Using Compositional Preorders in the Verification of Sliding Window Protocol

Roope Kaivola*

Department of Computer Science
PO Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland

Abstract The main obstacle to automatic verification of temporal logic properties of finite-state systems is the state explosion problem. One way to alleviate this is to replace components of a system with smaller ones and verify the required properties from the smaller system. This approach leads to notions of compositional property-preserving equivalences and preorders. Previously we have shown that the NDFD preorder is the weakest preorder which is compositional w.r.t. standard operators and preserves nexttime-less linear temporal logic properties. In this paper we describe a case study where NDFD preorder was used to verify semi-automatically both safety and liveness properties of the Sliding Window protocol for arbitrary channel lengths and realistic parameter values. In this process we located a previously undiscovered fault leading to lack of liveness in a version of the protocol.

1 Introduction

A promising approach to verification of finite-state concurrent systems is the use of propositional temporal logic as a specification language and the application of model-checking algorithms for the verification task (for an overview, see [4, 14]). In practice the main obstacle to this approach is that the execution graphs of realistic systems are too large for feasible model-checking, as in the general case the size of an execution graph is exponential in the number of concurrent processes. Several techniques have been proposed for alleviating this *state explosion* problem. Most of these are based on the fact that the components of a system are often reasonably small, even though the total system may be large [2, 10].

One particular technique is *compositional reduction*. Here the idea is to replace the individual components of a system by smaller ones before building the model of the whole system, and to do this in such a way that if the reduced model fulfils the required properties, then the full system does so as well. This approach leads to notions of compositional property-preserving equivalences and preorders between modules. Assuming that we have a preorder for which the required properties that hold of a model also hold of all models which are lower in the preorder, and which is congruent with respect to the composition operators

* e-mail: Roope.Kaivola@helsinki.fi

used in building the system, we can replace a module of a system with any module which is smaller in size but lies higher in the preorder, and if the required properties hold of the resulting system then they also hold of the original one.

In this paper we describe a case study where compositional preorders were used to verify safety and liveness properties of the Sliding Window or SW communication protocol [13]. The SW protocol forms the basis of the data transfer function of the HDLC protocol [6], and an extreme simplification of it is the Alternating Bit protocol, a stock example in the literature on formal verification. Although almost any method seems to cope with the AB protocol, analysis of the more general SW protocol is harder; even with small window sizes and channel lengths the state space becomes far too large for direct model-checking.

In the case study the components of the protocol were modelled by ordinary labelled transition systems, and the complete protocol was built from these by Basic LOTOS [1] operators. The properties that were required of the protocol were expressed using a transition-oriented variant of the propositional nexttime-less linear temporal logic. To express the requirements in a propositional language and to model the system in a finitary way, we used the technique of *data independence* [17, 12], which allows us to deduce that the system works correctly for all sequences of data, if it works correctly for data sequences of certain forms.

The notion of preorder used in the verification was the so-called *non-divergent failures divergences* or *NDFD* preorder. In [8, 9] it was shown that the related NDFD equivalence preserves all nexttime-less linear temporal logic properties, and in [16] that it is a congruence with respect to all Basic LOTOS operators. Moreover, in [9] it was shown that NDFD equivalence is the weakest or loosest possible criterion of equivalence which has these properties. These results can be generalised to the NDFD preorder in a natural way [7].

In practive the main task in the verification was the construction of an *abstraction* of the sender part of the protocol. This was done manually, guided by intuitions about the protocol. Next, it was verified that this abstraction is higher in the NDFD preorder than the sender combined with a channel of arbitrary length. This was done automatically, using the ARA toolset [15]. Finally, the abstraction was combined with the receiver end of the protocol, and it was verified that the resulting system fulfils the required properties of the protocol. In this extended abstract, only the main steps are presented. For a full analysis, see [7].

In the case study we succesfully verified that the SW-protocol meets its safety and liveness requirements for a variety of realistic window sizes and *arbitrary* channel lengths. A more surprising observation was that a variant of the protocol *fails* to fulfil usual liveness requirements. This variant arises from a modification which seems harmless enough that it has sometimes appeared in the literature. In our opinion this result illustrates well the advantages of rigorous verification.

The verification of the SW protocol has been studied extensively (see [7, Sect. 7.3] for a survey). Like the original proof [13], most work has been manual or concentrated on safety properties, often both. Closest to the current work is [11], where safety properties of the protocol are expressed in temporal logic and verified by direct model-checking, with limits set by the state-explosion problem.

2 Background

Definition 1. For any set A, 2^A denotes the powerset of A, A^* the set of finite strings of elements of A and A^ω the set of infinite strings of elements of A. Define $A^\infty = A^* \cup A^\omega$. If $\sigma = a_0 a_1 \ldots \in A^\infty$, $\sigma(n)$ denotes the element a_n and $\sigma[n\ldots]$ the string $a_n a_{n+1} \ldots$ We use $\cdot$ to denote catenation and $|\sigma|$ the length of σ.

