad \mu X \bullet \left(\begin{array}{l}(initColls \longrightarrow collisions := 0)\ \Box\\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB}\ ;\ collisions := collisions + x))\ \Box\\ (getColls\,!\,collisions \longrightarrow \mathbf{skip})\end{array}\right) ;\ X\end{array}\right)
\end{array}\right)
\end{array}\right) \setminus
$$

$$\{\!|\, reduce, detect, output, [set/get]FrameState, [set/get]Work, initColls, recColls, getColls, sync \,|\!\}$$

We obersve the new channels *output* and *getColls*. Moreover, all channels used int the right-hand parallel action that encapsulates shared data are now channels that model either method calls or direct access to shared variables. We now proceed applying the law sync-barrier-design.

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mu X \bullet (\mathit{next\_frame}\,?\,\mathit{frame} \longrightarrow (\mathbf{wait}\,0\,..\,\mathit{SF}_{TB}\,;\ \mathit{StoreFrame})) \blacktriangleleft \mathit{INP\_DL};\\
\mathit{setFrameState}\,!\,\mathit{currentFrame}\,!\,\mathit{state} \longrightarrow \mathbf{skip}\,;\ \mathit{reduce} \longrightarrow \mathbf{skip}\,;\ \mathit{sync} \longrightarrow X
\end{array}\right)\\
\quad [\![\{\mathit{currentFrame}, \mathit{state}\} \mid \{\!|\, \mathit{reduce}, \mathit{sync} \,|\!\} \mid \{\mathit{voxel\_map}, \mathit{work}\}]\!]\\
\left(\begin{array}{l}
\mu X \bullet \mathit{reduce} \longrightarrow \mathbf{skip}\,;\ \mathit{getFrameState}\,?\,\mathit{currentFrame}\,?\,\mathit{state} \longrightarrow\\
\mathbf{wait}\,0\,..\,\mathit{RPW}_{TB}\,;\ \mathit{ReduceAndPartitionWork};\\
\mathit{setWork}\,!\,\mathit{work} \longrightarrow \mathbf{skip}\,;\ \mathit{initColls} \longrightarrow \mathbf{skip}\,;\ \mathit{detect} \longrightarrow \mathbf{skip}\,;\ \mathit{sync} \longrightarrow X
\end{array}\right)\\
\quad [\![\{\mathit{voxel\_map}, \mathit{work}\} \mid \{\!|\, \mathit{detect}, \mathit{sync} \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet \mathit{detect} \longrightarrow \mathbf{skip}\,;\ \mathit{getWork}\,?\,\mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls}1 : \mathit{int} \bullet\\
\mathbf{wait}\,0\,..\,\mathit{CPC}_{TB}\,;\ (\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}1/\mathit{pcolls}!] \land i? = 1);\\
\mathit{recColls}\,!\,\mathit{colls}1 \longrightarrow \mathbf{skip}\,;\ \mathit{notify}\,!\,1 \longrightarrow \mathbf{skip}\,;\ \mathit{sync} \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, \mathit{detect}, \mathit{output}, \mathit{sync} \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet \mathit{detect} \longrightarrow \mathbf{skip}\,;\ \mathit{getWork}\,?\,\mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls}2 : \mathit{int} \bullet\\
\mathbf{wait}\,0\,..\,\mathit{CPC}_{TB}\,;\ (\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}2/\mathit{pcolls}!] \land i? = 2);\\
\mathit{recColls}\,!\,\mathit{colls}2 \longrightarrow \mathbf{skip}\,;\ \mathit{notify}\,!\,2 \longrightarrow \mathbf{skip}\,;\ \mathit{sync} \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, \mathit{detect}, \mathit{output}, \mathit{sync} \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet \mathit{detect} \longrightarrow \mathbf{skip}\,;\ \mathit{getWork}\,?\,\mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls}3 : \mathit{int} \bullet\\
\mathbf{wait}\,0\,..\,\mathit{CPC}_{TB}\,;\ (\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}3/\mathit{pcolls}!] \land i? = 3);\\
\mathit{recColls}\,!\,\mathit{colls}3 \longrightarrow \mathbf{skip}\,;\ \mathit{notify}\,!\,3 \longrightarrow \mathbf{skip}\,;\ \mathit{sync} \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, \mathit{detect}, \mathit{output}, \mathit{sync} \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet \mathit{detect} \longrightarrow \mathbf{skip}\,;\ \mathit{getWork}\,?\,\mathit{work} \longrightarrow \mathbf{var}\ \mathit{colls}4 : \mathit{int} \bullet\\
\mathbf{wait}\,0\,..\,\mathit{CPC}_{TB}\,;\ (\exists\, i? : \mathbb{Z} \bullet \mathit{CalcPartCollisions}[\mathit{colls}4/\mathit{pcolls}!] \land i? = 4);\\
\mathit{recColls}\,!\,\mathit{colls}4 \longrightarrow \mathbf{skip}\,;\ \mathit{notify}\,!\,4 \longrightarrow \mathbf{skip}\,;\ \mathit{sync} \longrightarrow X
\end{array}\right)\\
\quad [\![\varnothing \mid \{\!|\, \mathit{output}, \mathit{sync} \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\mu X \bullet \mathit{output} \longrightarrow \mathit{getColls}\,?\,\mathit{collisions} \longrightarrow\\
\mathbf{var}\ \mathit{colls} : \mathit{int} \bullet \mathbf{wait}\,0\,..\,\mathit{CC}_{TB}\,;\ \mathit{CalcCollisions};\\
(\mathit{output\_collisions}\,!\,\mathit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \mathit{OUT\_DL}\,;\ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}\right)\\
\quad [\![\{\mathit{currentFrame}, \mathit{state}, \mathit{voxel\_map}, \mathit{work}\} \mid \{\!|\, \mathit{sync} \,|\!\} \mid \varnothing]\!]\ \mathit{Cycle}
\end{array}\right)\\
\quad [\![\{\mathit{currentFrame}, \mathit{state}, \mathit{voxel\_map}, \mathit{work}\} \mid \{\!|\, \mathit{detect}, \mathit{start}, \mathit{sync} \,|\!\} \mid \varnothing]\!]\\
(\mu X \bullet \mathit{start} \longrightarrow \mathit{detect} \longrightarrow \mathbf{skip}\,;\ \mathit{sync} \longrightarrow X)
\end{array}\right)\\
\quad [\![\{\mathit{currentFrame}, \mathit{state}, \mathit{voxel\_map}, \mathit{work}\} \mid\\
\qquad \{\!|\, [\mathit{set}/\mathit{get}]\mathit{FrameState}, [\mathit{set}/\mathit{get}]\mathit{Work}, \mathit{initColls}, \mathit{recColls}, \mathit{getColls} \,|\!\} \mid \varnothing]\!]\\
\left(\begin{array}{l}
\ldots \,|\!|\!|\\
\left(\begin{array}{l}
\mathbf{var}\ \mathit{active} : \mathbb{P}\,(1\,..\,n) \bullet\\
\quad \mu X \bullet \left(\begin{array}{l}
(\mathit{start} \longrightarrow \mathit{active} := 1\,..\,n)\\
\quad \Box\\
(\mathit{notify}\,?\,x \longrightarrow \left(\begin{array}{l}
\mathit{active} := \mathit{active} \setminus \{x\};\\
\mathbf{if}\ \mathit{active} = \varnothing \longrightarrow \mathit{output} \longrightarrow \mathbf{skip}\\
[\!]\ \neg\, \mathit{active} = \varnothing \longrightarrow \mathbf{skip}\\
\mathbf{fi}
\end{array}\right))
\end{array}\right)\,;\ X
\end{array}\right)
\end{array}\right)
\end{array}\right)
$$

$\{\!|\, \mathit{reduce}, \mathit{detect}, \mathit{output}, \mathit{start}, \mathit{notify}, [\mathit{set}/\mathit{get}]\mathit{FrameState}, [\mathit{set}/\mathit{get}]\mathit{Work}, \mathit{initColls}, \mathit{recColls}, \mathit{getColls}, \mathit{sync} \,|\!\}$

Once again a control fragment arises as a consequence of applying the law. We distribute it into the reducer handler. This gives rise to the following *System* action.

$\sqsubseteq$ "distribution of control fragment using parallel step laws"

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mu X \bullet (next\_frame ? frame \longrightarrow (\mathbf{wait}\ 0 \mathrel{..} SF_{TB} ;\ StoreFrame)) \blacktriangleleft INP\_DL; \\
setFrameState ! currentFrame ! state \longrightarrow \mathbf{skip} ;\ reduce \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\{currentFrame, state\} \mid \{\!| reduce, sync |\!\} \mid \{voxel\_map, work\}]\!] \\
\left(\begin{array}{l}
\mu X \bullet reduce \longrightarrow \mathbf{skip} ;\ getFrameState ? currentFrame ? state \longrightarrow \\
\mathbf{wait}\ 0 \mathrel{..} RPW_{TB} ;\ ReduceAndPartitionWork ;\ setWork ! work \longrightarrow \mathbf{skip}; \\
initColls \longrightarrow \mathbf{skip} ;\ start \longrightarrow \mathbf{skip} ;\ detect \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\{voxel\_map, work\} \mid \{\!| detect, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls1 : int \bullet \\
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB} ;\ (\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\
recColls ! colls1 \longrightarrow \mathbf{skip} ;\ notify ! 1 \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls2 : int \bullet \\
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB} ;\ (\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\
recColls ! colls2 \longrightarrow \mathbf{skip} ;\ notify ! 2 \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls3 : int \bullet \\
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB} ;\ (\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\
recColls ! colls3 \longrightarrow \mathbf{skip} ;\ notify ! 3 \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls4 : int \bullet \\
\mathbf{wait}\ 0 \mathrel{..} CPC_{TB} ;\ (\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\
recColls ! colls4 \longrightarrow \mathbf{skip} ;\ notify ! 4 \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet output \longrightarrow getColls ? collisions \longrightarrow \\
\mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0 \mathrel{..} CC_{TB} ;\ CalcCollisions; \\
(output\_collisions ! colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL ;\ sync \longrightarrow X
\end{array}\right)
\end{array}\right) \\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \{\!| sync |\!\} \mid \varnothing]\!]\ Cycle
\end{array}\right) \\
[\![\{currentFrame, state, voxel\_map, work\} \mid \\
\quad \{\!| [set/get]FrameState, [set/get]Work, initColls, recColls, getColls, output |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\ldots \interleave \\
\left(\begin{array}{l}
\mathbf{var}\ active : \mathbb{P}(1 \mathrel{..} n) \bullet \\
\quad \mu X \bullet \left(\begin{array}{l}
(start \longrightarrow active := 1 \mathrel{..} n) \\
\quad \Box \\
(notify ? x \longrightarrow \left(\begin{array}{l}
active := active \setminus \{x\}; \\
\mathbf{if}\ active = \varnothing \longrightarrow output \longrightarrow \mathbf{skip} \\
[\!]\ \neg\ active = \varnothing \longrightarrow \mathbf{skip} \\
\mathbf{fi}
\end{array}\right))
\end{array}\right) ;\ X
\end{array}\right)
\end{array}\right)
\end{array}\right) \setminus \\
\{\!| reduce, detect, output, start, notify, [set/get]FrameState, [set/get]Work, initColls, recColls, getColls, sync |\!\}
$$

This concludes Stage 3. All remaining control behaviour has been removed and as a result, additional shared data was introduced. In the next and last stage of the SH phase we factor out the *MArea* action and collapse parallelism between actions that encapsulate shared data in the same object in the program.

5.3.5 Stage 4

The encapsulated data that has emerged is now extracted in a separate local action *MArea*.

$$MArea \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ currentFrame : RawFrame \bullet \\
\mathbf{var}\ state : StateTable \bullet \\
\qquad \mu X \bullet \left(\begin{array}{l} (setFrameState\,?\,v_1\,?\,v_2 \longrightarrow currentFrame, state := v_1, v_2) \mathrel{\Box} \\ (getFrameState\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}) \end{array}\right) ;\ X
\end{array}\right) \mathbin{|||} \\
\left(\begin{array}{l}
\mathbf{var}\ work : Partition \bullet \\
\qquad \mu X \bullet \left(\begin{array}{l} (setWork\,?\,v \longrightarrow work := v) \mathrel{\Box} \\ (getWork\,!\,work \longrightarrow \mathbf{skip}) \end{array}\right) ;\ X
\end{array}\right) \mathbin{|||} \\
\left(\begin{array}{l}
\mathbf{var}\ collisions : int \bullet \\
\qquad \mu X \bullet \left(\begin{array}{l} (initColls \longrightarrow collisions := 0) \mathrel{\Box} \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB} ;\ collisions := collisions + x)) \mathrel{\Box} \\ (getColls\,!\,collisions \longrightarrow \mathbf{skip}) \end{array}\right) ;\ X
\end{array}\right) \mathbin{|||} \\
\left(\begin{array}{l}
\mathbf{var}\ active : \mathbb{P}\,(1\,..\,n) \bullet \\
\qquad \mu X \bullet \left(\begin{array}{l} (start \longrightarrow active := 1\,..\,n) \\ \quad \Box \\ (notify\,?\,x \longrightarrow \left(\begin{array}{l} active := active \setminus \{x\}; \\ \mathbf{if}\ active = \varnothing \longrightarrow output \longrightarrow \mathbf{skip} \\ [\!]\ \neg\ active = \varnothing \longrightarrow \mathbf{skip} \\ \mathbf{fi} \end{array}\right)) \end{array}\right) ;\ X
\end{array}\right)
\end{array}\right)$$

Because the shared variables *currentFrame*, *state*, *work* and *collisions* are fields of the same object in the program (class `CDxMission`), we collapse the parallelism of recursions into a single recursion to reflect that the underlying methods synchronise on a common lock.

$\sqsubseteq$ "specialised laws to collapse parallelisms of recursions in *MArea*"

$$MArea \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ currentFrame : RawFrame \bullet \\
\mathbf{var}\ state : StateTable \bullet \\
\mathbf{var}\ work : Partition \bullet \\
\mathbf{var}\ collisions : int \bullet \\
\qquad \mu X \bullet \left(\begin{array}{l} (setFrameState\,?\,v_1\,?\,v_2 \longrightarrow currentFrame, state := v_1, v_2) \mathrel{\Box} \\ (getFrameState\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}) \mathrel{\Box} \\ (setWork\,?\,v \longrightarrow work := v) \mathrel{\Box} \\ (getWork\,!\,work \longrightarrow \mathbf{skip}) \mathrel{\Box} \\ (initColls \longrightarrow collisions := 0) \mathrel{\Box} \\ (recColls\,?\,x \longrightarrow (\mathbf{wait}\ 0\,..\,RC_{TB} ;\ collisions := collisions + x)) \mathrel{\Box} \\ (getColls\,!\,collisions \longrightarrow \mathbf{skip}) \end{array}\right) ;\ X
\end{array}\right) \mathbin{|||} \\
\left(\begin{array}{l}
\mathbf{var}\ active : \mathbb{P}\,(1\,..\,n) \bullet \\
\qquad \mu X \bullet \left(\begin{array}{l} (start \longrightarrow active := 1\,..\,n) \\ \quad \Box \\ (notify\,?\,x \longrightarrow \left(\begin{array}{l} active := active \setminus \{x\}; \\ \mathbf{if}\ active = \varnothing \longrightarrow output \longrightarrow \mathbf{skip} \\ [\!]\ \neg\ active = \varnothing \longrightarrow \mathbf{skip} \\ \mathbf{fi} \end{array}\right)) \end{array}\right) ;\ X
\end{array}\right)
\end{array}\right)$$

The precise laws needed for the above refinement are future work and furthermore they are in general likely to be non-compositional. Namely, because collapsing parallelism above results in an action that is less willing to communicate in that nondeterministic waits may be introduced. Above, that is $\mathbf{wait}\ 0 \mathinner{..} RC_{TB}$. The refinement may be justified by reducing time budgets in other places but this is future work.

