Chapter 16

Specifying ATC Procedures

As suggested by the Difference Condition in Chapter 15.2, we must analyse ATC procedures to explain what could have gone wrong. We shall specify these procedues in the formal specification language TLA+, based on the tense logic TLA, which was devised for the purpose of verifying procedures and algorithms against their specifications. First, we undertake a superficial analysis of the procedure which will help us formulate the TLA+ modules describing the procedures at a somewhat high level, but one sufficient for our analysis in this example.

Expert knowledge tells us that the ATC procedures which are involved consist of

- receiving the flight information (FI), consisting of flight plan and flight progress data on the aircraft, from previous control facility
- handling the aircraft
- updating the FI
- transmitting the FI to the next control facility

Additionally, there is a transmission constraint:

- the FI transmitted from the previous ATC facility is reliably (no losses; without alteration) received by the next ATC facility

There are thus a number of state predicates to consider at each point of progress of the flight:

$\langle 3a \rangle$ ATC procedures are implemented correctly
$\langle 3b \rangle$ FI is correct
$\langle 3c \rangle$ AC position and direction of flight (AC-PD) is consistent with FI

These state predicates must be duplicated, one for each ATC facility. Thus the state predicates for these procedures for the entire example can be partitioned naturally into three; those for SATCC, those for LATCC and those for BATCC:

$\langle 31\rangle$ SATC procedures are implemented correctly
$\langle 32\rangle$ FI is correct at SATC
$\langle 33\rangle$ AC-PD is consistent with FI at SATC
$\langle La\rangle$ LATC procedures are implemented correctly
$\langle Lb\rangle$ FI are correct at LATC
$\langle Lc\rangle$ AC-PD is consistent with FI at LATC
$\langle Ba\rangle$ BATC procedures are implemented correctly
$\langle Bb\rangle$ FI are correct at BATC
$\langle Bc\rangle$ AC-PD is consistent with FI at BATC

We must also make the connection (again, this is *expert knowledge*) amongst the FI at the three centers:

$\{SLtrans\}$ FI from SATC are transmitted without change to LATC
$\{LBtrans\}$ FI from LATC are transmitted without change to BATC

and each of these processes can be further decomposed:

[*SLhuman*] FI from SATC are correctly entered by SATC controller for transmission to LATC
[*SLchannel*] FI from SATC are transmitted reliably (no loss, no change) over the channel to LATC
[*LBhuman*] FI from LATC are correctly entered by LATC controller for transmission to BATC
[*LBchannel*] FI from LATC are transmitted reliably (no loss, no change) over the channel to BATC

The main task of formal specifications[1] is to describe entities and their interactions in a completely unambiguous manner. The descriptions are written in a language that has a well defined semantics, so the specification can be compared against – or executed by – a machine or computer program. The logic EL that we shall use for proving correctness and relative sufficiency of the WBA is an extension of TLA, thus ensuring TLA+ may be used for specifying procedures in the WBA itself.

[1]We are not concerned about informal specifications in this work. These are documents written in an (often natural, and therefore unprecise) language, readable by humans, but less useful for technical implementations we intend to achieve.

It is a straightforward exercise to write TLA+ modules *ATCproc*, which defines a generic ATC handling of 'flight slips', and *ATCtrans* which defines the transmission of FI from one ATCC to another, as in $\langle 3a \rangle$, $\langle 3b \rangle$, $\langle 3c \rangle$. Thus one may write a TLA+ specification which defines the complete handover sequence from SATC, LATC to BATC (with MATC as a correct alternative), which includes the *ATCproc* for SATC, LATC and BATC, and *ATCtrans* for SATC-to-LATC and for LATC-to-BATC and LATC-to-MATC. These TLA+ modules are to be found in Section 16.2. The formulas defining the relevant state predicates and the way they are modified when correct or false data are entered and transmitted lead to an expression of the specification contained in these modules as a TLA *predicate-action diagram*, a PAD. But first we must introduce the temporal logic specification language TLA+.