Definition 2. Fix an infinite *transition alphabet* Σ, which does not contain the *empty transition label* τ. Define $\Sigma_\tau = \Sigma \cup \{\tau\}$. A *labelled transition system (lts)* is a triple $L = (S, s, \Delta)$, where S is a finite set of states, $s \in S$ a unique initial state, and $\Delta \subseteq S \times \Sigma_\tau \times S$ a finite transition relation. If $\rho \in \Sigma_\tau^*$, we write $s_0 \xrightarrow{\rho} s_n$ iff there are $s_1, \ldots, s_{n-1}$ such that for all $0 \leq i < n$, $(s_i, \rho(i), s_{i+1}) \in \Delta$, and $s_0 \xrightarrow{\rho}$ iff there is an s_n such that $s_0 \xrightarrow{\rho} s_n$. If $\rho \in \Sigma_\tau^\infty$, we write $s_0 \xrightarrow{\rho}$ iff there are $s_1, s_2, \ldots$ such that for all $i \geq 0$, $(s_i, \rho(i), s_{i+1}) \in \Delta$. If $\sigma \in (\Sigma^* \cup \Sigma^\infty)$, we write $s_0 \stackrel{\sigma}{\Longrightarrow} s_n$ ($s_0 \stackrel{\sigma}{\Longrightarrow}$) iff there is some $\rho \in (\Sigma_\tau^* \cup \Sigma_\tau^\infty)$ such that $s_0 \xrightarrow{\rho} s_n$ ($s_0 \xrightarrow{\rho}$) and σ is the string obtained from ρ by removing all τ symbols.

We use parallel composition, hiding and renaming operators to combine ltss. We abuse notation slightly in the following, and write for example **hide** $sc*$ **in** L to mean that all labels beginning with sc are hidden, and $L[sc*/cr*]$ to mean that each label of the form $scxx$ is renamed to $crxx$.

Definition 3. Let $L_1 = (S_1, s_1, \Delta_1)$, $L_2 = (S_2, s_2, \Delta_2)$ be ltss and G, H sets $G = \{g_1, \ldots, g_n\} \subset \Sigma$ and $H = \{h_1, \ldots, h_n\} \subset \Sigma$. Then

- $L_1 |[g_1, \ldots, g_n]| L_2$ is the lts $(S_1 \times S_2, (s_1, s_2), \Delta)$, where $((t,u), l, (t',u')) \in \Delta$ iff either $l \in G$, $(t, l, t') \in \Delta_1$ and $(u, l, u') \in \Delta_2$, or $l \notin G$, $(t, l, t') \in \Delta_1$ and $u = u'$, or $l \notin G$, $(u, l, u') \in \Delta_2$ and $t = t'$,
- **hide** $g_1, \ldots, g_n$ **in** L_1 is the lts (S_1, s_1, Δ), where $(t, l, t') \in \Delta$, iff either $l \notin G$ and $(t, l, t') \in \Delta_1$ or $l = \tau$ and there is a $g_i \in G$ such that $(t, g_i, t') \in \Delta_1$,
- $L_1[h_1/g_1, \ldots, h_n/g_n]$ is the lts (S_1, s_1, Δ), where $(t, l, t') \in \Delta$, iff either $l \notin G$ and $(t, l, t') \in \Delta_1$ or $l = h_i$ and $(t, g_i, t') \in \Delta_1$.

Definition 4. Let $L = (S, \Delta, s)$ be a labelled transition system. We say that

- $\rho \in \Sigma_\tau^\infty$ is a *maximal behaviour* of L iff either ρ is infinite and $s \xrightarrow{\rho}$, or ρ is finite and there is some s' such that $s \xrightarrow{\rho} s'$ and $s' \xrightarrow{a}$ for no $a \in \Sigma_\tau$,
- $\sigma \in \Sigma^\omega$ is an *infinite trace* of L iff $s \stackrel{\sigma}{\Longrightarrow}$,
- $\sigma \in \Sigma^*$ is a *divergence trace* of L iff there is some s' such that $s \stackrel{\sigma}{\Longrightarrow} s' \xrightarrow{\tau^\omega}$,
- $(\sigma, A) \in \Sigma^* \times 2^\Sigma$ is a *nondivergent failure* of L iff σ is not a divergence trace of L, and there is some s' such that $s \stackrel{\sigma}{\Longrightarrow} s'$ and $s' \stackrel{a}{\Longrightarrow}$ for no $a \in A$,

We use *maxbeh*(L) to denote the set of maximal behaviours of L, *inftr*(L) the set of infinite traces of L, *divtr*(L) the set of divergence traces of L, and *ndfail*(L) the set of nondivergent failures of L.