For the $System$ action we obtain the following definition.

$$
\begin{array}{l}
System \mathrel{\widehat=} \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mu X \bullet (next\_frame ? frame \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} SF_{TB} ;\ StoreFrame)) \blacktriangleleft INP\_DL; \\
setFrameState ! currentFrame ! state \longrightarrow \mathbf{skip} ;\ reduce \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\{currentFrame, state\} \mid \{\!| reduce, sync |\!\} \mid \{voxel\_map, work\}]\!] \\
\left(\begin{array}{l}
\mu X \bullet reduce \longrightarrow \mathbf{skip} ;\ getFrameState ? currentFrame ? state \longrightarrow \\
\mathbf{wait}\ 0 \mathinner{..} RPW_{TB} ;\ ReduceAndPartitionWork ;\ setWork ! work \longrightarrow \mathbf{skip}; \\
initColls \longrightarrow \mathbf{skip} ;\ start \longrightarrow \mathbf{skip} ;\ detect \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\{voxel\_map, work\} \mid \{\!| detect, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls1 : int \bullet \\
\mathbf{wait}\ 0 \mathinner{..} CPC_{TB} ;\ (\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\
recColls ! colls1 \longrightarrow \mathbf{skip} ;\ notify ! 1 \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls2 : int \bullet \\
\mathbf{wait}\ 0 \mathinner{..} CPC_{TB} ;\ (\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\
recColls ! colls2 \longrightarrow \mathbf{skip} ;\ notify ! 2 \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls3 : int \bullet \\
\mathbf{wait}\ 0 \mathinner{..} CPC_{TB} ;\ (\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\
recColls ! colls3 \longrightarrow \mathbf{skip} ;\ notify ! 3 \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| detect, output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip} ;\ getWork ? work \longrightarrow \mathbf{var}\ colls4 : int \bullet \\
\mathbf{wait}\ 0 \mathinner{..} CPC_{TB} ;\ (\exists i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\
recColls ! colls4 \longrightarrow \mathbf{skip} ;\ notify ! 4 \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| output, sync |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mu X \bullet output \longrightarrow getColls ? collisions \longrightarrow \\
\mathbf{var}\ colls : int \bullet \mathbf{wait}\ 0 \mathinner{..} CC_{TB} ;\ CalcCollisions; \\
(output\_collisions ! colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL ;\ sync \longrightarrow X
\end{array}\right)
\end{array}\right) \\
\quad [\![\{currentFrame, state, voxel\_map, work\} \mid \{\!| sync |\!\} \mid \varnothing]\!]\ Cycle
\end{array}\right) \\
[\![\{currentFrame, state, voxel\_map, work\} \mid \\
\quad \{\!| [set/get]FrameState, [set/get]Work, initColls, recColls, getColls, output |\!\} \mid \varnothing]\!] \\
{\color{red} MArea}
\end{array}\right) \setminus \\
\quad \{\!| reduce, detect, output, start, notify, [set/get]FrameState, [set/get]Work, initColls, recColls, \\
\qquad getColls, sync |\!\}
\end{array}
$$

This concludes Stage 4 and thereby the SH phase. We finally present the entire process for SH.

5.3.6 Process

The complete process for the SH phase is presented below. Its state and data operations are in fact the same as those of *CDxE_MH*, apart from *SetCollisionsFromParts* having been removed.

system *CDxE_SH* $\widehat{=}$ **begin**

state *CDxSHState* == *CDxMHState*

$$
\begin{array}{l}
\underline{\ Init\ }\\
CDxSHState'\\
\hline
currentFrame' = \mathbf{new}\ RawFrame\\
state' = \mathbf{new}\ StateTable\\
voxel\_map' = \mathbf{new}\ HashMap[Vector2d, List[Motion]]\\
work' = \mathbf{new}\ Partition(4)\\
collisions' = 0
\end{array}
$$

$$
\begin{array}{l}
\underline{\ StoreFrame\ }\\
\Delta CDxSHState\\
frame? : Frame\\
\hline
\exists\, posns, posns' : Frame;\ motions, motions' : Frame \mid\\
\quad \mathrm{dom}\, posns = \mathrm{dom}\, motions \land \mathrm{dom}\, posns' = \mathrm{dom}\, motions' \bullet\\
\left(\begin{array}{l}
posns' = frame? \land\\
motions' =\\
\quad \{a : \mathrm{dom}\, posns' \bullet a \mapsto \mathbf{if}\ a \in \mathrm{dom}\, posns\ \mathbf{then}\ (posns'\ a) -_V (posns\ a)\ \mathbf{else}\ ZeroV\} \land\\
posns = F(currentFrame) \land motions = G(currentFrame, state) \land\\
posns' = F(currentFrame') \land motions' = G(currentFrame', state')
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\underline{\ ReduceAndPartitionWork\ }\\
\Delta CDxSHState\\
\hline
currentFrame' = currentFrame \land state' = state\\
\exists\, posns : Frame;\ motions : Frame \mid \mathrm{dom}\, posns = \mathrm{dom}\, motions \bullet\\
\left(\begin{array}{l}
posns = F(currentFrame) \land motions = G(currentFrame, state) \land\\
\left(\begin{array}{l}
\forall\, a_1, a_2 : Aircraft \mid \{a_1, a_2\} \subseteq \mathrm{dom}\, posns \bullet\\
\quad (a_1, a_2) \in CalcCollisionSet(posns, motions) \Rightarrow\\
\quad\quad \left(\begin{array}{l}
\exists\, l : List[Motion] \mid l \in voxel\_map'.values().elems() \bullet\\
\quad MkMotion(a_1, posns\ a_1 -_V motions\ a_1, posns\ a_1) \in l.elems() \land\\
\quad MkMotion(a_2, posns\ a_2 -_V motions\ a_2, posns\ a_2) \in l.elems()
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\underline{\ CalcPartCollisions\ }\\
\Xi CDxSHState\\
pcolls! : int\\
i? : 1 \mathrel{..} 4\\
\hline
pcolls! =\\
\quad \# \left\{\begin{array}{l}
a_1 : Aircraft;\ a_2 : Aircraft \mid\\
\quad \left(\begin{array}{l}
\exists\, l : List[Motion] \mid l \in work.getDetectorWork(i?).elems() \bullet\\
\exists\, v_1, v_2 : Vector;\ w_1, w_2 : Vector \bullet\\
\quad MkMotion(a_1, v_1, w_1) \in l.elems() \land\\
\quad MkMotion(a_2, v_2, w_2) \in l.elems() \land\\
\quad collide((v_1, w_1 -_V v_1), (v_2, w_2 -_V v_2))
\end{array}\right)
\end{array}\right\} \ \mathrm{div}\ 2
\end{array}
$$

$$
\begin{array}{l}
\underline{\quad CalcCollisions \quad} \\
\Xi CDxSHState \\
colls! : \mathbb{N} \\
\hline
\exists\, posns : Frame;\ motions : Frame \mid \mathrm{dom}\, posns = \mathrm{dom}\, motions \bullet \\
\left(\begin{array}{l}
posns = F(currentFrame) \land motions = G(currentFrame, state) \land \\
\exists\, collset : \mathbb{F}\,(Aircraft \times Aircraft) \mid collset = CalcCollisionSet(posns, motions) \bullet \\
\quad (\#\, collset = 0 \land colls! = 0) \lor (\#\, collset > 0 \land colls! \geq (\#\, collset)\ \mathrm{div}\ 2)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
InputFrameHandler \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet (next\_frame\,?\,frame \longrightarrow (\mathbf{wait}\,0\,..\,SF_{TB}\ ;\ StoreFrame)) \blacktriangleleft INP\_DL; \\
setFrameState\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}\ ;\ reduce \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
ReducerHandler \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet reduce \longrightarrow \mathbf{skip}\ ;\ getFrameState\,?\,currentFrame\,?\,state \longrightarrow \\
\mathbf{wait}\,0\,..\,RPW_{TB}\ ;\ ReduceAndPartitionWork\ ;\ setWork\,!\,work \longrightarrow \mathbf{skip}; \\
initColls \longrightarrow \mathbf{skip}\ ;\ start \longrightarrow \mathbf{skip}\ ;\ detect \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
DetectorHandler1 \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls1 : int \bullet \\
\mathbf{wait}\,0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls1/pcolls!] \land i? = 1); \\
recColls\,!\,colls1 \longrightarrow \mathbf{skip}\ ;\ notify\,!\,1 \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
DetectorHandler2 \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls2 : int \bullet \\
\mathbf{wait}\,0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls2/pcolls!] \land i? = 2); \\
recColls\,!\,colls2 \longrightarrow \mathbf{skip}\ ;\ notify\,!\,2 \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
DetectorHandler3 \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls3 : int \bullet \\
\mathbf{wait}\,0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls3/pcolls!] \land i? = 3); \\
recColls\,!\,colls3 \longrightarrow \mathbf{skip}\ ;\ notify\,!\,3 \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
DetectorHandler4 \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet detect \longrightarrow \mathbf{skip}\ ;\ getWork\,?\,work \longrightarrow \mathbf{var}\ colls4 : int \bullet \\
\mathbf{wait}\,0\,..\,CPC_{TB}\ ;\ (\exists\, i? : \mathbb{Z} \bullet CalcPartCollisions[colls4/pcolls!] \land i? = 4); \\
recColls\,!\,colls4 \longrightarrow \mathbf{skip}\ ;\ notify\,!\,4 \longrightarrow \mathbf{skip}\ ;\ sync \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
OutputCollisionsHandler \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mu X \bullet output \longrightarrow getColls\,?\,collisions \longrightarrow \\
\mathbf{var}\ colls : int \bullet \mathbf{wait}\,0\,..\,CC_{TB}\ ;\ CalcCollisions; \\
(output\_collisions\,!\,colls \longrightarrow \mathbf{skip}) \blacktriangleleft OUT\_DL\ ;\ sync \longrightarrow X
\end{array}\right)
\end{array}
$$

$$
Cycle \mathrel{\widehat=} \big(\mu X \bullet \mathbf{wait}\ FRAME\_PERIOD\ ;\ sync \longrightarrow X\big)
$$

$$
\begin{array}{l}
System \mathrel{\widehat=} \\
\quad \left(\left(\left(\begin{array}{l}
InputFrameHandler \\
\quad [\![\{ currentFrame, state \} \mid \{\!| reduce, sync |\!\} \mid \{ voxel\_map, work \}]\!] \\
(ReducerHandler \\
\quad [\![\{ voxel\_map, work \} \mid \{\!| detect, sync |\!\} \mid \varnothing]\!] \\
(DetectorHandler1 \\
\quad [\![\varnothing \mid \{\!| detect, sync |\!\} \mid \varnothing]\!] \\
(DetectorHandler2 \\
\quad [\![\varnothing \mid \{\!| detect, sync |\!\} \mid \varnothing]\!] \\
(DetectorHandler3 \\
\quad [\![\varnothing \mid \{\!| detect, sync |\!\} \mid \varnothing]\!] \\
(DetectorHandler4 \\
\quad [\![\varnothing \mid \{\!| sync |\!\} \mid \varnothing]\!] \\
OutputCollisionsHandler)))))
\end{array} \right) \right) \\
\quad [\![\{ currentFrame, state, voxel\_map, work \} \mid \{\!| sync |\!\} \mid \varnothing]\!] \; Cycle \right) \setminus \\
\quad \{\!| reduce, detect, output, start, notify, sync, \\
\qquad setFrameState, getFrameState, setWork, getWork, initColls, recColls, getColls |\!\}
\end{array}
$$

$$
\begin{array}{l}
MArea \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ currentFrame : RawFrame \bullet \\
\mathbf{var}\ state : StateTable \bullet \\
\mathbf{var}\ work : Partition \bullet \\
\mathbf{var}\ collisions : int \bullet \\
\qquad \mu X \bullet \left(\begin{array}{l}
(setFrameState\,?\,v1\,?\,v2 \longrightarrow (currentFrame := v1\,;\ \ state := v2)) \mathrel{\Box} \\
(getFrameState\,!\,currentFrame\,!\,state \longrightarrow \mathbf{skip}) \mathrel{\Box} \\
(setWork\,?\,v \longrightarrow work := v) \mathrel{\Box} \\
(getWork\,!\,work \longrightarrow \mathbf{skip}) \mathrel{\Box} \\
(initColls \longrightarrow collisions := 0) \mathrel{\Box} \\
(recColls\,?\,x \longrightarrow (\mathbf{wait}\,0\,..\,RC_{TB}\,;\ \ collisions := collisions + x)) \mathrel{\Box} \\
(getColls\,!\,collisions \longrightarrow \mathbf{skip})
\end{array} \right) ;\ X
\end{array} \right) \interleave \\
\left(\begin{array}{l}
\mathbf{var}\ active : \mathbb{P}\,(1\,..\,4) \bullet \\
\qquad \mu X \bullet \left(\begin{array}{l}
(start \longrightarrow active := \{1, 2, 3, 4\}) \\
\quad \Box \\
(notify\,?\,x \longrightarrow \left(\begin{array}{l}
active := active \setminus \{x\}; \\
\mathbf{if}\ active = \varnothing \longrightarrow output \longrightarrow \mathbf{skip} \\
[\!]\ \neg\ active = \varnothing \longrightarrow \mathbf{skip} \\
\mathbf{fi}
\end{array} \right))
\end{array} \right) ;\ X
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\bullet\ Init\ ; \left(\begin{array}{l}
System \\
\quad [\![\{ currentFrame, state, voxel\_map, work \} \mid \\
\qquad \{\!| setFrameState, getFrameState, setWork, getWork, \\
\qquad\quad initColls, recColls, getColls, start, notify, sync |\!\} \mid \varnothing]\!] \\
MArea
\end{array} \right)
$$

end

Parts of the process that could not be parsed due to limitations of the *Circus* parser in CZT are highlighted.

5.4 Phase AR

In the AR phase we carry out algorithmic refinement. This also replaces (class) values by references to objects. We will not discussed this phase in as much detail as the previous phases for our example. There are, however, some refinements which are noteworthy and we shall briefly examine.

