ELSEVIER

Science of Computer Programming 29 (1997) 23–52

Science of Computer Programming

Verification of XTP context management closing procedure in style of TLA[1]

Tatjana Kapus*, Zmago Brezočnik

Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova ul. 17, SI-2000 Maribor, Slovenia

Abstract

Modular specification and compositional verification of the context management closing procedure of Xpress Transfer Protocol (XTP Protocol Definition, Revision 3.6) in style of Lamport's Temporal Logic of Actions is considered. It is assumed that a full-duplex association over a pair of lossy channels between two contexts on different hosts is being closed, such that any data for that connection have already been transmitted successfully to the hosts. Thus, the case of the closing procedure is considered where both contexts are active and synchronized initially, and only control messages have to be sent until the association is closed. © 1997 Elsevier Science B.V.

Keywords: Communication protocols; Connection management; Compositional verification; Temporal logic of actions

1. Introduction

Xpress Transfer Protocol is a light-weight transport protocol for high-speed networks [5]. It allows for reliable sequenced data delivery by establishing a full-duplex connection between end hosts. We show how the essential part of the context management closing procedure of XTP, described in XTP Protocol Definition, Revision 3.6 [15], can be specified and verified in style of Lamport's Temporal Logic of Actions [10]. Its basic idea is that the (logical) specification language contains unprimed and primed variables. A specified system is looked upon as a (possibly infinite) state transition system. Its possible transitions are not described e.g. by an event-state table (cf. [6]), but rather by action formulae. An action formula is like a first-order predicate logic formula, except that it may contain primed variables. An unprimed variable denotes

* Corresponding author. E-mail: kapus@uni-mb.si.

[1] Work supported by the Ministry of Science and Technology, Republic of Slovenia.

PII S0167-6423(96)00028-7

the value of the variable in the state just before the (atomic) action execution, and a primed variable its value in a possible next state after execution. A design specification of a protocol can thus actually be written by listing action formulae describing possible actions. The operational style can also be used for writing a requirement specification. Temporal operators can be used for specifying liveness and also for expressing requirement specifications in a more declarative way. The idea is also employed e.g. in [9, 13, 7], and demonstrated in [12].

Assume that a non-multicast *association*, meaning a connection in XTP terminology, between an instance of XTP at a host A and an instance of XTP at a host B has been established over a pair of lossy channels, one for carrying messages in each direction. The set of state variables for an instance of XTP at an end host is called a *context*. One therefore also says that the association exists between a context at host A and a context at host B. The association consists of a simplex data stream A-to-B and a simplex data stream B-to-A. Host A is the sending host for stream A-to-B and the receiving host for B-to-A, and symmetrically for host B. Likewise, the context at each host is the sender for one of the data streams, and the receiver for the other one. Each context manages both the outgoing and the incoming data stream as well as the potential for sending and receiving control information. In [15], a context is looked upon as consisting of a part for managing the outgoing data stream, called an *output data stream*, and a part for managing the incoming data stream, called an *input data stream*. When the output data stream is in *active* state, it can receive data from its host (i.e., XTP user or application), and transmit them over a lossy channel to the other context and host according to XTP specifications in order to achieve reliable data transfer. Typically, an acknowledgement/retransmission scheme for error control is used. When the input data stream of the context at the other host is in *active* state, it takes the data from the channel, also acts according to the XTP rules, and puts correctly received data into a *data receive queue*, wherefrom the data are read by the receiving host. A simplex data stream thus consists of an active output data stream at its sending context and of an active input data stream at its receiving context. If the output data stream is *inactive* (we shall also say 'closed'), no data can be sent or retransmitted, and if the input data stream is *inactive*, no data from the incoming channel can be received, no (negative) acknowledgements sent, and no data from the receive queue delivered to the application. A simplex data stream is *closed*, when the data streams at both ends are inactive. A context is *active* if at least its output or input data stream is not inactive. If both are inactive, the context is closed, i.e., it passes to *null* state.

When all data for the association have been sent, or for other reasons, the association is closed. In order for the association to be closed, both contexts must be closed by the context management *closing procedure*, so as to be prepared to be activated for a new association. A context can be closed in several ways. It may be released when its output and input data stream have been closed. It may be aborted when information exchange between the hosts becomes impossible. It may also be ended by a local application whenever it decides so, but we shall not consider this possibility.

In other words, in order to close the association, both simplex data streams must be closed. Graceful or ungraceful (forced) closure for a simplex data stream is possible. A graceful closure means that a data stream was closed after all data for the data stream had been correctly transmitted by its output data stream and acknowledged by the receiving input data stream, and delivered from the data receive queue to the receiving application. This means that the input data stream had been provided an opportunity to ask for retransmissions. In this paper, we consider the correctness of the closing procedure for the following possible situation. We assume that all data for both simplex data streams have already been correctly transferred and acknowledged, but there may still be some data in the data receive queue of each one, waiting to be read by the receiving host, when the closing of the association starts. We assume both contexts are active, and that both output and input data stream of each context is active. We also assume that the contexts are synchonized with each other relating data. The consequence of these assumptions is that no delay of releasing a context, described in [15], and no aspects of XTP data transfer need be modelled, except for the delivery of the receive data queues.

Information in XTP is transferred in packets. Each packet contains a header with bitflags for controlling the operation of XTP and a field for packet-type identification. For our purpose, only flags WCLOSE (i.e., "write-close"), RCLOSE ("read-close"), END for the control of association closing, and SREQ ("status request") flag for controlling the acknowledgement policy have to be modelled. We only need packets with CNTL or DIAG in their packet-type field. CNTL packets are *control packets* for exchanging state information between end hosts. DIAG packets are *diagnostic packets*. A CNTL packet with SREQ set is called CNTL-*request packet*. Upon receiving such a packet, a context must immediatelly send a response to the sender of the request. If the context is active, a CNTL packet with information about its current status, known as CNTL-*response packet*, is sent. In our case, the status information consists of WCLOSE, RCLOSE, and END bits only. If the context is already closed, a DIAG packet is sent, telling the sender that there is no active context at the other side.

Each simplex data stream must be closed with a handshake consisting of two packets, one from each host, involving the WCLOSE, RCLOSE, and/or END bits. Closure of a data stream can be initiated by the sending or the receiving host for the stream.

Basically, the closing procedure in our case works as follows. With *graceful close*, the sending context for a data stream sends a CNTL-request packet with WCLOSE bit set telling the receiving context that the sending host wants to close its output data stream (in fact, its outgoing simplex data stream), and starts a timer. It continues to transmit CNTL-request packets with WCLOSE set at timeout intervals. Upon receiving such a packet, the receiving context waits until its receive data queue is empty. We consider the closing procedure for the CONFIRM mode of closing [15]. This means that the context must then still wait for a confirmation from the application, before it can close the input data stream. After obtaining the confirmation, it closes its input data stream, and sets RCLOSE in outgoing packets. The sending

context stops the retransmission and closes the output data stream when it gets a CNTL packet with RCLOSE bit set. With *ungraceful close*, the receiving host decides to close its input data stream (i.e., its incoming simplex data stream), the context at the host discards any data left in its receive queue, sends a CNTL-request packet with RCLOSE bit set to the sending context, starts a timer, and continues to send CNTL-request packets with RCLOSE set at timeout intervals. Upon receiving such a packet, the sending context closes its output data stream and sets WCLOSE in outgoing packets. The receiving context stops the retransmission and closes its input data stream upon receiving a control packet with WCLOSE bit set. A context sets END in a CNTL response to a CNTL-request packet if it has been closed upon receiving the request.

Every control packet sent by a context carries the status information about both its output and input data stream. Closing of both data streams may also be initiated simultaneously. Therefore, the procedure of closing the simplex data streams can be highly concurrent. For example, it is possible to achieve the closure of both simplex data streams, and thus of both contexts and of the association, by sending just one CNTL packet in each direction, a CNTL-request packet with WCLOSE and RCLOSE bits set in one direction, and a CNTL-response packet with WCLOSE, RCLOSE and END bits set in the other one. In [15], the closing procedure for a context is described by mutually dependent semi-formal state machines, one for the output and one for the input data stream. We are interested in formally specifying and verifying concurrent behaviour and cooperation of input and output data stream state machines at both hosts in sending control packets for closing the association. We assume the only thing that can go wrong is loss of packets in the channels, and not, for example, that a host can go down. No real time aspects will be modelled.

We write the design specification of the procedure as a parallel composition of a number of components. The components specifications are written in such a way that the composition practically satisfies the conjunction of the specifications. A straightforward way to prove a requirement specification of the procedure is then by first constructing the design specification of the complete system by the conjunction, and prove that the latter implies the requirements. However, we avoid the intermediate step by providing proof rules for proving typical required properties directly from the components specifications. Since communication protocols procedures typically consist of specifically constructed sender, receiver, and channel components, the compositional verification could have been simplified by developing two proof rules for exactly this type of systems.

The paper is organized as follows. Section 2 briefly describes the temporal logic of actions (TLA) used. The system model, basic notions regarding specification, important system operations, and basic proof rules for them are presented in Section 3. Section 4 contains a design specification of the simplified XTP closing procedure written in style of Lamport's TLA together with an explanation. Required properties of the procedure are also specified there. In Section 5 we list necessary proof rules of TLA and propose some property-specific and system-specific proof rules for compositional

verification. Proofs of the required properties of the closing procedure applying these rules are outlined in Section 6. A discussion of the results and future work concludes the paper.

2. The temporal logic of actions

We use quite a usual linear-time temporal logic (cf. [12]). The alphabet of our temporal logic language consists of denumerable sets of local individual variables $\mathscr{X}$, global individual variables $\mathscr{Y}$, local proposition variables $\mathscr{V}$, and global proposition variables $\mathscr{U}$. Further, it consists of at most denumerable sets of function and predicate symbols, the latter including equality. It contains Boolean connectives $\neg$, $\rightarrow$ (implication), the quantifier $\exists$, and temporal operators $'$, $\bigcirc$ ("next"), **unless**, $\exists\!\!\!\exists$, the latter being a "temporal" quantifier for quantification over local variables. *Local variables* may change their value from state to state, *global* may not. Like in [10], local variables must not occur bound under temporal operators other than $\exists\!\!\!\exists$.

Terms *Term*, temporally unquantified formulae *UForm* and formulae *Form* are defined as follows. Let $x \in \mathscr{X}$, $y \in \mathscr{Y}$, $v \in \mathscr{V}$, $u \in \mathscr{U}$, $w \in \mathscr{Y} \cup \mathscr{U}$, and $z \in \mathscr{X} \cup \mathscr{V}$. Let $t \in Term$, $F_u, G_u \in UForm$, and $F, G \in Form$. Then

$$
\begin{aligned}
t &::= x \mid y \mid x' \mid f(t_1, \ldots, t_n) \\
F_u &::= v \mid u \mid v' \mid P(t_1, \ldots, t_n) \mid \neg F_u \mid F_u \rightarrow G_u \mid \bigcirc F_u \mid F_u \ \textbf{unless}\ G_u \mid \exists w F_u \\
F &::= F_u \mid \exists\!\!\!\exists z F \mid \neg F \mid F \rightarrow G \mid \exists w F
\end{aligned}
$$

for any n-ary function symbol f or predicate symbol P from our sets. Propositional constants (**true**, **false**), other Boolean connectives ($\wedge, \vee, \leftrightarrow$), and universal quantifiers can be introduced as usual abbreviations. The same holds for common temporal operators, for example, $\Box F \equiv F$ **unless false** (i.e., "always F"), $\Diamond F \equiv \neg\Box\neg F$ (i.e., "eventually F").

A variable is called *primed*, if followed by $'$, and *unprimed* otherwise. A *state formula* is one that does not contain temporal operators including $'$. A formula containing at most unprimed, primed variables, and Boolean operators, is called an *action formula*, or sometimes simply an *action*.

