Abstract

ASTRAL is a formal specification language for realtime systems. It is intended to support formal software development, and therefore has been formally defined. This paper focuses on formally proving the mathematical correctness of ASTRAL specifications.

ASTRAL is provided with structuring mechanisms that allow one to build modularized specifications of complex systems with layering. In this paper, we exploit and enhance ASTRAL's structure and provide a proof method that allows one to build well structured proofs.

Correctness proofs in ASTRAL can be divided into two categories: inter-level proofs and intra-level proofs. The former deal with proving that the specification of level i+1 is consistent with the specification of level i, while the latter deal with proving that the specification of level i is correct. In this paper we concentrate on intra-level proofs.

Key Words: Formal Methods, Formal specification and verification, Realtime systems, Timing requirements, State machines, ASLAN, TRIO.
1. Introduction

ASTRAL is a formal specification language for realtime systems. It is intended to support formal software development, and therefore has been formally defined. [GK 91a] discusses the rational of ASTRAL's design and demonstrates how the language builds on previous language experiments. [GK 91b] discusses how ASTRAL's semantics are specified in terms of TRIO, which is a formal realtime logic. It also outlines how ASTRAL specifications can be formally analyzed by translating them into TRIO and then using the TRIO validation theory.

Recently, a number of approaches have been proposed to build formal proofs for real-time systems [Ost 89, ACD 90, CHS 90, Suz 90, FMM 91, GF 91]. Many of them exploit the so called "dual language approach" [Pnu 77, Ost 89] where a system is modeled as an abstract machine (e.g., a finite state machine or a Petri net) and its properties are described through some assertion language (e.g., a logic or an algebraic language). However, they are based on low level formalisms, i.e., abstract machines and/or assertion languages that are not provided with modularization and abstraction mechanisms. As a consequence, the proofs lack structure, which makes them unsuitable for dealing with complex real-life systems.

On the contrary, ASTRAL is provided with structuring mechanisms that allow one to build modularized specifications of complex systems with layering [GK 91a, GK 91b]. In this report further details of the ASTRAL environment components and the critical requirements components, which were not fully developed in previous papers, are presented.

Formal proofs in ASTRAL can be divided into two categories: inter-level proofs and intra-level proofs. The former deal with proving that the specification of level i+1 is consistent with the specification of level i, while the latter deal with proving that the specification of level i is consistent and satisfies the stated critical requirements. This report concentrates on intra-level proofs.

In the next section a brief overview of ASTRAL is presented along with an example system, which will be used throughout the remainder of the report for illustrating specific features of ASTRAL. Section 3 discusses how to represent assumptions about the environment in which the system is to run as well as the representation of critical requirements for the system. Section 4 presents a formal framework for generating proof obligations in ASTRAL. Finally, in section 5 some conclusions from this research are presented and possible future directions are proposed.

2. Overview of ASTRAL

ASTRAL uses a state machine process model and has types, variables, constants, transitions, and invariants. A realtime system is modeled by a
collection of state machine specifications and a single global specification. Each state machine specification represents a process type of which there may be multiple instances in the system. The process being specified is thought of as being in various states with one state differentiated from another by the values of the state variables. The values of these variables evolve only via well defined state transitions, which are specified with Entry and Exit assertions and have an explicit nonnull duration. State variables and transitions may be explicitly exported by a process. This makes the variable values readable by other processes and the transitions callable by the external environment; exported transitions cannot be called by another process. Interprocess communication is via the exported variables, and is accomplished by inquiring about the value of an exported variable for a particular instance of the process. A process can inquire about the value of any exported variable of a process type or about the start or end time of an exported transition.

The ASTRAL computation model views the values of all variables being modified by a transition as being changed by the transition in a single atomic action that occurs when the transition completes execution. Thus, if a process is inquiring about the value of an exported variable while a transition is being executed by the process being queried, the value obtained is the value the variable had when the transition commenced. Start(Op_i, t) is a predicate that is true if and only if transition Op_i starts at time t and there is no other time after t and before the current time when Op_i starts (i.e., t is the time of the last occurrence of Op_i). For simplicity, the functional notation Start(Op_i) is adopted as a shorthand for "time t such that Start(Op_i, t)"; whenever the quantification of the variable t (whether existential or universal) is clear from the context. Start-k(Op_i) is used to give the start time of the kth previous occurrence of Op_i. Inquiries about the end time of a transition Op_i may be specified similarly using End(Op_i) and End-k(Op_i).

In ASTRAL a special variable called Now is used to denote the current time. The value of Now is zero at system initialization time. ASTRAL specifications can refer to the current time ("Now") or to an absolute value for time that must be less than or equal to the current time. That is, in ASTRAL one cannot express values of time that are to occur in the future. To specify the value that an exported variable var had at time t, ASTRAL provides a past(var, t) function. The past function can also be used with the Start and End predicates. For example the expression "past(Start(Op), t) = t" is used to specify that transition Op started at time t.

The type ID is one of the primitive types of ASTRAL. Every instance of a process type has a unique id. An instance can refer to its own id by using "Self". There is also an ASTRAL specification function IDTYPE, which returns the type of the process that is associated with the id.

For inquiries where there is more than one instance of that type, the inquiry is preceded by the unique id of the desired instance, followed by a period. Process instance ids that are used in a process specification must be explicitly imported. For example, i.Start(Op) gives the last start time that
transition Op was executed by the process instance whose unique id is i. The exception to this occurs when the process instance performing the inquiry is the same as the instance being queried. In this case the preceding id and period may be dropped.

An ASTRAL global specification contains declarations for all of the process instances that comprise the system and for any constants or nonprimitive types that are shared by more than one process type. Globally declared types and constants must be explicitly imported by a process type specification that requires them.

The computation model for ASTRAL is based on nondeterministic state machines and assumes maximal parallelism, noninterruptable and nonoverlapping transitions in a single process instance, and implicit one-to-many (multicast) message passing communication, which is instantaneous.

Critical requirements for the system being designed are represented as invariants and schedules in an ASTRAL specification. Global invariants and global scheduling constraints are part of the global specification. The global invariants represent properties that need to be proved about the realtime system as a whole. The global scheduling requirements should specify the ordering of transitions from different process instances and the time between executions of different transitions. Process invariants represent properties that must hold for each process instance, and process schedules represent the scheduling requirements that must be satisfied for each process instance.

ASTRAL also allows assumptions about the external environment to be specified in an environment clause and assumptions about the system context in which a process is to run to be specified in an imported variable clause. The optional environment clause describes the pattern of invocation of external transitions; it is a time-invariant formula (i.e., it must be true at any point of time). If Op_i is an exported transition, Call(Op_i) may be used in the environment clause to denote the time of the last occurrence of the call to Op_i (with the same syntactic conventions as Start(Op_i) and End(Op_i)), and Call-k(Op_i) denotes the time of the kth previous occurrence of the call. The imported variable clause describes patterns of value changes to imported variables, including timing information about any transitions exported by other processes that may be used by the process being specified (e.g., Start(Op_i) and End(Op_i)).

A detailed description of ASTRAL and of its underlying motivations is provided in [GK 91a], which also contains a complete specification of a phone system example. In this report only the concepts of ASTRAL that are needed to present the proof theory are discussed in detail. These concepts are illustrated via a simple example that is a variation of the packet assembler described in [Zav 87]:

---

1Note that there may be a delay from the time a transition Op_i is called until it is actually started.
"The system contains an object which assembles data items (in the order in which it receives them) into fixed-size packets, and sends these packets to the environment. It also contains a fixed number of other objects, each of which receives data items from the environment on a particular channel and sends those items to the packet maker. The packet maker sends a packet to the environment as soon as it is full of data items.

Each data receiver attaches a channel identifier to each incoming data item; these channel identifiers are included with the data items in the outgoing packets.

If a data receiver does not receive a new data item within a fixed time since the last item arrived, its channel is considered closed until the next data item arrives. Notifications of channel closings are put into the outgoing packets as well as data items. If all channels are closed then the packet maker should send an incomplete packet to the environment rather than wait for data to complete it."

In the remainder of this report we refer to this system as the CCITT system.

The appendix contains a complete ASTRAL specification of the CCITT system. It consists of a packet maker process specification, an input process specification (of which there are N instances), and the global specification. The input process specification, which corresponds to the data receiver in Zave's system description, contains two transitions: New_Info and Notify_Timeout. 2.

New_Info is an exported transition, with no Entry conditions, that models the receipt of data items from the environment. It has duration N_I_Dur.

TRANSITION New_Info(x:Info)  N_I_Dur
  EXIT
    Msg[Data_Part] = x
    & Msg[Count] = Msg[Count]' + 1
    & Msg[ID_Part] = Self
    & ~Channel_Closed

In ASTRAL Exit assertions, variable names followed by a ' indicate the value that the variable had when the transition fired.

Notify_Timeout is used to notify the packet maker process that an input process has not received any input from the environment within some fixed time (constant Input_Tout in the specification). It has duration N_T_Dur.

2An earlier version of this specification that did not take into account the environment and with different invariants and schedules was presented in [GK 91b].
The packet maker specification also has two transitions: Process_Msg and Deliver, which correspond to processing a message from an input channel and delivering a packet, respectively.

TRANSITION Process_Msg(R_id:Receiver_ID)  P_M_Dur  
ENTRY  
   LIST_LEN(Packet) < Maximum  
|  ( EXISTS t1: Time (Receiver[R_id].End(New_Info) = t1 & t1 > Previous(R_id))  
|  |  Receiver[R_id].Msg[Data_Part]=Closed  
|  &  past(Receiver[R_id].Msg[Data_Part],Previous(R_id)) ≠ Closed )  
EXIT  
   Packet = Packet' CONCAT LIST(Receiver[R_id].Msg)  
&  Previous(R_id) BECOMES Now  

TRANSITION Deliver  Del_Dur  
ENTRY  
   LIST_LEN(Packet) = Maximum  
|  ( LIST_LEN(Packet) > 0  
|  &  ( EXISTS t:Time (Start(Deliver, t) &  Now - t ≥ Del_Tout)  
|  |  Now = Del_Tout - Del_Dur + N_I_Dur))  
EXIT  
   Output = Packet'  
&  Packet = EMPTY  

3. Environmental Assumptions and Critical Requirements

In addition to specifying system state (through process variables and constants) and system evolution (through transitions), an ASTRAL specification also defines desired system properties and assumptions on the behavior of the environment that interacts with the system. Assumptions about the behavior of the environment are expressed in environment clauses and imported variable clauses, and desired system properties are expressed through invariants and schedules. Because these components are critical to the ASTRAL proof theory and were not fully developed in previous reports, they are discussed in more detail in this section.

3.1 Environment Clauses

An environment clause formalizes the assumptions that must always hold on the behavior of the environment to guarantee some desired system
properties. They are expressed as first-order formulas involving the calls of the exported transitions, which are denoted Call(Opi) (with the same syntactic conventions as for Start(Opi)). For each process p there is a local environment clause, Env_p, which expresses the assumptions about calls to the exported transitions of process p. There is also a global environment clause, Env_G, which is a formula that may refer to all exported transitions in the system.

