Model Checking and Fault Tolerance

Glenn Bruns and Ian Sutherland

Bell Labs, Lucent Technologies
{grb,ian}@research.bell-labs.com

Abstract. We present an algebraic approach to the model checking of fault-tolerant systems. Fault models and fault-handling mechanisms are modelled using special-purpose process operators. Besides providing for natural models, special-purpose operators allow systems with large state spaces to be verified using systems with small state spaces. To support this verification technique we show that a kind of simulation relation on processes preserves all process operators in tyft/tyxt format.

1 Introduction

Model checking – in which a system model is automatically checked to see if it satisfies a temporal logic formula – has two serious limitations. The *size problem* is that the state space of the model can grow exponentially with the state space of its components. Additionally, model checking algorithms for many logics have high time complexity. The *generality problem* is that model checking tells us only that some instance of a system satisfies a property. We usually want to know that a system works properly over a range of parameter values and initial conditions.

These limitations are especially serious in the application of model checking to fault-tolerant systems. By modelling failures of a system component one can increase its possible interactions with other components and thus dramatically increase the state space of the system. Furthermore, one usually wants to show that a fault tolerance mechanism is general purpose; in such cases it is not enough to show that it works for a particular underlying system.

Here we present an approach to the model checking of fault-tolerant systems based on process algebra. To model faults and fault-handling mechanisms we define new, special-purpose process operators. A faulty version of a process is modelled by applying a fault operator to it. For example, suppose process P models a system or system component. To obtain a crash-faulty version of P we define a new operator Cr and apply it to P. Similarly, a fault-tolerant or fault-detecting version of a process is defined by applying a process operator to it. This approach has modelling advantages because it is natural to understand faults and fault-handling mechanisms (e.g., triple-modular redundancy [19]) as behavior transformers that are independent of an underlying process or computation.

Defining faults and fault-handling mechanisms as process operators also has technical advantages. We show that if a temporal property holds of a fault-handling mechanism applied to a simple underlying process, then it holds of the mechanism applied to more complex underlying processes. In this way both the size and generality problems of model checking can be reduced. The technique is an application of a general abstraction principle for processes. To apply it here we have had to show that a kind of simulation relation on processes is preserved by every process operator that we might want to define.

Our work uses and extends results on rule formats for structured operational semantics [12, 5]. These results show that certain properties hold of every process built of operators defined in a specific format. The use of special-purpose process operators in specification is discussed by Bloom [4], but not specifically for modelling faults and fault-handling mechanisms. The use of the simulation relation in verification has been explored by Lynch [16], but not for the verification of temporal properties. Our work also draws on the new Process Algebra Compiler (PAC) [8], which takes a process language definition and produces a version of the Concurrency Workbench of North Carolina [9] that can analyze terms of the language.

Our verification approach is property-based: we verify that particular temporal properties hold of a system. This approach is more flexible than that used in [17], in which one shows that a fault tolerance mechanism applied to a faulty system yields behavior equivalent to a nonfaulty system. For example, consider a nuclear reactor protection system, which responds to faults in the reactor by shutting the system down. The protection system ensures safety, but does not ensure the other properties we expect of a normally operating reactor.

The paper is organized as follows. In the next section we show how faults and fault-handling mechanisms can be cleanly modelled using process operators. We then define the simulation relation on processes and show how it helps in verifying properties of complex systems. We illustrate the usefulness of our technique by using it to verify properties of British Rail's slow-scan design. Proofs of the theorems are given in the full version of the paper.

2 Modelling Faults

In hardware fault tolerance, a *fault* is "a physical defect which may or may not cause a failure" [14]. Software has no faults in this sense. We extend the notion of fault here to include design errors, and consider faults of both hardware and software.

Faults are rarely modelled directly, as that would require their precise nature and location to be known. Instead they are usually modelled by the way they transform the behavior of a system. For example, a crash fault extends the behavior of a system by allowing a crash to occur in any state. In this paper faults are modelled using process operators that conform to the *tyft/tyxt* rule format [12]. These rules, like the rules of CCS (see Appendix), are used to define the semantics of process operators. We do not have space to present the tyft/tyxt rule format here. However, we should point out that the format is very general: it has been used to describe the operators of CCS, ACP, and other process algebras, as well as the operators used in this paper.

Consider an operator Cr that models crash faults. A process with crash faults should be able to evolve silently in any state to a state in which no action is possible.

$$\frac{P \xrightarrow{\alpha} P'}{Cr(P) \xrightarrow{\alpha} Cr(P')} \qquad \frac{}{Cr(P) \xrightarrow{\tau} 0}$$

The rule on the left states that $Cr(P)$ can do whatever P can do. The rule on the right states that $Cr(P)$ can also crash. Notice that the behavior of $Cr(P)$ depends only on the behavior of P, not its structure. In our definition $Cr(P)$ need not eventually crash. This approach differs from that of [2], in which a crash-faulty system is one that has crashed, not one that may crash.