Definition 5. Let L and L' be ltss. We say that L is lower than L' in the *nondivergent failures divergences preorder* or *NDFD preorder* and write $L \overset{\text{ndfd}}{\prec} L'$ iff

- *inftr*$(L) \subseteq$ *inftr*(L'),

- $divtr(L) \subseteq divtr(L')$, and
- $ndfail(L) \subseteq ndfail(L') \cup (divtr(L') \times 2^{\Sigma})$.

For a full analysis of the NDFD preorder and its relation to other related failure-based preorders, see [16]. As far as the current paper is concerned, its two important properties are that and it preserves all maximal finite and infinite traces, and thereby also all nexttime-less linear temporal logic properties, and it is congruent with respect to the composition operators above,

Proposition 6 [16]. *If* $L_1 \overset{\text{ndfd}}{\prec} L_1'$ *and* $L_2 \overset{\text{ndfd}}{\prec} L_2'$, *then*

- $L_1[[g_1, \ldots, g_n]]L_2 \overset{\text{ndfd}}{\prec} L_1'[[g_1 \ldots, g_n]]L_2'$,
- **hide** $g_1, \ldots, g_n$ **in** $L_1 \overset{\text{ndfd}}{\prec}$ **hide** $g_1, \ldots, g_n$ **in** L_1' *and*
- $L_1[h_1/g_1, \ldots, h_n/g_n] \overset{\text{ndfd}}{\prec} L_1'[h_1/g_1, \ldots, h_n/g_n]$.

Most temporal logics used in program specification are essentially state-based, describing properties of execution states and sequences of them. However, here we would like to make reference to transitions and their labels directly. To this purpose we define a variant of the usual nexttime-less linear temporal logic LTL'. Similar extensions from state-based to transition-based logics are discussed in [3]. In [7] we describe a general model where both states and transitions of a transition system carry labels and both of these can be referred to in the logic. However, in the current case study ordinary ltss suffice.

Definition 7. The formulae of the logic $tLTL'$ are defined by the abstract syntax: $\phi ::= \top \mid \neg\phi \mid \phi \vee \phi' \mid \phi \mathcal{U} \phi' \mid \phi \mathcal{U}^a \phi'$, where a varies over Σ. We use $\bot$, $\wedge$, $\Rightarrow$ and $\Leftrightarrow$ as derived operators in the usual way, and define $\Diamond\phi = \top \mathcal{U} \phi$, $\Box\phi = \neg\Diamond\neg\phi$, $\langle a \rangle \phi = \top \mathcal{U}^a \phi$ and $\langle a \rangle = \top \mathcal{U}^a \top$. If ϕ is a formula, we use $\Sigma(\phi)$ to denote the set of labels $a \in \Sigma$ occurring in operators $\mathcal{U}^a$ in ϕ.

We say that ϕ *is true of* a sequence $\sigma \in \Sigma_\tau^\infty$, denoted by $\sigma \models \phi$, according to the following inductive rules: $\sigma \models \top$ always, $\sigma \models \neg\phi$ iff not $\sigma \models \phi$, $\sigma \models \phi \vee \psi$ iff $\sigma \models \phi$ or $\sigma \models \psi$, $\sigma \models \phi \mathcal{U} \psi$ iff there is some $0 \leq i \leq |\sigma|$ such that $\sigma[i \ldots] \models \psi$ and $\sigma[j \ldots] \models \phi$ for all $0 \leq j < i$, and finally, $\sigma \models \phi \mathcal{U}^a \psi$ iff there is some $0 \leq i < |\sigma|$ such that $\sigma(i) = a$, $\sigma[i+1 \ldots] \models \psi$, and for all $0 \leq j < i$, $\sigma(j) \neq a$ and $\sigma[j \ldots] \models \phi$. We say that ϕ is true of an lts L and write $L \models \phi$ iff $\sigma \models \phi$ for all $\sigma \in maxbeh(L)$.

Here, $\mathcal{U}$ is the standard *until* operator. The less familiar $\phi \mathcal{U}^a \psi$ operator says that ψ holds immediately after the next transition labelled with a, such a transition exists, and until that moment ϕ holds. The derived operator $\langle a \rangle \phi$ says that ϕ holds after the next transition labelled with a, and there is such a transition, and $\langle a \rangle$ that some future transition is labelled with a.

Proposition 8 [8, 7]. *If* ϕ *is a* $tLTL'$*-formula,* $L' \models \phi$ *and* $L \overset{\text{ndfd}}{\prec} L'$, *then* $L \models \phi$.

We use to following simple result to ignore uninteresting labels in a model.

Proposition 9 [7]. *If* ϕ *is a* $tLTL'$*-formula and* $\Pi \subseteq \Sigma$ *a set of labels such that* $\Sigma(\phi) \cap \Pi = \emptyset$, *then* $L \models \phi$ *iff* **hide** Π **in** $L \models \phi$.