In the CCITT example there is a local environment clause for the input process and a global clause. The local clause states that for each input process, the time between two consecutive calls to transition New_Info is not less than the duration of New_Info and that there will always be a call to New_Info before the time-out expires:

\[(\exists t : \text{Time}\ (\text{Call-2 (New\_Info, t)}) \Rightarrow \not\exists t : \text{Time}\ (\text{Call (New\_Info)}) - \text{Call-2 (New\_Info)} \geq N\_I\_Dur)\]
& \ (\text{Now} \geq \text{Input\_Tout} \rightarrow \exists t : \text{Time}\ (\text{Call (New\_Info, t)})
& \ (\text{Now} - \text{Call (New\_Info)} < \text{Input\_Tout})

The global environment clause states that exactly N/L calls to transition New_Info are cyclically produced, with time period N/L*P_M_Dur + Del_Dur (where P_M_Dur is the duration of transition Process_Message, Del_Dur is the duration of Deliver, and L denotes a constant that is used to specify that N/L processes are producing messages):

\[\forall t : \text{Time}\ (t \mod (N/L*P\_M\_Dur + \text{Del\_Dur}) = 0 \rightarrow \exists S : \text{Set\_Of\_Receiver}\]
& \ (|S| = N/L
& \ \forall i : \text{Receiver\_ID}\ (i \in S \leftrightarrow \text{Receiver}[i].\text{Call}(\text{New\_Info}) = t))
& \ \forall t : \text{Time}\ (t \mod (N/L*P\_M\_Dur + \text{Del\_Dur}) \neq 0 \rightarrow \forall i : \text{Receiver\_ID}\ (\sim \text{Receiver}[i].\text{Call}(\text{New\_Info}, t)))

3.2 Imported Variable Clauses

Each process p may also have an imported variable clause, IV_p. This clause formalizes assumptions that process p makes about the context provided by the other processes in the system. For example IV_p contains assumptions about the timing of transitions exported by other processes that p uses to synchronize the timing of its transitions. It also contains assumptions about when variables exported by other processes change value. For instance, p might assume that some imported variable changes no more frequently than every 10 time units.

\[3\) For simplicity, the traditional cardinality operator, | |, is used even though it is not an ASTRAL operator.
In the CCITT example only the Packet_Maker process has an imported variable clause. It states that the ends of transition New_Info executed by input processes follow the same periodic behavior as the corresponding calls. The clause is similar to the global environment clause.

3.3 Invariant Clauses

Invariants state properties that must initially be true and must be guaranteed during system evolution, according to the traditional meaning of the term. Invariants can be either local to some process or global. These properties must be true regardless of the environment or the context in which the process or system is running. \( I_p \) denotes the local invariant of process \( p \) and \( I_G \) denotes the global invariant.

\( I_p \) is a formula that can refer to local variables and local transitions (i.e., \( \text{Start}(\text{Op}_i) \), \( \text{End}(\text{Op}_i) \)) of process \( p \). It can not refer to any \( \text{Call}(\text{Op}_i) \), even if transition \( \text{Op}_i \) belongs to process \( p \). In addition, \( I_p \) can refer to imported variables, provided that they occur within a past construct in such a way that the meaning of the construct is the value of the imported variable in an absolute time instant (e.g., a constant time).

\( I_G \) can refer to all variables and transitions exported by processes (we call such transitions and variables global.)

In the CCITT example the global invariant consists of two clauses. The first clause states that every input data will be output within H1 time units after it is input, but not sooner than H2 time units:

\[
\text{FORALL } i: \text{Receiver}_\text{ID}, t_1: \text{Time}, x: \text{Info} \\
(t_1 \leq \text{Now}-H1 & \text{past(Receiver}[i].\text{End(New_Info}(x)),t_1) = t_1 \\
\rightarrow \text{EXISTS } t_2: \text{Time}, k: \text{Integer} \\
( t_2 \geq t_1+H2 & t_2 \leq \text{Now} & \text{Change(Output},t_2) \\
& 0 < k & k \leq \text{LIST}_\text{LEN(past(Output},t_2)) \\
& \text{past(Output}[k][\text{Data}_\text{Part}],t_2) = \text{past(Receiver}[i].\text{Msg}[\text{Count}],t_1) \\
& \text{past(Output}[k][\text{ID}_\text{part}],t_2) = \text{Receiver}[i].\text{Id})
\]

The other global clause states that no message is output other than those produced by the input processes.

The Input process local invariant states that after Input_Tout time units have elapsed without receiving any new message a time-out occurs, and that the last message received is kept until a Deliver time-out occurs.

The Packet_Maker's local invariant, in contrast, is a more complex formula that states the following properties:

11. Changes in Output occur at, and only at, the end of a Deliver:

\[
\text{FORALL } t: \text{Time} (\text{Change(Output}, t) \leftrightarrow \text{past(End(Deliver), t) = t})
\]

12. No new messages are generated by the packet assembler:
FOR ALL \( k : \text{integer} \) (\( k > 0 \) \& \( k \leq \text{LIST\_LEN}(\text{Output}) \)
\[ \rightarrow \text{EXISTS} \; i : \text{Receiver\_ID}, \; t : \text{Time} \] (\( t < \text{Now} \) \& \( \text{past}(\text{Receiver}[i].\text{Msg}, t) = \text{Output}[k] \))

13. The order that messages appear in an output packet is the order in which they were processed from a channel:
FOR ALL \( k : \text{integer} \) (\( k > 0 \) \& \( k < \text{LIST\_LEN}(\text{Output}) \)
\[ \rightarrow \text{EXISTS} \; t_1, t_2 : \text{Time} \] 
(\( t_1 < t_2 < \text{Now} \) \& \( \text{past}(\text{End}(\text{Process\_Message}), t_1) = t_1 \) \& \( \text{past}(\text{End}(\text{Process\_Message}), t_2) = t_2 \) \& \( \text{Output}[k] = \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t_1)], t_1) \) \& \( \text{Output}[k+1] = \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t_2)], t_2) \))

14. The order is also preserved across output packets:
EXIST \( t : \text{Time} \) (\( \text{Start}-2(\text{Deliver}, t) \) \& \( \text{End}(\text{Deliver}) > \text{Start}(\text{Deliver}) \)
\[ \rightarrow \text{EXISTS} \; t_1, t_2 : \text{Time} \] 
(\( t_1 < t_2 < \text{Now} \) \& \( \text{past}(\text{End}(\text{Process\_Message}), t_1) = t_1 \) \& \( \text{past}(\text{End}(\text{Process\_Message}), t_2) = t_2 \) \& \( \text{past}(\text{Output}[\text{past}(\text{LIST\_LEN}(\text{Output}), \text{Start}(\text{Deliver})), \text{Start}(\text{Deliver})) = \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t_1)], t_1) \) \& \( \text{Output}[1] = \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t_2)], t_2) \))

15. Every message in Output was previously in Packet and if Output changes Now then all of the elements of Packet have not changed from when they were put into the packet until now:
FOR ALL \( k : \text{integer} \)
(\( k > 0 \) \& \( k \leq \text{LIST\_LEN}(\text{Output}) \)
\[ \leftrightarrow \text{EXISTS} \; t : \text{Time} \] 
(\( t \leq \text{End}(\text{Deliver}) \) \& \( \text{past}(\text{End}(\text{Process\_Message}), t) = t \) \& \( \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t)], t) = \text{Output}[k] \) \& \( \text{FOR ALL} \; t_1 : \text{Time} \) (\( t_1 \geq t \) \& \( t_1 < \text{End}(\text{Deliver}) \)
\[ \rightarrow \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t_1)], t) = \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t)], t_1)) \))

16. FOR ALL \( t_1 : \text{Time} \) (\( t_1 \leq \text{Now} - \text{H3} \) \& \( \text{past}(\text{End}(\text{Process\_Msg}), t_1) = t_1 \)
\[ \rightarrow \text{EXISTS} \; t_2 : \text{Time} \] 
(\( t_2 > t_1 \) \& \( \text{Now} \geq t_2 \) \& \( \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t_2)], t_2) = \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t)], t_1) \) \& \( \text{FOR ALL} \; t : \text{Time} \) (\( t \geq t_1 \) \& \( t < t_2 \)
\[ \rightarrow \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t_1)], t) = \text{past}(\text{Packet}[\text{past}(\text{LIST\_LEN}(\text{Packet}), t)], t_1)) \))

### 3.4 Schedule Clauses

Schedules are additional system properties that are required to hold under more restrictive hypothesis than invariants. As mentioned above, invariants must hold for any system instantiation that satisfies the given transition specification, irrespective of the behavior of the external
environment. Unlike invariants, the validity of a schedule may be proved using assumptions expressed in the associated environment and/or imported variable clauses.

Like invariants, schedules may be either local or global and obey suitable scope rules in the same style as invariants. Unlike invariants, however, they may refer to calls to exported transitions, since they are more concerned with the interaction between the system and the environment. The schedule for process $p$ is denoted $Sc_p$ and the global schedule is denoted $Sc_G$. Typically, a schedule clause states properties about the reaction time of the system to external stimuli and on the number of requests that can be "served" by the system. This motivates the term "schedule".

Because there may be several ways to assure that a schedule is satisfied, such as giving one transition priority over another or making additional assumptions about the environment, and because this kind of decision should often be postponed until a more detailed design phase, in ASTRAL the schedules are not required to be proved. It is important, however, to know that the schedule is feasible. That is, it is important to know that if further restrictions are placed on the specification and/or if further assumptions are made about the environment, then the schedule can be met. For this reason, a further assumptions and restrictions clause may be included as part of a process specification. Unlike other components of the ASTRAL specification this clause is only used as guidance to the implementer; it is not a hard requirement. The details of this clause are given in the next subsection.

In the CCITT example the global schedule states that the time that elapses between the call of a New Info transition and the delivery of the message it produced is equal to $N/L*P_M_Dur + N_I_Dur + Del_Dur$:

$$\forall i:Receiver_ID, t_1:Time, x:Info$$

$$(t_1 \leq \text{Now} - N/L*P_M_Dur - N_I_Dur - Del_Dur$$

$\& \text{past}(\text{Receiver}[i].\text{Call}(\text{New}_{\text{Info}}(x)), t_1) = t_1$$

$\rightarrow \exists t_2:Time, k:Integer$

$$(t_2 = t_1 + N/L*P_M_Dur + N_I_Dur + Del_Dur$$

$\& 0 < k \leq \text{LIST}_\text{LEN}(\text{past}(\text{Output}, t_2))$$

$\& \text{Change}(@\text{Output}, t_2) \& \text{past}(\text{Output}[k][\text{Data}_\text{Part}], t_2) = x$$

$\& \text{past}(\text{Output}[k][\text{ID}_\text{Part}], t_2) = \text{Receiver}[i].\text{Id})$$

The local schedule for the Input process states that there is no delay between a call of New Info and the start of its execution.

The Packet Maker's schedule states that the transition Deliver is executed cyclically and that a packet is always delivered with $N/L$ elements:

$$\exists t:Time \quad (\text{End-2}(\text{Deliver}, t))$$

$$\rightarrow \text{End}(\text{Deliver}) - \text{End-2}(\text{Deliver}) = N/L*P_M_Dur + Del_Dur$$

$$\& \forall t: Time \quad (\text{past}(\text{End}(\text{Deliver}), t) = t$$

$$\rightarrow \text{LIST}_\text{LEN}(\text{past}(\text{Output}, t)) = N/L)$$

10
3.5 Further Assumptions and Restrictions Clause

As mentioned before, schedules can be guaranteed by exploiting further assumptions about the environment or restrictions on the system behavior. Such assumptions constitute a separate part of the process specification, the further assumptions and restrictions clause, $\text{FAR}_p$. For reasons that will be clear later, this clause is only local to processes. It consists of two parts: a further environment assumptions section and a further process assumptions section.

The further environment assumptions section, $\text{FEnv}_p$, obeys the same syntactic rules as $\text{Env}_p$. It simply states further hypotheses on—and, therefore, further restricts—the admissible behaviors of the environment interacting with the system. Of course, it cannot contradict previous general assumptions on the environment expressed in $\text{Env}_p$ and $\text{Env}_G$.

A further process assumptions section, $\text{FPA}_p$ restricts the possible system implementations by specifying suitable selection policies in the case of nondeterministic choice between several enabled transitions or by further restricting constants. In general, $\text{FPA}_p$ reduces the level of nondeterminism of the system specification\(^4\). The two parts that comprise the further environment assumptions section are a transition selection part, $\text{TS}_p$, and a constant refinement part, $\text{CR}_p$.

The transition selection part consists of a sequence of clauses of the following type:

$$\{\text{OpSet}_i\} <\text{Boolean Condition}_i> \{\text{ROpSet}_i\}$$

where

- $\{\text{OpSet}_i\}$ defines a set of transitions. For convenience, suitable short hand notations are provided to group the definition of several sets into a single formula.
- $\{\text{ROpSet}_i\}$ defines a restricted but nonempty set of transitions that must be a subset of the set defined by $\{\text{OpSet}_i\}$\(^5\).
- $<\text{Boolean Condition}_i>$ is a boolean condition on the state of process $p$.