16.1 Introducing TLA+

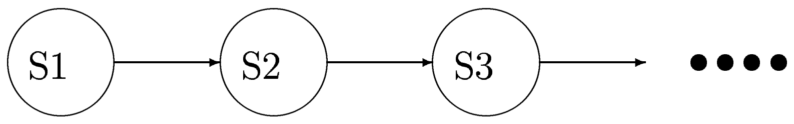
We aim to give just enough information about TLA+ to enable the reader to understand the modules we write and how they are used. This explanation does assume a basic knowledge and facility with the symbolism of first-order logic and data structures roughly equivalent to what one would pick up from a discrete mathematics course for computer science students. For a more thorough introduction to TLA and TLA+, we suggest [Lad97], which is intended to give a gentler introduction than the original [Lam94c]; further information on TLA+ can be found in [Lam96], [Lam97] and [Blu97].

Like every formal language, TLA+ has a well-defined syntax and semantics. A TLA+ specification is organized as a collection of modules with a well-defined appearance. We will discuss module *ATCtrans* (figure 16.1), since it uses most of the TLA+ features we need.

The module is delimited from other modules by bars on top and bottom. The top bar contains the module's name. Between these bars the body of the module is placed. The body is a list of statements, where a statement can be a *declaration*, a *definition*, an *assumption* or a *theorem*. To improve readability, we also use horizontal lines (midbars) to mark a separation of module parts. These midbars are purely decorative and do not have any formal meaning. Every part of the module may be given a name at will. Some of these names have meaning (namely, those identical with the classes of statements just mentioned), and some do not. The five parts of the module *ATCtrans* are named **DECLARATIONS**, **ASSUMPTIONS**, **PREDICATES**, **ACTIONS** and **DEFINITION**. Of these, **PREDICATES** and **ACTIONS** are actually types of **DEFINITION**. We introduce them for readability, to separate state predicates and actions, which are otherwise distinguishable by their syntax. But formally they are just types of **DEFINITION**.

A specification in TLA+ defines a process (or *state machine* to computer scientists) which has a *state*, consisting of an assignment of value for each of its variables. The state changes as time progresses. This change is taken to be discrete, but this is no restriction since continuous change can easily be modelled [Lam93b]. What is it that changes? The values that the variables have. These values are all taken to be sets (as defined by axiomatic first-order set theory, also known as Zermelo-Fraenkel set theory or ZF) because all data types can satisfactorily be defined as sets, and it has been argued that dealing just with sets makes logical life a lot easier.

Let us think of a state as a 'snapshot' of the state of the world. Then we can show how the world changes by continually taking snapshots, arranging them in an unending series (because time is unending – well, relatively speaking) and comparing pairs of snapshots to see what has changed. Thus the following picture, in which our 'snapshots' are curiously round instead of square:



Such an entire sequence of snapshots is called a *behavior*. A TLA+ specification module is a description in logic which behaviors either *satisfy* or not. It defines, in other words, a set of behaviors, namely those behaviors which satisfy it, and thus distinguishes these behaviors from those others which do not.

In order to compare across snapshots, we must be able to identify things in more than one snapshot. Things which persist across snapshots, but which may change value or attributes as they do so, are called *variables*. Things which must stay the same are called *constants*. But one should observe that constants are in fact *mathematical* variables, in that they have determinate but (usually) unspecified values in a behavior. For example, *ChannelSize* is a constant in *ATCtrans*. We do not know its value, but this value remains the same in all states of a behavior. In contrast, the variable *channel*, which is a sequence of messages that have been sent and not yet received, has different values in different states: messages are sent in various states and received in later states of the behavior, and what's 'in' the channel is constantly changing.

But back to syntax. Constants or variables must be declared to be such in the **DECLARATIONS** part of a TLA+ module. All constants/variables must be declared before being used. The **DECLARATIONS** part also contains statements concerning which other previously-defined modules are to be included (by either **extends** statements or **instance** statements) in the current module. The **extends** statement functions just like a macro: the named module's contents are simply to be included as they are, right there in the current module. One can think of it as an instruction to remove the **extends** statement and replace it by the contents of the named module. In contrast, an **instance** statement changes names of definitions, variables and constants. We shall explain **instance** later.