SENDER
Variables
$d[0\ldots(tw-1)]$: table of data items; i, j, k : int /* x, a: temporary variables */
Initially $i = 0, j = 0, k = 0$
loop infinitely
choose nondeterministically
$i < tw \longrightarrow$ receive(data source, $d[i]$); $i := i + 1$;
[] $j < i \longrightarrow$ send(channel, $(j \oplus k, d[j])$); $j := j + 1$;
[] $j = i \wedge i > 0 \longrightarrow$ receive(timeout); $j := 0$;
[] $\top \longrightarrow$ receive(ack channel, x); $a := (x \ominus k) + 1$;
if $(a \leq tw)$ then
$d[0\ldots(tw-1-a)] := d[a\ldots(tw-1)]$;
$i := \max(0, i-a)$; $j := \max(0, j-a)$; $k := x \oplus 1$
end if;
end choose;
end loop;

RECEIVER
Variables
$t[0\ldots(rw-1)]$: table of data items; $r[0\ldots(rw-1)]$: table of booleans
e : int; /* x, y, s: temporary variables */
Initially $e = 0$, $r[0\ldots(rw-1)] = (\bot, \ldots, \bot)$
loop infinitely
receive(channel, (x, y)); $s := x \ominus e$;
if $(s < rw)$ then $r[s] := \top$; $t[s] := y$; end if;
while $(r[0] \neq \bot)$ do
send(data target, $t[0]$);
$t[0\ldots(rw-2)] := t[1\ldots(rw-1)]$; $r[0\ldots(rw-2)] := r[1\ldots(rw-1)]$;
$r[rw-1] := \bot$; $e := e \oplus 1$;
end while;
send(ack channel, $(e \ominus 1)$);
end loop;

Notation: $x \oplus y = (x + y) \bmod (w + 1)$ and $x \ominus y = (x - y) \bmod (w + 1)$

Figure1. Sender and receiver

3 Sliding Window protocol

The object of the case study is a unidirectional version of the Sliding Window protocol [13], which tries to provide reliable transmission of data over an unreliable communication medium. The system consists of six components. The *sender* receives data items from an external *data source*, sends messages consisting of a sequence number and a data item to the *channel*, receives acknowledgement messages from the *acknowledgement channel*, and receives timeout signals from an unspecified external source. The *receiver* reads messages from the channel, forwards correctly received data items to an external *data target*, and sends

acknowledgement messages consisting of a sequence number to the acknowledgement channel. The channel and acknowledgement channel may distort or lose messages but may not change their order. Three parameters to the protocol are the *transmission window* size tw the *reception window* size rw, and the maximum value w for the sequence numbers. These must fulfil the constraint $tw+rw \leq w+1$ [13]. The sender and receiver processes are described in Figure 1.

We have specified only a very simple timeout mechanism for the protocol: instead of presenting explicitly the setting and cancelling of timers, we have specified that the sender may receive a timeout signal in situations where it cannot send any new data items to the channel. We make the assumption that timeouts should not occur prematurely, i.e. when some acknowledgement is still on its way to the sender. This is not reflected in the description so far, but will be taken into account when modelling the protocol. In reality this could be achieved by setting the timeout delay to be longer than the maximum message transit time from the sender to the receiver and back. The assumptions regarding timeouts are only needed for liveness properties of the protocol, not for safety.

The aim of the protocol is to transmit a sequence of data items from the data source to the target in the correct order. Reflecting this, the requirements to the protocol which we tried to verify were:

SAFE The data delivered to the target should be correct, i.e. the data sequence transmitted to the target before any given moment should be a prefix of the data sequence received from the source before that moment.

LIVE1 If n data items are read from the source along an execution sequence, and the execution sequence is complete under the assumption that no more data is available for reading from the source, then n data items should be delivered to the target along the execution sequence.

LIVE2 If infinitely many data items are read from the data source along an execution sequence, then infinitely many data items should be delivered to the target along that execution sequence.

LIVE3 The sender should accept data from the source, i.e. for any moment in a maximal execution sequence there should be a future moment when the sender reads a data item from the source, unless no more data is available.

The liveness properties are expected to hold under the fairness condition:

FAIR Let $i \in \{0, \ldots, w\}$ be any sequence number. If messages with sequence number i are sent to the channel infinitely often, then messages with that sequence number will also be transmitted successfully to the receiver infinitely often. The same holds for the acknowledgement channel.