\(^4\)This remark supplies a first explanation of the fact that $\text{FEnv}_k$ are exclusively local. In fact, because of the maximal parallelism assumption, there is never a need to select between several processes that are eligible for execution.

\(^5\)Presently, $\{\text{OpSet}_i\}$ and $\{\text{ROpSet}_i\}$ use static notations; i.e., they must define sets or set classes whose elements can be computed at "compile time". More dynamic computations are needed, e.g., by using variables denoting transition indexes, but this is left for future developments.
The operational semantics of the transition selection part is defined as follows.

1. At any given time the set of enabled transitions, \(\{ET\}\), is evaluated by the process abstract machine.

2. Let \(\{OpSet_i\}, <\text{Boolean Condition}_i>\) be a pair such that ET is \(\{OpSet_i\}\) and \(<\text{Boolean Condition}_i>\) holds. Notice that such a pair does not necessarily exist.

3. If there are pairs that satisfy condition 2, then the set of transitions that actually are eligible for firing is the union of all \(\{ROpSet_i\}\) corresponding to the above pairs \(\{OpSet_i\}, <\text{Boolean Condition}_i>\) that are satisfied.

4. If no such pair exists, the set of transitions eligible for firing is \(\{ET\}\).

The constant refinement part is a sequence of clauses that may restrict the values that system constants can assume w.r.t. what is stated in the remaining part of system specification. For example, one can further restrict a constant T1 that is bounded between 0 and 100, by stating that T1's value is actually between 10 and 50, or that it is exactly 5.

Notice that the further assumptions and restrictions section can only restrict the set of possible behaviors. That is, if \(\{B\}\) denotes the set of system behaviors that are compatible with the system specification without the FAR clause and \(\{RB\}\) denotes the set of behaviors that are compatible with the system specification including the FAR clause, then it is easy to verify that \(\{RB\}\) is contained in \(\{B\}\).

For the CCITT system two different further assumptions clauses were used with the Packet_Maker process. The first contains both a constant refinement part and a transition selection part. The CR part states that the time-out of transition Deliver is 0 and that the packet length is equal to \(N/L\).

\[\text{Del\_Tout} = 0 \& \text{Maximum} = N/L\]

The TS part states that the Process_Message transition has higher priority than Deliver.

\[\{\text{Process\_Message, Deliver}\} \text{ TRUE} \ {\text{Process\_Message}}\]

The second further assumptions clause contains only a constant refinement part, which states that Deliver's time-out is \(N/L \cdot P\_M\_Dur + \text{Del\_Dur}\) and that Maximum = \(N\).

Either of these further assumptions clauses is sufficient to prove that the schedules are met.

\[\text{\footnote{Notice that the set of clauses TS}_k\text{ can be automatically transformed in such a way that at least one pair satisfying the condition of point 3 always exists and that the semantics of the system coincides with the above description of points 1 through 4. Thus, point 4 has been included only for the user's convenience. It is not semantically necessary.}}\]
4 Intra-level Proof Obligations in ASTRAL

Proof obligations in ASTRAL can be divided into two categories: inter-level proofs and intra-level proofs. The former deal with proving that the specification of level i+1 is consistent with the specification of level i, while the latter deal with proving that the specification of level i is consistent with its critical requirements. In this report only intra-level proofs are presented. However, it is first necessary to present some notation.

Let S denote a top level ASTRAL specification. S is composed of a set of process specifications \( P_p \) and a global specification G. Each \( P_p \), in turn, is composed of a set of transitions \( Op_{p1},...,Op_{pn} \), a local invariant \( I_p \), a local schedule \( Sc_p \), a local environment \( Env_p \), imported variable assumptions \( IV_p \), a further local environment \( FEnv_p \) and a further process assumption \( FPA_p \), and an initial clause \( Init-State_p \). Moreover, every transition \( Op_{pj} \) is described by entry and exit clauses denoted \( EN_{pj} \) and \( EX_{pj} \), respectively. Finally, every further process assumption \( FPA_p \) is composed of two parts: the first, denoted \( CR_p \), describes some further hypothesis on constant values and the second, denoted \( TS_p \), restricts the non-determinism of the abstract machine. The global specification G is made up of a global invariant \( I_G \), a global schedule \( Sc_G \) and a global environment \( Env_G \) clause.

Proving that S satisfies its critical requirements can be partitioned into the following proof obligations:

1) Every process specification \( P_p \) guarantees its local invariant \( I_p \);
2) Every process specification \( P_p \) guarantees its local schedule \( Sc_p \);
3) The specification S guarantees the global invariant \( I_G \);
4) The specification S guarantees the global \( Sc_G \);

For soundness the following proof obligations are also needed:
5) The imported variable assumptions \( IV_p \) are guaranteed by the specification S.
6) All the assumptions about the environment (\( Env_p \), \( FEnv_p \) and \( Env_G \)) are consistent.

In what follows a formal framework for these proof obligations is presented.

4.1 ASTRAL Abstract Machine

\[\text{7 There are also optional EXCEPT/EXIT pairs that specify responses to exception conditions for a transition.}\]
An informal description of the ASTRAL computational model is given in [GK91a, GK91b]. However, a formal description of the ASTRAL abstract machine is needed in order to carry out the ASTRAL proofs.

The semantics of the ASTRAL abstract machine is defined by three axioms. The first axiom states that the time interval spanning from the starting to the ending of a given transition is equal to the specified duration of the transition.

\[ \forall t : \text{Time}, \ Op : \text{Trans}_{of\ p} \quad \text{[A1]} \]
\[
(\text{Now} - t \geq T_{Op} \rightarrow \\
(\text{past}(\text{Start}(Op),t) = t \leftrightarrow \text{past}(\text{End}(Op),t + T_{Op}) = t + T_{Op})),
\]

where \( T_{Op} \) represents the duration of transition \( Op \).

The second axiom states that if a processor is idle and some transitions are enabled then one transition will fire. Let \( S_T \) denote the set of transitions of process \( p \).

\[ \forall t : \text{Time} \ (\exists d : \text{Time}, \ S' \subset S_T : \text{SET OF Trans}_{of\ p} \quad \text{[A2]} \]
\[
(\forall t_1 : \text{Time}, \ Op : \text{Trans}_{of\ p} \\
( t_1 \geq t - d \& t_1 < t \& Op \in S_T \& \text{past}(\text{Start}(Op),t_1) < \text{past}(\text{End}(Op),t) \& \\
S' \subseteq S_T \& S' \neq \text{EMPTY} \& \\
\forall \ Op' : \text{Trans}_{of\ p} (Op' \in S_T \rightarrow \text{Eval}_{\text{Entry}}(Op',t)) \& \\
\forall \ Op' : \text{Trans}_{of\ p} (Op' \notin S_T \rightarrow \neg \text{Eval}_{\text{Entry}}(Op',t)) \\
\rightarrow \text{UNIQUE Op'} : \text{Trans}_{of\ p} (Op' \in S_T \& \text{past}(\text{Start}(Op'),t) = t)),
\]

where \( \text{Eval}_{\text{Entry}}(Op,t) \) is a function that given a transition \( Op \) and a time instant \( t \) evaluates the entry condition \( EN_{Op} \) of transition \( Op \) at time \( t \).

Because the ASTRAL model implies that the starting time of a transition equals the time in which its entry condition was evaluated, the \( \text{Eval}_{\text{Entry}} \) function is introduced to prevent the occurrence of a contradiction. More specifically, when the entry condition of transition \( Op \) refers to the last start (2nd last, etc.) of itself, the evaluation at time \( t \) of \( \text{Start}(Op) \) in the entry condition should refer to the value of \( \text{Start} \) immediately before the execution of \( Op \) at time \( t \). Since \( Op \) has a nonnull duration this can be expressed by evaluating \( \text{Start}(Op) \) at a time \( t' \) which is prior to \( t \) and such that transition \( Op \) has not fired in the interval \([t', t)\).

Finally, the third axiom states that for each processor the transitions are nonoverlapping.

\[ \forall t_1, t_2 : \text{Time}, \ Op : \text{Trans}_{of\ p} \quad \text{[A3]} \]
\[
(\text{Start}(Op) = t_1 \& \text{End}(Op) = t_2 \& t_1 < t_2 \\
\rightarrow \neg (\exists t_3 : \text{Time}, \ Op' : \text{Trans}_{of\ p} (t_3 > t_1 \& t_3 < t_2 \& \text{End}(Op') = t_3)) \& \\
\forall t_3 : \text{Time} \\
( t_3 \geq t_1 \& t_3 < t_2 \\
\& \exists Op' : \text{Trans}_{of\ p} (\text{Start}(Op') = t_3 \rightarrow Op = Op' \& t_3 = t_1)),
\]

\[ 4.2 \quad \text{Local Invariant Proof Obligations} \]
The local invariant $I_p$ represents a property that must hold for every state the process $p$ could be in. Furthermore, the invariant describes properties that are independent from the environment. Therefore, the proof of the invariant $I_p$ may not make use of any assumption about the environment, imported variables or the system behavior as described by $Env_p$, $FEnv_p$, $IV_p$ and $FPAp$.

To prove that the specification of process $p$ ($P_p$) guarantees the local invariant one needs to show that:

1) $I_p$ holds in the initial state of process $p$, and
2) If $P_p$ is in a state in which $I_p$ holds, then for every possible evolution of $P_p$, $I_p$ will hold.

The first proof consists of showing that the following implication is valid:

$$\text{Init State}_p \& \text{Now} = 0 \rightarrow I_p$$

To carry out the second proof one assumes that the invariant $I_p$ holds until a given time $t_0$ and proves that $I_p$ will hold for every time $t > t_0$. Without loss of generality, one can assume that $t$ is equal to $t_0 + \Delta$, for some fixed $\Delta$ greater than zero, and show that the invariant holds until $t_0 + \Delta$.

In order to prove that $I_p$ holds until time $t_0 + \Delta$ it may be necessary to make assumptions on the possible sequences of events that occurred within the interval $[t_0 - H, t_0 + \Delta]$, where $H$ is a constant a priori unbounded, and where by event is meant the starting or ending of some transition $OP_{pj}$ of $P_p$.

Let $\sigma$ denote one such sequence of events. A formula $F_\sigma$ composed of the sequence of events that belong to $\sigma$ can be algorithmically associated with $\sigma$. For each event occurring at time $t$ one has:

- $\text{Eval_Entry}(Ob_{pj}, t) \& \text{past}(\text{Start}(Op_{pj}, t), t)$ if the event is the start of $Op_{pj}$
- $\text{past}(EX_{pj}, t) \& \text{past}(\text{End}(Op_{pj}, t), t)$ if the event is the end of $Op_{pj}$.

Then the prover’s job is to show that for any $\sigma$:

$$\text{A1 \& A2 \& A3} \rightarrow F_\sigma \& \text{FORALL } t: \text{Time } (t \leq t_0 \rightarrow \text{past}(I_p, t))$$
$$\rightarrow \text{FORALL } t_1: \text{Time } (t_1 > t_0 \& t_1 \leq t_0 + \Delta \rightarrow \text{past}(I_p, t_1))$$
Notice that, as a particular case, the implication is trivially true if \( F_\sigma \) is contradictory, since this would mean that \( \sigma \) is not feasible.

A1, A2 and A3 are the axioms defining the semantics of the ASTRAL abstract machine, which were presented in the previous section.

Note that, during the proof of the invariant \( I_P \), one is not allowed to make use of any assumption on the environment, imported variables or the system behavior described by \( \text{Env}_p \), \( \text{FEnv}_p \), \( \text{IV}_p \) and \( \text{FPA}_p \). Due to the similarity of the proof techniques used for proving the local schedule we will not present the proof of a local invariant.

### 4.3 Local Schedule Proof Obligations

As discussed in section 3, the local schedule \( S_{cp} \) of a process \( P_p \) describes some further properties that \( P_p \) must satisfy when the assumptions on the behavior of both the environment and \( P_p \) hold. These assumptions are described by the local environment clause \( \text{Env}_p \), the imported variables assumption \( \text{IV}_p \), the further assumption on the environment \( \text{FEnv}_p \) and the system assumption clause \( \text{FPA}_p \).