The **PREDICATES** part contains definitions of *state predicates*, which are expressions in first-order logic – actually ZF set theory – containing constants and variables. The **ACTIONS** part contains definitions of *actions*: expressions of ZF set theory containing not only constants and variables but also *primed variables.*

The meaning of a state predicate is straightforward. In a given state (snapshot), the predicate is either true or false, given the values of its variables in that state. The story for actions is a little more complicated. Actions are actually binary relations on states – that is, a definition of an action is a comparison between two states which contain the same variables. The values of those variables in the first state are represented by the (unprimed) variables, and the values in the second state of the pair being compared are represented by the *primed variables.* Thus an action definition such as

$$x' = x + 1$$

is a true comparison between two states just in case the value of x in the second is 1 greater than the value of x in the first. (We leave the question of what $x + 1$ is when x is any set to the set theory textbooks to explain – it is well-defined!).

The actions in *ATCtrans* are *send* and *receive* actions for messages, defined in the standard computer-science manner, using notation for operations on sequences (which come in the module *Sequences*, which *ATCtrans* **extends**, but which we don't include here). So *Send(msg)* is a definition parametrised by the symbol *msg* (that means that in any use, the symbol *msg* has to be quantified or to be substituted by a variable, constant or expression), which says that *msg* is a member of the set *Messages* (*Messages* is a constant), that the length of the sequence *channel* (i.e., the number of elements it contains) must be less than the constant *ChannelSize*, and that the value of the channel after is the value before, with the new message concatenated at the end (that is what $channel \circ \langle msg \rangle$ means). The *Receive* action requires that the channel is not empty, and it removes an element from the head of the channel (this is what occurs when the new value of *channel* is the *Tail* of the old – the *Tail* operator just yields the previous value with the first element missing. This requires that the previous sequence actually *have* a first element to be removed – this being the point of specifying that it be non-empty).

The manner in which we write the conjunction, as a 'bulleted list', vertically aligned with the conjunction sign before each element of the list, aids the readability of complex logical formulae, as argued in [Lam94a]. It is a form of pretty-printing for logical formula. Indentation takes the place of parentheses, and the conjunction and disjunction symbols are placed before conjuncts and disjuncts to distinguish them.

The final definition part defines *SomeAction* as being either a *Send* or *Receive* action, and defines *Safety* as being a formula that is *Init* conjoined with a formula involving *SomeAction*. The *Init* subformula is straightforward – it was defined earlier in the module to mean that the channel is empty. The formula $[A]_f$, where f is a state function (a variable or a term involving variables), is defined to mean $A \vee (f' = f)$, that is, either A, or f remains the same. So $[\mathit{SomeAction}]_{\mathit{channel}}$ means $\mathit{SomeAction} \vee (\mathit{channel}' = \mathit{channel})$, which says that either *SomeAction* is true or *channel* doesn't change value. The symbol $\Box$ means 'always', which means 'at every step' in the future. (We shall also later use the symbol $\Diamond$, 'eventually', which can be defined as $\neg\Box\neg$: $\Diamond A$, A eventually comes to pass, if and only if $\neg\Box\neg A$, it is not the case that A is always false.) In other words, $\Box[\mathit{SomeAction}]_{\mathit{channel}}$ means that 'at every step in the future, either *SomeAction* occurs or *channel* doesn't change value'. This defines the behavior of the process. A process satisfying this formula *Safety* must start with an empty channel, and then after that either adjacent states are identical ($\mathit{channel}' = \mathit{channel}$), or they are related by *SomeAction*, i.e., either by a *Send* of some message, or by a *Receive*. So that says that every change in the variable *channel* in a behavior that satisfies the formula *Safety* must be caused by a *Send* as defined, or by a *Receive* as defined. The purpose of the entire module, then, is expressed in the formula *Safety* – that is how things are supposed to work according to this module.

Back to the **Assumptions**. These are exactly what they say they are. The assumptions must be fulfilled in any behavior that satisfies the module. For example, no specification (no instance of the module) corresponds to a case in which the *ChannelSize* is 0; or in which the not-a-message $\bot_{\mathit{Messages}}$ is in fact a *Message*; or in which a *msg* is not a pair.

The formula *Safety* contains a specification of what is known as a *safety property* of behaviors. It constrains the behavior step by step into only changing in specified ways. Behaviors can also exhibit *liveness properties*. A liveness property says roughly that something will eventually happen, but it doesn't say when. Any arbitrary set of behaviors can in fact be characterised as the conjunction of a safety property with a liveness property [AS85]. The usual liveness properties included in TLA specifications are those of *Weak Fairness* and *Strong Fairness*. We don't appear to need liveness properties to handle this example – although the ATC procedures should maybe include some liveness properties, it doesn't seem to change the outcome of the analysis. Liveness properties are logically complex to handle, so we'd just as soon not deal with them here, and refer the interested reader to [Lad97].