4 Modelling the protocol

As the first step in the verification of the protocol, we appeal to a syntactic property of the system called *data-independence* [17, 12], which allows us to verify the requirements to the system just for certain representative data sequences or certain data sources and conclude that the system works correctly with all possible sequences of data items. We shall omit the details here and just state:

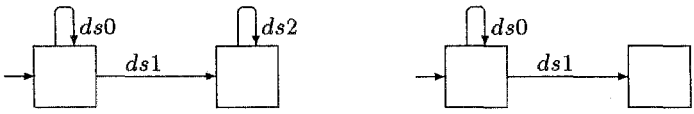

Figure2. Data source ltss D_{safe} (left) and D_{live} (right)

- The protocol satisfies the requirement SAFE iff on any maximal execution of the protocol connected to the data source D_{safe} in Fig. 2, the sequence of data items delivered to the target belongs to the language $0^\infty \cdot (1 \cdot 2^\infty \cup \epsilon)$.
- Assuming that the protocol satisfies the safety requirement, it satisfies the requirements LIVE1-3 iff on any maximal execution of the protocol connected to the data source D_{live} in Fig. 2, either the data item 1 or infinitely many 0:s are delivered to the target.

The analysis showing that it is enough to consider these data sources and sequences [7, App. A and Sect. 3.3] is similar to that of [17, 12]. However, for safety properties they consider the data sequences $0^* \cdot 1 \cdot 0^* \cdot 2 \cdot 0^\omega$, whereas here we found it more convenient to use the data sequences $0^\infty \cdot (1 \cdot 2^\infty \cup \epsilon)$. For liveness properties, we use sequences $0^\infty \cdot 1$ instead of 0^ω of [17, 12] to cover also finite sequences of data.

To model the protocol by labelled transition systems, we fix the set of data items as $\{0, 1, 2\}$ and set some upper bounds n_c and n_a for the capacity of the channel and the acknowledgement channel. Having decided the values of w, tw, rw, n_c and n_a, we build a model of the system from six basic ltss: **S** modelling the sender process, **R** modelling the receiver process, **C** modelling a channel with the capacity of one message, **A** modelling an acknowledgement channel with the capacity of one message, $\mathbf{C_F}$ modelling a faulty channel with the capacity of one message, and $\mathbf{A_F}$ modelling a faulty acknowledgement channel with the capacity of one message.

The ltss S and R can be constructed straightforwardly from the descriptions in Fig. 1. The lts C is just a one-place buffer and easy to construct. To represent the unreliability of the channel, we use the lts C_F, a one-place buffer which may lose the message it has received. Distorted messages are assumed to be detected by some checksum mechanism and discarded, to be treated as lost.

The lts modelling the whole protocol is constructed from the basic ltss as described in Figure 3. We use synchronisation labels **ds**y to correspond to transmission of data item y from data source to sender, **sc**xy to messages from sender to channel, **cr**xy to messages from channel to receiver, **ra**x to acknowledgement messages from receiver to ack. channel, **as**x to ack. messages from ack. channel to sender, **rt**y to transmission of data items from receiver to target, and **tim** to the timeout signal. Here x ranges over $\{0, \ldots, w\}$ and y over $\{0, 1, 2\}$.

The lts P modelling the whole protocol can be formally defined as follows:

$$SC_0 = S$$
$$SC_{i+1} = \textbf{hide } sc* \textbf{ in } (SC_i[sc*/cr*] \,||[sc*, tim]||\, C)$$

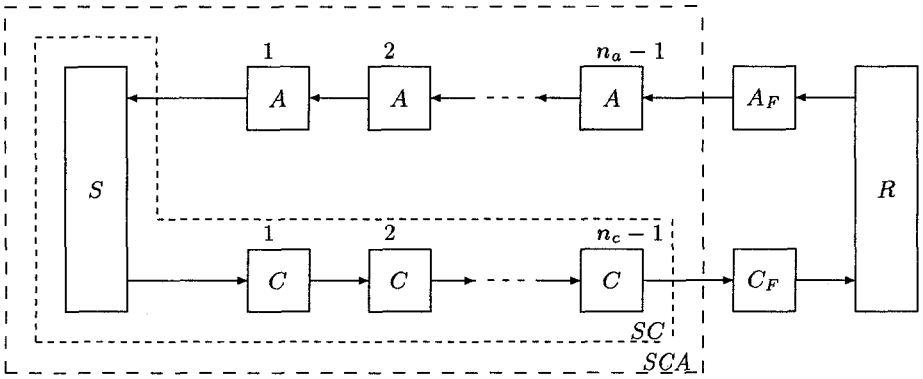

Figure3. Structure of the model

$$
\begin{aligned}
SCA_0 &= SC \;=\; SC_{n_c-1} \\
SCA_{i+1} &= \textbf{hide } as* \textbf{ in } (SCA_i[as*/ra*] \,||[as*, tim]||\, A) \\
SCA &= SCA_{n_a-1} \\
SC_F A &= SCA[sc*/cr*] \,||[sc*, tim]||\, C_F \\
SC_F A_F &= SC_F A[as*/ra*] \,||[as*, tim]||\, A_F \\
P &= SC_F A_F \,||[cr*, ra*, tim]||\, R
\end{aligned}
$$

To capture the assumption that timeouts do not occur prematurely, we synchronised all components of the system with respect to the *tim* action so that a timeout could occur only if all components agree on that. A *tim* transition is allowed in C, C_F, A and A_F only when the corresponding buffer is empty, and in R only when the receiver cannot send any acknowledgements.