Since operators defined using tyft/tyxt rules can be regarded as behavior transformers, they are a natural way to capture fault models. Furthermore, they are modular in the sense that they are defined separately from any underlying process. Compare our modelling of crash faults to that used in [15], in which the modelling of crash faults is embedded in the system description. Other basic fault models, such as fail-stop faults, incorrect-computation faults, and byzantine faults, can also be described with process operators. Intermittent faults can also be modelled easily.

Fault-handling mechanisms are also modelled naturally with process operators. Consider a watchdog timer that detects a fault by performing action `det` ("detect") if n `tick` actions occur before action `cl` ("clear") occurs. The timer is modelled by operator WD.

$$\frac{P \xrightarrow{\alpha} P'}{\mathrm{WD}_{n,\mathtt{cl}}(i,P) \xrightarrow{\alpha} \mathrm{WD}_{n,\mathtt{cl}}(i,P')} \ \alpha \notin \{\mathtt{tick}, \mathtt{cl}\}$$

$$\frac{P \xrightarrow{\mathtt{tick}} P'}{\mathrm{WD}_{n,\mathtt{cl}}(i,P) \xrightarrow{\mathtt{tick}} \mathrm{WD}_{n,\mathtt{cl}}(i-1,P')} \ i > 0 \qquad \frac{P \xrightarrow{\mathtt{tick}} P'}{\mathrm{WD}_{n,\mathtt{cl}}(0,P) \xrightarrow{\mathtt{tick}} \mathtt{det}\,.P'}$$

$$\frac{P \xrightarrow{\mathtt{cl}} P'}{\mathrm{WD}_{n,\mathtt{cl}}(i,P) \xrightarrow{\mathtt{cl}} \mathrm{WD}_{n,\mathtt{cl}}(n,P')}$$

The first rule shows that actions not in $\{\mathtt{tick}, \mathtt{cl}\}$ are ignored by the timer. The second and third rules show that a `tick` action decrements the timer, and when time out occurs the action `det` is produced. The fourth rule shows that action `cl` will reset the timer.

3 Simulation

Our approach to the size problem involves the notion of one process *simulating* another. Roughly, process Q simulates process P if Q can perform any action that P can, and can continue to do so indefinitely. Simulation is useful in verification because if Q simulates P then many of the properties of Q will also hold of P. For example, if Q cannot perform certain undesirable actions, then neither can P. We verify that a process P with a large state space satisfies a property ϕ

by verifying that ϕ holds of a process Q that has a small state space and that simulates P.

When we use this technique, we show that the simulates relation holds between two processes that model the same system. Showing this can be expensive for systems with large state spaces. However, if the models are built out of components, one model can be shown to simulate the other by showing that a component of one simulates the corresponding component of the other. For this approach to work, it must be the case that replacing a subprocess of P by one that simulates it yields a new process that simulates P. We state this formally using the following terminology: If $\mathcal{R}$ is a binary relation on processes and O is a collection of process operators, O *preserves* $\mathcal{R}$ if, for every n–ary operator o in O and processes $P_1, \ldots, P_n$ and $Q_1, \ldots, Q_n$, $P_i \mathrel{\mathcal{R}} Q_i$ for $1 \leq i \leq n$ implies $o(P_1, \ldots, P_n) \mathrel{\mathcal{R}} o(Q_1, \ldots, Q_n)$. We need the simulation relation we adopt to be preserved by all process operators we use.

To avoid dependence on CCS, we define simulation here on an arbitrary *transition system* $(S, A, \{\stackrel{a}{\rightarrow} \mid a \in A\})$, where S is a set of states, A is a set of actions, and $\{\stackrel{a}{\rightarrow} \mid a \in A\}$ is a set of labelled transition relations.

Definition 1. A binary relation $\mathcal{R}$ on S is a *simulation* if $P \mathrel{\mathcal{R}} Q$ implies that, for all a in A:

Whenever P $\stackrel{a}{\rightarrow}$ P', there exists Q' such that Q $\stackrel{a}{\rightarrow}$ Q' and $P' \mathrel{\mathcal{R}} Q'$.

Process Q *simulates* P if there exists a simulation $\mathcal{R}$ such that $P \mathrel{\mathcal{R}} Q$. Process Q *bisimulates* P if there exists a symmetric simulation $\mathcal{R}$ such that $P \mathrel{\mathcal{R}} Q$.

We write $P < Q$ if Q simulates P and $P \simeq Q$ if P bisimulates Q. The relation $<$ is a preorder and $\simeq$ is an equivalence relation.

The transition systems we use, like that of CCS, have a distinguished action τ that represents internal activity of a process. "Internal" means that observers of a process P cannot tell how many τ actions P is performing along with its externally visible actions. This interpretation of τ requires that we adopt a slightly different notion of simulation. Let $(S, A, \{\stackrel{a}{\rightarrow} \mid a \in A\})$ be a transition system with $\tau \in A$. The corresponding *weak transition system* is $(S, A, \{\stackrel{a}{\Rightarrow} \mid a \in A'\})$, where $\stackrel{\tau}{\Rightarrow}$ is $(\stackrel{\tau}{\rightarrow})^*$ (the reflexive, transitive closure of $\stackrel{\tau}{\rightarrow}$), and $\stackrel{a}{\Rightarrow}$ is $\stackrel{\tau}{\Rightarrow} \circ \stackrel{a}{\rightarrow} \circ \stackrel{\tau}{\Rightarrow}$ (the composition of the relations $\stackrel{\tau}{\Rightarrow}$, $\stackrel{a}{\rightarrow}$, and $\stackrel{\tau}{\Rightarrow}$), for a in $A - \{\tau\}$.