To prove that the specification of process \( P (P_p) \) guarantees the local schedule \( S_{cp} \) it is necessary to show that:

1. \( S_{cp} \) holds in the initial state of process \( P \), and
2. If \( P \) is in a state in which \( S_{cp} \) holds, then for every possible evolution of \( P \) compatible with \( \text{FPA}_p \), when the environment behavior is described by \( \text{Env}_p \) and \( \text{FEnv}_p \), and the imported variables behavior is described by \( \text{IV}_p \), \( S_{cp} \) will hold.

Note that one can also assume that the local invariant \( I_P \) holds; i.e., \( I_P \) can be used as a lemma. The initial state proof obligation is similar to the proof obligation for the local invariant case; however, the further hypothesis on the values of some constants expressed by \( \text{CR}_p \) can be used:

\[
\text{Init\_State}_p \land \text{Now} = 0 \land \text{CR}_p \Rightarrow S_{cp}
\]

The second proof obligation is also similar to the local invariant proof. However, in this case events may be external calls of exported transitions \( \text{Op}_{pj} \) in addition to the starting and ending of all transitions of \( P \).

Thus, in this case any formulas \( F_\sigma \) is composed of the sequence of events that belongs to \( \sigma \), and for each event occurring at time \( t \) we have:

\[
\text{past}(\text{EN}_{pj}, t) \land \text{past}(\text{Start}(\text{Op}_{pj}, t) = t) \text{ if the event is the start of } \text{Op}_{pj} \text{ or}
\]

\[
\text{past}(\text{EX}_{pj}, t) \land \text{past}(\text{End}(\text{Op}_{pj}, t) = t) \text{ if the event is the end of } \text{Op}_{pj}.
\]
past(Call(Op_{pj}, t) = t) if the event is the call of Op_{pj} from the external environment.

The prover's job is to show that for any \( \sigma \):

\[
\text{A1} \land \text{A2'} \land \text{A3} \land \text{A4} \land \text{Env}_p \land \text{FEnv}_p \land \text{IV}_p \land \text{CR}_p \land \text{F}_\sigma \land \forall t : \text{Time} (t \leq t_0 \Rightarrow \text{past} (\text{Sc}_p, t)) \\
\rightarrow \forall t_1 : \text{Time} (t_1 > t_0 \land t_1 \leq t_0 + \Delta \Rightarrow \text{past} (\text{Sc}_p, t_1))
\]

where A2'and A4 are defined in what follows:

A2' is an axiom derived from A2 by taking into account the TS\(_p\) section, which restricts the nondeterminism of the machine, and that the exported transitions can fire only if they are called by the environment.

The TS\(_p\) section can be viewed as the definition of a function TS: \( 2^{\{\text{Op}_1, \ldots, \text{Op}_n\}} \rightarrow 2^{\{\text{Op}_1, \ldots, \text{Op}_n\}} \), having as domain and range the powerset of the transitions of process \( P_p \). Its semantics is the following: denoting with ET the set of enabled transitions then TS(ET) returns a restricted set of enabled transitions, ET', where ET' \( \subseteq \) ET. The processor will nondeterministically select which transition to fire from the transitions in ET'.

Let ST denote the set of transition of process p:

\[
\begin{align*}
\forall t : \text{Time} & (\exists d : \text{Time}, S' \subseteq \text{SET OF Trans_of}_p \quad [A2'] \\
& \quad \forall t_1 : \text{Time}, \text{Op} : \text{Trans_of}_p \\
& \quad \left( t_1 \geq t - d \land t_1 < t \land \text{Op} \in S' \land \text{past(Start(\text{Op}), t_1)} < \text{past(End(\text{Op}), t)} \\
& \quad \land \ D \subseteq S' \land S' \neq \text{EMPTY} \\
& \quad \land \forall \text{Op}' : \text{Trans_of}_p \left( \text{Op}' \in S' \land \text{Eval_Entry}'(\text{Op}', t) \right) \\
& \quad \land \forall \text{Op}' : \text{Trans_of}_p \left( \text{Op}' \notin S' \land \neg \text{Eval_Entry}'(\text{Op}', t) \right) \\
& \quad \rightarrow \text{UNIQUE} \text{Op}' : \text{Trans_of}_p \left( \text{Op}' \in \text{TS}(S') \land \text{past(Start(\text{Op'}), t)} \right) \\
\end{align*}
\]

where Eval_Entry'(Op',t) = Eval_Entry(Op',t) \& Issued_call(Op') if Op' is exported, and Eval_Entry'(Op',t) = Eval_Entry(Op',t) if Op' is not exported.

A4 states that Issued_call(Op) is true iff the environment has called transition Op and transition Op has not fired since then:

\[
\begin{align*}
\forall \text{Op} : \text{Trans_of}_p \quad [A4] \\
& \quad \left( \exists t_1 : \text{Time} \\
& \quad \left( t_1 \leq \text{Now} \land \text{Call(\text{Op}, t_1)} \\
& \quad \land \forall t : \text{Time} (t \geq t_1 \land t \leq \text{Now} \land \neg \text{Start(\text{Op}, t)} \\
& \quad \rightarrow \text{past(Issued_call(\text{Op}, t))} \\
& \quad \land \left( \exists t_1 : \text{Time} (t_1 \leq \text{Now} \land \text{Start(\text{Op}, t_1)} \\
& \quad \land \forall t : \text{Time} (t > t_1 \land t \leq \text{Now} \land \neg \text{Call(\text{Op}, t)} \\
& \quad \rightarrow \neg \text{past(Issued_call(\text{Op}, t))} \\
\right)
\end{align*}
\]
Note that in this case the user is allowed to use the further assumptions represented by Envp, IVp, FEnvp and CRp.

**Example**

Consider the local schedule of process Packet_Maker (Scpm):

\[
\begin{align*}
&\text{EXISTS } t: \text{Time} \ (\ \text{End-2(Deliver, } t)) \\
&\quad \rightarrow \text{End(Deliver) - End-2(Deliver) = N/L*P_M_Dur + Del_Dur} \\
&\quad \& \ \text{FORALL } t: \text{Time} \ (\ \text{past(End(Deliver), } t) = t \rightarrow \text{LIST_LEN(past(Output, } t)) = N/L) \\
\end{align*}
\]

To prove Scpm the imported variables assumption IVpm and the second further process assumptions FPApm of process Packet Maker are used:

**IVpm**:

\[
\begin{align*}
&\text{FORALL } t: \text{Time} \ ((t - N_I_Dur) \mod (N/L*P_M_Dur + Del_Dur) = 0 \\
&\quad \rightarrow \text{EXISTS } S: \text{Set_of_Receiver_ID} \\
&\quad \quad ( \ |S| = N/L \\
&\quad \quad \& \ \text{FORALL } i: \text{Receiver_ID} \ (i \in S \leftrightarrow \text{Receiver}[i].\text{End(New_Info)} = t)) \\
&\quad \& \ \text{FORALL } t: \text{Time} \ ((t - N_I_Dur) \mod (N/L*P_M_Dur + Del_Dur) \neq 0 \\
&\quad \quad \rightarrow \text{FORALL } i: \text{Receiver_ID} \ (\neg \text{Receiver}[i].\text{End(New_Info)} = t)) \\
&\quad \& \ \text{FORALL } i: \text{Receiver_ID} \ (\text{Receiver}[i].\text{Msg[Data_Part]} \neq Closed)
\end{align*}
\]

**FPApm**:

\[
\begin{align*}
&\text{Del_Tout} = N/L*P_M_Dur + Del_Dur \\
&\text{Maximum} = N
\end{align*}
\]

Consider a time instant \( p_0 \) such that Scpm holds until \( p_0 \); it is necessary to prove that Scpm holds until \( p_0 + \Delta \), where \( \Delta \) is big enough to require an End(Deliver) to occur within \( (p_0, p_0 + \Delta) \). Without loss of generality, assume that:

1) at time \( p_0 \) transition Deliver ends and

2) \( \Delta = N/L*P_M_Dur + Del_Dur \).

Now, by [A1] one can deduce that at time \( p_0 - Del_Dur \) a Start(Deliver) occurred.

The Entry assertion of Deliver states that Deliver fires either when the buffer is full or when the time-out expires and at least one message has been processed.

**EnDel**:

\[
\begin{align*}
&\text{LIST_LEN(Packet)} = \text{Maximum} \\
&\quad | ( \ \text{LIST_LEN(Packet)>0} \\
&\quad \& \ \text{EXISTS } t: \text{Time} \\
&\quad \quad ( \ \text{Start(Deliver, } t) \ & \text{Now - } t = \text{Del_Tout}) \\
&\quad \ | \ \text{Now = Del_Tout + N_I_Dur - Del_Dur})
\end{align*}
\]
Because Sc\textsubscript{pm} holds until p\textsubscript{0} and from the Exit assertion for Deliver it is known that:

1) For all t less than or equal to p\textsubscript{0} and such that an end of transition Deliver occurred, Output contains N/L messages at time t (Sc\textsubscript{pm}), and

2) The content of Output at the end of Deliver is equal to the content of Packet at the beginning of Deliver (Exit assertion of Deliver).

From this one can conclude that at time t - Del\_Dur the packet contained N/L messages, i.e., it was not full. As a consequence transition Deliver has fired because the time-out has expired.

Furthermore, assume as lemma L1 that Process\_Message is disabled every time Deliver fires (this lemma will be proved later).

The Entry condition of Process\_Message is:

\[
\text{LIST\_LEN(Packet)} < \text{Maximum} \\
\& \ (\text{Receiver[R\textsubscript{id}].End(New\_Info)} > \text{Previous(R\textsubscript{id})} \\
\| (\text{Receiver[R\textsubscript{id}].Msg[Data\_Part]} = \text{Closed} \\
\& \text{past(Receiver[R\textsubscript{id}].Msg[Data\_Part], Previous(R\textsubscript{id})]} ≠ \text{Closed})
\]

and since:

1) the buffer is not full (Sc\textsubscript{pm}), and

2) no notification of closed channel can arrive (IV\textsubscript{pm})

one can conclude that no new message is available when Deliver fires (L1).

IV\textsubscript{pm} states that N/L messages are received every N/L*P\_M\_Dur + Del\_Dur time units. As a consequence:

1) the N/L messages output at time p\textsubscript{0} have been received before time p\textsubscript{0} - Del\_Dur - N/L*P\_M\_Dur, in order to allow Process\_Message to process each of them, and

2) they have been received after the second last occurrence of Delivery prior to p\textsubscript{0} (because of L1).
Thus, one can conclude that the N/L messages output at time \( p_0 \) have been received in the interval:

\[
\text{(Start-2(Deliver), } p_0 - \text{Del}_\text{Dur} - \text{N/L}\times\text{P}\_\text{M}\_\text{Dur}],
\]

i.e., \((p_0 - 2\times\text{Del}_\text{Dur} - \text{N/L}\times\text{P}\_\text{M}\_\text{Dur}, p_0 - \text{Del}_\text{Dur} - \text{N/L}\times\text{P}\_\text{M}\_\text{Dur}]\)

because of \text{Sc}_{pm}.

As a consequence of \text{IV}_{pm}, N/L new messages will arrive after \( N/L\times\text{P}\_\text{M}\_\text{Dur} + \text{Del}_\text{Dur} \) time units from the last arrival, i.e., in the interval \((p_0 - \text{Del}_\text{Dur}, p_0]\).

Thus, at time \( p_0 \) \text{Process\_Message} will become enabled and the N/L messages will be processed within time \( p_0 + N/L\times\text{P}\_\text{M}\_\text{Dur} \), since \text{Deliver} is disabled until that time. Moreover, at time \( p_0 + N/L\times\text{P}\_\text{M}\_\text{Dur} \) \text{Process\_Message} will be disabled, since there are exactly N/L messages to \text{process}.

Thus, at time \( p_0 + N/L\times\text{P}\_\text{M}\_\text{Dur} \) the buffer contains N/L messages and \text{Deliver} fires because the time-out has expired. Also, at time \( p_0 + N/L\times\text{P}\_\text{M}\_\text{Dur} + \text{Del}_\text{Dur} \) \text{Deliver} ends and the length of the Output buffer will be equal to \( N/L \) (Exit clause of \text{Deliver})

Therefore, the schedule will hold until time \( p_0 + N/L\times\text{P}\_\text{M}\_\text{Dur} + \text{Del}_\text{Dur} \).