To verify that the protocol satisfies the safety requirement, we express the property *the sequence of data items delivered to the target belongs to the language* $0^\infty \cdot (1 \cdot 2^\infty \cup \epsilon)$ by the following formula ϕ_{safe}

$$
(\langle rt1 \rangle \Rightarrow [rt1](\neg\langle rt0 \rangle \wedge \neg\langle rt1 \rangle)) \wedge (\langle rt2 \rangle \Rightarrow [rt2](\neg\langle rt0 \rangle \wedge \neg\langle rt1 \rangle)) \wedge (\langle rt2 \rangle \Rightarrow [rt1])
$$

and verify that $D_{safe} \,||[ds*]||\, P \models \phi_{safe}$. Similarly, to verify the liveness requirements, we try to verify that $D_{live} \,||[ds*]||\, P \models \phi_{live}$, where ϕ_{live} is the formula

$$
\phi_{fair} \Rightarrow (\Box\Diamond\langle rt0 \rangle \vee \Diamond\langle rt1 \rangle)
$$

and the fairness constraint is expressed by the following formula ϕ_{fair}

$$
\bigwedge_{0 \le i \le w} (\Box\Diamond(\bigvee_{0 \le j \le 2} \langle sci j \rangle) \Rightarrow \Box\Diamond(\bigvee_{0 \le j \le 2} \langle cri j \rangle)) \;\wedge\; \bigwedge_{0 \le i \le w} (\Box\Diamond\langle rai \rangle \Rightarrow \Box\Diamond\langle asi \rangle)
$$

5 Verification

For verifying both safety and liveness properties of the SW-protocol, we used the general approach of constructing an abstraction $\overline{S}$ on the basis of the sender lts S. The particular abstractions used for safety and liveness properties are different, but the techniques and reasoning are very similar. Because of this, we shall just discuss liveness properties here. Full details are in [7].

Before constructing the abstraction for liveness, we can ease the verification by a couple of simple techniques. First of all, the fixed data source D_{live} produces only a fraction of all the potential data sequences, which means that a large part of the state space of P will not be reachable in $D_{live}\,|[ds*]|\,P$. Therefore, rather than build P first and then compose it with D_{live}, it is better to start by composing the sender S with D_{live} and then build the other ltss on that basis; the resulting ltss will be the same up to isomorphism. Secondly, when trying to determine the validity of ϕ_{live} over a model, by Prop. 9 we are allowed to hide labels $ds*$ and tim which do not appear in ϕ_{live}. What is more, as **hide** distributes over parallel composition as long as the hidden labels are not used for synchronisation, we can hide $ds*$ immediately after the data source is combined with the sender.

Let $\overline{S}$ denote the abstraction, and define $\overline{DS} = \textbf{hide } ds* \textbf{ in } (D_{live}|[ds*]|\overline{S})$. Furthermore, let $\overline{DP}$ denote the system built by combining $\overline{DS}$ with the faulty elements C_F and A_F and the receiver R, in the same way as P was built on the basis of SCA. The essential properties of the abstraction are

A: $(\textbf{hide } ds* \textbf{ in } D_{live}|[ds*]|S) \overset{\text{ndfd}}{\prec} \overline{DS}$

B: $\textbf{hide } sc* \textbf{ in } (\overline{DS}[sc*/cr*]|[sc*, tim]|C) \overset{\text{ndfd}}{\prec} \overline{DS}$

C: $\textbf{hide } as* \textbf{ in } (\overline{DS}[as*/ra*]|[as*, tim]|A) \overset{\text{ndfd}}{\prec} \overline{DS}$

D: $\overline{DP} \models \phi_{live}$

Here A states that the abstraction is higher in the NDFD-preorder than the sender combined with D_{live}, and B and C that the abstraction alone is higher in the preorder than when combined with a one-place channel or acknowledgement channel. On the basis of compositionality of $\overset{\text{ndfd}}{\prec}$, it is easy to see from claims A-C that $(\textbf{hide } ds* \textbf{ in } (D_{live}|[ds*]|SCA)) \overset{\text{ndfd}}{\prec} \overline{DS}$, no matter what the channel capacities n_c and n_a are. From claim D and Prop. 8 it follows then that $D_{live}|[ds*]|P \models \phi_{live}$, as required. This approach to dealing with channels of arbitrary length has appeared previously in [18].