If Q simulates P in the weak transition system then we write $P \ll Q$ and say that Q *weakly simulates* P. If P bisimulates Q in the weak transition system we write $P \approx Q$, and say that P and Q are *observation equivalent*. Weak simulation is a preorder and observation equivalence is an equivalence relation.

The operators of CCS preserve weak simulation, but we wish to extend CCS with new operators. If we wish to use weak simulation as our notion of "simulation", we must be able to show that the combination of the CCS operators and the new operators preserve weak simulation. The semantics of our new operators are defined using the tyft/tyxt rule format [12]. It is shown in [12] that the bisimulates relation (and, by a slight modification of the argument, the simulates relation) is preserved by any collection of operators defined in tyft/tyxt format that satisfy an additional condition called *well-foundedness*.

Weak simulation is *not* preserved by every collection of operators defined by well-founded tyft/tyxt rules. There is, however, a natural additional condition to impose on the rules that ensures that weak simulation is preserved.

Definition 2. A tyft/tyxt rule of the form

$$\frac{\{t_i \xrightarrow{a_i} t'_i \mid i \in I\}}{t \xrightarrow{a} t'}$$

is said to be *valid for weak transitions* if, whenever $t_i \overset{a_i}{\Rightarrow} t'_i$ for all $i \in I$, $t \overset{a}{\Rightarrow} t'$.

Theorem 3. *If R is a well-founded set of tyft/tyxt rules for a system with a τ action, and the rules of R are valid for weak transitions, then weak simulation and observation equivalence are preserved by the function symbols of R.*

Like the proof in [12], the heart of the proof is an induction on proofs of transitions from a set of tyft/tyxt rules, but using transitions of the weak transition system. The additional hypothesis that the rules are valid for weak transitions is needed to make the modified induction work.

Unfortunately, the rules for the CCS operator "+" are not valid for weak transitions, so instead we will use a slightly stronger definition of simulation. This notion of simulation is defined, like weak simulation, by describing a derived transition system. Let $(S, A, \{\xrightarrow{a} \mid a \in A\})$ be a transition system with $\tau \in A$. The corresponding *weak-plus transition system* is $(S, A, \{\overset{a}{\Rightarrow}_+ \mid a \in A\})$, where $\overset{a}{\Rightarrow}_+$ is $(\xrightarrow{\tau})^* \circ \xrightarrow{a} \circ (\xrightarrow{\tau})^*$. Informally, the weak-plus transition system corresponds to an interpretation of τ in which observers cannot tell the difference between the occurrence of many τ actions and few τ actions, but can tell whether a τ action has occurred at all.

We write $P \ll_+ Q$ if Q simulates P in the weak-plus transition system. If $P \ll_+ Q$ we say that Q *weakly-plus simulates* P. If P bisimulates Q in the weak-plus transition system, we write $P \approx_+ Q$, and say that P and Q are *observation-plus equivalent*. Weak-plus simulation is a preorder and observation-plus equivalence is an equivalence relation.

Proposition 4. *Weak-plus simulation implies weak simulation. Observational-plus equivalence implies observation equivalence.*

We define the notion of a set of tyft/tyxt rules being valid for weak-plus transitions exactly as in Definition 2, with "weak" replaced by "weak-plus". We can then obtain the following result.

Theorem 5. *If R is a well-founded set of tyft/tyxt rules for a system with a τ action, and the rules of R are valid for weak-plus transitions, then weak-plus simulation and observation-plus equivalence are preserved by the function symbols of R.*

All rules for the basic CCS operators are valid for weak-plus transitions, so this result allows us to use weak-plus simulation with CCS as the basic algebra.

Are there less restrictive notions of "simulation" than weak-plus simulation that have the properties we need? However we define simulation, we would at least like it to (1) imply weak simulation, and (2) be preserved by any collection of operators defined by well-founded tyft/tyxt rules, possibly with some additional condition as in the theorems above. The following result is relevant.

Theorem 6. *If $\leq$ is any binary relation on labelled transition systems with a τ action such that:*

- *$P \leq Q$ implies $P \ll Q$.*
- *$\leq$ is preserved by any set of function symbols defined by a well-founded set of tyft/tyxt rules for a system with a τ action that are valid for weak-plus transitions.*

then $P \leq Q$ implies that $P \ll_+ Q$.

This result shows that the only way to get an acceptable notion of "simulation" that is not just a special case of weak-plus simulation would be to remove or replace the requirement that the tyft/tyxt rules are valid for weak-plus transitions. While such notions of "simulation" may exist, it would seem to require a major modification of the proof in [12] to establish that such a notion of "simulation" is preserved by new operators defined by tyft/tyxt rules.