Formulae are interpreted in infinite sequences of states. We assume a fixed first-order interpretation that assigns functions and relations on some domain of values to the function and predicate symbols. A global valuation η assigns values from the domain to the global individual variables, and a Boolean value to every global proposition variable. Each *state* s_i of an infinite state sequence $\sigma = (s_0, s_1, \ldots)$ does the same for the local individual and proposition variables, respectively. Let $\sigma, \eta \models F$ indicate that F is *true in* σ *under* η. The meaning of the operators is as usual, except for existential quantification of local variables, which is like in [10]:

$$\sigma, \eta \models \exists\!\!\!\exists z F \quad \text{iff} \quad \text{there exist } \sigma', \sigma'' \text{such that } \sigma \simeq \sigma', \sigma' \approx_z \sigma'', \text{ and } \sigma'', \eta \models F,$$

where $\sigma \simeq \sigma'$ means that the infinite sequences differ at most in how many times each state is repeated consecutively, and $\sigma \approx_z \sigma'$ means that the sequences differ at most in values they assign to z. Informally, $\exists z F$ says that we do not care about the value of z, but F must hold. x' means the value of the variable x in the next state. v' means that v is true in the next state. Let $\sigma[i..]$ denote the suffix of σ starting at s_i. We use the reflexive **unless** operator:

$$\sigma, \eta \models F \textbf{ unless } G \text{ iff } \sigma[i..], \eta \models F, \text{ for all } i \geqslant 0, \text{ or}$$
$$\text{there exists } j \geqslant 0, \text{ such that } \sigma[j..], \eta \models G \text{ and } \sigma[k..], \eta \models F$$
$$\text{for all } k, 0 \leqslant k < j.$$

Notice that we have an initial semantics. A formula F is *valid* if it is true in any σ under any η. A formula is *closed* iff all occurrences of global variables in it are bound. A *model* of a closed formula F is any infinite state sequence in which F is true. Let $[\![F]\!]$ denote the set of all models of F.

An action formula can be interpreted in a pair of states. For example, $x' = x + 1$ is true at any pair of states (s, s') such that $s'(x) = s(x) + 1$. It may be treated as an analogon of a programming language assignment $x := x + 1$.

We call our logic a temporal logic of actions because it contains $'$ and the following abbreviations relating actions (cf. [10]). Let $x_1, \ldots, x_n$ be local (individual or proposition) variables, let $\bar{x}$ denote a finite set of local variables, and let A be an action formula containing at most the variables from $\bar{x}$ as free local variables. Then[2]

$$\textbf{unch}(x_1, \ldots, x_n) \equiv x_1' = x_1 \wedge \ldots \wedge x_n' = x_n,$$

$$[A]_{\bar{x}} \equiv A \vee \bigwedge_{x \in \bar{x}} x' = x \quad \text{(interpreted as "either action } A$$
$$\text{is executed or nothing happens"),}$$

$$\langle A \rangle_{\bar{x}} \equiv A \wedge \bigvee_{x \in \bar{x}} x' \neq x \quad \text{("action } A \text{ is executed"),}$$

$$En(A) \equiv \exists y_1, \ldots, y_n A(y_1/x_1', \ldots, y_n/x_n') \quad \text{("action } A \text{ is enabled"),}$$

where $x_1, \ldots, x_n$ are local variables occurring free in A, and $y_1, \ldots, y_n$ corresponding global variables, substituted for primed versions of the local variables.

The following abbreviations will also be needed:

$$WF_{\bar{x}}(A) \equiv \Diamond\Box En(\langle A \rangle_{\bar{x}}) \rightarrow \Box\Diamond\langle A \rangle_{\bar{x}},$$

$$SF_{\bar{x}}(A) \equiv \Box\Diamond En(\langle A \rangle_{\bar{x}}) \rightarrow \Box\Diamond\langle A \rangle_{\bar{x}}.$$

[2] Strictly speaking, we should write $\leftrightarrow$ in place of $=$ for every x that is a proposition variable in the definitions.

3. The system model and specification

A complete program or algorithm can be looked upon as a closed system. A *closed system* Π is similar to the fair state transition system that lies implicitly behind Lamport's TLA (also, cf. [12]): $\Pi = (V, St, \mathscr{T}, \Theta, \mathscr{W}, \mathscr{F})$ with

- V – the *variables* of Π, a finite subset of the set of local variables of the logic that Π may access.
  $V = O \cup I$ – the union of *observable* variables O and *internal* variables I, such that $O \cap I = \emptyset$.
- St – the set of all possible *states*, valuations of all local variables of the logic.
- $\mathscr{T}$ – a finite set of possible *actions* including an *idling action* τ_I. An action $\tau \in \mathscr{T}$ is a binary relation on states St, defined by an action formula ρ_τ that contains at most local variables from V as free local variables.
  $(s, s') \in \tau$ iff ρ_τ is true at (s, s').
  $\rho_{\tau_I} \equiv \bigwedge_{x \in V}(x' = x)$.
- Θ – a state formula containing at most the variables from V as free local variables, an *initial condition* for V.
- $\mathscr{W} = \{W_1, \ldots, W_m\}$ – *weak fairness* requirements, $W_i = (E_i, T_i) \in \mathscr{W}$, $E_i \subseteq T_i \subseteq \mathscr{T} - \{\tau_I\}$.
- $\mathscr{F} = \{F_1, \ldots, F_n\}$ – *strong fairness* requirements, $F_i = (E_i, T_i) \in \mathscr{F}$, $E_i \subseteq T_i \subseteq \mathscr{T} - \{\tau_I\}$.

A part of a complete program that communicates with other parts of it can be looked upon as an open system. It is almost like a closed one, except that set O contains a special local proposition variable μ. It is introduced for asserting whether an action is controlled by the program part represented by the system, i.e., whether it is an action of that program part, or is controlled, i.e., executed, by its environment. Thus, an *open system* is a fair state transition system $\Pi = (V, St, \mathscr{T}, \Theta, \mathscr{W}, \mathscr{F})$ with

- V – like in closed system, except that $\mu \in O$, and $O - \{\mu\} = O_c \cup O_u$ – the union of *changeable* observable variables O_c and *unchangeable* observable variables O_u, such that $O_c \cap O_u = \emptyset$.
- St – like in closed system.
- $\mathscr{T}$ – like in closed system, except that it includes beside τ_I also an *environment action* τ_E defined by

$$\rho_{\tau_E} \equiv \bigwedge_{x \in I \cup O_u} (x' = x) \wedge \neg \mu'.$$

For $\tau \neq \tau_E$, μ' appears as a conjunct of the action formula ρ_τ describing τ, e.g.,

$$\rho_{\tau_I} \equiv \bigwedge_{x \in V - \{\mu\}} (x' = x) \wedge \mu'.$$

- Θ – a state formula over $V - \{\mu\}$.

- $\mathcal{W}$ and $\mathcal{F}$ – like in closed system, except that for each pair (E_i, T_i) in the requirements, $E_i, T_i \subseteq \mathcal{T} - \{\tau_I, \tau_E\}$.

  Actions $\mathcal{T} - \{\tau_I, \tau_E\}$ are called *diligent*.

Internal variables are those that are neither of interest to the system user nor can be accessed by the environment. They are only needed to achieve desired behaviour of observable variables. In a closed system, observable variables are those that are required to have specific values during computation by the user. In open systems, we distinguish between two kinds of observable variables. Unchangeable variables are those for which a specifier of the system knows in advance, possibly without knowing the environment of the system, that they cannot be changed by the environment, whereas their values are important for the user. Changeable variables are those that the environment may possibly change. Variable μ cannot be changed by the environment, but is treated as an observable one because its value is important when the system is executed concurrently with another open system.

A *computation* of (open or closed) system Π is any infinite sequence σ of states such that for every pair of states (s_i, s_{i+1}), $(s_i, s_{i+1}) \in \tau$, $\tau \in \mathcal{T}$, initial condition is true at s_0, and such that it satisfies all the fairness requirements.

We know that $\mu \notin V$ and $\tau_E \notin \mathcal{T}$ in a closed system. Then we can define the following for open and closed systems. We say that an action $\tau \in \mathcal{T} - \{\tau_I, \tau_E\}$ is *enabled* at s_i iff there exists a state $s \in St$ such that $\langle \rho_\tau \rangle_{V-\mu}$ is true at (s_i, s), i.e., iff $En(\langle \rho_\tau \rangle_{V-\mu})$ is true at s_i. It is *executed* at state s_i of σ iff $\langle \rho_\tau \rangle_{V-\mu}$ is true at (s_i, s_{i+1}). A set of actions T is enabled at s_i iff some action $\tau \in T$ is enabled at s_i. A set of actions $T \subseteq \mathcal{T} - \{\tau_I, \tau_E\}$ is executed at s_i iff some action $\tau \in T$ is executed at s_i. Notice that only an action that changes some variables at a state is treated as executed at the state.

The meaning of weak fairness requirements is equal for closed and open systems. A state sequence σ is *weakly fair* with respect to every $(E_i, T_i) \in \mathcal{W}$ iff: if E_i is enabled continuously from some state on in σ, then T_i is executed infinitely often in σ. For closed systems, a state sequence σ is *strongly fair* with respect to every $(E_i, T_i) \in \mathcal{F}$ iff: if E_i is enabled infinitely often in σ, then T_i is executed infinitely often in σ.

The meaning of strong fairness requirements of an open system is a bit different in order that they be realizable (e.g., see [1]).

Let $\mathcal{C}(\Pi)$ denote the set of computations of open or closed system Π. Let $\rho \approx_I \tau$ mean that infinite state sequences ρ and τ differ at most in values they assign to the variables from I. Let $[\![\Pi]\!]$ denote the set of *behaviours* of Π. We define: $[\![\Pi]\!] = \{\sigma \in St^\omega | \exists \rho, \tau \in St^\omega (\sigma \simeq \rho \wedge \rho \approx_I \tau \wedge \tau \in \mathcal{C}(\Pi))\}$. Thus, as usual, the values of internal variables are abstracted away, and it is the behaviours that will be described by our TLA. Also, the set of behaviours is *stuttering closed*, i.e., closed under finite repetition of states. Stuttering-closedness of sets of behaviours allows for hierarchical verification and refinement (e.g., [1]).

A closed formula F of our TLA is *stuttering insensitive* iff $[\![F]\!]$ is stuttering closed. The basic rule for a formula of our TLA to be stuttering insensitive is that it must not contain the operator $\bigcirc$ (cf. [16]). It is only retained for axiomatisation. A *specification*

is any closed formula of our TLA that is stuttering insensitive. A specification S is a *specification of a (closed or open) system* Π with observable variables O iff its only free variables are from O and $[\![\Pi]\!] \subseteq [\![S]\!]$. We then say that Π *satisfies* S, or that S is *valid in* Π, and denote it by $\Pi \models S$. If $[\![\Pi]\!] = [\![S]\!]$, then S is a *precise specification of* Π.

A straightforward way to write a precise specification of a closed or open system is to directly describe its meaning with a TLA formula. Let $\Pi = (V, St, \mathscr{T}, \Theta, \mathscr{W}, \mathscr{F})$ be an open system not containing any internal variables, such that for each $(E_i, T_i) \in \mathscr{W}$, $E_i = T_i$, and likewise for $\mathscr{F}$. Then the TLA formula is as follows:

$$S_\Pi \equiv \Theta \wedge \Box [\bigvee_{\tau \in \mathrm{T} - \{\tau_I\}} \rho_\tau]_{V-\{\mu\}} \wedge \bigwedge_{(E_i, T_i) \in \mathscr{W}} WF_{V-\{\mu\}}(\bigvee_{\tau \in E_i} \rho_\tau)$$

$$\wedge \bigwedge_{(E_i, T_i) \in \mathscr{F}} SF_{V-\{\mu\}}(\bigvee_{\tau \in E_i} \rho_\tau).$$

Alternatively, if we have a TLA formula $I \wedge \Box[N]_{\bar{x}} \wedge WF \wedge SF$, where I is a state formula, N a disjunction of action formulae, and WF (SF) a conjunction of formulae $WF_{\bar{x}}(N_i)$ ($SF_{\bar{x}}(N_i)$, respectively) for some disjuncts N_i of N, then, if the disjuncts of N are properly formed, the formula can be thought of as a precise specification of such an open system with I taken for Θ, $\bar{x}$ for $V - \{\mu\}$, with nonidling actions described by the disjuncts of N, and with weak and strong fairness requirements consisting of pairs of the sets of actions described by action formulae N_i occurring in WF and SF, respectively. We shall write our precise specifications in this "canonical" way.