To complete the proof it is necessary to give an inductive proof of lemma \text{L1}, which states that \text{Process\_Message} is disabled every time \text{Deliver} fires.

Initially, the first time that \text{Deliver} fires, \text{Process\_Message} is disabled.

In fact, the first \( N/L \) \text{End(New\_Info)} occur at time \( N\_I\_\text{Dur} \) (\text{IV}_{pm}).

\text{Transition} \text{Process\_Message} will finish processing these messages at time \( N\_I\_\text{Dur} + N/L\times\text{P}\_\text{M}\_\text{Dur} \), and at that time \text{Deliver} will become enabled.

Since no \text{End(New\_Info)} can occur in \( (N\_I\_\text{Dur}, N\_I\_\text{Dur} + N/L\times\text{P}\_\text{M}\_\text{Dur} + \text{Del}_\text{Dur}) \) (by \text{IV}_{pm}), then at time \( N\_I\_\text{Dur} + N/L\times\text{P}\_\text{M}\_\text{Dur} \) transition \text{Process\_Message} is disabled and \text{Deliver} fires.

Now suppose that when \text{Deliver} fires \text{Process\_Message} is disabled; it is necessary to prove that \text{Process\_Message} is again disabled the next time \text{Deliver} fires.

Let \( q_0 \) be the time when \text{Deliver} starts; by hypothesis at time \( q_0 \) \text{Process\_Message} is disabled. As a consequence the messages in \text{Packet} at time \( q_0 \) have been received in the interval \((q_0 - \text{Del}_\text{Dur} - N/L\times\text{P}\_\text{M}\_\text{Dur}, q_0 - N/L\times\text{P}\_\text{M}\_\text{Dur}] \) (\text{Sc}_{pm}).

Thus, by \text{IV}_{pm} the next N/L messages will arrive in the interval \((q_0, q_0 + \text{Del}_\text{Dur}]\). Furthermore, the time-out for \text{Deliver} will expire at time \( q_0 + N/L\times\text{P}\_\text{M}\_\text{Dur} + \text{Del}_\text{Dur} \). Therefore, \text{Deliver} cannot fire before that time unless the buffer is full.

At time \( q_0 + \text{Del}_\text{Dur} \) \text{Process\_Message} will become enabled, and it will fire until either all messages have been processed or the buffer becomes...
full. At time $q_0 + \text{Del}_\text{Dur} + N/L*P_\text{M}_\text{Dur}$ the N/L messages that arrived in the interval $(q_0, q_0 + \text{Del}_\text{Dur}]$ will be processed and since no new message can arrive before $q_0 + \text{Del}_\text{Dur} + N/L*P_\text{M}_\text{Dur}$ at that time Process_Message will be disabled. Similarly, at that time Deliver will be enabled and thus will fire.

### 4.4 Global Invariant Proof Obligations

Given an ASTRAL specification $S$ composed of $n$ processes, the state of $S$ can be defined as the tuple $<s_1, \ldots, s_n>$, where $s_p$ represents the state of process $P_p$. The global invariant $I_G$ of $S$ describes the properties that must hold in every state of $S$.

To prove that $I_G$ is guaranteed by $S$ it is necessary to prove that:

1) $I_G$ holds in the initial state of $S$, and

2) If $S$ is in a state in which $I_G$ holds, then for every possible evolution of $S$, $I_G$ will hold.

Since the initial state of $S$ is the tuple $<\text{Init}_\text{State}_1, \ldots, \text{Init}_\text{State}_n>$, where each $\text{Init}_\text{State}_p$ represents the initial state of process $P_p$, to prove point 1) one needs to prove the validity of the following logical implication:

$$\forall_{p=1}^n (\text{Init}_\text{State}_p) \& \text{Now} = 0 \rightarrow I_G$$

Point 2 can be proved in a manner very similar to the local invariant case. However in this case the sequences of events $\sigma$ will contain events for exported transitions belonging to any process of $S$. Moreover, the local invariant of each process $P_p$ composing $S$ can be used to prove that every $\sigma$ preserves the global invariant. Thus, in this case any formula $F_\sigma$ is composed of the sequence of events that belong to $\sigma$. For each event occurring at time $t$:

$$\text{Eval}\_\text{Entry}(\text{Obpj}, t) \& \text{past}(\text{Start}(\text{Oppj}, t), t) \quad \text{if the event is the start of } \text{Oppj}$$

or

$$\text{past}(\text{EXpj}, t) \& \text{past}(\text{End}(\text{Oppj}, t), t) \quad \text{if the event is the end of } \text{Oppj}.$$ 

Then the prover's job is to show that for any $\sigma$:

$$A1 \& A2 \& A3 \mid -F_\sigma \& \forall \text{Time } t_0 : (t \leq t_0 \rightarrow \text{past}(I_G, t)) \rightarrow \forall \text{Time } t_1 : (t_1 > t_0 \& t_1 \leq t_0 + \Delta \rightarrow \text{past}(I_G, t_1))$$

Note that during the above proofs the local invariants can be used as lemmas.
Example

Consider only the second part of the global invariant $I_G$:

\[
\begin{align*}
& \text{FORALL } i: \text{Receiver}_I D, t_1: \text{Time}, x: \text{Info} \\
& \quad (t_1 \leq \text{Now} - H_1 & \&\ past(\text{Receiver}[i].\text{End}(\text{New}_\text{Info}(x)), t_1) = t_1) \\
& \quad \rightarrow \ \text{EXISTS } t_2: \text{Time}, k: \text{Integer} \\
& \quad \quad (t_2 \geq t_1 + H_2 & \&\ t_2 \leq \text{Now} \\
& \quad \quad & \&\ k > 0 & \&\ k \leq \text{LIST}_\text{LEN}(\text{past}(\text{Output}, t_2)) \\
& \quad \quad & \&\ \text{Change}(\text{Output}, t_2) & \&\ \text{past}(\text{Output}[k][\text{Data}_\text{Part}], t_2) = x \\
& \quad \quad & \&\ \text{past}(\text{Output}[k][\text{Count}], t_2) = \text{past}(\text{Receiver}[i].\text{Msg}[\text{Count}], t_1) \\
& \quad \quad & \&\ \text{past}(\text{Output}[k][\text{ID}_\text{Part}], t_2) = \text{Receiver}[i].\text{Id})
\end{align*}
\]

We use the local invariant $I_1$ of the process $\text{Packet}_\text{Maker}$ as a lemma.

\[
\begin{align*}
& (I_1) \ \text{FORALL } t: \text{Time} \ (\text{Change}(\text{Output}, t) \leftrightarrow \text{past}(\text{End}(\text{Deliver}), t) = t)
\end{align*}
\]

The above invariant relates the predicate $\text{Chg}(\text{Output}, t)$ to the ending time of transition $\text{Deliver}$. The predicate $\text{Chg}(\text{Output}, t)$ is defined so that it is true iff $\text{Output}$ changes value at time $t$, otherwise it is false.

Thus, by using $I_1$ the invariant can be rewritten as follows:

\[
\begin{align*}
& \text{EXISTS } t_1: \text{Time}, x: \text{Info}, \\
& \quad \quad (t_1 \leq \text{Now} - H_1 & \&\ past(\text{Receiver}[i].\text{End}(\text{New}_\text{Info}(x)), t_1) = t_1) \\
& \quad \quad \rightarrow \ \text{EXISTS } t_2: \text{Time}, k: \text{Integer} \\
& \quad \quad \quad (t_2 \geq t_1 + H_2 & \&\ t_2 \leq \text{Now} \\
& \quad \quad \quad & \&\ k > 0 & \&\ k \leq \text{LIST}_\text{LEN}(\text{past}(\text{Output}, t_2)) \\
& \quad \quad \quad & \&\ \text{past}(\text{End}(\text{Deliver}), t_2) = t_2 & \&\ \text{past}(\text{Output}[k][\text{Data}_\text{Part}], t_2) = x \\
& \quad \quad \quad & \&\ \text{past}(\text{Output}[k][\text{Count}], t_2) = \text{past}(\text{Receiver}[i].\text{Msg}[\text{Count}], t_1) \\
& \quad \quad \quad & \&\ \text{past}(\text{Output}[k][\text{ID}_\text{Part}], t_2) = \text{Receiver}[i].\text{Id})
\end{align*}
\]

We can now use the local invariant $I_5$ of the process $\text{Packet}_\text{Maker}$ to further rewrite the global invariant.

\[
\begin{align*}
& \text{FORALL } k: \text{Integer} \ (k > 0 & \&\ k \leq \text{LIST}_\text{LEN}(\text{Output}) \\
& \quad \leftrightarrow \ \text{EXISTS } t: \text{Time} \\
& \quad \quad (t < \text{End}(\text{Deliver}) & \&\ \text{past}(\text{End}(\text{Process}_\text{Message}), t) = t \\
& \quad \quad & \&\ \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t)], t) = \text{Output}[k] \\
& \quad \quad & \&\ \text{FORALL } t_1: \text{Time} \ (t_1 \geq t & \&\ t_1 < \text{End}(\text{Deliver}) \\
& \quad \quad \quad \rightarrow \ \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t)], t) = \\
& \quad \quad \quad \quad \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t)], t_1)))
\end{align*}
\]

$I_5$ states that for every message in $\text{Output}$ there exists a time $t$ prior to the end of transition $\text{Deliver}$ such that at time $t$:

1) the message was the last element of variable $\text{Packet}$,
2) the transition $\text{Process}_\text{Message}$ ended and,
3) between $t$ and the end of transition $\text{Deliver}$ the elements already in $\text{Packet}$ are not modified.
In other words, \( t \) is the time instant in which Packet_Maker has finished processing the message.

By using \( I_5 \) the global invariant can be rewritten as follows:

\[
\text{FORALL } i: \text{Receiver_ID}, t_1: \text{Time}, x: \text{Info} \\
( t_1 \leq \text{Now} - H_1 \& \text{past}(\text{Receiver}[i]. \text{End}(\text{New}_\text{Info}(x)), t_1) = t_1) \\
\rightarrow \exists t_2, t_3: \text{Time}, k: \text{Integer} \\
( t_2 \geq t_1 + H_2 \& t_2 \leq \text{Now} \& t_3 < t_2 \\
\& k > 0 \& k \leq \text{LIST}_\text{LEN}(\text{past}(\text{Output}, t_2)) \\
\& \text{past}(\text{End}(\text{Deliver}), t_2) = t_2 \& \text{past}(\text{Output}[k][\text{Data}_\text{Part}], t_2) = x \\
\& \text{past}(\text{Output}[k][\text{Count}], t_2) = \text{past}(\text{Receiver}[i]. \text{Msg}[\text{Count}], t_1) \\
\& \text{past}(\text{Output}[k][\text{ID}_\text{Part}], t_2) = \text{Receiver}[i]. \text{Id} \\
\& \text{past}(\text{End}(\text{Process}_\text{Message}), t_3) = t_3 \\
\& \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN} (\text{Packet}), t_3)], t_3) = \text{past}(\text{Output}[k], t_2) \\
\& \text{FORALL } t: \text{Time} \left( t \geq t_3 \& t < t_2 \\
\rightarrow \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t_3)], t_3) = \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t_3)], t_3) \right) \\
\) \\
\]

Thus, our goal is to prove that both \( t_2 \) and \( t_3 \) exist under the hypothesis stated by the above premise. For this purpose we will use the following lemma (\( L_2 \)):

\[
\text{FORALL } i: \text{Receiver_ID}, t_1: \text{Time}, x: \text{Info} \\
( t_1 \leq \text{Now} - \text{N}_I_\text{Dur} \& \text{past}(\text{Receiver}[i]. \text{End}(\text{New}_\text{Info}(x)), t_1) = t_1) \\
\rightarrow \exists t_3: \text{Time} \\
( t_3 \leq \text{Now} \\
\& \text{past}(\text{End}(\text{Process}_\text{Mesg}), t_3) = t_3 \\
\& \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN} (\text{Packet}), t_3)][\text{Data}_\text{Part}], t_3) = x \\
\& \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t_3)][\text{Count}], t_3) = \text{past}(\text{Receiver}[i]. \text{Msg}[\text{Count}], t_1) \\
\& \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t_3)][\text{ID}_\text{Part}], t_3) = \text{Receiver}[i]. \text{Id}) \\
\]

\( L_2 \) states that every received message will be processed within \( \text{N}_I_\text{Dur} \) time units. Thus, using \( L_2 \) we conclude that \( t_3 \) exists if \( H_1 \) is greater than or equal to \( \text{N}_I_\text{Dur} \).