Tyft/tyxt is a fairly general format that can define the semantics of the basic CCS operators as well as our additional operators. The above theorem applies to all the operators we will use to construct models. In particular, if Q weakly-plus simulates P, then $Cr(Q)$ weakly-plus simulates $Cr(P)$.

4 An Abstraction Principle

Here we show that the weakly simulates relation preserves certain temporal properties. The preserved properties are those that can be expressed in a fragment of the modal mu-calculus [13], a simple but expressive temporal logic.

Formulas of the *diamond-free modal mu-calculus* have the following abstract syntax, where L ranges over sets of actions and X ranges over propositional variables:

$$\phi :== X \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\![L]\!]\phi \mid \nu X.\phi \mid \mu X.\phi$$

The propositional connectives have their usual meaning. Formula $[\![L]\!]\phi$ holds of a process P if every weak transition from P leads to a process satisfying formula ϕ. The fixed-point operators allow properties to be defined recursively. They are binders; free occurrences of X in ϕ are bound in $\nu X.\phi$ and $\mu X.\phi$.

Falsity, written `ff`, is an abbreviation for $\mu X.X$. We also write $[\![\alpha_1, \ldots, \alpha_n]\!]$ as an abbreviation for $[\![\{\alpha_1, \ldots, \alpha_n\}]\!]$, and $[\![-L]\!]$ as an abbreviation for $[\![Act - L]\!]$.

The semantics of a formula is given as the set of processes that it satisfies. Let $\mathcal{T} = (S, A, \{\xrightarrow{a} \mid a \in A\})$ be a transition system, with τ in A. Let $\mathcal{V}$ be a *valuation* that maps propositional variables to subsets of S. We write $()$ for the valuation that maps all variables to the empty set, and $\mathcal{V}[X := \mathcal{E}]$ for the valuation that is like $\mathcal{V}$ except that X is mapped to $\mathcal{E}$. Let f be a monotonic, set-valued function. By the Knaster-Tarski fixed-point theorem [18] we know that f has greatest and least fixed-points, written νf and μf respectively.

$$
\begin{aligned}
\|X\|_{\mathcal{V}}^{\mathcal{T}} &\stackrel{\text{def}}{=} \mathcal{V}(X) \\
\|\phi_1 \wedge \phi_2\|_{\mathcal{V}}^{\mathcal{T}} &\stackrel{\text{def}}{=} \|\phi_1\|_{\mathcal{V}}^{\mathcal{T}} \cap \|\phi_2\|_{\mathcal{V}}^{\mathcal{T}} \\
\|\phi_1 \vee \phi_2\|_{\mathcal{V}}^{\mathcal{T}} &\stackrel{\text{def}}{=} \|\phi_1\|_{\mathcal{V}}^{\mathcal{T}} \cup \|\phi_2\|_{\mathcal{V}}^{\mathcal{T}} \\
\|[\![L]\!]\phi\|_{\mathcal{V}}^{\mathcal{T}} &\stackrel{\text{def}}{=} \{E \in S \mid \forall E' \in S, a \in L. E \stackrel{a}{\Rightarrow} E' \text{ implies } E' \in \|\phi\|_{\mathcal{V}}^{\mathcal{T}}\} \\
\|\nu X.\phi\|_{\mathcal{V}}^{\mathcal{T}} &\stackrel{\text{def}}{=} \nu f, \text{ where } f(\mathcal{E}) \stackrel{\text{def}}{=} \|\phi\|_{\mathcal{V}[X:=\mathcal{E}]}^{\mathcal{T}} \\
\|\mu X.\phi\|_{\mathcal{V}}^{\mathcal{T}} &\stackrel{\text{def}}{=} \mu f, \text{ where } f(\mathcal{E}) \stackrel{\text{def}}{=} \|\phi\|_{\mathcal{V}[X:=\mathcal{E}]}^{\mathcal{T}}
\end{aligned}
$$

If ϕ is a closed formula and $P \in \|\phi\|_{()}^{\mathcal{T}}$ then we write $P \models \phi$ and say that P *satisfies* ϕ.

Formulas of the diamond-free modal mu-calculus can express safety properties [1]. However, the logic can also express weak liveness properties, which state that something will eventually happen in the absence of deadlock. The rough intuition of the following theorem is that nothing bad can happen in a process if nothing bad can happen in a process that simulates it.

Theorem 7. *Let $\mathcal{T} = (S, A, \{\stackrel{a}{\rightarrow} \mid a \in A\})$ be a transition system with τ in A, let P and Q be elements of S, and let ϕ be a formula of the diamond-free modal mu-calculus. Then*

$$(Q \models \phi \text{ and } P \ll Q) \Rightarrow P \models \phi.$$

Proof. This result is proved directly in the modal mu-calculus in [7], and proved using a logic with infinitary conjunction and disjunction in [3].