Different operations can be performed on systems. Let the result of *renaming* a variable in an open or closed system be the system with all occurrences of the variable substituted by a new variable. Let the result of *hiding* an observable variable in an open or closed system be the system with the variable moved from the set O to I (internal variables are already hidden).

Two open systems are *compatible* iff the conjunction of their initial conditions does not imply **false** and no unchangeable variable of one system can be changed by the other.

Let $\Pi_i = (V_{\Pi_i}, St_{\Pi_i}, \mathscr{T}_{\Pi_i}, \Theta_{\Pi_i}, \mathscr{W}_{\Pi_i}, \mathscr{F}_{\Pi_i})$, $i = 1, 2$, be compatible open systems. *Parallel composition* of Π_1 and Π_2, denoted $\Pi_1 \parallel \Pi_2$, is the closed system, obtained by first renaming internal variables and μ in such a way that for the newly obtained sets of internal variables, $I_{\Pi_i} \cap I_{\Pi_j} = \emptyset$, $I_{\Pi_i} \cap O_{\Pi_j} = \emptyset$, for $i = 1, 2$, $j = 1, 2$, $i \neq j$, and $\mu_{\Pi_1} \neq \mu_{\Pi_2}$, and then hiding μ_{Π_1} and μ_{Π_2} in the fair transition system $\Pi = (V, St, \mathscr{T}, \Theta, \mathscr{W}, \mathscr{F})$ with the renamed variables, where

- $V = V_{\Pi_1} \cup V_{\Pi_2}$, $I = I_{\Pi_1} \cup I_{\Pi_2}$, and $O = O_{\Pi_1} \cup O_{\Pi_2}$.
- St – the set of all possible valuations of local variables of TLA.
- $\mathscr{T}$ – the set of actions, obtained from $\mathscr{T}_{\Pi_1}$ and $\mathscr{T}_{\Pi_2}$. $\tau \in \mathscr{T}$ iff it is described by:

$$\rho_\tau = \rho_{\tau_{\Pi_i}} \wedge \rho_{\tau_{\mathrm{E}\Pi_j}} \wedge \bigwedge_{x \in O_{c\Pi_j} - O_{c\Pi_i}} (x' = x),$$

for $i, j = 1, 2$, $i \neq j$, where $\tau_{\Pi_i} \in \mathscr{T}_{\Pi_i} - \{\tau_{\mathrm{E}\Pi_i}\}$.

- $\Theta = \Theta_{\Pi_1} \wedge \Theta_{\Pi_2}$.
- $\mathscr{W} = \{W_1, \ldots, W_m\}$ – weak fairness requirements.
  Every $W_i = (E_i, T_i) \in \mathscr{W}$ consists of sets $E_i, T_i \subseteq \mathscr{T}$. $W_i = (E_i, T_i)$ is in $\mathscr{W}$ iff there exists $(E_{\Pi_j}, T_{\Pi_j}) \in \mathscr{W}_{\Pi_j}$ for $j = 1$ or $j = 2$ such that for all $\tau \in \mathscr{T}$:
  - $\tau \in E_i \Leftrightarrow \exists \tau_{\Pi_j} \in E_{\Pi_j}(\rho_\tau \equiv \rho_{\tau_{\Pi_j}} \wedge \rho^r)$ where ρ^r is the rest of the description of τ, and
  - $\tau \in T_i \Leftrightarrow \exists \tau_{\Pi_j} \in T_{\Pi_j}(\rho_\tau \equiv \rho_{\tau_{\Pi_j}} \wedge \rho^r)$, where ρ^r is the rest of the description of τ.
- $\mathscr{F} = \{F_1, \ldots, F_n\}$ – strong fairness requirements.
  Every $F_i = (E_i, T_i) \in \mathscr{F}$ consists of sets $E_i, T_i \subseteq \mathscr{T}$. They are obtained from $\mathscr{F}_{\Pi_j}$, $j = 1, 2$, like the weak fairness requirements.

In parallel composition, diligent actions of a component execute *interleaved* with the actions of the other one. Parallel composition of more than two open systems could be defined analogously.

Using semantic arguments, it can be proved that possible behaviours of the results of the operations are such that the following proof rules are valid (e.g., [8]).

Let $\|_{i=1}^n \Pi_i$ denote parallel composition of compatible open systems $\Pi_1, \ldots, \Pi_n$, $n \geqslant 2$, each Π_i containing the set of observable variables O_i. Then the *parallel composition rule* is valid:

$$\frac{\Pi_i(O_i) \models S_i, \quad i = 1, \ldots, n}{\|_{i=1}^n \Pi_i(O_i) \models \exists\mu(\exists\mu_1, \ldots, \mu_n(\bigwedge_{i=1}^n S_i[\mu_i/\mu] \wedge Uneq \wedge Unch \wedge Who) \wedge \Box\mu)}$$

where

$$Unch(\mu_i) \equiv \bigwedge_{u \in (\cup_{j=1}^n O_{c_j}) - O_{c_i}} \Box(\mu_i' \to u' = u), \quad i = 1, \ldots, n,$$

$$Unch \equiv \bigwedge_{i=1}^n Unch(\mu_i), \quad Uneq \equiv \Box \bigwedge_{\substack{i,j=1 \\ i \neq j}}^n \neg(\mu_i \wedge \mu_j), \quad Who \equiv \Box(\mu \leftrightarrow \bigvee_{i=1}^n \mu_i).$$

Compatibility of the components ensures that the parallel composition has a nonempty set of behaviours. Clearly, the *renaming rule*, where $\Pi[v/u]$ denotes Π with an observable variable u renamed for v and $S[v/u]$ denotes S with v substituted for u, the *hiding rule* with $\Pi \backslash x$ denoting the result of hiding an observable variable x in Π, the *consequence rule*, and the *conjunction rule* are valid, in order of appearance:

$$\frac{\Pi \models S}{\Pi[v/u] \models S[v/u]} \qquad \frac{\Pi \models S}{\Pi \backslash x \models \exists x S} \qquad \frac{\Pi \models S_1, \quad S_1 \to S_2}{\Pi \models S_2} \qquad \frac{\Pi \models S_1, \quad \Pi \models S_2}{\Pi \models S_1 \wedge S_2}$$

It could be proved that the last five rules without the conjunction rule together with precise specifications of open systems as axioms are complete for proving properties of systems constructed from the open systems using the operations, relative to the

completeness of a TLA proof system (cf. [14]). The conjunction rule is only needed for convenience.

4. Specification of the closing procedure

4.1. A design specification

A design specification of the closing procedure must be a precise specification. Let *CXT* and *CHN* denote an open system, representing a context at a host, and one representing a unidirectional lossy channel, respectively. The procedure is specified as a parallel composition of such systems with properly renamed observable variables:

$$CLS \equiv CXT_A \parallel CHN_{AB} \parallel CHN_{BA} \parallel CXT_B$$

where

$$CXT_A \equiv CXT[cxt_A/cxt, z_{OA}/z_O, z_{IA}/z_I, st_{OA}/st_O, st_{IA}/st_I, tmr_A/tmr, rq_A/rq]$$

is the context at host A, $CHN_{AB} \equiv CHN[z_{OA}/z_S, z_{IB}/z_R]$ is the channel for sending messages from A to B, and symmetrically for CXT_B, CHN_{BA}.

Let ε denote an empty sequence, $\bullet$ concatenation of sequences, $[m]$ a sequence with the only element m, $Head(z)$ the first element of a nonempty sequence z, and $Tail(z)$ the rest of it.

The following messages are sent in *CLS*:

$$M = \{(C,W,S),(C,R,S),(C,W,R,S),(C),$$
$$(C,W),(C,R),(C,W,R),(C,W,R,E),(D)\}.$$

C means that the message is a CNTL packet. (D) represents the DIAG packet saying that the context at a host is not active. If a message from M contains S (for SREQ), W (for WCLOSE), R (for RCLOSE), or E (for END), it means that the corresponding bit would be set in the XTP packet it represents.

Beside μ, the open system *CHN* contains two observable changeable variables, z_S and z_R, both ranging over sequences of the messages from M. The design specification of *CHN* is as follows:

$$CHN \models S_{CHN}$$

where

$$S_{CHN} \equiv z_S = z_R = \varepsilon \wedge \Box[((loss(z_S,z_R) \vee \bigvee_{m\in M} pass(m,z_S,z_R)) \wedge \mu') \vee \neg\mu']_{\{z_S,z_R\}}$$
$$\wedge WF_{\{z_S,z_R\}}((loss(z_S,z_R) \vee \bigvee_{m\in M} pass(m,z_S,z_R)) \wedge \mu')$$
$$\wedge \bigwedge_{M_1\in 2^M-\emptyset} SF_{\{z_S,z_R\}}(\bigvee_{m\in M_1} (pass(m,z_S,z_R) \wedge \mu'))$$

and

$$loss(z_S, z_R) \equiv z_S \neq \varepsilon \wedge z_S' = Tail(z_S) \wedge \mathsf{unch}(z_R),$$

$$pass(m, z_S, z_R) \equiv z_S \neq \varepsilon \wedge Head(z_S) = m \wedge z_S' = Tail(z_S) \wedge z_R' = z_R \bullet [m].$$

Local variable z_S represents the sender's end of the channel, and z_R the receiver's end of the channel. Initially, the channel is empty. One can imagine that the environment of the channel can put messages into z_S. The weak fairness requirement ensures that the channel takes each message from z_S and nondeterministically either loses it (*loss*) or puts it into z_R (*pass*), where it waits to be removed by the environment. The strong fairness requirement ensures that messages are not lost every time if sent sufficiently often on z_S by the environment. Although *CHN* is an open system, the strong fairness requirements are like in closed systems, because we know the *CXT* systems that make up the environment of *CHN* in *CLS* cannot disable a *pass* action.

The open system *CXT* contains beside μ variables $V_{CXT} = \{cxt, st_O, st_I, z_O, z_I, tmr, rq\}$. Local variables z_O and z_I range over sequences of the messages from M and represent the sender's end of the context's outgoing channel and the receiver's end of its incoming channel, respectively. Let $x \in T$ mean that x ranges over the values from T. The range of the other variables is as follows: $cxt \in \{act, null\}$, $st_O \in \{act, wcl, inact\}$, $st_I \in \{act, rcl, dlv, cls, inact\}$, $rq \in \{empty, full, dsc\}$, and $tmr \in \{off, set\}$.

The variable *cxt* indicates the state of the context. st_O describes the state of the output data stream, st_I the state of the input data stream of the context, and *rq* the state of the data receive queue. There is one timer for the context, represented by *tmr*.

We shall now provide a design specification of *CXT*, followed by an explanation.

Let the following action formulae represent sending of a message $m \in M$ on z_O and receipt of a message $m \in M$ from z_I, respectively:

$$send(m, z_O) \equiv z_O' = z_O \bullet [m],$$

$$rec(m, z_I) \equiv z_I \neq \varepsilon \wedge Head(z_I) = m \wedge z_I' = Tail(z_I).$$

Let $recE(z_I) \equiv rec((C, W, R, E), z_I)$, $recD(z_I) \equiv rec((D), z_I)$, let $recC(z_I)$ denote receipt of any CNTL packet with SREQ not set, and $recS(z_I)$ receipt of any packet with SREQ set.