Thus, we can use the local invariant \( I_6 \) of the process Packet_Maker to prove that also \( t_2 \) exists and that \( t_2 > t_3 \).

\[
\text{FORALL } t_1: \text{Time} \left( t_1 \leq \text{Now} - H_3 \& \text{past}(\text{End}(\text{Process}_\text{Message}), t_1) = t_1 \\
\rightarrow \exists t_2: \text{Time} \\
( t_2 > t_1 \& t_2 \leq \text{Now} \& \text{past}(\text{End}(\text{Deliver}), t_2) = t_2 \\
\& \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t_1)], t_1) = \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t_1)], t_2) \\
\& \text{FORALL } t: \text{Time} \left( t \geq t_1 \& t < t_2 \\
\rightarrow \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t_1)], t_1) = \text{past}(\text{Packet}[\text{past}(\text{LIST}_\text{LEN}(\text{Packet}), t_1)], t_2) \right) \\
\) \\
\]
In fact using I₆ we conclude that if \( t₃ \leq \text{Now} - H₃ \) then \( t₂ \) exists and is greater than \( t₃ \). Thus, by assuming \( H₁ \) equals \( H₃ + N_{\text{I-Dur}} \) we proved that \( t₂ \) exists.

We now prove the lemma L₂, that is:

\[
\text{FORALL } i : \text{Receiver_ID}, t₁ : \text{Time}, x : \text{Info},
\]
\[
(t₁ \leq \text{Now} - N_{\text{I-Dur}} \& \text{past}(\text{Receiver}[i].\text{End}(\text{New_INFO}(x)), t₁) = t₁)
\]
\[
\rightarrow \text{EXISTS } t₃ : \text{Time}, k : \text{Integer}
\]
\[
(t₃ \leq \text{Now} \& \text{past}(\text{End}(\text{Process_Msg}), t₃) = t₃
\]
\[
\& \text{past}(\text{Packet}[\text{past}(\text{LIST_LEN}(\text{Packet}), t₃)][\text{Data_Part}], t₃) = x
\]
\[
\& \text{past}(\text{Packet}[\text{past}(\text{LIST_LEN}(\text{Packet}), t₃)][\text{Count}], t₃) = \text{past}(\text{Receiver}[i].\text{Msg}[\text{Count}], t₁)
\]
\[
\& \text{past}(\text{Packet}[\text{past}(\text{LIST_LEN}(\text{Packet}), t₃)][\text{ID_Part}], t₃) = \text{Receiver}[i].\text{Id})
\]

Consider an instant \( p₀ \) such that L₂ holds until that time. We have to prove that L₂ will hold until time \( p₀ + \Delta \), where \( \Delta \) is equal to \( N_{\text{I-Dur}} \). Since L₂ holds until \( p₀ \) then all messages received before \( p₀ - N_{\text{I-Dur}} \) have been processed before \( p₀ \), i.e., they have been processed within \( N_{\text{I-Dur}} \) time units. Thus, to prove that L₂ holds at \( p₀ + N_{\text{I-Dur}} \) we must show that every message received in the interval \( (p₀ - N_{\text{I-Dur}}, p₀] \) is processed within \( N_{\text{I-Dur}} \) time units.

The local invariant I₃ of process Input allows us to conclude that all the messages will be kept for at least \( N_{\text{I-Dur}} \) time units.

Consider the worst case which is represented by \( N \) new messages \( m₁, ..., m_N \) arriving at time \( p₀ \), i.e., that the process Packet_Maker has to process \( N \) different messages within \( N_{\text{I-Dur}} \) time units. Each message \( m_i, 1 \leq i \leq N \) is a unique tuple \(<x_i, c_i, id_i>\), where \( x_i \) is the information, \( c_i \) is an integer identifying the message and \( id_i \) is the ID of the Input process that created \( x_i \).

If \( N \) new messages arrive at time \( p₀ \) then no messages have been produced in \( [p₀ - N_{\text{I-Dur}}, p₀) \), because the duration of transition New_INFO is \( N_{\text{I-Dur}} \). Thus all messages received before \( p₀ - N_{\text{I-Dur}} \) have been processed by transition Process_Message (L₂). Therefore, at time \( p₀ \) process Packet_Maker will be either idle or executing transition Deliver.

Consider the latter case, i.e., at time \( p₀ \) process Packet_Maker is executing transition Deliver. Therefore it will be idle at time \( p₀ + \text{Del_Dur} \).

The Entry clause of Deliver requires that at least one message is in Packet in order to fire and since the Exit clause of Deliver empties Packet we conclude that between two executions of Deliver there must be at least one execution of Process_Message. Thus, as soon as process Packet_Maker is idle Process_Message fires (Axiom A₂). Thus, in the worst case it will fire at time \( p₀ + \text{Del_Dur} \).
Transition Process_Message lasts P_M_Dur time units, and thus at time \( p_0 + \text{Del}_\text{Dur} + P_M_Dur \) Process_Message will end. By looking at the Entry - Exit clauses of Process_Message we have that Packet will contain the message \( m_1 \).

After Process_Message has fired transition Deliver is enabled if the timeout has expired or the buffer is full. In the former case the process Packet_Maker will select non deterministically whether to execute transition Process_Message or Deliver (Axiom A2), while in the latter Deliver will fire.

Thus if Process_Message is executed \( m_2 \) is processed within \( 2*P_M_Dur + \text{Del}_\text{Dur} \) time units; otherwise, if Deliver is executed, then Process_Message will fire and \( m_2 \) will be processed within \( 2*P_M_Dur + 2*\text{Del}_\text{Dur} \).

Since we want to prove that all the messages are processed within \( N_I_Dur \) time units we will assume that Deliver is executed after Process_Message. In fact in such a case we have to prove that we can process \( N-2 \) messages in \( N_I_Dur - 2*P_M_Dur - 2*\text{Del}_\text{Dur} \) time units while if we assumed that Process_Message fired we had to prove that we can process \( N-2 \) messages in \( N_I_Dur - 2*P_M_Dur - 2*\text{Del}_\text{Dur} \) time units.

After \( m_2 \) is processed transition Deliver could be enabled. Thus, after \( 2*P_M_Dur + 3*\text{Del}_\text{Dur} \) time units Process_Message will process \( m_3 \).

As a consequence the \( N \)-th message will be processed within \( N*P_M_Dur + N*\text{Del}_\text{Dur} \) time units.

Consider now that at time \( p_0 \) the processor was idle. In such a case by using the same kind of reasoning we can conclude that the \( N \) messages will be processed within \( N*P_M_Dur + (N-1)*\text{Del}_\text{Dur} \) time units.

Since by hypothesis \( N_I_Dur \) is greater than or equal to \( N*(P_M_Dur + \text{Del}_\text{Dur}) \) every received message will be processed within time \( p_0 + N_I_Dur \), i.e., L2 holds at time \( p_0 + N_I_Dur \) as well.

### 4.5 Global Schedule Proof Obligations

The global schedule \( S_{CG} \) of the specification \( S \) describes some further properties that \( S \) must satisfy, when all its processes satisfy their own schedules and the assumptions on the behavior of the global environment hold.

Thus, to prove that \( S_{CG} \) is consistent with \( S \) one has to show that:

1) \( S_{CG} \) holds in the initial state of \( S \), and

2) If \( S \) is in a state in which \( S_{CG} \) holds, then for every possible evolution of \( S \), \( S_{CG} \) will hold.

\[ 8 \] For simplicity we will denote with \( m_i \), the \( i \)-th processed message
In both proofs one can assume that the global invariant $I_G$ and every local invariant $I_p$ and local schedule $Sc_p$ holds as well as the global environment assumptions $Env_G$. Note that none of the local environment assumptions ($Env_p$ and $FEnv_p$) may be used to prove the validity of the global schedule.

The first proof requires the validity of the formula:

$$\bigwedge_{p=1}^{n} (Init\_State_p) \& Now = 0 \& Env_G \rightarrow Sc_G$$

The second proof requires the construction of the sequences of events $\sigma$. Each $\sigma$ will contain calling, starting and ending of exported transitions belonging to any process $P_p$ of $S$.

Thus, each formula $F_{\sigma}$ will be composed of the sequence of events that belong to $\sigma$. For each event occurring at time $t$:

- $\text{past}(En_{pj}, t) \& \text{past}(\text{Start}(Op_{pj}, t) = t)$ if the event is the start of $Op_{pj}$ or
- $\text{past}(Ex_{pj}, t) \& \text{past}(\text{End}(Op_{pj}, t) = t)$ if the event is the end of $Op_{pj}$.
- $\text{past}(\text{Call}(Op_{pj}, t) = t)$ if the event is the call of $Op_{pj}$ from the external environment.

Then the prover's job is to show that for any $\sigma$:

$$A1 \& A2'' \& A3 \& A4 \& Env_G \mid -F_{\sigma}$$

& $\forall t: Time \ (t \leq t_0 \rightarrow \text{past}(Sc_G, t))$\n
$\rightarrow \forall t_1: Time \ (t_1 > t_0 \& t_1 \leq t_0 + \Delta \rightarrow \text{past}(Sc_G, t_1))$

where $A2''$ is an axiom derived from $A2$ by taking into account that the exported transitions can fire only if they are called by the environment.

$\forall t: Time \ (\exists d: Time, S'T: \text{SET OF} \ Trans\_of\_p \ [A2'']$\n
$\forall t_1: Time, Op: \text{Trans}\_of\_p$\n
$(t_1 \geq t - d \& t_1 < t \& \text{Op ISIN} S_T \& \text{past}(\text{Start}(Op), t_1) < \text{past}(\text{End}(Op), t)$\n
$\& S'T \subseteq S_T \& S'T \neq \text{EMPTY}$\n
$\& \forall Op': \text{Trans}\_of\_p (\text{Op' ISIN} S'T \rightarrow \text{Eval\_Entry'}(Op', t))$\n
$\& \forall Op': \text{Trans}\_of\_p (\text{Op' ~ISIN} S'T \rightarrow \neg\text{Eval\_Entry'}(Op', t))$\n
$\rightarrow \exists \text{UNIQUE} Op': \text{Trans}\_of\_p (\text{Op' ISIN} S'T \& \text{past}(\text{Start}(Op'), t) = t))$,\n
$^9$ The global invariant, the local invariants and the local schedules can be used as lemmas to prove the global schedule.
where \( \text{Eval}_{ parents}(O_p', t) = \text{Eval}_{ parents}(O_p, t) \land \text{Issue}_{ call}(O_p') \), iff \( O_p' \) is exported and \( \text{Eval}_{ parents}(O_p', t) = \text{Eval}_{ parents}(O_p, t) \), iff \( O_p' \) is not exported.

**Example**

Consider the global schedule \( S_{CG} \):

\[
\forall i: \text{Receiver} _{ ID} , t_1: \text{Time} , x: \text{Info} \quad (t_1 \leq \text{Now} - \frac{N}{L} \cdot \text{P_M Dur} - \text{N_I Dur} - \text{Del Dur} \\
\land \text{past}(\text{Receiver}[i].\text{Call}(\text{New}_\text{Info}(x)), t_1) = t_1 \\
\rightarrow \exists t_2: \text{Time} , k: \text{Integer} \\
\quad (t_2 = t_1 + \frac{N}{L} \cdot \text{P_M Dur} + \text{N_I Dur} + \text{Del Dur} \\
\land k > 0 \land k \leq \text{LIST}\_\text{LEN}(\text{past}(\text{Output}, t_2)) \\
\land \text{Change}(\text{Output}, t_2) \land \text{past}(\text{Output}[k][\text{Data}_\text{Part}], t_2) = x \\
\land \text{past}(\text{Output}[k][\text{ID}_\text{Part}], t_2) = \text{Receiver}[i].\text{Id})
\]

Consider a time instant \( p_0 \) such that at \( p_0 - \frac{N}{L} \cdot \text{P_M Dur} - \frac{N}{L} \cdot \text{P_M Dur} - \text{Del Dur} \) a \( \text{call}(\text{New}_\text{Info}(x)) \) occurred and suppose \( S_{CG} \) holds until \( p_0 \). Our goal is to prove that \( S_{CG} \) holds until \( p_0 + \Delta \) as well. We assume \( \Delta \) equal to \( \frac{N}{L} \cdot \text{P_M Dur} + \text{Del Dur} \).