We shall need to use the module-inclusion operator **instance** in other modules, so we explain the various different versions of it. Basically, **instance** is like *extends* in that it brings definitions across, but unlike in that it doesn't bring declarations. So the variables and constants in the definitions must be declared in the current module; *or* they may be syntactically substituted by variables and constants that are declared in the current module. Furthermore, the definition names are prepended with the name of the module instance they are associated with. This allows one to instantiate, say, three versions of the same module, and the definitions of each will be distinguished by their names. The versions of **instance** are:

* "**instance** `modulelist`"
adds the declarations and definitions, **not** assumptions and theorems, from the module(s) of `modulelist` to the current module. (See module *Landing_Norms* for example.)

* "**instance** `modulename` $x_1 \leftarrow expr_1$, ... , $x_k \leftarrow expr_k$"
adds the declarations and definitions from module `modulename` to the current module and assigns to each symbol x_i (where $1 \leq x_1 \leq k$) a definition of symbol $expr_i$. Therefore $expr_i$ must be a defined symbol of module `modulename`.

* "`symbol` $\triangleq$ **instance** `modulename` $x_1 \leftarrow expr_1$, ... , $x_k \leftarrow expr_k$"
is the same as the statement above, except that it defines symbols `symbol`.x_i instead of x_i. This is useful if a module is included more than once to obtain separate copies of the same specification (e.g. to guarantee the same type of behaviour for several components – see module *This_ATCcomm_history* for example).

16.2 Physical Subsystems

Since the considerations in Chapter 15.2 suggested that a possible error during handling or during the transmission of flightdata between the air traffic control centers (called a *handoff* in aviation terminology) may be a key cause of the incident, we shall focus on the function of the communication systems.

16.2.1 Inter-ATC Communication

We write specifications of the ATC handoff procedures as TLA+ modules using first-in-first-out (FIFO) channels to handoff between air traffic control centers (ATCCs) and using sets to represent the data (the "flight slips") of flights being handled within a specific ATCC, as required in TLA. Module *ATCtrans* (Figure 16.1) in the previous section defines the basic actions on the lowest system level. An ATCC is connected to the world by a *channel*. It can send data to this channel or receive data when it is available from the channel. These definitions of actions are insufficient at this level to characterize the whole communication system. However they can be used by modules specifying a more sophisticated level of the system. So we define a hierarchy of modules enhancing the system (and getting closer to reality) step by step. This hierarchy is illustrated in figure 16.2, and is defined using the **instance** and **extends** definition facilities of TLA+.

Module *ATCproc* (figure 16.3) introduces a storage, in which an ATCC stores messages it gets from a channel or which should be sent to a channel. The former is defined by action *Download*, the latter by action *Upload(fid)*. The definitions of these actions use *Send(msg)* and *Receive* from the instantiated module *ATCtrans*. In *ATCproc* assumptions are made that the channel can contain at least one message and that all messages are valid. "To be valid" in this context means that all messages in scope have the form "$\langle \mathit{fid}, \mathit{fdata} \rangle$". An upload of a message with a particular flight ID (fid) to the channel is possible only if this message is currently present in the storage. It is removed from the storage as soon as the message is transmitted. A message can be downloaded only if the maximal storage size is not exceeded.

We have so far specified the upload and download actions one ATCC can perform. To establish communication between ATCCs, at least two ATCCs need to be included in the specification; and so we instantiate *ATCproc* twice (once for each ATCC) in the next module and define a *Handoff* action (Figure 16.4).