We take all the variables from V_{CXT} to be observable. They all seem interesting to us for expressing correctness criteria. Since communication is only possible over the channels, only z_O and z_I are changeable. Here is our design specification S_{CXT}, $CXT \models S_{CXT}$:

$$\begin{aligned} S_{CXT} \equiv\ & cxt = st_O = st_I = act \wedge z_O = z_I = \varepsilon \wedge tmr = off \\ & \wedge (rq = empty \vee rq = full) \\ & \wedge \Box[((Lcl \vee Tout \vee Dlr \vee Cnf \vee Rec) \wedge \mu') \\ & \quad \vee (\mathsf{unch}(cxt, st_O, st_I, tmr, rq) \wedge \neg \mu')]_{V_{CXT}} \end{aligned}$$

$$\wedge WF_{V_{CXT}}(Tout \wedge \mu') \wedge WF_{V_{CXT}}(Dlr \wedge \mu')$$
$$\wedge WF_{V_{CXT}}(Cnf \wedge \mu') \wedge WF_{V_{CXT}}(Rec \wedge \mu'),$$

where

$$Lcl \equiv LWc1 \vee LWc2 \vee LWc3 \vee LRc1 \vee LRc2,$$
$$Tout \equiv TtO1 \vee TtO2 \vee TtI1 \vee TtI2,$$
$$Dlr \equiv Rdrq1 \vee Rdrq2,$$
$$Cnf \equiv Cnf1 \vee Cnf2, \text{ and}$$
$$Rec \equiv RecSR \vee RecSW \vee RecSWR \vee RecCal \vee RecCR \vee RecCW \vee RecCWR$$
$$\vee RecED \vee RecC \vee RecS \vee RecD \text{ with}$$
$$RecSR \equiv RecSr1 \vee RecSr2 \vee RecSr3 \vee RecSr4$$
$$RecSW \equiv RecSw1 \vee RecSw2 \vee \cdots \vee RecSw12$$
$$RecSWR \equiv RecSwr1 \vee RecSwr2 \vee RecSwr3 \vee RecSwr4$$
$$RecCR \equiv RecCr1 \vee RecCr2 \vee RecCr3$$
$$RecCW \equiv RecCw1 \vee RecCw2 \vee RecCw3 \vee RecCw4 \vee RecCw5 \vee RecCw6$$
$$RecCWR \equiv RecCwr1 \vee RecCwr2 \vee RecCwr3 \vee RecCwr4$$

are disjunctions of the following action formulae describing possible diligent actions of CXT, not yet accompanied by μ'.

Local wclosing:

{just local wclosing}

$$LWc1 \equiv cxt = act \wedge st_O = act \wedge (st_I = act \vee st_I = dlv \vee st_I = cls)$$
$$\wedge send((C,W,S),z_O) \wedge st'_O = wcl \wedge tmr' = set \wedge \mathsf{unch}(cxt,z_I,st_I,rq)$$

{just local wclosing}

$$LWc2 \equiv cxt = act \wedge st_O = act \wedge (st_I = rcl \vee st_I = inact)$$
$$\wedge send((C,W,R,S),z_O) \wedge st'_O = wcl \wedge tmr' = set \wedge \mathsf{unch}(cxt,z_I,st_I,rq)$$

{local wclosing + local rclosing}

$$LWc3 \equiv cxt = act \wedge st_O = act \wedge st_I = act$$
$$\wedge send((C,W,R,S),z_O) \wedge rq' = dsc \wedge st'_O = wcl \wedge st'_I = rcl \wedge tmr' = set$$
$$\wedge \mathsf{unch}(cxt,z_I)$$

Local rclosing:

{just local rclosing}

$$LRc1 \equiv cxt = act \wedge st_O = act \wedge st_I = act$$
$$\wedge send((C,R,S),z_O) \wedge rq' = dsc \wedge st'_I = rcl \wedge tmr' = set$$
$$\wedge \mathsf{unch}(cxt,z_I,st_O)$$

{just local rclosing}

$LRc2 \equiv cxt = act \wedge (st_O = wcl \vee st_O = inact) \wedge st_I = act$
$\wedge send((C,W,R,S),z_O) \wedge rq' = dsc \wedge st_I' = rcl \wedge tmr' = set$
$\wedge \mathsf{unch}(cxt,z_I,st_O)$

Receipt of (*CNTL, RCLOSE, SREQ*):

{just wclose and answer to RCLOSE}

$RecSr1 \equiv rec((C,R,S),z_I) \wedge cxt = act \wedge (st_I = act \vee st_I = dlv \vee st_I = cls)$
$\wedge send((C,W),z_O) \wedge st_O' = inact \wedge tmr' = off \wedge \mathsf{unch}(cxt,st_I,rq)$

{wclose, also local rclosing, and answer to RCLOSE}

$RecSr2 \equiv rec((C,R,S),z_I) \wedge cxt = act \wedge st_I = act$
$\wedge send((C,W,R,S),z_O) \wedge rq' = dsc \wedge st_O' = inact \wedge st_I' = rcl$
$\wedge tmr' = set \wedge \mathsf{unch}(cxt)$

{just wclose and answer to RCLOSE}

$RecSr3 \equiv rec((C,R,S),z_I) \wedge cxt = act \wedge st_I = rcl$
$\wedge send((C,W,R),z_O) \wedge st_O' = inact \wedge \mathsf{unch}(cxt,st_I,tmr,rq)$

{wclose, answer to RCLOSE; also end}

$RecSr4 \equiv rec((C,R,S),z_I) \wedge cxt = act \wedge st_I = inact$
$\wedge send((C,W,R,E),z_O) \wedge st_O' = inact \wedge tmr' = off \wedge cxt' = null$
$\wedge \mathsf{unch}(st_I,rq)$

Receipt of (*CNTL, WCLOSE, SREQ*):

{pass to delivering of *rq* and answer to WCLOSE}

$RecSw1 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge st_O = act \wedge st_I = act \wedge rq = full$
$\wedge send((C),z_O) \wedge st_I' = dlv \wedge \mathsf{unch}(cxt,st_O,tmr,rq)$

{pass to delivering of *rq* and answer to WCLOSE}

$RecSw2 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge (st_O = wcl \vee st_O = inact)$
$\wedge st_I = act \wedge rq = full$
$\wedge send((C,W),z_O) \wedge st_I' = dlv \wedge \mathsf{unch}(cxt,st_O,tmr,rq)$

{*rq* is empty; get wait for confirmation and answer to WCLOSE}

$RecSw3 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge st_O = act \wedge st_I = act \wedge rq = empty$
$\wedge send((C),z_O) \wedge st_I' = cls \wedge \mathsf{unch}(cxt,st_O,tmr,rq)$

{*rq* is empty; get wait for confirmation and answer to WCLOSE}

$RecSw4 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge (st_O = wcl \vee st_O = inact)$
$\wedge st_I = act \wedge rq = empty$
$\wedge send((C,W),z_O) \wedge st_I' = cls \wedge \mathsf{unch}(cxt,st_O,tmr,rq)$

{*rq* is *empty*; get wait for confirmation, also wclosing, and answer to WCLOSE}

$RecSw5 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge st_O = act \wedge st_I = act \wedge rq = empty$
$\wedge send((C,W,S),z_O) \wedge st_O' = wcl \wedge st_I' = cls \wedge tmr' = set$
$\wedge \mathsf{unch}(cxt,rq)$

{just rclose and answer to WCLOSE}

$RecSw6 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge st_O = act \wedge (st_I = rcl \vee st_I = inact)$
$\wedge send((C,R),z_O) \wedge st_I' = inact \wedge tmr' = off \wedge \mathsf{unch}(cxt,st_O,rq)$

{just rclose and answer to WCLOSE}

$RecSw7 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge st_O = wcl \wedge (st_I = rcl \vee st_I = inact)$
$\wedge send((C,W,R),z_O) \wedge st'_I = inact \wedge \mathsf{unch}(cxt, st_O, tmr, rq)$

{rclose and answer to WCLOSE; also end}

$RecSw8 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge st_O = inact \wedge (st_I = rcl \vee st_I = inact)$
$\wedge send((C,W,R,E),z_O) \wedge st'_I = inact \wedge tmr' = off \wedge cxt' = null$
$\wedge \mathsf{unch}(st_O, rq)$

{rclose, also wclosing, and answer to WCLOSE}

$RecSw9 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge st_O = act \wedge (st_I = rcl \vee st_I = inact)$
$\wedge send((C,W,R,S),z_O) \wedge st'_O = wcl \wedge st'_I = inact \wedge tmr' = set$
$\wedge \mathsf{unch}(cxt, rq)$

{just answer to WCLOSE}

$RecSw10 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge st_O = act \wedge (st_I = dlv \vee st_I = cls)$
$\wedge send((C),z_O) \wedge \mathsf{unch}(cxt, st_O, st_I, tmr, rq)$

{just answer to WCLOSE}

$RecSw11 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge (st_O = wcl \vee st_O = inact)$
$\wedge (st_I = dlv \vee st_I = cls)$
$\wedge send((C,W),z_O) \wedge \mathsf{unch}(cxt, st_O, st_I, tmr, rq)$

{wclosing and answer to WCLOSE}

$RecSw12 \equiv rec((C,W,S),z_I) \wedge cxt = act \wedge st_O = act \wedge (st_I = dlv \vee st_I = cls)$
$\wedge send((C,W,S),z_O) \wedge st'_O = wcl \wedge tmr' = set \wedge \mathsf{unch}(cxt, st_I, rq)$

Read *rq*:

$Rdrq1 \equiv cxt = act \wedge st_I = act \wedge rq = full$
$\wedge rq' = empty \wedge \mathsf{unch}(cxt, z_O, z_I, st_O, st_I, tmr)$

$Rdrq2 \equiv cxt = act \wedge st_I = dlv \wedge rq = full$
$\wedge st'_I = cls \wedge rq' = empty \wedge \mathsf{unch}(cxt, z_O, z_I, st_O, tmr)$

Rclose confirmed:

$Cnf1 \equiv cxt = act \wedge (st_O = act \vee st_O = wcl) \wedge st_I = cls$
$\wedge st'_I = inact \wedge \mathsf{unch}(cxt, z_O, z_I, st_O, tmr, rq)$

$Cnf2 \equiv cxt = act \wedge st_O = inact \wedge st_I = cls$
$\wedge st'_I = inact \wedge cxt' = null \wedge \mathsf{unch}(z_O, z_I, st_O, tmr, rq)$

Receipt of (*CNTL, WCLOSE, RCLOSE, SREQ*):

{wclose, rclose, answer to WCLOSE, RCLOSE; also end}

$RecSwr1 \equiv rec((C,W,R,S),z_I) \wedge cxt = act \wedge (st_I = rcl \vee st_I = inact)$
$\wedge send((C,W,R,E),z_O) \wedge st'_O = inact \wedge st'_I = inact \wedge tmr' = off$
$\wedge cxt' = null \wedge \mathsf{unch}(rq)$

{wclose, pass to delivering, answer to WCLOSE, RCLOSE}

$RecSwr2 \equiv rec((C,W,R,S),z_I) \wedge cxt = act \wedge st_I = act \wedge rq = full$
$\wedge send((C,W),z_O) \wedge st'_O = inact \wedge st'_I = dlv \wedge tmr' = off$
$\wedge \mathsf{unch}(cxt, rq)$

{wclose, get wait for confirmation, answer to WCLOSE, RCLOSE}

$$RecSwr3 \equiv rec((C,W,R,S),z_I) \wedge cxt = act \wedge st_I = act \wedge rq = empty \wedge send((C,W),z_O) \wedge st'_O = inact \wedge st'_I = cls \wedge tmr' = off \wedge \mathsf{unch}(cxt,rq)$$