By considering the local invariant \( I_1 \) of process Packet Maker we know that at time \( p_0 \) Deliver ended, and by axiom A1 at time \( p_0 - \text{Del Dur} \), transition Deliver started.

By considering \( \text{Env}_G \) we can deduce that:

1) At time \( p_0 - \frac{N}{L} \cdot \text{P_M Dur} - \frac{N}{L} \cdot \text{P_M Dur} - \text{Del Dur} \), \( \frac{N}{L} \) calls to \( \text{New}_\text{Info} \) occurred,

2) At time \( p_0 - \frac{N}{L} \cdot \text{P_M Dur} \), \( \frac{N}{L} \) calls to \( \text{New}_\text{Info} \) occur, and

3) No \( \text{call}(\text{New}_\text{Info}) \) can occur in \( (p_0 - \frac{N}{L} \cdot \text{P_M Dur} - \frac{N}{L} \cdot \text{P_M Dur} - \text{Del Dur}, p_0 - \frac{N}{L} \cdot \text{P_M Dur}) \)

Using the local schedule of process Packet Maker we know that at time \( p_0 + \frac{N}{L} \cdot \text{P_M Dur} + \text{Del Dur} \) another \( \text{End}(\text{Deliver}) \) occurs, and by again using \( I_1 \) of process Packet Maker at that time Output changes value.

Thus, the time interval between the \( \frac{N}{L} \) call(\( \text{New}_\text{Info} \)) at time \( p_0 - \frac{N}{L} \cdot \text{P_M Dur} \) and the change in the value of Output at time \( p_0 + \frac{N}{L} \cdot \text{P_M Dur} + \text{Del Dur} \) is equal to:

\[
[p_0 + \frac{N}{L} \cdot \text{P_M Dur} + \text{Del Dur}] - [p_0 - \frac{N}{L} \cdot \text{P_M Dur}] = \frac{N}{L} \cdot \text{P_M Dur} + \frac{N}{L} \cdot \text{P_M Dur} + \text{Del Dur}
\]

We still have to prove that the output packet contains all and only the \( \frac{N}{L} \) messages produced by the input packet:

The local invariant \( I_2 \) of process Packet Maker states that no messages are produced by that process, thus the output packet will contain only
received messages. All the messages received in 

$[0, p_0 - N \_ I \_ Dur - N/L*P \_ M \_ Dur - Del \_ Dur]$ have already been output, since the local 
schedule holds until $p_0$; therefore, the only received messages after $p_0 - N \_ I \_ Dur - N/L*P \_ M \_ Dur - Del \_ Dur$ are those that arrived at time $p_0 - N \_ I \_ Dur$. Furthermore, Output has not changed value in 

$[p_0 - N \_ I \_ Dur, p_0 + N/L*P \_ M \_ Dur + Del \_ Dur]$.

As a consequence every message in Output at time $p_0 + N/L*P \_ M \_ Dur + Del \_ Dur$ was received at time $p_0 - N \_ I \_ Dur$. Since we know from the local 
schedule of process Packet Maker that Output contains $N/L$ messages at 
time $p_0 + N/L*P \_ M \_ Dur + Del \_ Dur$, it is trivial to prove that at that time 
Output contains all and only the messages received at time $p_0 - N \_ I \_ Dur$.

Thus, one can conclude that $\text{Sc}_G$ holds in $p_0 + \Delta$.

4.6 Imported Variable Proof Obligation

When proving the local schedule of a process $p$ one can use the 
assumptions about the imported variables expressed by $\text{IV}_p$. Therefore, 
these assumptions must be checked against the behavior of the processes 
that they are imported from.

The proof obligation guarantees that the local environment, local 
schedule and local invariant of every process of $S$ except $p$, and the global 
environment, invariant and schedule imply the assumptions on the 
imported variables of process $p$:

$$A1 \& A2 \& A3 \& \bigwedge_{i \neq p} \text{Envi}_i \& \bigwedge_{i \neq p} \text{Ii} \& \bigwedge_{i \neq p} \text{Sci} \& \text{Env}_G \& \text{I}_G \& \text{Sc}_G \rightarrow \text{IV}_p$$

Example

Consider the first two assumptions on the imported variables of process 
Packet_Maker ($\text{IV}_{pm}$):

$$\forall t: \text{Time} \ ( (t - N \_ I \_ Dur) \mod (N/L*P \_ M \_ Dur + Del \_ Dur) = 0 \\ \rightarrow \exists S: \text{Set of Receiver ID} \\
( |S| = N/L \\ \& \forall i: \text{Receiver ID} \ (i \isin S \leftrightarrow \text{Receiver}[i].\text{End}(\text{New Info}) = t))$$

$$\& \forall t: \text{Time} \ ((t - N \_ I \_ Dur) \mod (N/L*P \_ M \_ Dur + Del \_ Dur) \neq 0 \\ \rightarrow \forall i: \text{Receiver ID} \ (\neg \text{Receiver}[i].\text{End}(\text{New Info}) = t))$$

The global environment clause states that:

$$\forall t: \text{Time} \ ( t \mod (N/L*P \_ M \_ Dur + Del \_ Dur) = 0 \\ \rightarrow \exists S: \text{Set of Receiver ID} \\
( |S| = N/L \\ \& \forall i: \text{Receiver ID} \ (i \isin S \leftrightarrow \text{Receiver}[i].\text{Call}(\text{New Info}) = t))$$

$$\& \forall t: \text{Time} \ ( t \mod (N/L*P \_ M \_ Dur + Del \_ Dur) \neq 0 \\ \rightarrow \forall i: \text{Receiver ID} \ (\neg \text{Receiver}[i].\text{Call}(\text{New Info}, t))$$
Using the local schedule of process Input we can rewrite the above formula as follows:

\[
\text{FORALL } t: \text{Time} \ (t \ \text{MOD} \ (N/L \ast P_{M\_Dur} + \text{Del\_Dur}) = 0 \\
\rightarrow \ \text{EXISTS } S: \text{Set\_of\_Receiver\_ID} \\
\hspace{1.5cm} (|S| = N/L \\
\hspace{2.5cm} \& \ \text{FORALL } i: \text{Receiver\_ID} \ (i \ \text{ISIN} \ S \leftrightarrow \text{Receiver}[i].\text{Start}(\text{New\_Info}) = t))
\]

\[
\& \ \text{FORALL } t: \text{Time} \ (t \ \text{MOD} \ (N/L \ast P_{M\_Dur} + \text{Del\_Dur}) \neq 0 \\
\rightarrow \ \text{FORALL } i: \text{Receiver\_ID} \ (\neg \text{Receiver}[i].\text{Start}(\text{New\_Info}, t))
\]

Using axiom A1 we have that:

\[
\text{FORALL } t: \text{Time} \ (t \ \text{MOD} \ (N/L \ast P_{M\_Dur} + \text{Del\_Dur}) = 0 \\
\rightarrow \ \text{EXISTS } S: \text{Set\_of\_Receiver\_ID} \\
\hspace{1.5cm} (|S| = N/L \\
\hspace{2.5cm} \& \ \text{FORALL } i: \text{Receiver\_ID} \\
\hspace{3.5cm} (i \ \text{ISIN} \ S \leftrightarrow \text{Receiver}[i].\text{End}(\text{New\_Info}) = t + N_I\_Dur))
\]

\[
\& \ \text{FORALL } t: \text{Time} \ (t \ \text{MOD} \ (N/L \ast P_{M\_Dur} + \text{Del\_Dur}) \neq 0 \\
\rightarrow \ \text{FORALL } i: \text{Receiver\_ID} \ (\neg \text{Receiver}[i].\text{End}(\text{New\_Info}, t + N_I\_Dur))
\]

which can be rewritten as:

\[
\text{FORALL } t: \text{Time} \ (t - N_I\_Dur) \ \text{MOD} \ (N/L \ast P_{M\_Dur} + \text{Del\_Dur}) = 0 \\
\rightarrow \ \text{EXISTS } S: \text{Set\_of\_Receiver\_ID} \\
\hspace{1.5cm} (|S| = N/L \\
\hspace{2.5cm} \& \ \text{FORALL } i: \text{Receiver\_ID} \\
\hspace{3.5cm} (i \ \text{ISIN} \ S \leftrightarrow \text{Receiver}[i].\text{End}(\text{New\_Info}) = t))
\]

\[
\& \ \text{FORALL } t: \text{Time} \ ((t - N_I\_Dur) \ \text{MOD} \ (N/L \ast P_{M\_Dur} + \text{Del\_Dur}) \neq 0 \\
\rightarrow \ \text{FORALL } i: \text{Receiver\_ID} \ (\neg \text{Receiver}[i].\text{End}(\text{New\_Info}) = t))
\]

which is the IV pm.

4.7 Environment Consistency Proof Obligations

Every process $P_p$ of $S$ may contain two clauses describing assumptions on the behavior of the external environment, $Env_p$ and $FEnv_p$. These clauses are used to prove the local schedule of $P_p$ as discussed in sections 3.1 and 3.5. The global specification of $S$ also contains a clause describing assumptions on the system environment behavior $Env_G$.

For soundness, it is necessary to verify that none of the environmental assumptions contradict each other, i.e., that a behavior satisfying the global as well as the local assumptions can exist. This requires proving that the following formula is satisfiable:

\[
\Lambda_i \exists_l Env_i \ \& \ \Lambda_i \exists_l FEnv_i \ \& \ Env_G
\]

If the above formula is satisfiable, then there exists at least one environment behavior that fulfills all the environmental assumptions. Conversely, if it is not satisfiable, no such environment can exist and some (or possibly all) of the assumptions on the environment have to be modified.
Example

The local environment of process Input is:

\[ ( \exists t: \text{Time} \ ( \text{Call-2(New\_Info, t)} ) \]
\[ \rightarrow \text{Call(New\_Info) - Call-2(New\_Info) \geq N\_I\_Dur} \]
\& ( \text{Now} \geq \text{Input\_Tout} \]
\[ \rightarrow ( \exists t: \text{Time} \ ( \text{Call(New\_Info, t)} ) \& \text{Now} - \text{Call(New\_Info)} < \text{Input\_Tout} ) \]

Process Packet\_Maker has no local environment, while the global environment is:

\[ \forall t: \text{Time} \ ( t \ \text{MOD} \ ( N/L \times P\_M\_Dur + \text{Del\_Dur} ) = 0 \]
\[ \rightarrow ( \exists S: \text{Set\_of\_Receiver\_ID} \]
\[ ( |S| = N/L \]
\& \ ( \forall i: \text{Receiver\_ID} \ ( i \in S \leftrightarrow \text{Receiver}[i].\text{Call(New\_Info, t)} ) \]
\& \ ( \forall t: \text{Time} \ ( t \ \text{MOD} \ ( N/L \times P\_M\_Dur + \text{Del\_Dur} ) \neq 0 \]
\[ \rightarrow ( \forall i: \text{Receiver\_ID} \ ( \neg \text{Receiver}[i].\text{Call(New\_Info)} = t ) ) \]

Neither the Input nor the Packet\_Maker process have any further assumptions about the environment and thus, we have to verify the satisfiability of the conjunction of the two formulas above.

This will result in a set of equations among the constants of the specification. If the set of equations can be solved, then the environment is satisfiable otherwise it is not.

5. Conclusion and future directions

In this report the environment and critical requirements clauses, which were only briefly sketched in previous papers, were presented in detail. The intra-level proof obligations were also presented and an example proof was demonstrated.

All of the proofs for the CCITT specification have been completed. In addition, the proofs of five different schedules that can be guaranteed by using different further assumptions clauses have also been completed. The proofs of these schedules did not require any new or changed invariants.