This theorem is useful because a process with many states can sometimes be simulated by a process with few states. Imagine that our goal is to show that $Cr(P)$ satisfies a formula ϕ of the diamond-free modal mu-calculus. Suppose that P' is a process that simulates P but has fewer states. Suppose also that $Cr(P')$ satisfies ϕ. Then $Cr(Q) \ll Cr(P)$ because Cr preserves weak simulation and $Cr(Q)$ satisfies ϕ by this theorem. Note that by Prop. 4 the weakly-plus simulates relation also preserves formulas of the diamond-free modal mu-calculus.

The following fact is a simple corollary of the theorem.

Proposition 8. *Let $\mathcal{T} = (S, A, \{\stackrel{a}{\rightarrow} \mid a \in A\})$ be a transition system with τ in A, let P and Q be elements of S, and let ϕ be a formula of the diamond-free modal mu-calculus. Then*

$$P \approx Q \Rightarrow (P \models \phi \Leftrightarrow Q \models \phi)$$

5 Analysis of the Slow-Scan System

Here we describe the slow-scan system and present three models of the system. The first model is written in CCS. The second model, written in CCS plus new operators, shows how special-purpose operators improve the modelling of faults and fault-handling mechanisms. The third model is like the second except that some components are modelled more realistically. We show that it satisfies some important properties by appealing to the simpler second model.

5.1 The Slow-Scan System

A railway signalling interlocking adjusts, at the request of the signal operator, the setting of colored light signals and direction controlling points in the railway to permit the safe passage of trains. An example of a modern interlocking is the *Solid State Interlocking* (SSI) [11], developed by British Rail in collaboration with GEC ALSTHOM Signalling and Westinghouse Signals. Figure 1 shows an SSI and the devices it controls. Operator commands are broadcast over a high-speed communication link to track-side functional modules (TFMs), which control signals and points.

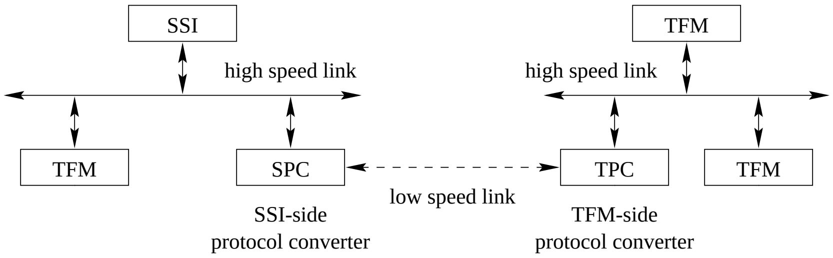

Fig. 1. The slow-scan system

It is important that TFM and communication link failures be detected quickly by the SSI. Instead of sending signal and point commands only as needed, the SSI sends a message of the form $\langle \text{TFM-address}, \text{state} \rangle$ to every attached TFM about once every second. The SSI then waits at most a few milliseconds for the addressed TFM to respond with the current state of its attached device. This command-cycling scheme provides fast failure detection, and allows the interlocking to take immediate action to maintain the system in a safe state.

In some rail networks the signals and points controlled by a single SSI lie many miles apart. The TFMs in such a network are connected by long high-speed links. A cheaper *low-grade link* (LGL) cannot directly replace a high-speed link because it does not have the bandwidth needed for the TFM command-cycling scheme. However, a low-speed link is feasible if used with a pair of *protocol converters*. The *slow-scan* system, shown in Figure 1, is built from a low-grade link and two protocol converters. The SSI-side protocol converter (SPC) accepts TFM commands once every second, and responds immediately with track-side device status, but only sends TFM commands along the low-grade link occasionally. The TFM-side protocol converter (TPC) sends a command to each attached TFM once every second. Responses from the TFMs are occasionally sent by the TPC to the SPC to update the SPC's device status information. The SPC appears to the SSI as a TFM and the TPC appears to a TFM as an SSI.

5.2 A CCS Model of the System

Our model is based on the model in [6]. The SSI and TFM's are taken as outside the scope of the model. The model therefore contains a process for the LGL, the TPC, the SPC, and a system clock that synchronizes the TPC and SPC.

We model the LGL as two unidirectional communication links, each with the capacity to buffer one message. Relabelling function f_c maps $\mathtt{in}$ to $\mathtt{c1}$, $\overline{\mathtt{out}}$ to $\overline{\mathtt{c2}}$, $\overline{\mathtt{out}}_u$ to $\overline{\mathtt{c2}}_u$, and all other actions to themselves. Relabelling function f_s maps $\mathtt{in}$ to $\mathtt{s1}$, $\overline{\mathtt{out}}$ to $\overline{\mathtt{s2}}$, $\overline{\mathtt{out}}_u$ to $\overline{\mathtt{s2}}_u$, and all other actions to themselves.

$$
\begin{aligned}
\mathit{Comm} &\stackrel{\text{def}}{=} \mathtt{in}(m).\mathit{Comm}'(m) + \overline{\mathtt{out}}_u\,.\mathit{Comm} + \mathtt{fail}\,.\mathit{CommF}\\
\mathit{Comm}'(m) &\stackrel{\text{def}}{=} \overline{\mathtt{out}}(m).\mathit{Comm} + \mathtt{fail}\,.\mathit{CommF}\\
\mathit{CommF} &\stackrel{\text{def}}{=} \mathtt{in}\,.\mathit{CommF} + \overline{\mathtt{out}}_u\,.\mathit{CommF}\\
\mathit{LGL} &\stackrel{\text{def}}{=} \mathit{Comm}[f_c] \mid \mathit{Comm}[f_s]
\end{aligned}
$$

Actions $\mathtt{in}$ and $\overline{\mathtt{out}}$ model the input and output ports of a link. The link can always read the input port but can only write on the output port if a message is buffered. However, the link can synchronize on $\overline{\mathtt{out}}_u$ when no message is buffered. Action $\mathtt{fail}$ models a link failure. The only failure we model is the spontaneous break of the communication medium. Once a link fails it no longer synchronizes on its output port.