In principle, we now have everything we need to explain the communication between ATCCs. However, to use the specification for the proof of possible errors concerning the actions defined, we need to specify under which circumstances we interpret an action to be executed incorrectly. To differentiate the case in which the *Handoff* was performed correctly from that in which it was not, we introduce a *history variable.* The purpose of a history variable is to record a part of a state for 'posterity', as it were. In figure 16.5 we present the module *ATCcomm_history* using the variable *persistent_data* to state whether the data during handoff were changed or not. We intend that *persistent_data* is a record containing the fid and flight data of the aircraft (AC) whose data is transferred currently (whether it is precisely such a thing, or whether it can be something else, will depend on the use we make of it in the module). We define the handoff to be executed correctly if the history variable remains unchanged. Intuitively one would argue vice versa: The fact that the data has not changed allows us to assume that it was transferred correctly. Similarly we define an action to be incorrect if the value of the history variable changes during execution. To be able to differentiate between these two possibilities, we have split the *Handoff* action into $\mathit{Handoff}_{\mathit{correct}}(x, y)$ and $\mathit{Handoff}_{\mathit{incorrect}}(x, y)$.

Finally, we define the possible communications between the three ATCCs involved in this incident. See the module *This_ATCcomm_history* in Figures 16.6 and 16.7. Since three ATCCs are involved, we define three instances of the module *ATCcomm_history* with a complete initialized set of variables, one for for each ATCC. Including a fourth copy of *ATCcomm_history* by an extension statement avoids redefining the history variable and assumptions.

This completes the definition of (correct and incorrect) ATC handoff procedures, and we shall now see how to use them in the analysis of the incident, using TLA *Predicate-Action Diagrams*, a form of transition diagram defined using logical formulas.

module *ATCtrans*

DECLARATIONS

extends $Naturals, Sequences$

CONSTANTS $ChannelSize, Messages, \perp_{Messages}, msg$

VARIABLE $channel$

ASSUMPTIONS

ASSUME $ChannelSize > 0$

ASSUME $\perp_{Messages} \notin Messages$

ASSUME $msg \in Messages \Rightarrow \exists fid, fdata : msg = \langle fid, fdata \rangle$

PREDICATES

$Init \triangleq channel = \langle\,\rangle$

ACTIONS

$$Send(msg) \triangleq \begin{array}{l} \land\ msg \in Messages \\ \land\ Len(channel) < ChannelSize \\ \land\ channel' = channel \circ \langle msg \rangle \end{array}$$

$$Receive \triangleq \begin{array}{l} \land\ channel \neq \langle\,\rangle \\ \land\ channel' = Tail(channel) \end{array}$$

DEFINITION

$$SomeAction \triangleq \begin{array}{l} \lor\ \exists\, msg : Send(msg) \\ \lor\ Receive \end{array}$$

$$Safety \triangleq \begin{array}{l} \land\ Init \\ \land\ \Box[SomeAction]_{channel} \end{array}$$

Figure 16.1: Sample Module *ATCtrans*

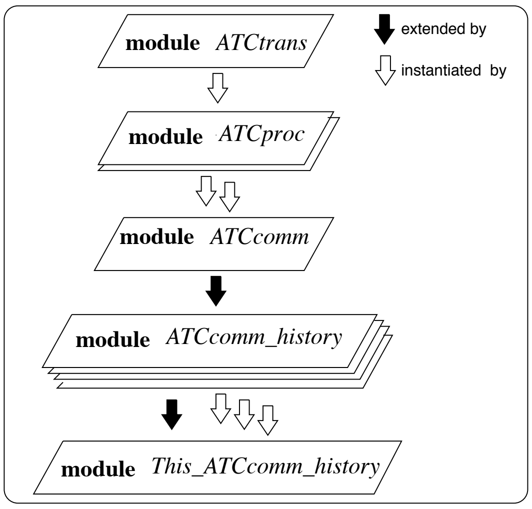


Figure 16.2: Hierarchy of ATC Communications Specifications

module *ATCproc*

DECLARATIONS

extends $Naturals, Sequences$

CONSTANTS $ChannelSize, StorageSize, Messages, \bot_{Messages}, msg$

VARIABLES $storage, channel$

$ATCtx \triangleq$ **instance** $ATCtrans$

ASSUMPTIONS

ASSUME $ChannelSize > 0$

ASSUME $\bot_{Messages} \notin Messages$

ASSUME $msg \in Messages \Rightarrow \exists fid, fdata : msg = \langle fid, fdata \rangle$

PREDICATES

$$\begin{aligned} Init \triangleq\ & \land\ channel = \langle\,\rangle \\ & \land\ storage = \langle\,\rangle \end{aligned}$$