{wclose, continue rclosing on a past request, answer to WCLOSE, RCLOSE}

$$RecSwr4 \equiv rec((C,W,R,S),z_I) \wedge cxt = act \wedge (st_I = dlv \vee st_I = cls) \wedge send((C,W),z_O) \wedge st'_O = inact \wedge tmr' = off \wedge \mathsf{unch}(cxt,st_I,rq)$$

tmr **runs out:**

$$TtO1 \equiv cxt = act \wedge st_O = wcl \wedge (st_I = act \vee st_I = dlv \vee st_I = cls) \wedge send((C,W,S),z_O) \wedge \mathsf{unch}(cxt,z_I,st_O,st_I,tmr,rq)$$

$$TtO2 \equiv cxt = act \wedge st_O = wcl \wedge (st_I = rcl \vee st_I = inact) \wedge send((C,W,R,S),z_O) \wedge \mathsf{unch}(cxt,z_I,st_O,st_I,tmr,rq)$$

$$TtI1 \equiv cxt = act \wedge st_O = act \wedge st_I = rcl \wedge send((C,R,S),z_O) \wedge \mathsf{unch}(cxt,z_I,st_O,st_I,tmr,rq)$$

$$TtI2 \equiv cxt = act \wedge st_O = inact \wedge st_I = rcl \wedge send((C,W,R,S),z_O) \wedge \mathsf{unch}(cxt,z_I,st_O,st_I,tmr,rq)$$

Receipt of (*CNTL*)**:**

$$RecCal \equiv rec((C),z_I) \wedge cxt = act \wedge (st_O = act \vee st_O = wcl \vee st_I = act \vee st_I = rcl \vee st_I = dlv \vee st_I = cls) \wedge \mathsf{unch}(cxt,z_O,st_O,st_I,tmr,rq)$$

Receipt of (*CNTL, RCLOSE*)**:**

$$RecCr1 \equiv rec((C,R),z_I) \wedge cxt = act \wedge (st_I = act \vee st_I = dlv \vee st_I = cls) \wedge st'_O = inact \wedge tmr' = off \wedge \mathsf{unch}(cxt,z_O,st_I,rq)$$

$$RecCr2 \equiv rec((C,R),z_I) \wedge cxt = act \wedge st_I = rcl \wedge st'_O = inact \wedge \mathsf{unch}(cxt,z_O,st_I,tmr,rq)$$

$$RecCr3 \equiv rec((C,R),z_I) \wedge cxt = act \wedge st_I = inact \wedge st'_O = inact \wedge tmr' = off \wedge cxt' = null \wedge \mathsf{unch}(z_O,st_I,rq)$$

Receipt of (*CNTL, WCLOSE*)**:**

$$RecCw1 \equiv rec((C,W),z_I) \wedge cxt = act \wedge st_I = act \wedge rq = empty \wedge st'_I = cls \wedge \mathsf{unch}(cxt,z_O,st_O,tmr,rq)$$

$$RecCw2 \equiv rec((C,W),z_I) \wedge cxt = act \wedge st_I = act \wedge rq = full \wedge st'_I = dlv \wedge \mathsf{unch}(cxt,z_O,st_O,tmr,rq)$$

$$RecCw3 \equiv rec((C,W),z_I) \wedge cxt = act \wedge (st_I = dlv \vee st_I = cls) \wedge \mathsf{unch}(cxt,z_O,st_O,st_I,tmr,rq)$$

$$RecCw4 \equiv rec((C,W),z_I) \wedge cxt = act \wedge st_O = act \wedge (st_I = rcl \vee st_I = inact) \wedge st'_I = inact \wedge tmr' = off \wedge \mathsf{unch}(cxt,z_O,st_O,rq)$$

$$RecCw5 \equiv rec((C,W),z_I) \wedge cxt = act \wedge st_O = wcl \wedge (st_I = rcl \vee st_I = inact) \wedge st'_I = inact \wedge \mathsf{unch}(cxt,z_O,st_O,tmr,rq)$$

$$RecCw6 \equiv rec((C,W),z_I) \wedge cxt = act \wedge st_O = inact \wedge (st_I = rcl \vee st_I = inact) \wedge st'_I = inact \wedge tmr' = off \wedge cxt' = null \wedge \mathsf{unch}(z_O,st_O,rq)$$

Receipt of (*CNTL*, *WCLOSE*, *RCLOSE*):

$$RecCwr1 \equiv rec((C,W,R),z_I) \wedge cxt = act \wedge (st_I = rcl \vee st_I = inact) \wedge st'_O = inact \wedge st'_I = inact \wedge tmr' = off \wedge cxt' = null \wedge \mathsf{unch}(z_O, rq)$$

$$RecCwr2 \equiv rec((C,W,R),z_I) \wedge cxt = act \wedge st_I = act \wedge rq = empty \wedge st'_O = inact \wedge st'_I = cls \wedge tmr' = off \wedge \mathsf{unch}(cxt, z_O, rq)$$

$$RecCwr3 \equiv rec((C,W,R),z_I) \wedge cxt = act \wedge st_I = act \wedge rq = full \wedge st'_O = inact \wedge st'_I = dlv \wedge tmr' = off \wedge \mathsf{unch}(cxt, z_O, rq)$$

$$RecCwr4 \equiv rec((C,W,R),z_I) \wedge cxt = act \wedge (st_I = dlv \vee st_I = cls) \wedge st'_O = inact \wedge tmr' = off \wedge \mathsf{unch}(cxt, z_O, st_I, rq)$$

Receipt of END or DIAG when context active:

$$RecED \equiv (recE(z_I) \vee recD(z_I)) \wedge cxt = act \wedge (st_I = rcl \vee st_I = inact) \wedge st'_O = inact \wedge st'_I = inact \wedge tmr' = off \wedge cxt' = null \wedge \mathsf{unch}(z_O, rq)$$

Receipt of C when no active context:

$$RecC \equiv recC(z_I) \wedge cxt = null \wedge \mathsf{unch}(cxt, z_O, st_O, st_I, tmr, rq)$$

Receipt of SREQ when no active context:

$$RecS \equiv recS(z_I) \wedge cxt = null \wedge send((D), z_O) \wedge \mathsf{unch}(cxt, st_O, st_I, tmr, rq)$$

Receipt of DIAG when no active context:

$$RecD \equiv recD(z_I) \wedge cxt = null \wedge \mathsf{unch}(cxt, z_O, st_O, st_I, tmr, rq)$$

Initially, the context, the input and output data stream are *act*ive, both channel ends are empty, the data from the receive queue have either already been read ($rq = empty$), or there are still some data in it ($rq = full$).

The state st_O of the output data stream changes as follows. When it is *act*ive, it can start closing. The *act*ive context sends a CNTL-request packet containing W, *set*s timer *tmr*, and st_O passes to *wcl*osing (see actions *LWc*1, *LWc*2, *LWc*3 and *RecSw*5, *RecSw*9, *RecSw*12 in the specification). When $st_O = wcl$, *tmr* runs and a timeout can occur (actions *TtO*1, *TtO*2). In this case, the context just sends a CNTL-request packet with W and *set*s *tmr* again. For any value of st_O, if the *act*ive context receives a packet containing R, E, or D, st_O closes (or remains closed, respectively), i.e., gets *inact*ive (actions *RecSR*, *RecSWR*, *RecCR*, *RecCWR*, *RecED*). The context includes W in each CNTL packet sent when st_O is equal to *wcl* or *inact*.

The input data stream behaves as follows. When st_I is *act*ive, it can start closing. In this case, the *act*ive context discards the data receive queue (rq gets the value *dsc*), sends a CNTL-request packet containing R, *set*s *tmr*, and changes st_I to *rcl*osing (actions *LWc*3, *LRc*1, *LRc*2, *RecSr*2). When $st_I = rcl$, *tmr* runs and a timeout can occur (actions *TtO*2, *TtI*1, *TtI*2). In this case, the context just sends a CNTL-request

packet with R and *sets tmr* again. If the *act*ive context receives a packet with W, E, or D when $st_I = rcl$ or $st_I = inact$, st_I closes, i.e., passes to *inact*ive, or remains closed, respectively (actions *RecSw*6–*RecSw*9, *RecSwr*1, *RecCw*4–*RecCw*6, *RecCwr*1, *RecED*).

It is possible that the *act*ive context receives a packet with W, when st_I is *act*ive. In this case, st_I passes to *cls* ("closing") if *rq* is already *empty* (actions *RecSw*3–*RecSw*5, *RecSwr*3, *RecCw*1, *RecCwr*2), or to *dlv* ("deliver") if *rq* is *full* (actions *RecSw*1, *RecSw*2, *RecSwr*2, *RecCw*2, *RecCwr*3). When $st_I = dlv$, an action (*Rdrq*2) must happen that reads data from *rq*, i.e., sets it to *empty*, and st_I passes to *cls*. The data can also be read from *rq* when st_I is still *act*ive (action *Rdrq*1). At *cls* state, the input data stream waits for a confirmation from the application (action *Cnf*1 or *Cnf*2), and then just changes st_I to *inact*ive.

The context includes R in each CNTL packet sent when st_I is equal to *rcl* or *inact*.

When *tmr* is *set*, it is stopped, i.e., assigned the value *off*, by the context, when it passes to *null* or when its output or input data stream passes to *inact*, but only if *tmr* does not need to run further for any of the streams. We allow for premature timeouts, i.e., a CNTL-request packet may be retransmitted before a response to the previous request can come.

The input and output data stream of the context may begin locally closing simultaneously (action *LWc*3). Also, it is possible that a data stream starts closing just when the context receives a CNTL-request. In this case, a CNTL-request may be sent in response (actions *RecSr*2, *RecSw*5, *RecSw*9, *RecSw*12).

The context passes to *null* upon receiving a packet containing E or D (action *RecED*), and whenever the input and output data stream get *inact*ive (*RecSr*4, *RecSw*8, *Cnf*2, *RecSwr*1, *RecCr*3, *RecCw*6, *RecCwr*1). The context passes to *null* in the same atomic action, in which the data streams both got *inact*ive.

We have a group of actions for the receipt of each possible message from M in the specification. This is to ensure that there are no unspecified receptions in *CXT*, i.e., at any state of *CXT*, an action should be enabled that can receive the message currently residing at the head of z_I. Notice that at any time, only some actions for receiving exactly one possible element of M are enabled. The weak fairness requirement for *Rec* in S_{CXT} together with the absence of unspecified receptions is intended to guarantee that each message at the head of z_I is eventually removed from z_I.

In S_{CXT}, there is no action for receiving E or D when the state of the input data stream is *act*, *dlv*, or *cls*. Since we do not allow for local ending of contexts, it is namely not possible that a packet with E or D appears in z_I while st_I has any of these values. This is because the context on the other side of the association can only be ended if it knows that the input data stream is closed or *rcl*osing. However, if st_I is *act*, *dlv*, or *cls*, any packet obtained by the context on the other side will say that the input data stream is neither closed nor *rcl*osing.

The receipt of any CNTL-request packet causes that a response is sent in the same atomic action (actions *RecSW*, *RecSR*, *RecSWR*, *RecS*). Upon receiving a packet without

S, the context just commits internal changes depending on the packet contents, but sends nothing (actions *RecCal*, *RecCW*, *RecCR*, *RecCWR*, *RecC*, *RecED*, *RecD*).

In S_{CXT}, the weak fairness requirement for *Tout* ensures that *tmr* will eventually time out if set long enough and that the information, for which *tmr* runs, will be retransmitted. Weak fairness of *Dlr* and *Cnf* ensures that the data from the receive queue will eventually be delivered to the host and that the input data stream will not wait indefinitely for a confirmation before closing, respectively.

We do not model interactions of the procedure with the user explicitly, since this is not needed for verification. Nevertheless, one can imagine that the transition from $st_O = act$ to $st_O = wcl$ in the output data stream (from $st_I = act$ to $st_I = rcl$ in the input data stream) indicates that the user requested closing of the outgoing data stream (the incoming data stream, respectively). When the output data stream (the input data stream) passes to $st_O = inact$ ($st_I = inact$, respectively) after the handshake initiated by the local request, this can be viewed as also representing issuing of a confirmation to the user, that his request for closing a simplex data stream was served. Passing to $st_I = cls$ can be thought of as also giving an indication to the user that the input data stream wants to close. A subsequent execution of *Cnf* then represents the user's confirmation that the stream may close. Also, the transition of *cxt* to *null* can be thought of as representing issuing of an indication to the user that the context is closed.

4.2. A requirement specification

We are now ready to specify important properties that are required to be satisfied by *CLS*. We assume that any side of a simplex data stream can start closing, independent of each other, and that each simplex data stream can start closing independent of the other.