The SPC is modelled as a parametrized process.

$$
\begin{aligned}
\mathit{SPC} &\stackrel{\text{def}}{=} \mathit{SPC}(c_0, s_0, 0)\\
\mathit{SPC}(c,s,i) &\stackrel{\text{def}}{=} \mathtt{comm\_in}(c').(\tau.\mathit{SPC}(c,s,i) + \tau.\mathit{SPC}(c',s,i)) +\\
&\qquad \overline{\mathtt{stat\_out}}(s).\mathit{SPC}(c,s,i) +\\
&\qquad \overline{\mathtt{c1}}(c).\mathit{SPC}(c,s,i) + \mathtt{s2}(s').\mathit{SPC}(c,s',0) + \mathtt{s2}_u\,.\mathit{SPC}(c,s,i) +\\
&\qquad \mathtt{mcs}\,.\,\overline{\mathtt{c1}}(c).(\text{if } i > n \text{ then } \mathtt{det}\,.\mathit{SPCF} \text{ else } \mathit{SPC}(c,s,i+1))\\
\mathit{SPCF} &\stackrel{\text{def}}{=} \mathtt{comm\_in}(c).\mathit{SPCF} + \mathtt{s2}(s).\mathit{SPCF} + \mathtt{s2}_u\,.\mathit{SPCF} + \mathtt{mcs}\,.\mathit{SPCF}
\end{aligned}
$$

Actions $\mathtt{comm\_in}$ and $\overline{\mathtt{stat\_out}}$ model the command input and status output ports of an SPC. A command can be received and a status message can be output at any time. Actions $\overline{\mathtt{c1}}$, $\mathtt{s2}$, and $\mathtt{s2}_u$ model the ports on which interaction with the LGL takes place. Action $\mathtt{mcs}$ models the reception of a clock tick. After each clock tick occurs a command is sent to the LGL. Action $\mathtt{det}$ models the detection of an LGL failure by the SPC. It arises if n clock ticks have occurred without the reception of a status message over the LGL.

The TPC model is similar:

$$
\begin{aligned}
TPC &\stackrel{\text{def}}{=} TPC(c_0, s_0, 0)\\
TPC(c,s,i) &\stackrel{\text{def}}{=} \overline{\texttt{comm\_out}}(c).(\tau.\,TPC(c,s,i) + \tau.\,TPC(c',s,i)) +\\
&\qquad \texttt{stat\_in}(s').\,TPC(c,s',i) +\\
&\qquad \overline{\texttt{s1}}(s).\,TPC(c,s,i) + \texttt{c2}(c').\,TPC(c',s,0) + \texttt{c2}_u\,.\,TPC(c,s,i) +\\
&\qquad \texttt{mct}\,.\,\overline{\texttt{s1}}(s).(\text{if } i > n \text{ then } \texttt{det}\,.\,TPCF \text{ else } TPC(c,s,i+1))\\
TPCF &\stackrel{\text{def}}{=} \texttt{stat\_in}(s).\,TPCF + \texttt{c2}(c).\,TPCF + \texttt{c2}_u\,.\,TPCF + \texttt{mct}\,.\,TPCF
\end{aligned}
$$

The TPC outputs commands and accepts status messages. It sends a status message to the LGL every clock tick. If n clock ticks occur without the reception of a command over the LGL then action `det` occurs.

Finally we model the clock process.

$$Clock \stackrel{\text{def}}{=} \texttt{tick}\,.\,\overline{\texttt{mcs}}\,.\,\overline{\texttt{mct}}\,.\,Clock$$

Action `tick` models a visible clock tick; actions $\overline{\texttt{mcs}}$ and $\overline{\texttt{mct}}$ model clock signals to the SPC and TPC.