Refinement of *CalcCollisions*

The *CalcCollisions* data operation is used by *OutputCollisionsHandler* to calculate the number of collisions from the other shared variables. We refine it by simple returning the value of *collisions*. For this, we have to prove the following operation refinement to discharge.

$$\begin{array}{l}
\textit{CalcCollisions} \\
\hline
\Xi \textit{CDxSHState} \\
\textit{colls}! : \mathbb{N} \\
\hline
\exists \textit{posns} : \textit{Frame};\ \textit{motions} : \textit{Frame} \mid \mathrm{dom}\,\textit{posns} = \mathrm{dom}\,\textit{motions} \bullet \\
\left(\begin{array}{l}
\textit{posns} = F(\textit{currentFrame}) \wedge \textit{motions} = G(\textit{currentFrame}, \textit{state}) \wedge \\
\exists \textit{collset} : \mathbb{F}\,(\textit{Aircraft} \times \textit{Aircraft}) \mid \textit{collset} = \textit{CalcCollisionSet}(\textit{posns}, \textit{motions}) \bullet \\
\quad (\#\,\textit{collset} = 0 \wedge \textit{colls}! = 0) \vee (\#\,\textit{collset} > 0 \wedge \textit{colls}! \geq (\#\,\textit{collset})\ \mathrm{div}\ 2)
\end{array}\right)
\end{array}$$

$$\sqsubseteq$$

$$\textit{colls} := \textit{collisions}$$

This refinement is used to simplify the *OutputCollisionsHandler* action as follows.

$$\begin{array}{l}
\quad \textit{OutputCollisionsHandler} \\
\sqsubseteq \text{“refinement of } \textit{CalcCollisions} \text{ and copy rule”} \\
\quad \left(\begin{array}{l}
\mu X \bullet \textit{output} \longrightarrow \textit{getColls}\,?\,\textit{collisions} \longrightarrow \\
\mathbf{var}\ \textit{colls} : \textit{int} \bullet \mathbf{wait}\ 0\,..\,CC_{TB}\ ;\ \textit{colls} := \textit{collisions}; \\
(\textit{output_collisions}\,!\,\textit{colls} \longrightarrow \mathbf{skip}) \blacktriangleleft \textit{OUT_DL}\ ;\ \textit{sync} \longrightarrow X
\end{array}\right) \\
\sqsubseteq \text{“elimination of local variable } \textit{colls} \text{ using a symmetric version of the law var-intro”} \\
\quad \left(\begin{array}{l}
\mu X \bullet \textit{output} \longrightarrow \textit{getColls}\,?\,\textit{collisions} \longrightarrow \mathbf{wait}\ 0\,..\,CC_{TB}; \\
(\textit{output_collisions}\,!\,\textit{collisions} \longrightarrow \mathbf{skip}) \blacktriangleleft \textit{OUT_DL}\ ;\ \textit{sync} \longrightarrow X
\end{array}\right) \\
\sqsubseteq \text{“elimination of time budget using the law narrow-time-budget-1”} \\
\quad \left(\begin{array}{l}
\mu X \bullet \textit{output} \longrightarrow \textit{getColls}\,?\,\textit{collisions} \longrightarrow \\
(\textit{output_collisions}\,!\,\textit{collisions} \longrightarrow \mathbf{skip}) \blacktriangleleft \textit{OUT_DL}\ ;\ \textit{sync} \longrightarrow X
\end{array}\right)
\end{array}$$

We thus obtain the simplified operation below for the handler outputting the collisions.

$$\begin{array}{l}
\textit{OutputCollisionsHandler} \ \widehat{=} \\
\quad \left(\begin{array}{l}
\mu X \bullet \textit{output} \longrightarrow \textit{getColls}\,?\,\textit{collisions} \longrightarrow \\
(\textit{output_collisions}\,!\,\textit{collisions} \longrightarrow \mathbf{skip}) \blacktriangleleft \textit{OUT_DL}\ ;\ \textit{sync} \longrightarrow X
\end{array}\right)
\end{array}$$

Its behaviour is now simply to read the value of the shared *collisions* variable and output it on the channel *output_collisions*.

Refinement of *MArea*

A second algorithmic refinement worth mentioning a (data) refinement of part of the *MArea* action.

$$
\begin{array}{l}
MArea \mathrel{\widehat=} \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ currentFrame : RawFrame \bullet \\
\mathbf{var}\ state : StateTable \bullet \\
\mathbf{var}\ work : Partition \bullet \\
\mathbf{var}\ collisions : int \bullet \\
\quad \mu X \bullet \left(\begin{array}{l}
(setFrameState ? v1 ? v2 \longrightarrow (currentFrame := v1 \mathbin{;}\ state := v2)) \Box \\
(getFrameState ! currentFrame ! state \longrightarrow \mathbf{skip}) \Box \\
(setWork ? v \longrightarrow work := v) \Box \\
(getWork ! work \longrightarrow \mathbf{skip}) \Box \\
(initColls \longrightarrow collisions := 0) \Box \\
(recColls ? x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB} \mathbin{;}\ collisions := collisions + x)) \Box \\
(getColls ! collisions \longrightarrow \mathbf{skip})
\end{array}\right) \mathbin{;}\ X
\end{array}\right) \interleave \\
\left(\begin{array}{l}
\mathbf{var}\ active : \mathbb{P}(1 \mathinner{..} 4) \bullet \\
\quad \mu X \bullet \left(\begin{array}{l}
(start \longrightarrow active := \{1,2,3,4\}) \\
\quad \Box \\
(notify ? x \longrightarrow \left(\begin{array}{l}
active := active \setminus \{x\}; \\
\mathbf{if}\ active = \varnothing \longrightarrow output \longrightarrow \mathbf{skip} \\
[\!]\ \neg\ active = \varnothing \longrightarrow \mathbf{skip} \\
\mathbf{fi}
\end{array}\right))
\end{array}\right) \mathbin{;}\ X
\end{array}\right)
\end{array}\right)
\end{array}
$$

The highlighted action above uses and abstract variable *active* to retain information about handlers that are still active in caclulating their collisions result. In Appendix A.6, we include a model for the `DetectorControl` class used to record this data in the program. With it, we refine *MArea* as follows.

$$
\begin{array}{l}
MArea \mathrel{\widehat=} \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ currentFrame : RawFrame \bullet \\
\mathbf{var}\ state : StateTable \bullet \\
\mathbf{var}\ work : Partition \bullet \\
\mathbf{var}\ collisions : int \bullet \\
\quad \mu X \bullet \left(\begin{array}{l}
(setFrameState ? v1 ? v2 \longrightarrow (currentFrame := v1 \mathbin{;}\ state := v2)) \Box \\
(getFrameState ! currentFrame ! state \longrightarrow \mathbf{skip}) \Box \\
(setWork ? v \longrightarrow work := v) \Box \\
(getWork ! work \longrightarrow \mathbf{skip}) \Box \\
(initColls \longrightarrow collisions := 0) \Box \\
(recColls ? x \longrightarrow (\mathbf{wait}\ 0 \mathinner{..} RC_{TB} \mathbin{;}\ collisions := collisions + x)) \Box \\
(getColls ! collisions \longrightarrow \mathbf{skip})
\end{array}\right) \mathbin{;}\ X
\end{array}\right) \interleave \\
\left(\begin{array}{l}
\mathbf{var}\ control : DetectorControl \bullet control := \mathbf{newM}\ DetectorControl; \\
\quad \mu X \bullet \left(\begin{array}{l}
(start \longrightarrow control . start()) \\
\quad \Box \\
\left(\begin{array}{l}
notify ? i \longrightarrow \\
\quad \left(\begin{array}{l}
control . notify(i); \\
\mathbf{if}\ control . done() = jtrue \longrightarrow output \longrightarrow \mathbf{skip} \\
[\!]\ \neg\ control . done() = jtrue \longrightarrow \mathbf{skip} \\
\mathbf{fi}
\end{array}\right)
\end{array}\right)
\end{array}\right) \mathbin{;}\ X
\end{array}\right)
\end{array}\right)
\end{array}
$$

The *control* object of type *DetectorControl* retains the number of active handlers by way of a boolean array. Synchronisations on *start* and *notify* result in calling the corresponding methods on the data object. The right-hand branch of the parallelism in fact models an *active* object now: that is a data object that interacts through synchronisations with other processes. The actual model for the class is thus a mixture of the Oh*Circus* class specification of the data object as well as the active part in the $MArea$ action.

6 Anchor S

In this section, we sketch the S Anchor model for our case study. Since the CD_x is more complex than our previous example in [6], we shall not attempt to specify the model in full detail here. Its exact shape moreover depends on the precise definition of SCJ*Circus*, which is still work in progress. Each type of SCJ*Circus* paragraph is discussed in a separate subsection in the remainder of this section.

6.1 CD_x Safelet

In the original CD_x program, the `setUp()` method is defined as follows.

```
public void setup() {
    Constants.PRESIMULATE = true;
    new ImmortalEntry().run();
    new Simulator().generate();
}
```

The first and third statements configure and instantiate a simulator and thus can be ignored. The `run()` method of the `ImmortalEntry` class, called in the second line, merely initialises a static field `frameBuffer` of this class. This is also part of the simulation, hence the actual content of `setup()` is void terms of our model. The implementation of `tearDown()` in the original program calls a static method `dumpResults()` but this merely reports results of the benchmark and thus is not relevant for the model either.

Because of the above the safelet SCJ*Circus* paragraph takes the same trivial shape as in [6].

$$
\begin{array}{l}
\mathbf{safelet}\ \mathit{CDxSafelet} \mathrel{\widehat=} \mathbf{begin} \\
\mathbf{setUp} \mathrel{\widehat=} \mathbf{skip} \\
\mathbf{tearDown} \mathrel{\widehat=} \mathbf{skip} \\
\mathbf{end}
\end{array}
$$

6.2 Mission Sequencer

We first have to introduce a mission identifier for the single mission of the parallel CD_x.

$$
\mathit{CDxMissionId} : \mathit{MissionId}
$$

The model of the mission sequencer is likewise identical to the one for the serial line.

$$
\begin{array}{l}
\mathbf{sequencer}\ \mathit{CDxMissionSequencer} \mathrel{\widehat=} \mathbf{begin} \\
\mathbf{state}\ \mathit{CDxMissionSequencerState} == [\, \mathit{mission\_done} : \mathit{boolean} \,] \\
\mathbf{initial} \mathrel{\widehat=} \mathit{mission\_done} := \mathit{jfalse} \\
\mathbf{getNextMission} \mathrel{\widehat=} \mathbf{res\ ret} : \mathit{MissionId} \bullet \\
\quad \left(\begin{array}{l}
\mathbf{if}\ \mathit{mission\_done} = \mathit{jfalse} \longrightarrow \\
\quad \left(\begin{array}{l} \mathit{mission\_done} := \mathit{jtrue}; \\ \mathbf{ret} := \mathit{CDxMissionId} \end{array} \right) \\
[\!]\ \neg\ \mathit{mission\_done} = \mathit{jfalse} \longrightarrow \mathbf{ret} := \mathit{nullMId} \\
\mathbf{fi}
\end{array} \right) \\
\mathbf{end}
\end{array}
$$

The only difference in comparison to [6] is the use of the name *CDxMissionId* rather than *ProtocolMission*. Strictly, we could even make our job easier here since `getNextMission()` is only called once in our program, as the mission does not terminate. Because of this we have no obligation to return **null** with the second call.

6.3 CD_x Mission

The SCJ*Circus* paragraph for the `CDxMission` class is more interesting as it takes care of the construction the various handlers and software events as well as encapsulates shared data via its *MArea* action.

mission *CDxMission* $\widehat{=}$ **begin**

$$
\begin{array}{l}
\textbf{state}\ \mathit{CDxMissionState} \\
\hline
\mathit{currentFrame} : \textbf{ref}\ \mathit{RawFrame} \\
\mathit{state} : \textbf{ref}\ \mathit{StateTable} \\
\mathit{work} : \textbf{ref}\ \mathit{Partition} \\
\mathit{collisions} : \mathit{int} \\
\mathit{control} : \textbf{ref}\ \mathit{DetectorControl} \\
\hline
\end{array}
$$

$$
\mathit{Init}\ \widehat{=}\
\left(\begin{array}{l}
\mathit{currentFrame} := \textbf{newM}\ \mathit{RawFrame}; \\
\mathit{state} := \textbf{newM}\ \mathit{StateTable}; \\
\mathit{work} := \textbf{newM}\ \mathit{Partition}(4); \\
\mathit{collisions} := 0
\end{array}\right)
$$

$$
\textbf{initialize}\ \widehat{=}\
\left(\begin{array}{l}
\textbf{var}\ \mathit{reduce} : \mathit{AperiodicEvent} \bullet \mathit{reduce} := \textbf{newEvent}\ \mathit{AperiodicEvent}(); \\
\textbf{var}\ \mathit{detect} : \mathit{AperiodicEvent} \bullet \mathit{detect} := \textbf{newEvent}\ \mathit{AperiodicEvent}(); \\
\textbf{var}\ \mathit{output} : \mathit{AperiodicEvent} \bullet \mathit{output} := \textbf{newEvent}\ \mathit{AperiodicEvent}(); \\
\mathit{control} := \textbf{newM}\ \mathit{DetectorControl}(\mathit{output}, 4)\ ;\ \ \mathit{DetectorControlInit}\,!\,\mathit{control} \longrightarrow \textbf{skip} \\
\textbf{var}\ h_1 : \mathit{InputFrameHandler} \bullet h_1 := \textbf{newHandler}\ \mathit{InputFrameHandler}(\textbf{self}, \mathit{reduce}); \\
\textbf{var}\ h_2 : \mathit{ReducerHandler} \bullet h_2 := \textbf{newHandler}(\mathit{reduce})\ \mathit{ReducerHandler}(\textbf{self}, \mathit{detect}, \mathit{control}); \\
\textbf{var}\ h_3 : \mathit{DetectorHandler} \bullet h_3 := \textbf{newHandler}(\mathit{detect})\ \mathit{DetectorHandler}(\textbf{self}, \mathit{control}, 1)); \\
\textbf{var}\ h_4 : \mathit{DetectorHandler} \bullet h_4 := \textbf{newHandler}(\mathit{detect})\ \mathit{DetectorHandler}(\textbf{self}, \mathit{control}, 2)); \\
\textbf{var}\ h_5 : \mathit{DetectorHandler} \bullet h_5 := \textbf{newHandler}(\mathit{detect})\ \mathit{DetectorHandler}(\textbf{self}, \mathit{control}, 3)); \\
\textbf{var}\ h_6 : \mathit{DetectorHandler} \bullet h_6 := \textbf{newHandler}(\mathit{detect})\ \mathit{DetectorHandler}(\textbf{self}, \mathit{control}, 4)); \\
\textbf{var}\ h_7 : \mathit{OutputCollisionsHandler} \bullet h_7 := \textbf{newHandler}(\mathit{output})\ \mathit{OutputCollisionsHandler}(\textbf{self}); \\
\textbf{register}\ h_1\ ;\ \textbf{register}\ h_2\ ;\ \textbf{register}\ h_3\ ;\ \textbf{register}\ h_4\ ;\ \textbf{register}\ h_5\ ;\ \textbf{register}\ h_6\ ;\ \textbf{register}\ h_7
\end{array}\right)
$$