Let in the sequel I, P, Q denote state formulae, A, N action formulae, F, G formulae, and V a set of local variables. Let P' denote an action formula obtained from a state formula P by replacing any local variables occurring free in P by equally named primed variables. Let $STAB\ P \equiv \Box(P \rightarrow \Box P)$, $F\ UNL\ G \equiv \Box(F \rightarrow F \text{ unless } G)$, and $F \rightsquigarrow G \equiv \Box(F \rightarrow \Diamond G)$ ("leadsto"). *STAB* and *UNL* remind us of UNITY operators [3]. Let temporal operators have higher priority than nontemporal ones. Priority of them is as follows, in decreasing order: $\neg$, $\wedge$ and $\vee$, $\rightarrow$, $\leftrightarrow$.

Let b be a nonempty subset of $\{W,R,E,D\}$. By $b \notin z$ we denote that a variable z ranging over sequences of the messages from M does not contain any message containing an element of b. Let $x \in \{c_1,\ldots,c_n\}$ denote $x = c_1 \vee \cdots \vee x = c_n$.

Define:

$$NoLRcl_B \equiv \Box(st_{IB} \neq rcl), NoLWcl_A \equiv \Box(st_{OA} \neq wcl)$$

$$SClw1_{AB} \equiv \Box(st_{OA} = inact \rightarrow st_{IB} = inact)$$

$$SClw2_{AB} \equiv st_{IB} \neq inact \text{ unless } rq = empty$$
$$SClw3_{AB} \equiv \Box(\{R,E,D\} \in z_{OB} \rightarrow st_{IB} = inact)$$
$$SClr_{AB} \equiv \Box(st_{IB} = inact \rightarrow st_{OA} = inact)$$
$$GRq_B \equiv \Box(rq_B = full \vee rq_B = empty)$$
$$GCl_{AB} \equiv st_{OA} = wcl \rightsquigarrow (st_{IB} = inact \wedge rq_B = empty \wedge st_{OA} = inact)$$
$$UCl_{AB} \equiv st_{IA} = rcl \rightsquigarrow (st_{OB} = inact \wedge st_{IA} = inact),$$

and likewise for A and B exchanged.

Let $TRst \equiv tmr_A = off \wedge tmr_B = off$.

$$ClAss \equiv (\Diamond(st_{OA} = wcl) \wedge \Diamond(st_{OB} = wcl)) \rightarrow \Diamond\Box(cxt_A = null \wedge cxt_B = null \wedge TRst)$$
$$UGCl \equiv ((\Diamond(st_{IA} = rcl) \wedge \Diamond(st_{OA} = wcl)) \vee (\Diamond(st_{IB} = rcl) \wedge \Diamond(st_{OB} = wcl))) \rightarrow \Diamond\Box(cxt_A = null \wedge cxt_B = null \wedge TRst)$$

Then we require the following to hold:

Theorem 1.

$$CLS \models NoLRcl_B \rightarrow SClw1_{AB} \wedge SClw2_{AB} \wedge SClw3_{AB} \wedge GRq_B \wedge GCl_{AB},$$
$$CLS \models NoLRcl_A \rightarrow SClw1_{BA} \wedge SClw2_{BA} \wedge SClw3_{BA} \wedge GRq_A \wedge GCl_{BA}.$$

Theorem 2.

$$CLS \models NoLWcl_A \rightarrow SClr_{AB}, CLS \models NoLWcl_B \rightarrow SClr_{BA}.$$

Theorem 3.

$$CLS \models ClAss \wedge UCl_{AB} \wedge UCl_{BA} \wedge UGCl.$$

Theorem 1 is about graceful closing of the simplex data streams. It says that if local closing of the input data stream for a simplex data stream never takes place, then its output data stream, which is so the only one that may initiate closing, can be closed only if the input data stream is already closed. The input data stream must deliver all the data before closing, and it must not say that it is closed in outgoing packets before getting a confirmation from the application. Theorem 2 is about ungraceful closing of the simplex data streams. It says that if the output data stream of a simplex data stream does not initiate closing, then its input data stream, which in this case is the only one that may initiate closing, can only get inactive when the output data stream is closed.

Theorem 3 is about graceful closing, ungraceful closing, and about the combination of both. Independent of which side initiates closing of some simplex data stream, they must eventually get closed, timers must be stopped, and both contexts must close.

5. Verification of the closing procedure

5.1. *Some useful proof rules*

To prove the theorems, we treat the components design specifications in the form $\Pi \models S$ as axioms. We first provide some useful proof rules for our TLA similar to those from [10]. Remember that in the absence of temporal operators, a primed variable of the form x' is treated like an unprimed variable named x', thus allowing pure assertional reasoning. Also, if an action formula A is proved assertionally valid, then $\Box A$ is valid.

INV1-rule:

$$\frac{\Box(I \wedge [N]_V \rightarrow I')}{I \wedge \Box[N]_V \rightarrow \Box I}$$

INV2-rule:

$$\vdash \Box(\Box I \rightarrow (\Box[N] \leftrightarrow \Box[N \wedge I \wedge I']))$$

STAB-rule:

$$\frac{\Box(P \wedge [N]_V \rightarrow P')}{\Box[N]_V \rightarrow STAB\, P}$$

UNL-rule:

$$\frac{\Box(P \wedge [N]_V \rightarrow A \vee P')}{\Box[N]_V \rightarrow P\ UNL\ A}$$

WF1-rule:

$$\frac{\begin{array}{l}\Box(P \wedge [N]_V \rightarrow P' \vee Q') \\ \Box(P \wedge \langle N \wedge A\rangle_V \rightarrow Q') \\ \Box(P \rightarrow En(\langle A\rangle_V))\end{array}}{\Box[N]_V \wedge WF_V(A) \rightarrow P \rightsquigarrow Q}$$

Analogous to these rules, we now propose some rules that enable us to prove important properties of a parallel composition without first constructing the conjunction of its component specifications using the parallel composition rule. For our purpose, it suffices to assume that there are no hidden variables in the specifications. Also, assume that the special observable variables do not appear free in properties to be proved by the rules.

INV1C-rule:

$$\frac{\begin{array}{l} \Pi_i \models I_i \wedge \Box[(N_i \wedge \mu') \vee (\mathsf{unch}(O_{ui}) \wedge \neg\mu')]_{V_i} \text{ for } i = 1,\ldots,n \\ \Box(\bigwedge_{i=1}^{n} I_i \rightarrow I) \\ \Box(I \wedge [N_i \wedge \mathsf{unch}(\bigcup_{j=1}^{n} V_j - V_i)]_{\cup_{j=1}^{n} V_j} \rightarrow I') \text{ for } i = 1,\ldots,n \end{array}}{\|_{i=1}^{n} \Pi_i \models \Box I}$$

STABC-rule:

$$\frac{\begin{array}{l} \Pi_i \models \Box[(N_i \wedge \mu') \vee (\mathsf{unch}(O_{ui}) \wedge \neg\mu')]_{V_i} \text{ for } i = 1,\ldots,n \\ \Box(P \wedge [N_i \wedge \mathsf{unch}(\bigcup_{j=1}^{n} V_j - V_i)]_{\cup_{j=1}^{n} V_j} \rightarrow P') \text{ for } i = 1,\ldots,n \end{array}}{\|_{i=1}^{n} \Pi_i \models STAB\, P}$$

UNLC-rule:

$$\frac{\begin{array}{l} \Pi_i \models \Box[(N_i \wedge \mu') \vee (\mathsf{unch}(O_{ui}) \wedge \neg\mu')]_{V_i} \text{ for } i = 1,\ldots,n \\ \Box(P \wedge [N_i \wedge \mathsf{unch}(\bigcup_{j=1}^{n} V_j - V_i)]_{\cup_{j=1}^{n} V_j} \rightarrow A \vee P') \text{ for } i = 1,\ldots,n \end{array}}{\|_{i=1}^{n} \Pi_i \models P\ UNL\ A}$$

WF1C-rule:

$$\frac{\begin{array}{l} \Pi_i \models \Box[(N_i \wedge \mu') \vee (\mathsf{unch}(O_{ui}) \wedge \neg\mu')]_{V_i} \text{ for } i = 1,\ldots,n \\ \Pi_k \models WF_{V_k}(A \wedge \mu') \text{ for some } k \in \{1,\ldots,n\} \\ \Box(P \wedge [N_i \wedge \mathsf{unch}(\bigcup_{j=1}^{n} V_j - V_i)]_{\cup_{j=1}^{n} V_j} \rightarrow P' \vee Q') \text{ for } i = 1,\ldots,n \\ \Box(P \wedge \langle N_k \wedge A \wedge \mathsf{unch}(\bigcup_{j=1}^{n} V_j - V_k)\rangle_{\cup_{j=1}^{n} V_j} \rightarrow Q') \\ \Box(P \rightarrow En(\langle A \rangle_{V_k})) \end{array}}{\|_{i=1}^{n} \Pi_i \models P \rightsquigarrow Q}$$

LOC-rule:

$$\frac{\Pi \models F \text{ for } \Pi \text{ an open system}}{\Pi \parallel \Pi_1 \parallel \ldots \parallel \Pi_n \models F \text{ for any compatible open systems } \Pi_1,\ldots,\Pi_n}$$

Intuitively, soundness of INV1C, STABC, UNLC, and WF1C rules follows from the fact that they just subsume the way of proving properties of parallel composition by first applying the parallel composition rule on precise specifications of the components for obtaining an action formula N describing possible actions of the complete system, and then proving the premise of the consequence rule using INV1 or one of the other TLA rules by cases on the actions of N. Soundness of LOC follows directly from the parallel composition rule. Properties that only contain unchangeable variables of Π as free local variables are good candidates to be proved by LOC-rule, thus possibly saving a lot of work if validity of the property can be proved locally.

Now, assume that we have any system composed like ours. Assume that it consists of two communication channels, Π_{AB}, Π_{BA}, like *CHN* except that a different set of messages M may be communicated in the system. Also, assume that they connect two sender/receiver processes, Π_A, Π_B, similar to *CXT* in the following. They communicate only over the channel variables. They may execute internal actions, not accessing the channels, sending actions, and receiving actions. Assume that they contain some actions that are both sending and receiving ones. Only actions for receiving one possible message from M are enabled at a process at a time, and all receiving actions are required to be weakly fair. It must also be ensured that there are no unspecified receptions. It is possible to ensure this already when writing a design specification of a receiving component by defining a receiving action for all possible combinations of heads and states of the component. Then the system satisfies the *channel liveness property*

$$\bigwedge_{M_1 \in 2^M - \emptyset} (\Box\Diamond \bigvee_{m \in M_1} send(m, z_{OA}) \rightarrow \Box\Diamond \bigvee_{m \in M_1} rec(m, z_{IB}))$$

with z_{OA} the channel variable at Π_A's end of its outgoing channel Π_{AB} and z_{IB} the one at Π_B's end of Π_{AB}, and symmetrically for channel Π_{BA}. This can be proved by first constructing a precise specification of the system by the parallel composition rule. Using the fairness requirements of the channel and the receiving actions, assuming messages do not progress over the channel, the liveness property can be shown to be implied by the precise specification, by contradiction.

Once ensuring the liveness for both channels, the following proof rules for proving leads to properties can be derived for this specific type of communication systems (cf. [9]). Let M_{AB}, M_{BA} be nonempty subsets of a set of messages M, and z_{OA}, z_{OB} and z_{IA}, z_{IB} the variables of the outgoing and incoming channel of Π_A and Π_B, respectively. Let P and Q be state formulae not containing the channel variables.