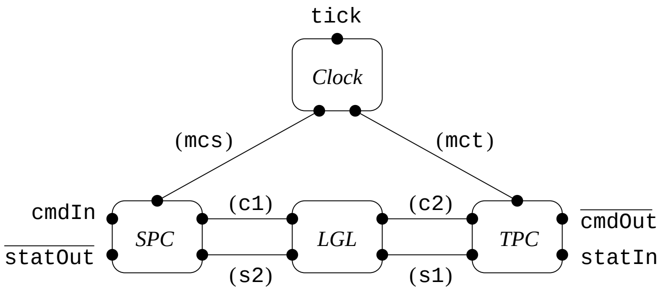

Fig. 2. Flow Diagram of the Slow-Scan System

The complete slow-scan model is obtained by putting the components together. Figure 2 is a flow diagram showing the components of the model and how they are connected through their ports. The model has about 26,000 states.

$$SS_1 \stackrel{\text{def}}{=} (SPC \mid LGL \mid TPC \mid Clock)\backslash\{\texttt{s1}, \texttt{s2}, \texttt{s2}_u, \texttt{c1}, \texttt{c2}, \texttt{c2}_u, \texttt{mcs}, \texttt{mct}\}$$

We are interested in knowing whether this model satisfies two main fault tolerance properties. First, are false alarms possible? In other words, can an LGL failure be detected before an LGL failure has occurred? Second, are failures always detected? These properties can be expressed as formulas of the diamond-free modal mu-calculus.

$$
\begin{aligned}
NoFalseAlarms &\stackrel{\text{def}}{=} \nu X.[\![\texttt{det}]\!]\,\texttt{ff} \wedge [\![-\,\texttt{fail}]\!]X\\
FailuresDetected &\stackrel{\text{def}}{=} \nu X.[\![-\,\texttt{fail}]\!]X \;\wedge\\
&\qquad\qquad [\![\texttt{fail}]\!](\mu Y.\nu Z.[\![-\,\texttt{tick}, \texttt{det}]\!]Z \;\wedge\; [\![\texttt{tick}]\!]Y)
\end{aligned}
$$

The property *NoFalseAlarms* can be understood recursively as "action `det` cannot occur, and after any non-`fail` action occurs, *NoFalseAlarms* holds". The property *FailuresDetected* is more difficult to understand because it incorporates the assumption that the clock continues to tick. The formula states that after any initial `fail` action occurs, there can be no path containing infinitely many `tick` actions and no `det` action.

Proposition 9. *SS_1 satisfies NoFalseAlarms and FailuresDetected.*

Proof. Verified with the Concurrency Workbench. □

The modelling of faults and of the fault detection mechanism is awkward here. In modelling the LGL we want to capture that a failure can occur in any state, so `fail` actions must be added to the model at several different points. The modelling of the nonfaulty LGL behavior and the modelling of faults are combined. Similarly, in the SPC and TPC models the modelling of the nonfaulty behavior and the modelling of the fault detection mechanism are combined.

Another problem with the model is that it has been simplified to make model checking feasible and to avoid committing to details that are not specified in the design of the system. For example, the model attempts to capture various SPC policies for sending commands along the low-grade link by allowing action $\overline{\texttt{c1}}$ to occur at any time. We have not shown that implementations of slow-scan will satisfy properties *NoFalseAlarms* and *FailuresDetected*.

In [10] the slow-scan system was modelled in a version of CCS extended with priorities. The use of priorities allows for more natural modelling and reduces the state space of the model. However, like the CCS model it suffers from awkward fault modelling and excessive simplification.

5.3 Modelling Slow-Scan with New Operators

We now show how the use of new operators improves the modelling of the slow-scan system. To model LGL faults we define new operator LF ("link failure"), which allows a failure to happen at any time, causing output actions of a link to be hidden.

$$\frac{P \xrightarrow{\alpha} P'}{\mathrm{LF}(P) \xrightarrow{\alpha} \mathrm{LF}(P')} \qquad \frac{}{\mathrm{LF}(P) \xrightarrow{\texttt{fail}} P \backslash\!\backslash \{\texttt{c2}, \texttt{s2}\}}$$

To revise the LGL model we first define a nonfaulty communications link. A faulty link is then defined as a nonfaulty link extended with a link failure. The advantage of using the fault operator LF would be greater if the LGL model was more complicated.

$$Comm \stackrel{\text{def}}{=} \texttt{in}(m).\,\overline{\texttt{out}}(m).Comm + \overline{\texttt{out}}_u\,.Comm$$
$$LGL \stackrel{\text{def}}{=} \mathrm{LF}(Comm[f_c]) \mid \mathrm{LF}(Comm[f_s])$$

To model the failure detection mechanism used in the SPC and TPC we use two new operators. Operator ex ("exception") is an exception handler for action $\texttt{det}$. A process $P \text{ ex } Q$ evolves to Q when P performs action $\texttt{det}$.

$$\frac{P \xrightarrow{\alpha} P'}{P \text{ ex } Q \xrightarrow{\alpha} P' \text{ ex } Q}\ \alpha \neq \texttt{det} \qquad \frac{P \xrightarrow{\texttt{det}} P'}{P \text{ ex } Q \xrightarrow{\texttt{det}} Q}$$

We also use the watchdog timer operator $\mathrm{WD}_{n,\texttt{c1}}$ of Section 2. It produces action $\texttt{det}$ if n clock ticks occur before action $\texttt{c1}$ occur.