cleanup $\widehat{=}$ **skip**

$$
\mathit{MArea}\ \widehat{=}\
\left(\left(\begin{array}{l}
\textbf{var}\ \mathit{currentFrame} : \textbf{ref}\ \mathit{RawFrame} \bullet \\
\textbf{var}\ \mathit{state} : \textbf{ref}\ \mathit{StateTable} \bullet \\
\textbf{var}\ \mathit{work} : \textbf{ref}\ \mathit{Partition} \bullet \\
\textbf{var}\ \mathit{collisions} : \mathit{int} \bullet \\
\mu X \bullet \left(\begin{array}{l}
(\mathit{setFrame}\,?\,\mathit{value} \longrightarrow \mathit{currentFrame} := \mathit{value}) \Box \\
(\mathit{getFrame}\,!\,\mathit{currentFrame} \longrightarrow \textbf{skip}) \Box \\
(\mathit{setState}\,?\,\mathit{value} \longrightarrow \mathit{state} := \mathit{value}) \Box \\
(\mathit{getState}\,!\,\mathit{state} \longrightarrow \textbf{skip}) \Box \\
(\mathit{setWork}\,?\,\mathit{value} \longrightarrow \mathit{work} := \mathit{value}) \Box \\
(\mathit{getWork}\,!\,\mathit{work} \longrightarrow \textbf{skip}) \Box \\
(\mathit{initCollsC} \longrightarrow \mathit{collisions} := 0\ ;\ \ \mathit{initCollsR} \longrightarrow \textbf{skip}) \Box \\
(\mathit{recCollsC}\,?\,x \longrightarrow \left(\begin{array}{l}
\textbf{wait}\ 0\,..\,RC_{TB}\ ;\ \ \mathit{collisions} := \mathit{collisions} + x); \\
\mathit{recCollsR} \longrightarrow \textbf{skip}
\end{array}\right) \Box \\
(\mathit{getColls}\,!\,\mathit{collisions} \longrightarrow \textbf{skip})
\end{array}\right)\ ;\ X
\end{array}\right)\right)
$$

end

Due to limitations of the tools for *Circus* it is at present not possible to parse the **initialize** action, indicated by the highlight above. The model corresponds directly to the SCJ code of this class recaptured below.

```
public class CDxMission extends Mission {
  /* Shared objects in mission memory. */

  public RawFrame currentFrame;
  public StateTable state;
  public Partition work;
  public int collisions;
  public DetectorControl control;

  /* Constructor of the class. */

  public CDxMission() {
    currentFrame = new RawFrame();
    state = new StateTable();
    work = new Partition(4);
    collisions = 0;
  }

  /* Initialisation method call by the SCJ infrastructure. */

  public void initialize() {
    AperiodicEvent reduce = new AperiodicEvent();
    AperiodicEvent detect = new AperiodicEvent();
    AperiodicEvent output = new AperiodicEvent();
    control = new DetectorControl(output, 4);
    InputFrameHandler h1 = new InputFrameHandler(this, reduce);
    ReducerHandler h2 = new ReducerHandler(this, detect, control, reduce);
    DetectorHandler h3 = new DetectorHandler(this, control, 1, detect);
    DetectorHandler h4 = new DetectorHandler(this, control, 2, detect);
    DetectorHandler h5 = new DetectorHandler(this, control, 3, detect);
    DetectorHandler h6 = new DetectorHandler(this, control, 4, detect);
    OutputCollisionsHandler h7 = new OutputCollisionsHandler(this, output);
    h1.register()
    h2.register();
    h3.register();
    h4.register();
    h5.register();
    h6.register();
    h7.register();
  }

  /* Clean-up method call by the SCJ infrastructure. */

  public void cleanup() { }

  /* Specifies the memory requirements of the mission (not modelled). */

  public long missionMemorySize() {
    return Constants.MISSION_MEMORY_SIZE;
  }
```

```
  /* Methods to access shared data, modelled by the MArea action. */

  public RawFrame getFrame() {
    return currentFrame;
  }

  public void setFrame(RawFrame frame) {
    currentFrame = frame;
  }

  public StateTable getState() {
    return state;
  }

  public void setState(StateTable state) {
    this.state = state;
  }

  public Partition getWork() {
    return work;
  }

  public void setWork(Partition work) {
    this.work = work;
  }

  public synchronized void initColls() {
    collisions = 0;
  }

  public synchronized void recColls(int n) {
    collisions += n;
  }

  public synchronized int getColls() {
    return collisions;
  }
}
```

The only deviation is the additional method `missionMemorySize()` which we do not model as we are not concerned with resource issues. The `synchronized` identifiers are implicit in the specification of *MArea*. We note that the above code is from the 'clean' version of the program which, unlike the runnable version, exclude any simulation code and is compliant with Version 0.78 of the SCJ Technology Specification.

6.4 CD_x Handlers

The SCJ program of the parallel CD_x consists of seven handlers.

- 1 x `InputFrameHandler` (periodic, running at maximal priority)
- 1 x `ReducerHandler` (aperiodic, running at normal priority)
- 4 x `DetectorHandler` (aperiodic, running at normal priority)
- 1 x `OutputCollisionsHandler` (aperiodic, running at maximal priority)

In terms of control, `InputFrameHandler` is released periodically by a timer and releases `ReducerHandler` by virtue of a software event. `ReducerHandler` releases all `DetectorHandler` instances, and the last active `DetectorHandler` releases `OutputCollisionsHandler` indirectly by calling `notify(int)` when finishing its work. We now discuss the SCJ*Circus* models for the four types of handlers in more detail.

6.4.1 InputFrameHandler

This is the only periodic handler of the application. It reads the next frame and deposits it in the global variable *currentFrame*. Before doing so it copies the content of the current frame into the *state* data structure. In fundamental terms *InputFrameHandler* is similar to *Handler*1 of the serial line example in [5].

periodic$(\mathit{FRAME\_PERIOD})$ **handler** *InputFrameHandler* $\widehat{=}$ **begin**

state *InputFrameHandlerState*
$\mathit{mission} : \mathit{MissionId}$
$\mathit{reduce} : \mathit{AperiodicEvent}$

initial $\mathit{InputFrameHandlerInit}(m : \mathit{MissionId}, \mathit{evt} : \mathit{AperiodicEvent}) \mathrel{\widehat{=}}$
$\mathit{mission} := m \,;\; \mathit{reduce} := \mathit{evt}$

handleAsyncEvent $\widehat{=}$

$$\left(\begin{array}{l}(\mathit{next\_frame}\,?\,\mathit{frame} \longrightarrow (\mathbf{wait}\;0\,..\,ST_{TB}\;;\;\mathit{StoreFrame}(\mathit{frame}))) \blacktriangleleft \mathit{INP\_DL};\\ \mathbf{fire}\;\mathit{reduce}\end{array}\right)$$

$\mathit{StoreFrame}(\mathit{frame} : \mathit{RawFrame}) \mathrel{\widehat{=}}$

$$\left(\begin{array}{l}\mathbf{var}\;\mathit{currentFrame} : \mathbf{ref}\;\mathit{RawFrame} \bullet\\ \mathit{getFrame}\,?\,f \longrightarrow \mathit{currentFrame} := f;\\ \mathbf{var}\;\mathit{state} : \mathbf{ref}\;\mathit{StateTable} \bullet\\ \mathit{getState}\,?\,s \longrightarrow \mathit{state} := s;\\ \ldots\\ \texttt{\{* Update } \mathit{currentFrame} \texttt{ and } \mathit{state} \texttt{ according to } \mathit{frame}\texttt{. *\}}\end{array}\right)$$

dispatch handleAsyncEvent

end

Since the handler is release periodically by a timer, the dispatch actions takes a simple form of just calling *handleAsyncEvent*. The behaviour of the handler action is to wait for communication on *next_frame* and then invoke *StoreFrame* while passing the frame object read from the hardware. The communication must occur within *INP_DL* time units from the start of each cycle. The **fire** construct is an extension of SCJ*Circus* used to fire a software event. It corresponds to a respective call to the `fire()` method of `AperiodicEvent`. The computation carried out by *StoreFrame* is mostly omitted; it emerges during the AR phase.

6.4.2 ReducerHandler

We now sketch the S model for *ReducerHandler*.

aperiodic handler *ReducerHandler* $\widehat{=}$ **begin**

state *ReducerHandlerState*

$mission : MissionId$
$detect : AperiodicEvent$
$control : \mathbf{ref}\ DetectorControl$

initial $ReducerHandlerInit(m : MisisonId, evt : AperiodicEvent, c : \mathbf{ref}\ DetectorControl) \mathrel{\widehat{=}}$
$\quad mission := m\ ;\ detect := evt\ ;\ control := c$

handleAsyncEvent $\widehat{=}$

$$\left(\begin{array}{l}
\mathbf{var}\ currentFrame : \mathbf{ref}\ RawFrame \bullet getFrame\,?\,f \longrightarrow currentFrame := f;\\
\mathbf{var}\ state : \mathbf{ref}\ StateTable \bullet getState\,?\,s \longrightarrow state := s;\\
\mathbf{var}\ work : \mathbf{ref}\ Partition \bullet getWork\,?\,w \longrightarrow work := w;\\
\mathbf{var}\ voxel_map : HashMap[Vector2d, List[Motion]] \bullet\\
\mathbf{wait}\ 0\,..\,RPW_{TB};\\
voxel_map := \mathbf{newP}\ HashMap();\\
\{\texttt{* Execute algorithm for voxel hashing and populate } voxel_map\texttt{. *}\}\\
voxel_map\,.\,put(\ldots);\\
work\,.\,clear();\\
\left(\begin{array}{l}
\mathbf{for}\ i = 0\ \mathbf{to}\ voxel_map\,.\,values()\,.size() - 1 \bullet\\
work\,.\,recordVoxelMotions(voxel_map\,.\,values()\,.\,get(i))
\end{array}\right);\\
initCollsC \longrightarrow \mathbf{skip}\ ;\ initCollsR \longrightarrow \mathbf{skip};\\
startC \longrightarrow \mathbf{skip}\ ;\ startR \longrightarrow \mathbf{skip};\\
\mathbf{fire}\ detect
\end{array}\right)$$

dispatch $release\_handler\,.\,ReducerHandlerId \longrightarrow$ **handleAsyncEvent**

end

The difference to, for instance, *Handler*2 of the serial line example in [6] is that this handler is released by a software event rather than an external event. The synchronisation on $release\_handler\,.\,ReducerHandlerId$ highlights this. In the P model of software events, the *release_handler* channel is used to cause the periodic release of a handler. The channel is parametrised by the id of the handler to be released. Details of the voxel hashing algorithm are again omitted; they are a concern for AR.

6.4.3 DetectorHandler

The aperiodic detection handler is specified below. Since we have four instances of this handler in the program, the process is parametrised by an identifier of type *int*.

$\textbf{aperiodic handler } \mathit{DetectorHandler} \mathrel{\widehat=} \mathit{hdl} : \mathit{HandlerId} \bullet \textbf{begin}$

$$\begin{array}{|l}
\textbf{state } \mathit{DetectorHandlerState} \\
\hline
\mathit{mission} : \mathit{MissionId} \\
\mathit{control} : \textbf{ref } \mathit{DetectorControl} \\
\mathit{id} : \mathit{int} \\
\hline
\end{array}$$

$$\begin{array}{l}
\textbf{initial } \mathit{DetectorHandlerInit}(m : \mathit{MissionId}, c : \textbf{ref } \mathit{DetectorControl}, n : \mathit{int}) \mathrel{\widehat=} \\
\quad \mathit{mission} := m \;;\; \mathit{control} := c \;;\; \mathit{id} := n
\end{array}$$

$$\begin{array}{l}
\mathit{CalcPartCollisions} \mathrel{\widehat=} \textbf{res } \mathit{pcolls} : \mathit{int} \bullet \\
\quad \left(\begin{array}{l}
\mathit{pcolls} := 0; \\
\textbf{var } \mathit{work} : \mathit{Partition} \bullet \mathit{getWork}\,?\,w \longrightarrow \mathit{work} := w; \\
\textbf{for } i = 0 \textbf{ to } \mathit{work} \,.\, \mathit{getDetectorWork}(\mathit{id}) \,.\, \mathit{size}() - 1 \bullet \\
\quad \left(\begin{array}{l}
\textbf{var } \mathit{motions} : \mathit{List}[\mathit{Motion}] \bullet \\
\mathit{motions} := \mathit{work} \,.\, \mathit{getDetectorWork}(i) \,.\, \mathit{get}(i); \\
\mathit{pcolls} := \mathit{pcolls} + \textbf{self} \,.\, \mathit{determineCollisions}(\mathit{motions});
\end{array}\right)
\end{array}\right)
\end{array}$$

$$\begin{array}{l}
\mathit{determineCollisions} \mathrel{\widehat=} \textbf{val } \mathit{motions} : \mathit{List}[\mathit{Motion}];\ \textbf{res ret} : \mathit{int} \bullet \\
\quad \{\texttt{* Algorithm for counting collisions. *}\} \\
\quad \textbf{ret} := \ldots
\end{array}$$

$$\begin{array}{l}
\textbf{handleAsyncEvent} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\textbf{var } \mathit{colls} : \mathit{int} \bullet \\
\quad \left(\begin{array}{l}
\textbf{wait } 0 \,..\, \mathit{CPC}_{TB} \;;\; \mathit{CalcPartCollisions}(\mathit{colls}); \\
\mathit{recCollsC}\,!\,\mathit{colls} \longrightarrow \mathit{recCollsR} \longrightarrow \textbf{skip}; \\
\mathit{notifyC}\,!\,\mathit{id} \longrightarrow \mathit{notifyR} \longrightarrow \textbf{skip}
\end{array}\right)
\end{array}\right)
\end{array}$$

$\textbf{dispatch } \mathit{release} \,.\, \mathit{DetectorHandlerId} \longrightarrow \textbf{handleAsyncEvent}()$

end

Details have been omitted concerning the algorithm that counts collisions in a voxel motion list (this is done inside the method *determineCollisions*). We have four instances of this process in the S anchor:

$$\mathit{DetectorHandler}(1) \parallel \mathit{DetectorHandler}(2) \parallel \mathit{DetectorHandler}(3) \parallel \mathit{DetectorHandler}(4)$$

6.4.4 OutputCollisionsHandler

This is a simple aperiodic handler that outputs the collisions.

$\textbf{aperiodic handler } \mathit{OutputCollisionsHandler} \mathrel{\widehat=} \textbf{begin}$

$$\begin{array}{|l}
\textbf{state } \mathit{OutputCollisionsHandlerState} \\
\hline
\mathit{mission} : \mathit{MissionId} \\
\hline
\end{array}$$

$\textbf{initial } \mathit{OutputCollisionsHandlerInit}(m : \mathit{MissionId}) \mathrel{\widehat=} \mathit{mission} := m$

$$\begin{array}{l}
\textbf{handleAsyncEvent} \mathrel{\widehat=} \textbf{var } \mathit{colls} : \mathit{int} \bullet \\
\quad \left(\begin{array}{l}
\mathit{getColls}\,?\,c \longrightarrow \mathit{colls} := c; \\
(\mathit{output_collisions}\,!\,\mathit{colls} \longrightarrow \textbf{skip} \blacktriangleleft \mathit{OUT_DL})
\end{array}\right)
\end{array}$$

$\textbf{dispatch } \mathit{release\_handler} \,.\, \mathit{OutputCollisionsHandlerId} \longrightarrow \textbf{handleAsyncEvent}$

end

The handler method first obtains the detected collisions using the *getColls* method provided by the mission class to access the shared *collisions* variable. It then outputs the collisions on the *output_collisions* channel, imposing a deadline on the communication to ensure that the hardware accepts the output within the required time interval.