Leadsto-via-(M_{AB}, z_{OA}, z_{IB})-rule:

$$\frac{\begin{array}{l} \Pi_A \models \Box[(N_A \wedge \mu') \vee (\mathsf{unch}(O_{uA}) \wedge \neg\mu')]_{V_A} \wedge WF_{V_A}(A \wedge \mu') \\ \Box(P \rightarrow En(\langle A \rangle_{V_A}), \quad \Box(P \wedge \langle N_A \wedge A \rangle_{V_A} \rightarrow \bigvee_{m \in M_{AB}} send(m, z_{OA})) \\ \Pi_B \models \Box[(N_B \wedge \mu') \vee (\mathsf{unch}(O_{uB}) \wedge \neg\mu')]_{V_B} \\ \Box(P \wedge [N_A \wedge \mathsf{unch}(V_B)]_{V_A \cup V_B} \rightarrow P' \vee Q') \\ \Box(P \wedge [N_B \wedge \mathsf{unch}(V_A)]_{V_A \cup V_B} \rightarrow P' \vee Q') \\ \Box(P \wedge \langle \bigvee_{m \in M_{AB}} (N_B \wedge rec(m, z_{IB})) \wedge \mathsf{unch}(V_A) \rangle_{V_A \cup V_B} \rightarrow Q') \end{array}}{\Pi_A \parallel \Pi_{AB} \parallel \Pi_{BA} \parallel \Pi_B \models P \rightsquigarrow Q}$$

Leadsto-via-(M_{BA}, z_{OB}, z_{IA})-(M_{AB}, z_{OA}, z_{IB})-rule:

$$\frac{\begin{array}{l} \Pi_A \models \Box[(N_A \wedge \mu') \vee (\mathsf{unch}(O_{uA}) \wedge \neg\mu')]_{V_A} \wedge WF_{V_A}(A \wedge \mu') \\ \Box(P \rightarrow En(\langle A \rangle_{V_A}), \quad \Box(P \wedge \langle N_A \wedge A \rangle_{V_A} \rightarrow \bigvee_{m \in M_{AB}} send(m, z_{OA})) \end{array}}{\cdots}$$

$$\frac{\begin{array}{l} \Pi_B \models \Box[(N_B \wedge \mu') \vee (\mathsf{unch}(O_{uB}) \wedge \neg\mu')]_{V_B} \\ \Box(P \wedge [N_A \wedge \mathsf{unch}(V_B)]_{V_A \cup V_B} \rightarrow P' \vee Q') \\ \Box(P \wedge [N_B \wedge \mathsf{unch}(V_A)]_{V_A \cup V_B} \rightarrow P' \vee Q') \\ \Box(P \wedge \langle \bigvee_{m \in M_{BA}} (N_A \wedge rec(m, z_{IA})) \wedge \mathsf{unch}(V_B) \rangle_{V_A \cup V_B} \rightarrow Q') \\ \Box(P \wedge \langle \bigvee_{m \in M_{AB}} (N_B \wedge rec(m, z_{IB})) \rangle_{V_B} \rightarrow \bigvee_{m \in M_{BA}} send(m, z_{OB})) \end{array}}{\Pi_A \parallel \Pi_{AB} \parallel \Pi_{BA} \parallel \Pi_B \models P \rightsquigarrow Q}$$

In fact, both rules rely on the existence of a weakly fair timeout action A in Π_A. The first rule requires that the messages from M_{AB} are retransmitted by action A of Π_A, so that eventually some element of M_{AB} is received at Π_B, whereupon Q holds. The second rule requires that the messages from M_{AB} are retransmitted by action A of Π_A, so that eventually a message from M_{BA} sent by Π_B is received at Π_A, whereupon Q holds. This rule requires that receipts of messages from M_{AB} at Π_B imply sending of some message from M_{BA} back. This is meant to be ensured by the actions that are receiving and sending ones at the same time. In CXT, these are the actions that receive CNTL-request messages and immediately send a reply.

If a formula has already been proved valid for a parallel composition, it can be taken as a valid assumption in proving another property by any of INV1C, STABC, UNLC, WF1C, or Leadsto-via rules. If G is one of the properties that appear in the conclusions of these rules, validity of $F \rightarrow G$ in a parallel composition may be proved by using F as a valid assumption in proving validity of G in the composition by one of the rules.

5.2. *Verification of the requirements*

In order to prove the theorems about the closing procedure, we need some lemmata. By symmetry of the closing procedure, assume that any lemma containing A and B is also valid if A and B are interchanged.

Lemma 1. *Let*

$$InvC_A \equiv cxt_A = null \leftrightarrow st_{OA} = inact \wedge st_{IA} = inact.$$

Then

$$CXT_A \models \Box InvC_A.$$

Proof. We first prove $InvC_A \wedge \Box[N]_V \rightarrow \Box InvC_A$ by INV1, for possible actions $[N]_V$ of CXT_A. Because the initial condition of CXT_A implies $InvC_A$, it follows that the precise specification of CXT_A implies $\Box InvC_A$, thus ensuring the validity of the lemma by the consequence rule. □

Lemma 2.

$CLS \models \Box Inv C_A$, $CLS \models \Box Inv C_B$.

Proof. From Lemma 1 by LOC. □

Lemma 3. *Let*

$$
\begin{aligned}
Inv_{AB} \equiv\ & (st_{OA} = act \wedge st_{IB} = act \wedge rq_B \in \{empty, full\} \\
& \wedge \{R,E,D\} \notin z_{IA} \wedge \{W,E,D\} \notin z_{OA} \wedge \{R,E,D\} \notin z_{OB} \wedge \{W,E,D\} \notin z_{IB}) \\
\vee\ & (st_{OA} = act \wedge st_{IB} = rcl \wedge rq_B = dsc \\
& \wedge \{E,D\} \notin z_{IA} \wedge \{W,E,D\} \notin z_{OA} \wedge \{E,D\} \notin z_{OB} \wedge \{W,E,D\} \notin z_{IB}) \\
\vee\ & (st_{OA} = inact \wedge st_{IB} = rcl \wedge rq_B = dsc \\
& \wedge \{E,D\} \notin z_{IA} \wedge \{E,D\} \notin z_{OB}) \\
\vee\ & (st_{OA} = inact \wedge st_{IB} = inact \wedge rq_B \in \{empty, dsc\}) \\
\vee\ & (st_{OA} = wcl \wedge st_{IB} = rcl \wedge rq_B = dsc \\
& \wedge \{E,D\} \notin z_{IA} \wedge \{E,D\} \notin z_{OA} \wedge \{E,D\} \notin z_{OB} \wedge \{E,D\} \notin z_{IB}) \\
\vee\ & (st_{OA} = wcl \wedge st_{IB} = act \wedge rq_B \in \{empty, full\} \\
& \wedge \{R,E,D\} \notin z_{IA} \wedge \{E,D\} \notin z_{OA} \wedge \{R,E,D\} \notin z_{OB} \wedge \{E,D\} \notin z_{IB}) \\
\vee\ & (st_{OA} = wcl \wedge st_{IB} = dlv \wedge rq_B = full \\
& \wedge \{R,E,D\} \notin z_{IA} \wedge \{E,D\} \notin z_{OA} \wedge \{R,E,D\} \notin z_{OB} \wedge \{E,D\} \notin z_{IB}) \\
\vee\ & (st_{OA} = wcl \wedge st_{IB} = cls \wedge rq_B = empty \\
& \wedge \{R,E,D\} \notin z_{IA} \wedge \{E,D\} \notin z_{OA} \wedge \{R,E,D\} \notin z_{OB} \wedge \{E,D\} \notin z_{IB}) \\
\vee\ & (st_{OA} = wcl \wedge st_{IB} = inact \wedge rq_B \in \{empty, dsc\} \\
& \wedge \{E,D\} \notin z_{OA} \wedge \{E,D\} \notin z_{IB})
\end{aligned}
$$

Then

$CLS \models \Box Inv_{AB}$.

Proof. By using INV1C, and either Lemma 1 for A and B side together with INV2 for using the fact about the contexts in N_A and N_B, or Lemma 2 for A and B side to take the invariants as assumptions. We also need the obvious fact, that if some kind of messages are not in a channel, then after executing an action, this will still hold if the action does not send this kind of messages into the channel. Also, if a message is not in a channel, it is also not at the head of it. Thus receiving actions for the message are not enabled. □

Lemma 4. $CXT_A \models \Box (st_{IA} \neq rcl \wedge st_{OA} \neq wcl \rightarrow tmr_A = off)$.

Proof. By using INV1 on the actions of CXT_A, analogous to the proof of Lemma 1. □

Lemma 5. $CXT_A \models \Box (st_{IA} \neq rcl) \rightarrow \Box (rq_A = full \vee rq_A = empty)$.

Proof. By taking $\Box (st_{IA} \neq rcl)$ as a valid assumption, and using INV2 and INV1 on the actions of CXT_A, analogous to the proof of Lemma 1. □

Lemma 6. $CXT_A \models STAB\,(st_{OA} = inact) \wedge STAB\,(st_{IA} = inact)$.

Proof. By using STAB on the actions of CXT_A and the consequence rule for proving each stability property separately, and then using the conjunction rule. □

In order to prove basic liveness properties of the procedure, we take the property that always, for any message at the head of the incoming channel variable of a context, an action that receives the message is enabled, for granted, i.e.

$$CLS \models \Box \bigwedge_{m \in M} (Head(z_{IA}) = m \rightarrow En(N_A \wedge rec(m, z_{IA}))),$$

and likewise for the B side.

Lemma 7. $CLS \models \bigwedge_{i=1}^{8} L_i$ *where*

$$\begin{aligned}
L_1 &\equiv (st_{OA} = wcl \wedge st_{IB} = act \wedge rq_B = empty) \\
&\quad \rightsquigarrow ((st_{OA} = wcl \wedge st_{IB} = cls) \vee (st_{OA} = wcl \wedge st_{IB} = rcl)) \\
L_2 &\equiv (st_{OA} = wcl \wedge st_{IB} = act \wedge rq_B = full) \\
&\quad \rightsquigarrow ((st_{OA} = wcl \wedge st_{IB} = act \wedge rq_B = empty) \\
&\quad \vee (st_{OA} = wcl \wedge st_{IB} = dlv \wedge rq_B = full) \vee (st_{OA} = wcl \wedge st_{IB} = rcl)) \\
L_3 &\equiv (st_{OA} = wcl \wedge st_{IB} = dlv \wedge rq_B = full) \\
&\quad \rightsquigarrow (st_{OA} = wcl \wedge st_{IB} = cls \wedge rq_B = empty) \\
L_4 &\equiv (st_{OA} = wcl \wedge st_{IB} = cls) \rightsquigarrow (st_{OA} = wcl \wedge st_{IB} = inact) \\
L_5 &\equiv (st_{OA} = wcl \wedge st_{IB} = rcl) \\
&\quad \rightsquigarrow ((st_{OA} = wcl \wedge st_{IB} = inact) \vee (st_{OA} = inact \wedge st_{IB} = rcl)) \\
L_6 &\equiv (st_{OA} = inact \wedge st_{IB} = rcl) \rightsquigarrow (st_{OA} = inact \wedge st_{IB} = inact) \\
L_7 &\equiv (st_{OA} = wcl \wedge st_{IB} = inact) \rightsquigarrow (st_{OA} = inact \wedge st_{IB} = inact) \\
L_8 &\equiv (st_{OA} = act \wedge st_{IB} = rcl) \\
&\quad \rightsquigarrow ((st_{OA} = inact \wedge st_{IB} = rcl) \vee (st_{OA} = wcl \wedge st_{IB} = rcl))
\end{aligned}$$

Proof. Let $(*, String)$ denote all the messages of M that contain *String*. $L3$ and $L4$ can be proved by WF1C-rule assuming $\Box Inv C_B$ and $\Box Inv_{AB}$, by weak fairness of Dlr and weak fairness of Cnf in CXT_B, respectively.
L_3 can also be proved without checking all the components of CLS in the following way. We first prove

$$CXT_B \models (st_{IB} = dlv \wedge rq_B = full) \rightsquigarrow (st_{IB} = cls \wedge rq_B = empty)$$

with help of Lemma 1, INV2, and WF1-rule by weak fairness of Dlr in CXT_B. Then, by LOC also $CLS \models (st_{IB} = dlv \wedge rq_B = full) \rightsquigarrow (st_{IB} = cls \wedge rq_B = empty)$.
From the latter and since $st_{OA} = wcl \wedge st_{IB} = dlv \wedge rq_B = full$ always implies $st_{IB} = dlv \wedge rq_B = full$, $CLS \models (st_{OA} = wcl \wedge st_{IB} = dlv \wedge rq_B = full) \rightsquigarrow (st_{IB} = cls \wedge rq_B = empty)$, because $\Box(F \rightarrow G)$ implies $F \rightsquigarrow G$ and by transitivity of $\rightsquigarrow$.
From Lemma 3, $CLS \models \Box(st_{IB} = cls \wedge rq_B = empty \rightarrow st_{OA} = wcl)$, and thus clearly $CLS \models L_3$.