ACTIONS

$$\begin{aligned} Upload(fid) \triangleq\ \exists\,!msg \in storage :\ & \land\ msg[1] = fid \\ & \land\ storage' = storage \setminus \{msg\} \\ & \land\ ATCtx.Send(msg) \end{aligned}$$

$$\begin{aligned} Download \triangleq\ & \land\ |storage| < StorageSize \\ & \land\ storage' = storage \cup \{Head(channel)\} \\ & \land\ ATCtx.Receive \end{aligned}$$

DEFINITION

$$\begin{aligned} SomeAction \triangleq\ & \lor\ \exists\, msg : Upload(msg) \\ & \lor\ Download \end{aligned}$$

$$\begin{aligned} Safety \triangleq\ & \land\ Init \\ & \land\ \Box[SomeAction]_{storage, channel} \end{aligned}$$

Figure 16.3: Module *ATCproc*

module *ATCcomm*

DECLARATIONS

extends *Naturals*, *Sequences*

CONSTANTS $ChannelSize, StorageSize, Messages, \bot_{Messages}, msg$

VARIABLES $x, y, x\_ATCdata, y\_ATCdata, storage, channel$

$X\_ATC \triangleq$ **instance** *ATCproc* **with** $storage \leftarrow x\_ATCdata$

$Y\_ATC \triangleq$ **instance** *ATCproc* **with** $storage \leftarrow y\_ATCdata$

$Vars = \langle x, y, x\_ATCdata, y\_ATCdata, channel \rangle$

ASSUMPTIONS

ASSUME $ChannelSize > 0$

ASSUME $\bot_{Messages} \notin Messages$

ASSUME $msg \in Messages \Rightarrow \exists fid, fdata : msg = \langle fid, fdata \rangle$

ASSUME $x, y \in \{SATC, BATC, MATC, LATC, ...\}$

ASSUME $\exists ! msg \in x\_ATCdata : msg[1] = fid$

PREDICATES

$$Init \triangleq \begin{array}{l} \land\ fid \in valid\_fids \\ \land\ channel = \langle\,\rangle \\ \land\ x\_ATCdata = \langle\,\rangle \\ \land\ y\_ATCdata = \langle\,\rangle \end{array}$$

ACTIONS

$$Handoff(x, y) \triangleq \begin{array}{l} \land\ X\_ATC.Upload(fid) \\ \land\ Y\_ATC.Download \end{array}$$

DEFINITION

$$Safety \triangleq \begin{array}{l} \land\ Init \\ \land\ \Box[Handoff(x, y)]_{Vars} \end{array}$$

Figure 16.4: Module *ATCcomm*

module *ATCcomm_history*

DECLARATIONS

extends $\mathit{Naturals}, \mathit{Sequences}, \mathit{ATCcomm}$

VARIABLE $\mathit{persistent\_data}$

CONSTANTS $\mathit{dest}, \mathit{xdests}, \mathit{ydests}$

ASSUMPTIONS

ASSUME $\mathit{persistent\_data}[\mathit{destination}] = \mathit{dest}$

ASSUME $\mathit{xdests} \cap \mathit{ydests} \neq \emptyset$

PREDICATES

$\mathit{Init} \triangleq \mathit{persistent\_data} = x\_\mathit{ATCdata}$

ACTIONS

$$\begin{aligned}\mathit{Handoff}_{\mathit{correct}}(x, y) \triangleq\ & \land\ \mathit{persistent\_data}[\mathit{destination}] \in \mathit{xdests}\\ & \land\ \mathit{Handoff}(x, y)\\ & \land\ \text{UNCHANGED } \mathit{persistent\_data}\end{aligned}$$

$$\begin{aligned}\mathit{Handoff}_{\mathit{incorrect}}(x, y) \triangleq\ & \land\ \mathit{persistent\_data}[\mathit{destination}] \in \mathit{ydests}\\ & \land\ \mathit{Handoff}(x, y)\\ & \land\ \mathit{persistent\_data}' \neq \mathit{persistent\_data}\end{aligned}$$

DEFINITION

$$\begin{aligned}\mathit{SomeAction} \triangleq\ & \lor\ \mathit{Handoff}_{\mathit{correct}}(x, y)\\ & \lor\ \mathit{Handoff}_{\mathit{incorrect}}(x, y)\end{aligned}$$

$$\begin{aligned}\mathit{Safety} \triangleq\ & \land\ \mathit{Init}\\ & \land\ \Box[\mathit{SomeAction}]_{\mathit{persistent\_data}}\end{aligned}$$