The abstraction $\overline{S}$ for liveness properties is described in Fig. 4. The two basic intuitions behind it are the following. First, it mimics the behaviour of the sender with channels of unspecified length. What this means is that sent messages do not necessarily appear or acknowledgements cause effect immediately. Secondly, in the context of the complete system often only a subset of all possible acknowledgement messages can actually occur. In the abstraction it is specified that after the reception of an unexpected acknowledgement message, the abstraction becomes 'chaotic', capable of performing any action (same idea appears in [5]).

The abstraction was formulated manually, aided by intuitions on which characteristics of the protocol were relevant for liveness properties. Once the abstraction was constructed, claims A-C were checked by the ARA toolset [15].

SENDER ABSTRACTION
Variables
$d[(-tw)\ldots(tw-1)]$: table of data items; i, j, k : int; ch : bool;
Initially $i = 0, j = 0, k = 0, ch = \bot$
loop infinitely
choose nondeterministically
ch $\longrightarrow$ perform any action; /* (also τ possible) */
[] $\neg ch \wedge i < tw$ $\longrightarrow$ receive(data source, $d[i]$); $i := i + 1$;
[] $\neg ch \wedge j < i$ $\longrightarrow$ send(channel, $(j \oplus k, d[j])$); $j := j + 1$;
if $j < 0$ then $j := rnd(j, \ldots, 0)$ end if;
[] $\neg ch \wedge j = i \wedge i > 0$ $\longrightarrow$ receive(timeout); $j := 0$;
[] $\neg ch$ $\longrightarrow$ receive(ack channel, x); $a := (x \ominus k) + 1$;
if $((a - \min(j, 0)) \leq tw)$ then
$d[(-tw)\ldots(tw-1-a)] := d[(-tw+a)\ldots(tw-1)]$;
$i := \max(-tw, i - a)$; $j := j - a$;
if $j < 0$ then $j := rnd(j, \ldots, 0)$ end if
$k := x \oplus 1$
else if $a \neq w + 1$ then $ch := \top$ end if;
end choose;
end loop;

Notation: $rnd(x, \ldots, y)$ returns a random element from $\{x, \ldots, y\}$

Figure4. Abstraction for liveness properties

Although originally designed for a slightly different semantic model, with added pre-processing steps it could deal with the NDFD preorder. Since no temporal logic model checker was available, claim D was also verified using the ARA toolset, by combining $\overline{DP}$ with tester processes trying to find a path violating ϕ_{live} [7, App. B]. The table below lists sizes of some ltss involved in the verification:

Parameters			Reachable nodes				
w	tw	rw	$\overline{DS}$	$D_{live} \parallel S$	$\overline{DS} \parallel C$	$\overline{DS} \parallel A$	$\overline{DP}$
1	1	1	22	12	56	95	61
3	2	2	98	48	280	723	915
5	3	3	266	120	838	2851	5179
6	3	3	310	140	977	3791	6018
7	4	4	562	240	1940	8117	21379

6 Faulty variant of the protocol

In the sender process of the SW protocol an acknowledgement is treated as valid, if it contains a sequence number anywhere in the current transmission window. In another variant of the protocol, occasionally found in the literature [11], an acknowledgement is treated as valid, if it contains a sequence number between the beginning of the transmission window and the sequence number to be sent

Step	Action	Explanation
1	$ds0$	sender receives data item 0 from source
2	$ds0$	sender receives data item 0 from source
3	$sc00$	sender sends first data item to channel
4	$cr00$	channel forwards it to receiver
5	$rt0$	receiver forwards it to target
6	$ra0$	receiver sends an ack to first data item to ack channel
7	τ	ack channel loses it
8	$sc10$	sender sends second data item to channel
9	$cr10$	channel forwards it to receiver
10	$rt0$	receiver forwards it to target
11	$ra1$	receiver sends an ack to second data item to ack channel
12	τ	ack channel loses it
13	tim	timeout: sender starts sending data items again
14	$sc00$	sender sends first data item to channel
15	$cr00$	channel forwards it to receiver
16	$ra1$	receiver sends an ack to **second** data item to ack channel
17	$as1$	ack channel forwards it to sender, **sender ignores it as invalid**
18	$sc10$	sender sends second data item to channel
19	$cr10$	channel forwards it to receiver
20	$ra1$	receiver sends an ack to second data item to ack channel
21	τ	ack channel loses it
22 –	repeating from 13 –	

Parameters: $w = 2$ (or any $w \geq 2$), $tw = 2$, $rw = 1$, $n_c = n_a = 1$.

Figure5. Failure of liveness

next. In the sender process this corresponds to changing the last if-statement from 'if $(a \leq tw)$' to 'if $(a \leq j)$'. This change to the protocol seems innocent enough: How could there be an acknowledgement to any of the data items in the remaining part of the transmission window, anyway? After all, these data items are only just to be sent.