6.4.5 Active Objects

We note that the Oh*Circus* class model of *DetectorControl* does not contain a **fire** statement. This, however, is needed to give a faithful model of this class. Below we capture the active behaviour of the *DetectorControl* instance used by the program by way of a an SCJ*Circus* paragraph **active**. It is also part of the S anchor.

$$\mathbf{active}\ DetectorControl \mathrel{\widehat=} \mathbf{begin}$$

$$\begin{array}{|l}\hline \mathbf{state}\ DetectorHandlerState \\ control : \mathbf{ref}\ DetectorControl \\ \hline\end{array}$$

$$Init \mathrel{\widehat=} DetectorControlInit\,?\,c \longrightarrow control := c$$

$$MArea \mathrel{\widehat=}$$
$$\left(\mu X \bullet \left(\begin{array}{l}(startC \longrightarrow control\,.\,start()\ ;\ \ startR \longrightarrow \mathbf{skip}) \\ \quad\Box \\ \left(\begin{array}{l} notifyC\,?\,i \longrightarrow \\ \quad\left(\begin{array}{l} control\,.\,notify(i); \\ \left(\begin{array}{l}(control\,.\,done() \mathrel{\&} \mathbf{fire}\ control\,.\,event \longrightarrow \mathbf{skip}) \\ \quad\Box \\ (\neg\ control\,.\,done() \mathrel{\&} \mathbf{skip})\end{array}\right); \\ notifyR \longrightarrow \mathbf{skip}\end{array}\right)\end{array}\right)\end{array}\right)\ ;\ X\right)$$

end

The state includes a reference to the detector control class object whose active behaviour is wrapped by the process. The *Init* action connects the process to this object via an input prefix on the *DetectorControlInit* channel. The communication is raised inside the *CDxMission* paragraph when the *control* object is created.

A Class Definitions

In this section we present the specification of Oh*Circus* classes of the program relevant to the models.

A.1 *RawFrame* class

class *RawFrame* $\widehat{=}$ **begin**

statics *RawFrameStatics*

$\mathbf{private}\ MAX\_PLANES : int;$
$\mathbf{private}\ MAX\_SIGNS : int$

sinit *RawFrameSInit*

$RawFrameStatics\,'$

$MAX\_PLANES' = 1000$
$MAX\_SIGNS' = 10 * MAX\_PLANES'$

state *RawFrameState*

$\mathbf{public}\ lengths : intArray$
$\mathbf{public}\ callsigns : byteArray$
$\mathbf{public}\ positions : floatArray$
$\mathbf{public}\ planeCnt : int$

$lengths \neq \mathbf{null} \land callsigns \neq \mathbf{null} \land positions \neq \mathbf{null}$
$lengths\,.\,length() = MAX\_PLANES$
$callsigns\,.\,length() = MAX\_SIGNS$
$positions\,.\,length() = 3 * MAX\_PLANES$
$0 \leq planeCnt \leq MAX\_PLANES$

initial *RawFrameInit*

$RawFrameState\,'$

$lengths' = \mathbf{newM}\ intArray(MAX\_PLANES)$
$callsigns' = \mathbf{newM}\ byteArray(MAX\_SIGNS)$
$positions' = \mathbf{newM}\ floatArray(3 * MAX\_PLANES)$
$planeCnt' = 0$

logical function *getCallSignOffset*

$\Xi RawFrameState$
$plane? : int$
$result! : int$

$0 \leq plane? < planeCnt$
$result! = \Sigma\{i : 0\,..\,plane? - 1 \bullet i \mapsto lengths\,.\,getA(i)\}$

logical function *getCallSign*

$\Xi RawFrameState$
$plane? : int$
$result! : \operatorname{seq} byte$

$0 \leq plane? < planeCnt$
$\# result! = lengths\,.\,getA(plane?)$
$\forall i : 1\,..\,lengths\,.\,getA(plane?) \bullet$
$\quad result!(i) = callsigns\,.\,getA(\mathbf{self}\,.\,getCallSignOffset(plane?) + i - 1)$

logical function *find*

$$
\begin{array}{l}
\Xi RawFrameState \\
a? : Aircraft \\
result! : \mathbb{Z} \\
\hline
result! = \left(\begin{array}{l}
\mathbf{if}\ (\exists_1\, i : 0 \mathrel{..} (planeCnt - 1) \bullet \mathbf{self} \,.\, getCallSign(i) = a?) \\
\mathbf{then}\ (\mu\, i : 0 \mathrel{..} (planeCnt - 1) \mid \mathbf{self} \,.\, getCallSign(i) = a?) \\
\mathbf{else} - 1
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathbf{public}\ copy(lengths : intArray, signs : byteArray, positions : floatArray) \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathbf{var}\ pos1, pos2, pos3, pos4 : int \bullet \\
pos1 := 0 \,;\ pos2 := 0 \,;\ pos3 := 0 \,;\ pos4 := 0; \\
\left(\begin{array}{l}
\mathbf{for}\ i = 0\ \mathbf{to}\ lengths \,.\, length() - 1 \bullet \\
\quad \mathbf{self} \,.\, lengths \,.\, setA(pos1, lengths \,.\, getA(i)); \\
\quad pos1 := pos1 + 1; \\
\quad \mathbf{self} \,.\, positions \,.\, setA(pos2, positions \,.\, getA(3 * i)); \\
\quad pos2 := pos2 + 1; \\
\quad \mathbf{self} \,.\, positions \,.\, setA(pos2, positions \,.\, getA(3 * i + 1)); \\
\quad pos2 := pos2 + 1; \\
\quad \mathbf{self} \,.\, positions \,.\, setA(pos2, positions \,.\, getA(3 * i + 2)); \\
\quad pos2 := pos2 + 1; \\
\quad \left(\begin{array}{l}
\mathbf{for}\ j = 0\ \mathbf{to}\ lengths \,.\, getA(i) - 1 \bullet \\
\quad \mathbf{self} \,.\, callsigns \,.\, setA(pos3, signs \,.\, getA(pos4 + j); \\
\quad pos3 := pos3 + 1
\end{array}\right); \\
\quad pos4 := pos4 + lengths.getA(i)
\end{array}\right); \\
planeCnt := lengths.length()
\end{array}\right)
\end{array}
$$

end

This class has only one non-logical method which is used to initialise the instance variables from a given set of arrays. It also introduces two static variables which, however, are merely used as constants. The remainder of the SCJ program accesses the fields of the class directly to obtain the position data of the aircrafts (all instance variables of the class are public).

A.2 *StateTable* class

Abstract version

The abstract *StateTable* class is not concerned with memory allocation issues.

class *StateTableA* $\widehat{=}$ **begin**

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{StateTableState}} \\ \hline
\textbf{private}\ \textit{posnMap} : \textit{HashMap}[\textit{CallSign}, \textit{Vector3d}] \\ \hline
\textit{posnMap} \neq \textbf{null} \\ \hline
\end{array}$$

initial *Init* $\widehat{=}$ $\textit{posnMap} := \textbf{newM}\ \textit{HashMap}$

The abstract *StateTable* class is not concerned with memory allocation issues.

public $\textit{put}(\textit{callsign} : \textit{CallSign}, x : \textit{float}, y : \textit{float}, z : \textit{float}) \ \widehat{=}$
$\quad \textit{posnMap} \,.\, \textit{put}(\textit{callsign}, \textbf{new}\ \textit{Vector3d}(x, y, z))$

public $\textit{get}(\textit{callsign} : \textit{CallSign}) \ \widehat{=}\ \textbf{ret} := \textit{posnMap} \,.\, \textit{get}(\textit{callsign})$

end

The *posnMap* member variable of type *HashMap* is used to store *Vector3d* objects under keys being *CallSign* objects. The model for the *HashMap* class is included in Appendix **??**.

Concrete version

The concrete *StateTable* class in comparison considers memory areas.

class *StateTableC* $\widehat{=}$ **begin**

$$\begin{array}{|l}
\multicolumn{1}{l}{\textbf{statics}\ \textit{StateTableStatics}} \\ \hline
\textit{MAX_AIRPLANES} : \textit{int} \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textbf{sinit}\ \textit{StateTableSInit}} \\ \hline
\textit{StateTableStatics}' \\ \hline
\textit{MAX_AIRPLANES}' = 10000 \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{StateTableState}} \\ \hline
\textbf{private}\ \textit{posnMap} : \textit{HashMap}[\textit{CallSign}, \textit{Vector3d}] \\
\textbf{private}\ \textit{allocatedVectors} : \textit{Vector3dArray} \\
\textbf{private}\ \textit{usedVectors} : \textit{int} \\
\textbf{private}\ r : \textit{StateTable_R} \\ \hline
\textit{posnMap} \neq \textbf{null} \land \textit{allocatedVectors} \neq \textbf{null} \\
0 \leq \textit{usedVectors} \leq \textit{allocatedVectors} \,.\, \textit{length}() \\ \hline
\end{array}$$

initial *Init* $\widehat{=}$

$$\left(\begin{array}{l}
r := \textbf{newM}\ \textit{StateTable_R}(\textbf{self}); \\
\textit{allocatedVectors} := \textbf{newM}\ \textit{Vector3dArray}(\textit{MAX_AIRPLANES}); \\
\left(\begin{array}{l}\textbf{for}\ \textit{index} = 0\ \textbf{to}\ \textit{allocatedVectors} \,.\, \textit{length}() - 1 \bullet \\ \quad \textit{allocatedVectors} \,.\, \textit{setA}(\textit{index}, \textbf{newM}\ \textit{Vector3d}()))\end{array}\right); \\
\textit{usedVectors} := 0
\end{array}\right)$$

public $put(callsign : CallSign, x : float, y : float, z : float) \mathrel{\widehat=}$

$$\left(\begin{array}{l} r \,.\, callsign := callsign; \\ r \,.\, x := x; \\ r \,.\, y := y; \\ r \,.\, z := z; \\ MemoryArea \,.\, getMemoryArea(\mathbf{self}) \,.\, executeInArea(r) \end{array}\right)$$

public $get(callsign : CallSign) \mathrel{\widehat=} \mathbf{ret} := posnMap \,.\, get(callsign)$

end

This class introduces mechanisms to solve memory allocation issues ensued by the dynamic allocation of data in mission memory. This is, in particular, the allocation of *Vector3d* objects. It utilises an inner class *StateTable_R* to execute code in mission memory that updates the *HashMap*.

Inner class *R* of *StateTable*

The inner class below is used to ensure that put operations carried out on the *HashMap* are executed in mission memory. This should, in principle, not be necessary, however, I suspect that adding elements to the *HashMap* causes dynamic allocation of data, too, in the original program of the CD_x. This is fundamentally an issue with the memory behaviour of libraries and subject to future research.

class $StateTable\_R \mathrel{\widehat=}$ **begin**

StateTable_RState

private $outer : StateTable$
public $callsign : CallSign$
public $x, y, z : float$
$outer \neq \mathbf{null}$

initial $Init(o : StateTable) \mathrel{\widehat=} outer := o$

public $run \mathrel{\widehat=}$

$$\left(\begin{array}{l} \mathbf{var}\ v : Vector3d \bullet v := outer \,.\, posnMap \,.\, get(callsign); \\ \mathbf{if}\ v = \mathbf{null} \longrightarrow \\ \quad \left(\begin{array}{l} v := outer \,.\, allocatedVectors \,.\, get(usedVectors); \\ usedVectors := usedVectors + 1; \\ outer \,.\, posnMap \,.\, put(callsign, v); \end{array}\right) \\ [\!]\ \neg\ v = \mathbf{null} \longrightarrow \mathbf{skip} \\ \mathbf{fi}\,; \\ v \,.\, x := x; \\ v \,.\, y := y; \\ v \,.\, z := z \end{array}\right)$$

end

Other than the potential problem of *HashMap* internally allocating data, I do not see why the content of the *run*() method cannot be executed in per-release memory. Are there downward references? Another issue is how we make explicit in SCJ*Circus* that a piece of code should run in a particular memory area. We might not want to do this via a class and data object as above. These are still open issues for the language.

A.3 *CallSign* class

The class *CallSign* is used to represent call sign objects in the program.

$$
\begin{array}{l}
\textbf{class}\ \mathit{CallSign} \mathrel{\widehat=} \textbf{begin}\\[6pt]
\quad \textbf{state}\ \mathit{CallSignState}\\
\qquad \textbf{private}\ \mathit{val} : \mathit{byteArray}\\
\qquad \hline
\qquad \mathit{val} \neq \textbf{null}\\[6pt]
\quad \textbf{initial}\ \mathit{Init} \mathrel{\widehat=} \textbf{val}\ v : \mathit{byteArray} \bullet \mathit{val} := v\\[6pt]
\quad \textbf{public}\ \mathit{hashCode} \mathrel{\widehat=} \textbf{ret} : \mathit{boolean} \bullet\\
\qquad \left(\begin{array}{l}
\textbf{var}\ h \bullet h := 0;\\
(\textbf{for}\ i = 0\ \textbf{to}\ \mathit{val}\,.\,\mathit{length}() - 1 \bullet h := h + \mathit{val}\,.\,\mathit{getA}(i));\\
\textbf{ret} := h
\end{array}\right)\\[6pt]
\quad \textbf{public}\ \mathit{equals} \mathrel{\widehat=} \textbf{val}\ \mathit{obj} : \mathit{Object};\ \textbf{res}\ \textbf{ret} : \mathit{boolean} \bullet\\
\qquad \textbf{ret} := \textbf{if}\ (\textbf{self} = \mathit{obj})\ \textbf{then}\ \mathit{jtrue}\ \textbf{else}\ \mathit{jfalse}\\[6pt]
\quad \textbf{public}\ \mathit{compareTo} \mathrel{\widehat=} \textbf{val}\ \mathit{obj} : \mathit{Object} \bullet \ldots\\[6pt]
\quad \textbf{end}
\end{array}
$$

The definition of the $\mathit{compareTo}(\mathit{obj} : \mathit{Object})$ method has been omitted; it is not central to the models presented in the report. Objects of *CallSign* are used as map keys in *StateTable*. This class is immutable.

A.4 *Vector2d* class

The class *Vector2d* is used to index the map that results from voxel hashing.

$$
\begin{array}{l}
\textbf{class}\ \mathit{Vector2d} \mathrel{\widehat=} \textbf{begin}\\[6pt]
\quad \textbf{state}\ \mathit{Vector2dState}\\
\qquad x : \mathbb{R};\\
\qquad y : \mathbb{R}\\[6pt]
\quad \textbf{initial}\ \mathit{Init} \mathrel{\widehat=} \textbf{val}\ v\_x : \mathbb{R};\ \textbf{val}\ v\_y : \mathbb{R} \bullet x := v\_x\ ;\ \ y := v\_y\\[6pt]
\quad \textbf{end}
\end{array}
$$

The class *Vector2d* is immutable too.