Future work will concentrate on defining the necessary inter-level proof obligations for ASTRAL.

References


GLOBAL Specification CCITT

PROCESSES
  Receiver: array [1..N] of Input,
  Assembler: Packet_Maker

TYPE
  Data,
  Message IS STRUCTURE OF
    (Data_Part: Data
     Count: Integer
     ID_Part: ID),
  Message_List IS LIST OF Message,
  Pos_Integer: TYPEDEF i: Integer (i > 0),
  Receiver_ID: TYPEDEF i:Pos_Integer (i ≤ N),
  Set_Of_Receiver_ID IS SET OF Receiver_ID,
  Info: TYPEDEF D:Data (D ≠ Closed)

CONSTANT
  N, L: Pos_Integer,

  /*N denotes the number of processes of type Input, L denotes a value such that the number of input processes producing messages at the same time is N/L*/

  Closed: Data,
  N_I_Dur, P_M_Dur, Del_Dur: Time
/*These are the duration for transitions New_Info, Process_Message, and Deliver*/

H1, H2: Time

/*H1, H2 are lower and upper bounds on the time for an input to be output*/

AXIOM
N MOD L =0

DEFINE
Change(L_Msg:Message_List,t:Time):Boolean ==
   EXISTS e: Time
   ( e >0 & e ≤ t
     & FORALL d: Time
        ( d ≥ t - e & d < t → past(L_Msg, d) ≠ past(L_Msg,t)))

ENVIRONMENT
/*The environment cyclically produces exactly N/L messages
every N/L*P_M_Dur + Del_Dur time units*/
FORALL t:Time (t MOD (N/L*P_M_Dur + Del_Dur) = 0 → EXISTS S: Set_Of_Receiver
   ( |S| =N/L^10
     & FORALL i:Receiver_ID
        (i ISIN S ↔ Receiver[i].Call(New_Info) = t)))
& FORALL t:Time (t MOD (N/L*P_M_Dur + Del_Dur) ≠ 0
   → FORALL i:Receiver_ID (∼Receiver[i].Call(New_Info, t)))

INVARIANT
/* Every data output was received sometime in the past */

^10For simplicity, we adopt the traditional cardinality operator, | |, even though it is not an ASTRAL operator.
FORALL k:Integer
  (k > 0 & k ≤ LIST_LEN(Output) & Output[k][Data_Part] ≠ Closed
  → EXISTS i:Receiver_ID, t:Time, j:Integer
  (t < Now & Receiver[i].Start-j(New_Info(Output[k][Data_Part]) = t))

& /* Every input data will be output within H1 time units after it is input, but not sooner than H2 time units*/

FORALL i:Receiver_ID, t1:Time, x:Info
  (t1≤Now-H1 & past(Receiver[i].End(New_Info(x)),t1) = t1
  → EXISTS t2:Time, k:Integer
    ( t2 ≥ t1+H2 & t2 ≤ Now & Change(Output,t2)
    & 0 < k & k ≤ LIST_LEN(past(Output,t2))
    & past(Output[k][Data_Part],t2)=x
    & past(Output[k][Count],t2) = past(Receiver[i].Msg[Count],t1)
    & past(Output[k][ID_part],t2) = Receiver[i].Id))

SCHEDULE

/*The time that elapses between the call of a New_Info transition and the delivery of the message it produced is equal to N/L*P_M_Dur + N_I_Dur + Del_Dur*/

FORALL i:Receiver_ID, t1:Time, x:Info
  (t1≤Now-H1 & past(Receiver[i].End(New_Info(x)),t1) = t1
  → EXISTS t2:Time, k:Integer
    ( t2 = t1 + N/L*P_M_Dur + N_I_Dur + Del_Dur
    & 0 < k & k ≤ LIST_LEN(past(Output,t2))
    & Change(Output,t2) & past(Output[k][Data_Part],t2)=x
    & past(Output[k][ID_part],t2) = Receiver[i].Id))

END CCITT
SPECIFICATION Input
LEVEL Top_Level

IMPORT
  Data, Message, Info, Closed, N_I_Dur

EXPORT
  New_Info, Msg

VARIABLE
  Msg: Message,
  Channel_Closed:Boolean

CONSTANT
  Input_Tout, N_T_Dur: Time

ENVIRONMENT

  ( EXISTS t:Time (Call-2 (New_Info, t)) →
    Call (New_Info) - Call-2 (New_Info) ≥ N_I_Dur )
& ( Now ≥ Input_Tout →
    EXISTS t:Time (Call (New_Info, t))
  & Now - Call(New_Info) < Input_Tout )

INITIAL
  ~Channel_Closed & Msg[Data_Part] ≠ Closed & Msg[Count]=0
INVARIANT

/* After Input_Tout time units have elapsed without receiving any new message a timeout occurs */

FORALL t1: Time
   (Start(New_Info, t1) & Now - t1 > Input_Tout
    → EXISTS t2: Time
       (Start(Notify_Timeout, t2) & t2 = t1 + Input_Tout))

& /* The last received message is kept until a timeout occurs */

FORALL t1: Time, x: Info
   ( (End(New_Info(x), t1) & Now - t1 < Input_Tout - N_I_Dur + N_T_Dur
      → Msg[Data_part] = x)
    &
      (End(New_Info(x), t1) & Now - t1 ≥ Input_Tout - N_I_Dur + N_T_Dur
       → Msg[Data_part] = Closed))

SCHEDULE
FORALL t: Time, x: Info
  (t ≤ Now → ((Call(New_Info(x)) = t) ↔ Start(New_Info(x)) = t))

TRANSITION New_Info(x:Info)  N_I_Dur
EXIT
   Msg[Data_Part] = x
& Msg[Count] = Msg[Count]' + 1
& Msg[ID_Part] = Self
& ~Channel_Closed
TRANSITION Notify_Timeout  N_T_Dur
ENTRY
   EXISTS t1: Time
      Start(New_Info,t1) & Now - t1 ≥ Input_Tout
      & ~Channel_Closed
EXIT
   Msg[Data_Part] = Closed
   & Msg[Count] = Msg[Count]' + 1
   & Msg[ID_Part] = Self
   & Channel_Closed

END Top_Level
END Input
SPECIFICATION Packet_Maker
LEVEL Top_Level
IMPORT
Receiver, Data, Message, Message_List, Pos_Integer, Receiver_ID,
Set_Of_Receiver_ID, Info, Closed, N, L, P_M_Dur, Del_Dur, N_I_Dur,
Msg

EXPORT
Output

VARIABLE
Packet: Message_List,
Previous(Receiver_ID): Time,
Output: Message_List

CONSTANT
Maximum: Pos_Integer,
Del_Tout, H3: Time
/*H3 denotes an upperbound for the time to deliver a message
   after it has been processed*/

IMPORTED VARIABLE CLAUSE
FORALL t: Time ((t - N_I_Dur) MOD (N/L*P_M_Dur + Del_Dur) = 0
   →
   EXISTS S: Set_Of_Receiver_ID
   ( | S | = N/L
       & FORALL i: Receiver_ID
           (i ISIN S ↔ Receiver[i].End(New_Info) = t))
   & FORALL t:Time ((t - N_I_Dur) MOD (N/L*P_M_Dur + Del_Dur) ≠ 0
   →
   FORALL i:Receiver_ID (~Receiver[i].End(New_Info) = t))
   & FORALL i:Receiver_ID (Receiver[i].Msg[Data_Part] ≠ Closed)
INITIAL

Packet = EMPTY
& FORALL i:Receiver_ID (Previous(i)=0)
& Output = EMPTY

INVARIANT

/*Changes in Output occur at and only at the end of a Deliver*/ (I1)
FORALL t: Time
(Change(Output, t) ↔ past(End(Deliver), t) = t)

& /* No new messages are generated by the packet assembler */ (I2)
FORALL k:Integer
(k>0 & k≤LIST_LEN(Output)
→ EXISTS i:Receiver_ID, t:Time
(t<Now & past(Receiver[i].Msg,t)=Output[k]) )

& /*The order that messages appear in an output packet is the order in which they were processed from the channels*/ (I3)
FORALL k:Integer
(k>0 & k<LIST_LEN(Output)
→ EXISTS t1,t2:Time
(t1 < t2 < Now
& past(End(Process_Message), t1) = t1
& past(End(Process_Message), t2) = t2
& Output[k] = past(Packet[past(LIST_LEN(Packet),t1)], t1)
& Output[k+1] = past(Packet[past(LIST_LEN(Packet),t2)], t2))
&
/* The order is also preserved across output packets */ (14)
EXISTS t:Time (Start-2(Deliver, t) & End(Deliver) > Start(Deliver))
→
EXISTS t1,t2:Time
(t1 < t2 < Now
& past(End(Process_Message, t1) = t1
& past(End(Process_Message, t2) = t2
& past(Output[past(LIST_LEN(Output), Start(Deliver)], Start(Deliver))
  = past(Packet[past(LIST_LEN(Packet), t1)], t1)
& Output[1] = past(Packet[past(LIST_LEN(Packet), t2)], t2))
&
/* Every message in Output was previously in Packet and (15)
all of the elements of Packet have not changed from when
they were put into the packet until the packet is output*/
FORALL k:Integer
(k>0 & k ≤ LIST_LEN(Output)
↔ EXISTS t:Time
 (t<End(Deliver) & past(End(Process_Message,t)=t
 & past(Packet[past(LIST_LEN(Packet), t)], t) = Output[k]
 & FORALL t1:Time
 (t1≥t & t1<End(Deliver)
  → past(Packet[past(LIST_LEN(Packet), t)], t) =
    past(Packet[past(LIST_LEN(Packet), t)], t1))))
&
FORALL t1:Time
(t1≤Now-H3 & past(End(Process_Msg),t1)=t1
→ EXISTS t2:Time
(t2>t1 & t2 ≤ Now & past(End(Deliver),t2)=t2
 & past(Packet[past(LIST_LEN(Packet), t1)], t1) =
    past(Output[past(LIST_LEN(Packet), t1)], t2)
 & FORALL t:Time
 (t≥t1 & t<t2
  → past(Packet[past(LIST_LEN(Packet), t1)], t1) =
    past(Packet[past(LIST_LEN(Packet), t1)], t2))))
SCHEDULE
/*The transition Deliver is activated cyclically. Furthermore, it always delivers a packet with N/L elements*/

EXISTS t:Time (End-2(Deliver,t))
→ End(Deliver) - End-2(Deliver) = N/L*P_M_Dur + Del_Dur
& FORALL t: Time (past(End(Deliver), t) = t
→ LIST_LEN(past(Output, t)) = N/L)

TRANSITION Process_Msg(R_id:Receiver_ID)  P_M_Dur
ENTRY
/*Packet is not full and either (a) the present message has been produced after the last message that has been processed from that channel, or (b) the value of the current message is Closed and the value of the previously processed message for that channel was not Closed*/
LIST_LEN(Packet) < Maximum
& (Receiver[R_id].End(New_Info) > Previous(R_id)
  | (Receiver[R_id].Msg[Data_Part]=Closed
      & past(Receiver[R_id].Msg[Data_Part],Previous(R_id)) ≠ Closed ))
EXIT
  Packet = Packet' CONCAT LIST(Receiver[R_id].Msg)
& Previous(R_id) BECOMES Now

TRANSITION Deliver  Del_Dur
ENTRY
/*Either Packet is full or Packet is not empty and the timeout elapsed from the last Deliver or from the initial time*/
LIST_LEN(Packet) = Maximum
| (LIST_LEN(Packet) > 0
  & (EXISTS t:Time (Start(Deliver, t) &  Now - t = Del_Tout)
      | Now = Del_Tout - Del_Dur + N_I_Dur))
EXIT
  Output = Packet'
& Packet = EMPTY
FURTHER ASSUMPTIONS #1
CONSTANT REFINEMENT
Del_Tout = 0, Maximum = N/L

TRANSITION SELECTION
{Process_Message, Deliver} TRUE {Process_Message}

FURTHER ASSUMPTIONS #2
CONSTANT REFINEMENT
Del_Tout = N/L*P_M_Dur + Del_Dur, Maximum = N

END Top_Level
END Packet_Maker