The safety properties of the variant can be verified as with the original protocol. However, when trying to verify the liveness properties, we were unable to produce an abstraction fulfilling the claims A-D above. After analysing the reasons behind this, it emerged that the variant actually *fails* to fulfil the liveness requirements. Fig. 5 describes one execution sequence of the variant where this failure appears. Although the sequence is fair, it fails to fulfil property LIVE3, requiring the system to accept data from the source; after the first two data items, no more data is accepted. Intuitively, although $as1$ is delivered to the sender infinitely often, it always arrives at a time when the sender regards it as invalid.

In our opinion this failure illustrates two issues. First, the change causing the protocol to fail liveness is natural and seems unimportant enough that, for example, [11] used the faulty variant when verifying safety properties of the protocol. We feel that this is a good example of the benefits of the rigorousness induced by a formal verification method. Secondly, the example reminds of the fact that liveness properties cannot be reliably verified unless divergences and

fairness conditions are properly kept track of. For example, if channel events were hidden and the services provided by the protocols compared with respect to observation equivalence, ignoring divergences, both versions would be considered equivalent. In effect, divergences corresponding to fair executions would be confused with ones corresponding to unfair executions, and ignored.

References

1. Bolognesi, T. & Brinksma, E.: Introduction to the ISO specification language LOTOS, in *The Formal Descr. Technique LOTOS*, North-Holland, 1989, pp. 23-73
2. Clarke, E. M. & Long, D. E. & McMillan, K. L.: Compositional model checking, in *Proceedings of the Fourth IEEE LICS*, 1989, pp. 353-362
3. De Nicola, R. & Vaandrager, F.: Action vs. state based logics for transition systems, in *Semantics of Sys. of Conc. Proc.*, LNCS vol. 469, Springer, 1990, pp. 407- 419
4. Emerson, E. A.: Temporal and modal logic, in van Leeuwen, J. (ed.): *Handbook of Theoretical Computer Science*, Elsevier/North-Holland, 1990, pp. 997-1072
5. Graf, S. & Steffen, B. & Lüttgen, G.: Compositional Minimisation of Finite State Systems Using Interface Spec., in *Formal Asp. of Comp.*, vol. 8, 1996, pp. 607-616
6. International Standards Organisation: *Data Communications – HDLC Unbalanced Class of Procedures*, Ref. No. ISO 6159, ISO, Geneva, 1980
7. Kaivola, R.: *Equivalences, Preorders and Compositional Verification for Linear Time Temp. Logic and Conc. Sys.*, A-1996-1, Univ. of Helsinki, Dept. of Comp. Sci., 1996, 176+9 p., also in `http://www.cs.Helsinki.FI/~rkaivola/research/ft.ps`
8. Kaivola, R. & Valmari, A.: Using truth-preserving reductions to improve the clarity of Kripke-models, in *CONCUR'91*, LNCS vol. 527, Springer, 1991, pp. 361-375
9. Kaivola, R. & Valmari, A.: The weakest compositional semantic equivalence preserving nexttime-less linear temporal logic, *CONCUR'92*, LNCS vol. 630, Springer, 1992, pp. 207-221
10. Manna, Z. & Pnueli, A.: *The Temporal Logic of Reactive and Concurrent Systems, vol. 1, Specification*, Springer, 1991
11. Richier, J. L. & Rodriguez, C. & Sifakis, J. & Voiron, J.: Verification in Xesar of the sliding window protocol, in *PSTV VII*, North-Holland, 1987, pp. 235-248
12. Sabnani, K.: An algorithmic technique for protocol verification, in *IEEE Transactions on Communications*, vol. 36, no. 8, 1988, pp. 924-931
13. Stenning, N. V.: A data transfer protocol, in *Computer Networks*, vol. 11, 1976, pp. 99-110
14. Stirling, C.: Modal and temporal logics, in Abramsky, S. & al. (eds.): *Handbook of Logic in Computer Science*, Oxford University Press, 1992, pp. 477-563
15. Valmari, A. & Kemppainen, J. & Clegg, M. & Levanto, M.: Putting advanced reachability analysis techniques together: the "ARA" tool, in *FME'93: Industrial-Strength Formal Methods*, LNCS vol. 670, Springer, 1993, pp. 597-616
16. Valmari, A. & Tienari, M.: Compositional failure-based semantic models for Basic LOTOS, in *Formal Aspects of Computing*, vol. 7, 1995, pp. 440-468
17. Wolper, P.: Expressing interesting properties of programs in propositional temporal logic, in *Proceedings of the 13th ACM POPL*, 1986, pp. 184-193
18. Wolper, P. & Lovinfosse, V.: Verifying Properties of Large Sets of Processes with Network Invariants, in *Proc. of International Workshop on Automatic Verification Methods for Finite State Systems*, LNCS vol. 407, Springer, 1990, pp. 68-80