A.5 *Partition* class

class *Partition* $\widehat{=}$ **begin**

$$\begin{array}{|l}
\textbf{state}\ \mathit{PartitionState} \\
\hline
\textbf{private}\ \mathit{parts} : \mathit{ListArray}[\mathit{List}[\mathit{Motion}]]; \\
\textbf{private}\ \mathit{counter} : \mathit{int} \\
\hline
\mathit{parts} \neq \textbf{null} \land 0 \leq \mathit{counter} < \mathit{parts.length}() \\
\hline
\end{array}$$

$$\begin{array}{l}
\textbf{initial}\ \mathit{Init}\ \widehat{=}\ \textbf{val}\ n : \mathit{int} \bullet \\
\quad \left(\begin{array}{l}
\mathit{parts} := \textbf{newM}\ \mathit{ListArray}(n); \\
\left(\begin{array}{l}
\textbf{for}\ \mathit{index} = 0\ \textbf{to}\ \mathit{parts}\,.\,\mathit{length}() - 1 \bullet \\
\qquad \mathit{parts}\,.\,\mathit{setA}(\mathit{index}, \textbf{newM}\ \mathit{LinkedList}())
\end{array} \right); \\
\mathit{counter} := 0
\end{array} \right)
\end{array}$$

$$\begin{array}{l}
\textbf{public sync}\ \mathit{clear}\ \widehat{=} \\
\quad \left(\begin{array}{l}
(\textbf{for}\ \mathit{index} = 0\ \textbf{to}\ \mathit{parts}\,.\,\mathit{length}() - 1 \bullet \mathit{parts}\,.\,\mathit{clear}()); \\
\mathit{counter} := 0
\end{array} \right)
\end{array}$$

$$\begin{array}{l}
\textbf{public sync}\ \mathit{recordVoxelMotions}(\mathit{motions} : \mathit{List}[\mathit{Motions}])\ \widehat{=} \\
\quad \left(\begin{array}{l}
\mathit{parts}\,.\,\mathit{getA}(\mathit{counter})\,.\,\mathit{add}(\mathit{motions}); \\
\mathit{counter} := (\mathit{counter} + 1) \bmod \mathit{parts}\,.\,\mathit{length}()
\end{array} \right)
\end{array}$$

$$\begin{array}{l}
\textbf{public sync}\ \mathit{getDetectorWork}\ \widehat{=}\ \textbf{val}\ \mathit{id} : \mathit{int}; \\
\quad \textbf{res ret} : \mathit{List}[\mathit{List}[\mathit{Motion}]] \bullet \textbf{ret} := \mathit{parts}\,.\,\mathit{getA}(\mathit{id} - 1)
\end{array}$$

end

A.6 *DetectorControl* class

class *DetectorControl* $\widehat{=}$ **begin**

$$\begin{array}{|l}
\textbf{state}\ \mathit{DetectorControlState} \\
\hline
\textbf{private}\ \mathit{idle} : \mathit{booleanArray} \\
\hline
\mathit{idle} \neq \textbf{null} \\
\hline
\end{array}$$

$$\textbf{initial}\ \mathit{DetectorControlInit}\ \widehat{=}\ \textbf{val}\ n : \mathit{int} \bullet \mathit{idle} := \textbf{newM}\ \mathit{booleanArray}(n)$$

$$\begin{array}{l}
\textbf{public sync}\ \mathit{start}\ \widehat{=} \\
\quad \textbf{for}\ \mathit{index} = 0\ \textbf{to}\ \mathit{idle}\,.\,\mathit{length}() - 1 \bullet \mathit{idle}\,.\,\mathit{setA}(\mathit{index}, \mathit{jfalse})
\end{array}$$

$$\textbf{public sync}\ \mathit{notify}\ \widehat{=}\ \textbf{val}\ \mathit{id} : \mathit{int} \bullet \mathit{idle}\,.\,\mathit{setA}(\mathit{id} - 1, \mathit{jtrue});$$

$$\begin{array}{l}
\textbf{function sync}\ \mathit{done}\ \widehat{=} \\
\quad \left(\begin{array}{l}
\textbf{ret} := \mathit{jtrue}; \\
\textbf{for}\ \mathit{index} = 0\ \textbf{to}\ \mathit{idle}\,.\,\mathit{length}() - 1 \bullet \\
\textbf{if}\ \mathit{idle}\,.\,\mathit{getA}(\mathit{index}) = \mathit{jfalse} \longrightarrow \textbf{ret} := \mathit{jfalse} \\
[\!]\ \neg\ \mathit{idle}\,.\,\mathit{getA}(\mathit{index}) = \mathit{jfalse} \longrightarrow \textbf{skip} \\
\textbf{fi}
\end{array} \right)
\end{array}$$

end

We note that the specification of the *done*() method is not complete, only capturing changes made to data. As explained in [6], we require a process / action model to give a full account of the active behaviour.

B Refinement Laws

This appendix summarises all significant refinement laws that are used throughout the refinement strategy.

B.1 *Circus* Laws

Circus **Law 1 (distr-prefix-seq)**

$$c \longrightarrow (A_1 \ ; \ A_2) \ \equiv \ (c \longrightarrow A_1) \ ; \ A_2$$

Circus **Law 2 (seq-to-par-1)**

$$A_1 \ ; \ A_2 \equiv ((A_1 \ ; \ c \longrightarrow \mathbf{skip}) \llbracket \, wrtV(A_1) \mid \{\!| \, c \, |\!\} \mid wrtV(A_2) \, \rrbracket (c \longrightarrow A_2)) \setminus \{\!| \, c \, |\!\}$$

provided $wrtV(A_1) \cap wrtV(A_2) = \varnothing$ **and** $wrtV(A_1) \cap usedV(A_2) = \varnothing$ **and** $c \notin usedC(A_1) \cup usedC(A_2)$

Circus **Law 3 (seq-to-par-2)**

$$A_1 \ ; \ A_2 \equiv ((A_1 \ ; \ c\,!\,x \longrightarrow \mathbf{skip}) \llbracket \, wrtV(A_1) \mid \{\!| \, c \, |\!\} \mid wrtV(A_2) \, \rrbracket (c\,?\,x \longrightarrow A_2)) \setminus \{\!| \, c \, |\!\}$$

provided $wrtV(A_1) \cap wrtV(A_2) = \varnothing$ **and** $wrtV(A_1) \cap usedV(A_2) = \{x\}$ **and** $c \notin usedC(A_1) \cup usedC(A_2)$

Circus **Law 4 (conj-to-par)**

$$Op_1 \land Op_2 \ \equiv \ Op_1 \llbracket \, wrtV(Op_1) \mid \varnothing \mid wrtV(Op_2) \, \rrbracket \, Op_2 \ \ \mathbf{provided} \ \ wrtV(Op_1) \cap wrtV(Op_2) = \varnothing$$

Circus **Law 5 (distr-var-hide)**

$$\mathbf{var}\, x : T \bullet (A \setminus cs) \ \equiv \ (\mathbf{var}\, x : T \bullet A) \setminus cs$$

Circus **Law 6 (distr-var-par)**

$$\mathbf{var}\, x : T \bullet (A_1 \llbracket \ldots \rrbracket A_2) \ \equiv \ (\mathbf{var}\, x : T \bullet A_1) \llbracket \ldots \rrbracket (\mathbf{var}\, x : T \bullet A_2)$$

Circus **Law 7 (remove-var)**

$$\mathbf{var}\, x : T \bullet A \ \equiv \ A \ \ \mathbf{provided} \ \ x \notin FV(A)$$

Circus **Law 8 (compact-write-sets-par)**

$$A_1 \llbracket \, ns_1 \mid cs \mid ns_2 \, \rrbracket A_2 \ \equiv \ A_1 \llbracket \, ns_1' \mid cs \mid ns_2' \, \rrbracket A_2$$

provided $(ns_1 \setminus ns_1') \cap wrtV(A_1) = \varnothing$ **and** $(ns_2 \setminus ns_2') \cap wrtV(A_2) = \varnothing$

Circus **Law 9 (distr-prefix-par-1)**

$$c\,?\,x \longrightarrow (A_1 \llbracket \, ns_1 \mid cs \mid ns_2 \, \rrbracket A_2) \ \equiv \ (c\,?\,x \longrightarrow A_1 \llbracket \, ns_1 \mid cs \cup \{c\} \mid ns_2 \, \rrbracket \, c\,?\,x \longrightarrow A_2)$$

provided $c \notin usedC(A_1)$ **and** $c \notin usedC(A_2)$

Circus **Law 10 (distr-prefix-par-2)**

$$(A_1 \llbracket \, ns_1 \mid cs \mid ns_2 \, \rrbracket A_2) \ ; \ c\,!\,x \longrightarrow \mathbf{skip} \ \equiv$$
$$(A_1 \ ; \ c\,?\,y \longrightarrow \mathbf{skip}) \llbracket \, ns_1 \mid cs \cup \{\!| \, c \, |\!\} \mid ns_2 \, \rrbracket (A_2 \ ; \ c\,!\,x \longrightarrow \mathbf{skip})$$

provided $c \notin usedC(A_1)$ **and** $c \notin usedC(A_2)$ **and** $x \notin ns_1$

Circus **Law 11 (lockstep-intro)**

$$(\mu X \bullet (A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2) \,;\; X) \equiv$$

$$\left(\begin{array}{l}(\mu X \bullet A_1 \,;\; sync \longrightarrow X) \\ \quad \llbracket ns_1 \mid cs \cup \{\!|\, Sync \,|\!\} \mid ns_2 \rrbracket \\ (\mu X \bullet A_2 \,;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, sync \,|\!\}$$

provided $sync \notin usedC(A_1) \cup usedC(A_2)$ **and** $wrtV(A_1) \cap usedV(A_2) = \varnothing$ **and** $wrtV(A_2) \cap usedV(A_1) = \varnothing$

Circus **Law 12 (replace-sync-chan-seq)**

$$\left(\begin{array}{l}(\mu X \bullet A_1 \,;\; c \,!\, x \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad \llbracket ns_1 \mid cs \mid ns_2 \rrbracket \\ (\mu X \bullet c \,?\, x \longrightarrow A_2 \,;\; sync \longrightarrow X)\end{array}\right) \setminus \{\!|\, c \,|\!\}$$

$$\equiv$$

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1 \,;\; c_1 \,!\, x \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad \llbracket ns_1 \mid cs \setminus \{\!|\, c \,|\!\} \mid ns_2 \rrbracket \\ (\mu X \bullet c_2 \,?\, x \longrightarrow A_2 \,;\; sync \longrightarrow X)\end{array}\right) \\ \quad \llbracket ns_1 \cup ns_2 \mid \{\!|\, c_1, c_2, sync \,|\!\} \mid \varnothing \rrbracket \\ \big(\mu X \bullet c_1 \,?\, x \longrightarrow c_2 \,!\, x \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X\big)\end{array}\right) \setminus \{\!|\, c_1, c_2 \,|\!\}$$

provided $\{\!|\, c, sync \,|\!\} \subseteq cs \wedge c \notin usedC(A_1) \cup usedC(A_2)$ **and** c_1 and c_2 are fresh channels

Circus **Law 13 (var-intro)**

$$A(x) \equiv \mathbf{var}\, v : T \bullet v := x \,;\; A(v)$$

provided v is not free in A

Circus **Law 14 (extract-var-prefix)**

$$c \,?\, x \longrightarrow (\mathbf{var}\, v : T \bullet A) \equiv \mathbf{var}\, v : T \bullet c \,?\, x \longrightarrow A$$

provided x and v are distinct variables

Circus **Law 15 (extract-var-seq)**

$$(\mathbf{var}\, v : T \bullet A_1) \,;\; A_2 \equiv (\mathbf{var}\, v : T \bullet A_1 \,;\; A_2)$$

provided v is not free in A_2

Circus **Law 16 (extract-var-rec)**

$$\mu X \bullet (\mathbf{var}\, v : T \bullet A) \equiv \mathbf{var}\, v : T \bullet (\mu X \bullet A)$$

provided v is initialised before use in A

Circus **Law 17 (distr-prefix-par-3)**

$$c \longrightarrow \mathbf{skip} \,;\; (\mathbf{skip} \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A) \equiv \mathbf{skip} \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (c \longrightarrow A)$$

provided $c \notin cs$

Circus **Law 18 (distr-prefix-par-4)**

$$c\,!\,x \longrightarrow \mathbf{skip}\,;\ (A_1 \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket\, A_2) \equiv (c\,!\,x \longrightarrow \mathbf{skip}\,;\ A_1) \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket\, (c\,?\,y \longrightarrow A_2)$$

provided $c \in cs$ **and** y is not free in A_2

Circus **Law 19 (distr-prefix-par-5)**

$$c\,?\,x \longrightarrow v := x\,;\ (A_1 \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket\, A_2) \equiv$$
$$(c\,?\,x \longrightarrow v := x\,;\ A_1) \llbracket\, ns_1 \cup \{v\} \mid cs \mid ns_2 \,\rrbracket\, (c\,?\,y \longrightarrow A_2)$$

provided $c \in cs$ **and** v and y are not free in A_2

Circus **Law 20 (extchoice-par-intro)**

$$((c \longrightarrow A_1)\,;\ A_2) \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket\, (c \longrightarrow A_3) \equiv$$
$$(((c \longrightarrow A_1) \Box (c_1 \longrightarrow B_1) \Box \ldots \Box (c_n \longrightarrow B_n))\,;\ A_2) \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket\, (c \longrightarrow A_3)$$

provided $c \in cs$ **and** c is distinct from all c_i (the B_i can be chosen arbitrarily)

Circus **Law 21 (seq-skip-left-intro)**

$$A \equiv \mathbf{skip}\,;\ A$$

Circus **Law 22 (par-skip-intro)**

$$\mathbf{skip} \equiv (\mathbf{skip} \llbracket\, \varnothing \mid \varnothing \mid \varnothing \,\rrbracket\, \mathbf{skip})$$

Circus **Law 23 (extend-sync-par)**

$$A_1 \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket\, A_2 \equiv A_1 \llbracket\, ns_1 \mid cs \cup cs' \mid ns_2 \,\rrbracket\, A_2$$

provided $cs' \cap (usedC(A_1) \cup usedC(A_2)) = \varnothing$

Circus **Law 24 (extchoice-comm)**

$$A_1 \Box A_2 \equiv A_2 \Box A_1$$

Circus **Law 25 (distr-rec-par-1)**

$$\mu X \bullet (A_1 \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket\, A_2)\,;\ c \longrightarrow X \equiv$$
$$(\mu X \bullet A_1\,;\ c \longrightarrow X) \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket\, (\mu X \bullet A_2\,;\ c \longrightarrow X)$$

provided $c \in cs$ **and** $c \notin usedC(A_1) \cup usedC(A_2)$ **and**
$wrtV(A_1) \cap usedV(A_2) = \varnothing$ **and** $wrtV(A_2) \cap usedV(A_1)$

Circus **Law 26 (distr-rec-par-2)**

$$\mu X \bullet (((c_1 \longrightarrow A_1 \Box c_2 \longrightarrow A_2)\,;\ A_3) \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket\, (c_1 \longrightarrow A_4))\,;\ X \equiv$$
$$(\mu X \bullet (c_1 \longrightarrow A_1 \Box c_2 \longrightarrow A_2)\,;\ A_3\,;\ X) \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket\, (\mu X \bullet c_1 \longrightarrow A_4 \longrightarrow X)$$

provided $\{c_1, c_2\} \subseteq cs$ **and** $c_1 \notin usedC(A_i)$ for all $i \in \{1,2,3,4\}$ **and**
$(wrtV(A_1) \cup wrtV(A_2) \cup wrtV(A_3)) \cap usedV(A_4) = \varnothing$ **and**
$wrtV(A_4) \cap (usedV(A_1) \cup usedV(A_2) \cup usedV(A_3)) = \varnothing$

Circus **Law 27 (elim-repeated-seq-rec)**

$$\mu X \bullet A\,;\ A\,;\ \ldots\,;\ A\,;\ X \equiv \mu X \bullet A\,;\ X$$

Circus **Law 28 (var-elim)**

$(\mathbf{var}\ v : T \bullet A) \equiv A$

provided v is not free in A

Circus **Law 29 (hidden-sync-intro)**

$A \equiv (c \longrightarrow A) \setminus \{\!\mid c \mid\!\}$ **provided** $c \notin usedC(A)$