Figure 16.5: Module *ATCcomm_history*

module *This_ATCcomm_history (Part1)*

DECLARATIONS

```
extends $Naturals, Sequences$
extends $ATCcomm\_history$
CONSTANTS $ChannelSize, StorageSize, Messages, \bot_{Messages}, msg$
VARIABLES $x, y, storage, channel, SATCdata, LATCdata, MATCdata$
$SL\_ATCcommhist \triangleq$ instance $ATCcomm\_history$ with
    $x\_ATCdata \leftarrow SATCdata, y\_ATCdata \leftarrow LATCdata,$
    $dest \leftarrow$ “FRA”, $persistent\_data \leftarrow NW052\_data$
    $xdests \leftarrow \{$“FRA”, “BRU”, ...$\}, ydests \leftarrow \{$“FRA”, “BRU”, “AMS”, ...$\}$
$LB\_ATCcommhist \triangleq$ instance $ATCcomm\_history$ with
    $x\_ATCdata \leftarrow LATCdata, y\_ATCdata \leftarrow BATCdata,$
    $dest \leftarrow$ “FRA”, $persistent\_data \leftarrow NW052\_data$
    $xdests \leftarrow \{$“BRU”, “AMS”, ...$\}, ydests \leftarrow \{$“BRU”, “AMS”, ...$\}$
$LM\_ATCcommhist \triangleq$ instance $ATCcomm\_history$ with
    $x\_ATCdata \leftarrow LATCdata, y\_ATCdata \leftarrow MATCdata,$
    $dest \leftarrow$ “FRA”, $persistent\_data \leftarrow NW052\_data$
    $xdests \leftarrow \{$“FRA”, ...$\}, ydests \leftarrow \{$“FRA”, “TRN”, ...$\}$
```

ASSUMPTIONS

```
ASSUME $ChannelSize > 0$
ASSUME $\bot_{Messages} \notin Messages$
ASSUME $msg \in Messages \Rightarrow \exists fid, fdata : msg = \langle fid, fdata \rangle$
ASSUME $x, y \in \{SATC, BATC, MATC, LATC, ...\}$
ASSUME $\exists! msg \in x\_ATCdata : msg[1] = fid$
ASSUME “FRA” $\notin LB\_ATCcommhist.ydests$
ASSUME “BRU” $\notin LM\_ATCcommhist.ydests$
```

PREDICATES

```
$Init \triangleq \land persistent\_data = SATCdata$
         $\land SATCdata = \langle \rangle$
         $\land LATCdata = \langle \rangle$
         $\land MATCdata = \langle \rangle$
```

Figure 16.6: Module *This_ATCcomm_history (Part 1)*

module *This_ATCcomm_history (continued)*

ACTIONS

$$
\begin{aligned}
StoL_{correct} &\triangleq SL\_ATCcommhist.Handoff_{correct}(SATC, LATC)\\
StoL_{incorrect} &\triangleq SL\_ATCcommhist.Handoff_{incorrect}(SATC, LATC)\\
LtoB_{correct} &\triangleq LB\_ATCcommhist.Handoff_{correct}(LATC, BATC)\\
LtoB_{incorrect} &\triangleq LB\_ATCcommhist.Handoff_{incorrect}(LATC, BATC)\\
LtoM_{correct} &\triangleq LM\_ATCcommhist.Handoff_{correct}(LATC, MATC)\\
LtoM_{incorrect} &\triangleq LM\_ATCcommhist.Handoff_{incorrect}(LATC, MATC)
\end{aligned}
$$

DEFINITION

$$
\begin{aligned}
SomeAction \triangleq\ & \lor\ StoL_{correct}\\
& \lor\ StoL_{incorrect}\\
& \lor\ LtoB_{correct}\\
& \lor\ LtoB_{incorrect}\\
& \lor\ LtoM_{correct}\\
& \lor\ LtoM_{incorrect}\\
Safety \triangleq\ & \land\ Init\\
& \land\ \Box[SomeAction]_{persistent\_data}
\end{aligned}
$$

Figure 16.7: Module *This_ATCcomm_history (Part 2)*