$L1$, $L2$, and $L5$ can be proved by Leadsto-via-$(\{(*,W,S)\}),z_{OA},z_{IB})$ exploiting weak fairness of *Tout* in CXT_A, assuming $\Box Inv_{AB}$ when proving $L1$ and $L2$, and assuming $\Box InvC_A$ and $\Box InvC_B$ in the proofs of $L1$, $L2$, and $L5$. $L6$ can be proved by Leadsto-via-$(\{(*,W),(D)\},z_{OA},z_{IB})$-$(\{(*,R,S)\},z_{OB},z_{IA})$ using weak fairness of *Tout* in CXT_B and assuming $\Box InvC_B$. $L7$ can be proved by Leadsto-via-$(\{(*,R),(D)\},z_{OB},z_{IA})$-$(\{(*,W,S)\},z_{OA},z_{IB})$ using weak fairness of *Tout* in CXT_A and assuming $\Box InvC_A$. $L8$ can be proved by Leadsto-via-$(\{(*,R,S)\}),z_{OB},z_{IA})$ by weak fairness of *Tout* in CXT_B and assuming $\Box Inv_{AB}$, $\Box InvC_A$ and $\Box InvC_B$. □

Lemma 8.

$$CLS \models st_{OA} = wcl \rightsquigarrow (st_{IB} = inact \wedge st_{OA} = inact)$$
$$CLS \models st_{IA} = rcl \rightsquigarrow (st_{OB} = inact \wedge st_{IA} = inact)$$

Proof. We will only outline the proof of the first liveness property. The other can be proved similarly. By the consequence rule, from Lemma 3,

$$\begin{aligned} CLS \vdash \Box(st_{OA} = wcl \rightarrow ((&st_{OA} = wcl \wedge st_{IB} = act \wedge rq_B = empty)\\ &\vee (st_{OA} = wcl \wedge st_{IB} = act \wedge rq_B = full)\\ &\vee (st_{OA} = wcl \wedge st_{IB} = dlv \wedge rq_B = full)\\ &\vee (st_{OA} = wcl \wedge st_{IB} = cls \wedge rq_B = empty)\\ &\vee (st_{OA} = wcl \wedge st_{IB} = rcl \wedge rq_B = dsc) \vee (st_{OA} = wcl \wedge st_{IB} = inact))). \end{aligned}$$

Using the temporal logic proof rule from [11],

$$\frac{F \rightsquigarrow (G_1 \vee G_2), \quad G_1 \rightsquigarrow G, \quad G_2 \rightsquigarrow G}{F \rightsquigarrow G},$$

we can prove

$$\begin{aligned} CLS \models\ & (st_{OA} = wcl \wedge st_{IB} = act \wedge rq_B = empty)\\ & \rightsquigarrow (st_{IB} = inact \wedge st_{OA} = inact) \end{aligned}$$

from $CLS \models L_1 \wedge L_4 \wedge L_5 \wedge L_6 \wedge L_7$, which follows by the consequence rule from Lemma 7, and likewise for the other possible combinations with $state_{OA} = wcl$. Then, by the conjunction and consequence rule, because $\Box(F \rightarrow G)$ implies $F \rightsquigarrow G$, and by transitivity of $\rightsquigarrow$ we obtain the first liveness property of the current lemma. □

Proof of Theorem 1. We will only sketch the proof of the requirement in one direction, namely, when the input data stream at B cannot be locally closed. Validity of the requirement for graceful closing in the other direction follows by symmetry. By the conjunction rule and propositional reasoning, it suffices to prove

$$CLS \models NoLRcl_B \rightarrow SClw1_{AB}, \quad CLS \models NoLRcl_B \rightarrow SClw2_{AB},$$
$$CLS \models NoLRcl_B \rightarrow SClw3_{AB}, \quad CLS \models NoLRcl_B \rightarrow GRq_B,$$
$$CLS \models NoLRcl_B \rightarrow GCl_{AB}.$$

The first requirement is proved easily by using the consequence rule on Inv_{AB} (Lemma 3) assuming $NoLRcl_B$.

To prove the second requirement, we first prove

$$CXT_B \models NoLRcl_B \rightarrow st_{IB} \neq inact\, UNL\, rq = empty$$

from the precise specification of CXT_B using the consequence rule, UNL, and INV2 assuming $NoLRcl_B$. Since initially, $st_{IB} \neq inact$ in CXT_B, also

$$CXT_B \models NoLRcl_B \rightarrow st_{IB} \neq inact \text{ unless } rq = empty,$$

and thus by LOC also CLS satisfies the requirement.

The third requirement can be proved by using the consequence rule on Inv_{AB}. Assuming $NoLRcl_B$, we can prove that $\Box Inv_{AB}$ implies $\Box(st_{IB} \neq inact \rightarrow \{R,E,D\} \notin z_{OB})$, thus proving the requirement.

The fourth requirement follows by symmetry and LOC from Lemma 4. The fifth requirement can be proved as follows. By Lemma 8 and the consequence rule,

$$CLS \models NoLRcl_B \rightarrow (st_{OA} = wcl \rightsquigarrow (st_{IB} = inact \wedge st_{OA} = inact)).$$

Also, assuming $NoLRcl_B$, by the consequence rule on Inv_{AB}, $CLS \models \Box(st_{IB} = inact \rightarrow rq_B = empty)$. By the conjunction rule, then validity of the fifth requirement follows. □

Proof of Theorem 2. We can prove this safety property easily by using the consequence rule on Inv_{AB} (Lemma 3) assuming $NoLWcl_A$ for direction A-to-B. The property for data stream B-to-A follows by symmetry. □

Proof of Theorem 3. We can prove

$$CLS \models \Box(st_{OA} = inact \wedge st_A = inact \rightarrow tmr_A = off \wedge cxt_A = null).$$

This follows by LOC from Lemmas 1 and 4, using the conjunction and consequence rule, and symmetrically for side B. By Lemma 8 and by symmetry, starting closing of the data stream in each direction either by setting wcl or rcl, eventually st_{OA}, st_{IA}, st_{OB}, and st_{IB} will be inactive. By Lemma 6, because $\Diamond P \wedge \Diamond Q \wedge STAB\, P \wedge STAB\, Q$ always implies $\Diamond\Box(P \wedge Q)$, $CLS \models \Diamond\Box(st_{OA} = st_{IA} = inact \wedge st_{OB} = st_{IB} = inact)$ holds, assuming that closing of both simplex data streams has been initiated. From the first temporal assertion in the proof, it follows that both timers and contexts will also get and remain *off* and *null*, respectively, which proves the theorem. □

6. Conclusion

We showed how the essential features of the procedure for closing contexts at each side of an XTP association can be specified and verified using a TLA. Our work could

be compared to that of [7]. There, a message-passing model is used instead of the original TLA shared-variables one. The former is useful for layered specification and verification of protocols, where specification module boundaries are not equal to process boundaries. By introducing appropriate notation for sending and receiving messages on channels, our design specification could also be modularized, for example, with regard to the two simplex data streams. Also, it would be possible to specify a context in terms of its input and output data stream. However, assuming the degree of concurrency in them as taken in our specification would make the way to get the specification of the complete system more complicated than if directly considering all possible concurrency in the contexts.

This is also because our verification problem is still of a moderate size. Since each data stream can in fact close independent of the other, we can say that the system has 144 global states, observing Inv_{AB} for one data stream, not looking at possible contents of channel variables. Mechanical support would be welcome for the reasons of reliable verification, but the verification is also quite manageable just by hand. Having appropriate proof rules, the majority of reasoning reduces just to simple checking if some small state formulae are preserved or transformed to some other simple state formulae by actions that in many cases do not even affect the formulae. This is usually true for communication protocols, since processes at different sides of communication links generally communicate only over a small number of variables. TLA also seems quite appropriate at least for writing protocol specifications by protocol developers that are not logic experts, because of its simple state-based model.

Although we used a TLA, our approach can also be compared to UNITY [3]. Whereas it is common to include some features that have arised with the work on TLA in UNITY (e.g., [4]), some of our proof rules can be seen as adding the UNITY style to TLA specifications. Beside a TLA, we also used the meta notation for satisfaction of specifications. We gave only an outline of proofs in this paper. It would be interesting to specify and verify the closing procedure without assuming that all the data have been transferred correctly. Errors have namely been found in this setting (e.g., by simulation [2]). However, then also at least some aspects of the data transfer part of the protocol would have to be included. And assuming a high degree of concurrency, more sophisticated ways of writing modular specifications of protocols and effectively reasoning about them would be needed. This will be a part of our future work.

References

[1] M. Abadi and L. Lamport, Composing specifications, *ACM TOPLAS* **15** (1993) 73–132.

[2] O. Catrina, Protocol analysis and verification methods, application to the Xpress Transport Protocol 4.0, in: P. Dembiński and M. Średniawa, eds., *Conf. Proc. PSTV'95*, Warsaw, Poland (1995) 365–380.

[3] K.M. Chandy and J. Misra, *Parallel Program Design – A Foundation* (Addison-Wesley, Reading, MA, 1988).

[4] P. Collette, Application of the composition principle to Unity-like specifications, in: M.-C. Gaudel and J.-P. Jouannaud, eds., *TAPSOFT '93*, Lecture Notes in Computer Science, Vol. 668 (Springer, Berlin, 1993) 230–242.

[5] W.A. Doeringer et al., A survey of light-weight transport protocols for high-speed networks, *IEEE Trans. Commun.* **38** (1990) 2025–2039.

[6] F. Halsall, *Data Communications, Computer Networks and Open Systems* (Addison-Wesley, Reading, MA, 1992).

[7] P. Herrmann and H. Krumm, Compositional specification and verification of high-speed transfer protocols, Research Report No. 540, Universität Dortmund, 1994.

[8] T. Kapus, True concurrency semantics and correctness of concurrent programs, Ph.D. Thesis, University of Maribor, Faculty of Technical Sciences, 1994 (in Slovene).

[9] S.S. Lam and A.U. Shankar, A relational notation for state transition systems, *IEEE Trans. Soft. Eng.* **16** (1990) 755–775.

[10] L. Lamport, The temporal logic of actions, *ACM TOPLAS* **16** (1994) 872–943.

[11] Z. Manna and A. Pnueli, The anchored version of the temporal framework, in: J.W. de Bakker, W.P. de Roever and G. Rozenberg, eds., *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Lecture Notes in Computer Science, Vol. 354 (Springer, Berlin, 1989) 201–284.

[12] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification* (Springer, Berlin, 1992).

[13] S.L. Murphy and A.U. Shankar, Connection management for transport layer: service specification and protocol verification, *IEEE Trans. Commun.* **39** (1991) 1762–1775.

[14] V. Nguyen, A. Demers, D. Gries and S. Owicki, A model and temporal proof system for networks of processes, *Distributed Comput.* **1** (1986) 7–25.

[15] XTP Protocol Definition, Revision 3.6, 11 January 1992, PEI 92-10, TU Berlin, Text and Protocol Conversion, derived from the original edited by Protocol Engines Incorporated.

[16] S. Zhou, R. Gerth and R. Kuiper, Transformations preserving properties and properties preserved by transformations in fair transition systems, in: E. Best, ed., *CONCUR '93*, Lecture Notes in Computer Science, Vol. 715 (Springer, Berlin, 1993) 353–367.