Circus **Law 30 (extract-hide-prefix)**

$c \longrightarrow (A \setminus cs) \equiv (c \longrightarrow A) \setminus cs$

provided $c \notin cs$

Circus **Law 31 (extract-hide-rec)**

$\mu X \bullet (A \setminus cs) \equiv (\mu X \bullet A) \setminus cs$

Circus **Law 32 (extract-hide-par-left)**

$(A_1 \setminus cs) [\![ns_1 \mid cs' \mid ns_2]\!] A_2 \equiv (A_1 [\![ns_1 \mid cs' \mid ns_2]\!] A_2) \setminus cs$

provided $cs \cap cs' = \varnothing$ **and** $cs \cap usedC(A_2) = \varnothing$

Circus **Law 33 (extract-hide-par-right)**

$A_1 [\![ns_1 \mid cs' \mid ns_2]\!] (A_2 \setminus cs) \equiv (A_1 [\![ns_1 \mid cs' \mid ns_2]\!] A_2) \setminus cs$

provided $cs \cap cs' = \varnothing$ **and** $cs \cap usedC(A_1) = \varnothing$

Circus **Law 34 (idem-par)**

$A \equiv (A [\![\varnothing \mid usedC(A) \mid \varnothing]\!] A)$ **provided** $wrtV(A) = \varnothing$ **and** A is deterministic

Circus **Law 35 (seq-op-comm)**

$A ;\ Op \equiv Op ;\ A$

provided $usedV(Op) \cup wrtV(A) = \varnothing$ **and** $usedV(A) \cup wrtV(Op) = \varnothing$

B.2 *Circus* Time Laws

Circus **Time Law 1 (narrow-time-budget-1)**

$$\mathbf{wait}\ t_1 \mathinner{..} t_2 \sqsubseteq \mathbf{wait}\ t_1' \mathinner{..} t_2' \ \ \mathbf{provided}\ \ t_1 \leq t_1' \ \ \mathbf{and}\ \ t_2' \leq t_2$$

Circus **Time Law 2 (narrow-time-budget-2)**

$$\mathbf{wait}\ w : t_1 \mathinner{..} t_2 \bullet A \sqsubseteq \mathbf{wait}\ w : t_1' \mathinner{..} t_2' \bullet A \ \ \mathbf{provided}\ \ t_1 \leq t_1' \ \ \mathbf{and}\ \ t_2' \leq t_2$$

Circus **Time Law 3 (time-prefix-elim)**

$$(c \mathbin{@} t \longrightarrow \mathbf{wait}\ t_1 - t) \blacktriangleleft d \ \equiv\ ((c \longrightarrow \mathbf{skip}) \blacktriangleleft d) \mathbin{|||} \mathbf{wait}\ t_1 \ \ \mathbf{provided}\ \ d \leq t_1$$

Circus **Time Law 4 (extract-inter-wait-seq)**

$$Op\ ;\ (A \mathbin{|||} \mathbf{wait}\ t) \ \equiv\ (Op\ ;\ A) \mathbin{|||} \mathbf{wait}\ t$$

provided Op is a data operation and $wrtV(Op) \cap FV(t) = \varnothing$

Circus **Time Law 5 (extract-inter-wait-var)**

$$\mathbf{var}\ x : T \bullet (A \mathbin{|||} \mathbf{wait}\ t) \ \equiv\ (\mathbf{var}\ x : T \bullet A) \mathbin{|||} \mathbf{wait}\ t$$

provided $x \notin FV(t)$

Circus **Time Law 6 (extract-inter-wait-waitblock)**

$$\mathbf{wait}\ w : t_1 \mathinner{..} t_2 \bullet (A(w) \mathbin{|||} \mathbf{wait}\ t - w) \ \equiv\ (\mathbf{wait}\ w : t_1 \mathinner{..} t_2 \bullet A(w)) \mathbin{|||} \mathbf{wait}$$

provided $t_2 \leq t$

Circus **Time Law 7 (extract-inter-wait-prefix)**

$$(c \mathbin{@} t \longrightarrow (A(t) \mathbin{|||} \mathbf{wait}\ (t_1 - t))) \blacktriangleleft d \ \equiv\ ((c \mathbin{@} t \longrightarrow A(t)) \blacktriangleleft d) \mathbin{|||} \mathbf{wait}\ t_1$$

provided $d \leq t_1$

Circus **Time Law 8 (remove-unused-time-prefix)**

$$c \mathbin{@} t \longrightarrow A \ \equiv\ c \longrightarrow A \ \ \mathbf{provided}\ \ t \notin FV(A)$$

Circus **Time Law 9 (remove-unused-wait-block)**

$$\mathbf{wait}\ w : T \bullet A \ \equiv\ \mathbf{wait}\ T\ ;\ A \ \ \mathbf{provided}\ \ w \notin FV(A)$$

Circus **Time Law 10 (distr-sync-deadline-seq)**

$$(c \longrightarrow (A_1\ ;\ A_2)) \blacktriangleleft d \ \equiv\ ((c \longrightarrow A_1) \blacktriangleleft d)\ ;\ A_2$$

Circus **Time Law 11 (split-time-budget-1)**

$$\mathbf{wait}\ 0 \mathinner{..} t \ \equiv\ \mathbf{wait}\ 0 \mathinner{..} t_1\ ;\ \mathbf{wait}\ 0 \mathinner{..} t_2 \ \ \mathbf{provided}\ \ t = t_1 + t_2$$

Circus **Time Law 12 (split-time-budget-2)**

$$\mathbf{wait}\ 0 \mathinner{..} t \sqsubseteq \mathbf{wait}\ 0 \mathinner{..} t_1\ ;\ \mathbf{wait}\ 0 \mathinner{..} t_2 \ \ \mathbf{provided}\ \ t_1 + t_2 \leq t$$

Circus **Time Law 13 (time-budget-op-comm)**

$$P(Op \; ; \; \mathbf{wait}\, t_1 \mathinner{\ldotp\ldotp} t_2) \equiv P(\mathbf{wait}\, t_1 \mathinner{\ldotp\ldotp} t_2 \; ; \; Op) \;\; \mathbf{provided} \;\; Op \text{ is a data operation}$$

This law is in fact non-compositional: it is a law about processes rather than actions. Hence, it only holds if the underlying action $Op \; ; \; \mathbf{wait}\, t_1 \mathinner{\ldotp\ldotp} t_2$ is embedded in a process P. The justification for the law comes from the structure and semantics of processes that prevents one from observing the precise time at which an (internal) state change takes place. It is proved by induction over the structure of processes.

Circus **Time Law 14 (distr-wait-seq-var)**

$$\mathbf{wait}\, t_1 \mathinner{\ldotp\ldotp} t_2 \; ; \; \mathbf{var}\, x : T \bullet A \equiv \mathbf{var}\, x : T \bullet (\mathbf{wait}\, t_1 \mathinner{\ldotp\ldotp} t_2 \; ; \; A)$$
$$\mathbf{provided} \;\; x \notin FV(t_1) \;\; \mathbf{and} \;\; x \notin FV(t_2)$$

Circus **Time Law 15 (distr-wait-seq-par)**

$$\mathbf{wait}\, t_1 \mathinner{\ldotp\ldotp} t_2 \; ; \; (Op_1 \llbracket \ldots \rrbracket Op_2) \equiv (\mathbf{wait}\, t_1 \mathinner{\ldotp\ldotp} t_2 \; ; \; Op_1) \llbracket \ldots \rrbracket (\mathbf{wait}\, t_1 \mathinner{\ldotp\ldotp} t_2 \; ; \; Op_2)$$
$$\mathbf{provided} \;\; Op_1 \text{ and } Op_2 \text{ are data operations}$$

Circus **Time Law 16 (zero-wait-intro)**

$$A \equiv \mathbf{wait}\, 0 \; ; \; A$$

B.3 High-level Patterns

High-level Law 1 (seq-share-1)

$$
\left(\begin{array}{l}
(\mu X \bullet A_1 \,;\; c\,!\,x \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns_1 \mid cs \mid ns_2 \rrbracket \\
(\mu X \bullet c\,?\,x \longrightarrow A_2 \,;\; sync \longrightarrow X)
\end{array}\right) \setminus \{\!| c |\!\}
$$

$$\equiv$$

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
(\mu X \bullet A_1 \,;\; c_1\,!\,x \longrightarrow \mathbf{skip} \,;\; c_3 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns_1 \mid (cs \setminus \{\!| c |\!\}) \cup \{\!| c_3 |\!\} \mid ns_2 \rrbracket \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2\,?\,x \longrightarrow A_2 \,;\; sync \longrightarrow X)
\end{array}\right) \setminus \{\!| c_3 |\!\} \\
\quad \llbracket ns_1 \cup ns_2 \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}
\mathbf{var}\ v : T \bullet \\
\quad \mu X \bullet \left(\begin{array}{l} (c_1\,?\,x \longrightarrow v := x) \,\Box \\ (c_2\,!\,v \longrightarrow \mathbf{skip}) \end{array}\right) ;\; X
\end{array}\right)
\end{array}\right) \setminus \{\!| c_1, c_2 |\!\}
$$

provided $\{\!| c, sync |\!\} \subseteq cs \land c \notin usedC(A_1) \cup usedC(A_2)$ **and** c_1, c_2 and c_3 are fresh channels

High-level Law 2 (seq-share-2)

$$
\left(\begin{array}{l}
(\mu X \bullet A_1 \,;\; c\,!\,x \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns_1 \mid cs_1 \mid ns_2 \cup ns_3 \cup \ldots \cup ns_n \rrbracket \\
(\mu X \bullet c\,?\,x \longrightarrow A_2 \,;\; sync \longrightarrow X) \\
\quad \llbracket ns_2 \mid cs_2 \mid ns_3 \cup ns_4 \cup \ldots \cup ns_n \rrbracket \\
(\mu X \bullet c\,?\,x \longrightarrow A_3 \,;\; sync \longrightarrow X) \\
\quad \ldots \\
\quad \llbracket ns_{n-1} \mid cs_{n-1} \mid ns_n \rrbracket \\
(\mu X \bullet c\,?\,x \longrightarrow A_n \,;\; sync \longrightarrow X)
\end{array}\right) \setminus \{\!| c |\!\}
$$

$$\equiv$$

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
(\mu X \bullet A_1 \,;\; c_1\,!\,x \longrightarrow \mathbf{skip} \,;\; c_3 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\
\quad \llbracket ns_1 \mid (cs_1 \setminus \{\!| c |\!\}) \cup \{\!| c_3 |\!\} \mid ns_2 \cup ns_3 \cup \ldots \cup ns_n \rrbracket \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2\,?\,x \longrightarrow A_2 \,;\; sync \longrightarrow X) \\
\quad \llbracket ns_2 \mid (cs_2 \setminus \{\!| c |\!\}) \cup \{\!| c_3 |\!\} \mid ns_3 \cup ns_4 \cup \ldots \cup ns_n \rrbracket \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2\,?\,x \longrightarrow A_3 \,;\; sync \longrightarrow X) \\
\quad \ldots \\
\quad \llbracket ns_{n-1} \mid (cs_{n-1} \setminus \{\!| c |\!\}) \cup \{\!| c_3 |\!\} \mid ns_n \rrbracket \\
(\mu X \bullet c_3 \longrightarrow \mathbf{skip} \,;\; c_2\,?\,x \longrightarrow A_n \,;\; sync \longrightarrow X)
\end{array}\right) \setminus \{\!| c_3 |\!\} \\
\quad \llbracket ns_1 \cup ns_2 \cup \ldots \cup ns_n \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}
\mathbf{var}\ v : T \bullet \\
\quad \mu X \bullet \left(\begin{array}{l} (c_1\,?\,x \longrightarrow v := x) \,\Box \\ (c_2\,!\,v \longrightarrow \mathbf{skip}) \end{array}\right) ;\; X
\end{array}\right)
\end{array}\right) \setminus \{\!| c_1, c_2 |\!\}
$$

provided $\{\!| c, sync |\!\} \subseteq cs_i \land c \notin usedC(A_i)$ for $i \in 1 \,..\, n$ **and** c_1, c_2 and c_3 are fresh channels

High-level Law 3 (par-share)

$$
\left(\begin{array}{l}
\mathbf{var}\, v : T \bullet \\
\quad \left(\begin{array}{l}
\mu X \bullet\ \mathit{start} \longrightarrow \mathbf{wait}\, 0 \mathinner{..} \mathit{Init}_{TB}\ ;\ \mathit{InitOp}; \\
\mathbf{var}\, x_1, x_2, \ldots, x_n : T \bullet \\
\left(\begin{array}{l}
(\mathit{record}\, ?\, x \longrightarrow (\mathbf{wait}\, 0 \mathinner{..} \mathit{RC}_{TB}\ ;\ x_1 := x)); \\
(\mathit{record}\, ?\, x \longrightarrow (\mathbf{wait}\, 0 \mathinner{..} \mathit{RC}_{TB}\ ;\ x_2 := x)); \\
\quad \ldots \\
(\mathit{record}\, ?\, x \longrightarrow (\mathbf{wait}\, 0 \mathinner{..} \mathit{RC}_{TB}\ ;\ x_n := x)); \\
\mathbf{wait}\, 0 \mathinner{..} \mathit{Merge}_{TB}\ ;\ \mathit{MergeOp}(\llbracket x_1, x_2, \ldots, x_n \rrbracket); \\
\mathit{output}\, !\, v \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X
\end{array}\right)
\end{array}\right)
\end{array}\right)
$$

$$\sqsubseteq$$

$$
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\, v : T \bullet \\
\quad \mu X \bullet \left(\begin{array}{l}
\mathit{init} \longrightarrow (\mathbf{wait}\, 0 \mathinner{..} \mathit{Init}_{TB}\ ;\ \mathit{InitOp}) \,\Box \\
\mathit{record}\, ?\, x \longrightarrow (\mathbf{wait}\, 0 \mathinner{..} \mathit{RC}_{TB}\ ;\ \mathit{MergeOp}(\llbracket x \rrbracket)) \,\Box \\
\mathit{output}\, !\, v \longrightarrow \mathbf{skip}
\end{array}\right) ;\ X
\end{array}\right) \\
\quad \llbracket \varnothing \mid \{\!|\, \mathit{init}, \mathit{record}, \mathit{output} \,|\!\} \mid \varnothing \rrbracket \\
\left(\mu X \bullet \mathit{init} \longrightarrow \mathit{start} \longrightarrow \left(\begin{array}{l}
(\mathit{record}\, ?\, y \longrightarrow \mathbf{skip}) \;|||\; \\
(\mathit{record}\, ?\, y \longrightarrow \mathbf{skip}) \;|||\; \\
\quad \ldots \\
(\mathit{record}\, ?\, y \longrightarrow \mathbf{skip})
\end{array}\right) ;\ \mathit{output}\, ?\, y \longrightarrow \mathbf{skip}\ ;\ \mathit{sync} \longrightarrow X\right)
\end{array}\right) \setminus
$$

$$\{\!|\, \mathit{init} \,|\!\}$$

provided *InitOp* and *MergeOp* are data operations **and**
$\mathit{wrtV}(\mathit{InitOp}) = \{v\} = \mathit{wrtV}(\mathit{MergeOp})$ **and** $\mathit{MergeOp}(b_1 \uplus b_2) = \mathit{MergeOp}(b_1)\ ;\ \mathit{MergeOp}(b_2)$

High-level Law 4 (par-share-control)

$$
\left(
\begin{array}{l}
\left(
\begin{array}{l}
(\mu X \bullet A\,;\ start \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X) \\
\quad [\![ns \mid \{\!|\, start, sync \,|\!\} \mid \varnothing]\!] \\
(\mu X \bullet start \longrightarrow \mathbf{var}\ v : T \bullet A_1\,;\ record\,!\,v \longrightarrow \mathbf{skip}\,;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X) \\
\quad [\![\varnothing \mid \{\!|\, start, output, sync \,|\!\} \mid \varnothing]\!] \\
(\mu X \bullet start \longrightarrow \mathbf{var}\ v : T \bullet A_2\,;\ record\,!\,v \longrightarrow \mathbf{skip}\,;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X) \\
\quad \ldots \\
\quad [\![\varnothing \mid \{\!|\, start, output, sync \,|\!\} \mid \varnothing]\!] \\
(\mu X \bullet start \longrightarrow \mathbf{var}\ v : T \bullet A_n\,;\ record\,!\,v \longrightarrow \mathbf{skip}\,;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X)
\end{array}
\right) \\
\quad [\![ns \mid \{\!|\, start, record, output, sync \,|\!\} \mid \varnothing]\!] \\
\left(
\mu X \bullet init \longrightarrow start \longrightarrow
\left(
\begin{array}{l}
(record\,?\,y \longrightarrow \mathbf{skip})\ |\!|\!| \\
(record\,?\,y \longrightarrow \mathbf{skip})\ |\!|\!| \\
\quad \ldots \\
(record\,?\,y \longrightarrow \mathbf{skip})
\end{array}
\right)
;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X
\right)
\end{array}
\right)
$$

$$\sqsubseteq$$

$$
\left(
\begin{array}{l}
(\mu X \bullet A\,;\ init \longrightarrow \mathbf{skip}\,;\ start \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X) \\
\quad [\![ns \mid \{\!|\, start, sync \,|\!\} \mid \varnothing]\!] \\
(\mu X \bullet start \longrightarrow \mathbf{var}\ v : T \bullet A_1\,;\ record\,!\,v \longrightarrow \mathbf{skip}\,;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X) \\
\quad [\![ns \mid \{\!|\, start, output, sync \,|\!\} \mid \varnothing]\!] \\
(\mu X \bullet start \longrightarrow \mathbf{var}\ v : T \bullet A_2\,;\ record\,!\,v \longrightarrow \mathbf{skip}\,;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X) \\
\quad \ldots \\
\quad [\![ns \mid \{\!|\, start, output, sync \,|\!\} \mid \varnothing]\!] \\
(\mu X \bullet start \longrightarrow \mathbf{var}\ v : T \bullet A_n\,;\ record\,!\,v \longrightarrow \mathbf{skip}\,;\ output\,?\,y \longrightarrow \mathbf{skip}\,;\ sync \longrightarrow X)
\end{array}
\right)
$$

provided $\{start, sync\} \cap usedC(A) = \varnothing$ **and**

$\{start, record, output, sync\} \cap usedC(A_i) = \varnothing$ for all $i : 1 \mathinner{..} n$

High-level Law 5 (sync-barrier-elim)

$$\left(\begin{array}{l}(\mu X \bullet start \longrightarrow A_1 \,;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs_1 \mid ns_2 \cup \ldots \cup ns_n]\!] \\ (\mu X \bullet start \longrightarrow A_2 \,;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad [\![ns_2 \mid cs_2 \mid ns_3 \cup \ldots \cup ns_n]\!] \\ \qquad \ldots \\ \quad [\![ns_{n-1} \mid cs_{n-1} \mid ns_n]\!] \\ (\mu X \bullet start \longrightarrow A_n \,;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)\end{array}\right)$$

$$\equiv$$

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet start \longrightarrow A_1 \,;\; notify\,!\,1 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad [\![ns_1 \mid cs_1 \setminus \{\!|\, done \,|\!\} \mid ns_2 \cup \ldots \cup ns_n]\!] \\ (\mu X \bullet start \longrightarrow A_2 \,;\; notify\,!\,2 \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X) \\ \quad [\![ns_2 \mid cs_2 \setminus \{\!|\, done \,|\!\} \mid ns_3 \cup \ldots \cup ns_n]\!] \\ \qquad \ldots \\ \quad [\![ns_{n-1} \mid cs_{n-1} \setminus \{\!|\, done \,|\!\} \mid ns_n]\!] \\ (\mu X \bullet start \longrightarrow A_n \,;\; notify\,!\,n \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X)\end{array}\right) \\ \quad [\![ns_1 \cup \ldots \cup ns_n \mid \{\!|\, start, notify, sync \,|\!\} \mid \varnothing]\!] \\ \left(\mu X \bullet start \longrightarrow \left(\begin{array}{l}(notify\,!\,1 \longrightarrow \mathbf{skip}) \,|||\, \\ (notify\,!\,2 \longrightarrow \mathbf{skip}) \,|||\, \\ \ldots \\ (notify\,!\,n \longrightarrow \mathbf{skip})\end{array}\right) ;\; done \longrightarrow \mathbf{skip} \,;\; sync \longrightarrow X\right)\end{array}\right) \setminus \{\!|\, notify \,|\!\}$$

provided $\{\!|\, start, done, sync \,|\!\} \subseteq cs_i \land \{start, done, sync\} \cap usedC(A_i) = \varnothing$ for all $i : 1 \mathinner{..} n$
and $notify$ is a fresh channel of type $\mathbb{N}$

High-level Law 6 (sync-barrier-control)

$$\left(\mu X \bullet start \longrightarrow \begin{pmatrix} (notify\,!\,1 \longrightarrow \mathbf{skip}) \mid\mid\mid \\ (notify\,!\,2 \longrightarrow \mathbf{skip}) \mid\mid\mid \\ \ldots \\ (notify\,!\,n \longrightarrow \mathbf{skip}) \end{pmatrix} ;\ done \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X \right)$$

$$\equiv$$

$$\left(\begin{array}{l} \left(\begin{array}{l} \big(\mu X \bullet reset \longrightarrow start \longrightarrow X ;\ sync \longrightarrow X\big) \\ \quad [\![\varnothing \mid \{\!|\, start, sync \,|\!\} \mid \varnothing]\!] \\ \big(\mu X \bullet start \longrightarrow notify\,!\,1 \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X\big) \\ \quad [\![\varnothing \mid \{\!|\, start, sync \,|\!\} \mid \varnothing]\!] \\ \big(\mu X \bullet start \longrightarrow notify\,!\,2 \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X\big) \\ \quad [\![\varnothing \mid \{\!|\, start, sync \,|\!\} \mid \varnothing]\!] \\ \quad \ldots \\ \quad [\![\varnothing \mid \{\!|\, start, sync \,|\!\} \mid \varnothing]\!] \\ \big(\mu X \bullet start \longrightarrow notify\,!\,n \longrightarrow \mathbf{skip} ;\ sync \longrightarrow X\big) \end{array}\right) \\ \quad [\![\varnothing \mid \{\!|\, reset, notify, sync \,|\!\} \mid \varnothing]\!] \\ \left(\begin{array}{l} \mathbf{var}\ active : \mathbb{P}\,(1 \mathinner{..} n) \bullet \\ \quad \mu X \bullet \left(\begin{array}{l} (reset \longrightarrow active := 1 \mathinner{..} n) \\ \quad \Box \\ (notify\,?\,x \longrightarrow \left(\begin{array}{l} active := active \setminus \{x\}; \\ \mathbf{if}\ active = \varnothing \longrightarrow done \longrightarrow \mathbf{skip} \\ [\!]\ \neg\ active = \varnothing \longrightarrow \mathbf{skip} \\ \mathbf{fi} \end{array}\right)) \end{array}\right) ;\ X \end{array}\right) \end{array}\right) \setminus \{\!|\, reset \,|\!\}$$

High-level Law 7 (sync-barrier-design)

$$
\begin{array}{l}
\left(\begin{array}{l}
(\mu X \bullet start \longrightarrow A_1 \ ; \ \ done \longrightarrow \mathbf{skip} \ ; \ \ sync \longrightarrow X) \\
\quad \llbracket ns_1 \mid cs_1 \mid ns_2 \cup \ldots \cup ns_n \rrbracket \\
(\mu X \bullet start \longrightarrow A_2 \ ; \ \ done \longrightarrow \mathbf{skip} \ ; \ \ sync \longrightarrow X) \\
\quad \llbracket ns_2 \mid cs_2 \mid ns_3 \cup \ldots \cup ns_n \rrbracket \\
\qquad \ldots \\
\quad \llbracket ns_{n-1} \mid cs_{n-1} \mid ns_n \rrbracket \\
(\mu X \bullet start \longrightarrow A_n \ ; \ \ done \longrightarrow \mathbf{skip} \ ; \ \ sync \longrightarrow X)
\end{array}\right) \\
\quad \sqsubseteq \\
\left(\begin{array}{l}
\big(\mu X \bullet reset \longrightarrow start \longrightarrow \mathbf{skip} \ ; \ \ sync \longrightarrow X\big) \\
\quad \llbracket \varnothing \mid \{\!|\, reset, start, sync \,|\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}
(\mu X \bullet start \longrightarrow A_1 \ ; \ \ notify\,!\,1 \longrightarrow \mathbf{skip} \ ; \ \ sync \longrightarrow X) \\
\quad \llbracket ns_1 \mid cs_1 \setminus \{\!|\, done \,|\!\} \mid ns_2 \cup \ldots \cup ns_n \rrbracket \\
(\mu X \bullet start \longrightarrow A_2 \ ; \ \ notify\,!\,2 \longrightarrow \mathbf{skip} \ ; \ \ sync \longrightarrow X) \\
\quad \llbracket ns_2 \mid cs_2 \setminus \{\!|\, done \,|\!\} \mid ns_3 \cup \ldots \cup ns_n \rrbracket \\
\qquad \ldots \\
\quad \llbracket ns_{n-1} \mid cs_{n-1} \setminus \{\!|\, done \,|\!\} \mid ns_n \rrbracket \\
(\mu X \bullet start \longrightarrow A_n \ ; \ \ notify\,!\,n \longrightarrow \mathbf{skip} \ ; \ \ sync \longrightarrow X)
\end{array}\right) \\
\quad \llbracket ns_1 \cup \ldots \cup ns_n \mid \{\!|\, start, notify, sync \,|\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}
\mathbf{var}\ active : \mathbb{P}\,(1 \mathinner{..} n) \bullet \\
\quad \mu X \bullet \left(\begin{array}{l}
(reset \longrightarrow active := 1 \mathinner{..} n) \\
\quad \Box \\
(notify\,?\,x \longrightarrow \left(\begin{array}{l}
active := active \setminus \{x\}; \\
\mathbf{if}\ active = \varnothing \longrightarrow done \longrightarrow \mathbf{skip} \\
[\!]\ \neg\ active = \varnothing \longrightarrow \mathbf{skip} \\
\mathbf{fi}
\end{array}\right))
\end{array}\right) \ ; \ \ X
\end{array}\right)
\end{array}\right) \setminus \{\!|\, reset, notify \,|\!\}
\end{array}
$$

provided $\{\!|\, reset, start, done, sync \,|\!\} \subseteq cs_i \wedge \{reset, start, done, sync\} \cap usedC(A_i) = \varnothing$ for all $i : 1 \mathinner{..} n$
and *notify* is a fresh channel of type $\mathbb{N}$

C Mock Objects

Dummy definitions that enable the parsing and type-checking of the models.

C.1 Unit Type

$[unit]$

Empty Tuple

```
%%Zword \emptytuple emptytuple
```

$emptytuple : unit$

C.2 Array Types

$Array[X]$
$getA : int \to X$
$setA : int \times X \to X$
$length : unit \to int$

$intArray == Array[int]$

$byteArray == Array[byte]$

$floatArray == Array[float]$

$booleanArray == Array[boolean]$

C.3 Classes Types

$[Object]$

$List[X]$
$elems : unit \to \mathbb{P}\, X$

$HashMap[X, Y]$
$get : X \to Y$
$values : unit \to List[Y]$

$CallSign$

$Vector2d$
$x : \mathbb{R}$
$y : \mathbb{R}$

Vector3d

$x : \mathbb{R}$
$y : \mathbb{R}$
$z : \mathbb{R}$

RawFrame

$planeCnt : int$
$positions : Array[\mathbb{R}]$
$getCallSign : int \rightarrow Aircraft$
$getCallSignOffset : int \rightarrow int$
$find : Aircraft \rightarrow int$

FrameBuffer

StateTable

$position\_map : HashMap[CallSign, Vector3d]$

Motion

Partition

$getDetectorWork : int \rightarrow List[List[Motion]]$

DetectorControl

C.4 Infrastructure Classes

AperiodicEvent

InputHandler

OutputHandler

ReducerHandler

DetectorHandler

DetectorHandlerArray

C.5 Auxiliary Functions

$MkCallSign : Aircraft \rightarrow CallSign$

$MkMotion : (Aircraft \times Vector \times Vector) \rightarrow Motion$

References

[1] The Real Numbers in Z. In *Proceedings of the 2nd BCS-FACS Northern Formal Methods Workshop, Ilkley, 14-15 July 1997*. British Computer Society.

[2] A. Cavalcanti, P. Clayton, and C. O'Halloran. From Control Law Diagrams to Ada via *Circus*. Technical report, University of York, Heslington, York, YO10 5DD, U.K., April 2008.

[3] A. Cavalcanti, P. Clayton, and C. O'Halloran. From control law diagrams to Ada via *Circus*. *Formal Aspects of Computing*, 23(4):465–512, July 2011.

[4] A. Cavalcanti, A. Sampaio, and J. Woodcock. A Refinement Strategy for *Circus*. *Formal Aspects of Computing*, 15(2–3):146–181, 2003.

[5] A. Cavalcanti, A. Wellings, J. Woodcock, K. Wei, and F. Zeyda. Safety-Critical Java in *Circus*. In *JTRES '11: Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems*, New York, NY, USA, September 2011. ACM.

[6] A. Cavalcanti, A. Wellings, J. Woodcock, K. Wei, and F. Zeyda. Development of Safety-Critical Java Programs using *Circus*. *Concurrency and Copmutation: Practice and Experience*, June 2012. Paper submitted for review.

[7] R. B. Jones. ICL ProofPower. *BCS FACS FACTS*, Series III, 1(1):10–13, 1992.

[8] T. Kalibera, J. Hagelbergand, F. Pizlo, A. Plsek, B. Titzer, and J. Vitek. CDx: A Family of Real-time Java Benchmarks. In *Proceedings of the 7th International Workshop on Java Technologies for Real-Time and Embedded Systems*, JTRES 2009, pages 41–50. ACM, September 2009.

[9] C. C. Morgan. Auxiliary variables in data refinement. *Information Processing Letters*, 29(6):293–296, February 1988.

[10] M. Oliveira, F. Zeyda, and A. Cavalcanti. A tactic language for refinement of state-rich concurrent specifications. *Science of Computer Programming*, 76(9):792–833, September 2011.

[11] F. Zeyda, M. Oliveira, and A. Cavalcanti. Mechanised support for sound refinement tactics. *Formal Aspects of Computing*, 24(1):127–160, January 2012.