To revise the SPC model we first define a nonfaulty SPC and the SPC failure state. The SPC is then defined as a nonfaulty SPC that can time out and evolve to the failure state.

$$\begin{aligned}
\mathit{SPCB} &\stackrel{\text{def}}{=} \mathit{SPCB}(c_0, s_0)\\
\mathit{SPCB}(c,s) &\stackrel{\text{def}}{=} \texttt{comm_in}(c').(\tau.\mathit{SPCB}(c,s) + \tau.\mathit{SPCB}(c',s)) +\\
&\quad\ \overline{\texttt{stat_out}}(s).\mathit{SPCB}(c,s) +\\
&\quad\ \overline{\texttt{c1}}(c).\mathit{SPCB}(c,s) + \texttt{s2}(s').\mathit{SPCB}(c,s') + \texttt{s2}_u\,.\mathit{SPCB}(c,s) +\\
&\quad\ \texttt{mcs}\,.\,\overline{\texttt{c1}}(c).\mathit{SPCB}(c,s)\\
\mathit{SPCF} &\stackrel{\text{def}}{=} \texttt{comm_in}(c).\mathit{SPCF} + \texttt{s2}(s).\mathit{SPCF} + \texttt{s2}_u\,.\mathit{SPCF} + \texttt{mcs}\,.\mathit{SPCF}\\
\mathit{SPC} &\stackrel{\text{def}}{=} \mathrm{WD}_{2,\texttt{c2}}(2, \mathit{SPCB}) \text{ ex } \mathit{SPCF}
\end{aligned}$$

The new TPC definition is similar. We let the new definitions of LGL, SPC, and TPC replace the old ones, and call the new model SS_2. We have defined our new process operators with the PAC and verified automatically that processes SS_1 and SS_2 are observation equivalent.

Proposition 10. *SS_2 satisfies NoFalseAlarms and FailuresDetected.*

Proof. Process SS_1 satisfies the formulas. The Concurrency Workbench verifies that SS_1 and SS_2 are observation equivalent, so by Prop. 8 process SS_2 satisfies the formulas. □

5.4 A More Realistic Model

The models of the last two sections are highly simplified. They do not capture the contents of command and status messages or the local variables of the SPC and TPC. We now examine a more realistic slow-scan model and show how our modelling of faults helps in showing that it satisfies properties *NoFalseAlarms* and *FailuresDetected*.

In our previous SPC model we allowed commands to be transmitted over the LGL at any time. We can instead model that a command is transmitted only when it differs from the previous command.

$$\begin{aligned}
\mathit{SPCB}(c,s) &\stackrel{\text{def}}{=} \texttt{comm_in}(c').\,\overline{\texttt{stat_out}}(s).\\
&\quad\ (\text{if } c = c' \text{ then } \overline{\texttt{c1}}\,.\mathit{SPCB}(c',s) \text{ else } \mathit{SPCB}(c,s)) +\\
&\quad\ \texttt{s2}(s').\mathit{SPCB}(c,s') + \texttt{s2}_u\,.\mathit{SPCB}(c,s) +\\
&\quad\ \texttt{mcs}\,.\,\overline{\texttt{c1}}(c).\mathit{SPCB}(c,s)
\end{aligned}$$

To complete the model we make an analogous change to the SPC. The resulting model, which we call SS_3, has about 250,000 states. This example illustrates how sensitive the size of the model is to increases in the size of components.

To show that SS_3 satisfies *NoFalseAlarms* and *FailuresDetected* we need only show that the revised versions of *SPCB* and *TPCB* are weakly-plus simulated by the original versions. The following proposition is straightforward to prove by exhibiting a simulation relation.

Proposition 11. *Let $SPCB'$ and $TPCB'$ be the revised versions of processes SPCB and TPCB. Then $SPCB' \ll_+ SPCB$ and $TPCB' \ll_+ TPCB$.*

Proposition 12. *SS_3 satisfies NoFalseAlarms and FailuresDetected.*

Proof. SS_2 satisfies the formulas by Prop. 10. Let $SPCB'$ and $TPCB'$ be the revised versions of processes *SPCB* and *TPCB*. Then $SS_3 \ll_+ SS_2$ by Prop. 11 and Theorem 5, since all operators of SS_3 are well-founded and valid for weak-plus transitions, and $SS_3 \ll SS_2$ by Prop. 4. Therefore by Theorem 7 SS_3 also satisfies the formulas. □

6 Future Work

One can define powerful new process operators for which implementation may be difficult. We have not described a general approach to implementing process operators or even what 'implementation' means. One notion of the implementation of an operator $\mathcal{O}$ is to find, for a process $\mathcal{O}(P)$, a CCS process P' such that $\mathcal{O}(P)$ and P' are observation equivalent. However, the implementation here might depend on P. A notion of the *modular* implementation of an operator $\mathcal{O}$ is to find a CCS process *context* C such that $C[P]$ is observation equivalent to $\mathcal{O}(P)$ for all CCS processes P. The proof that $C[P]$ and $\mathcal{O}(P)$ are observation equivalent could be done by exhibiting a bisimulation relation. This is the approach used by Bloom in [4]. We have not yet answered the interesting question: which tyft/tyxt operators have modular implementations in CCS?

A CCS

The terms of CCS represent processes that perform *actions*. The set *Act* of CCS actions contains *names* $(a, b, \ldots)$, *co-names* $(\overline{a}, \overline{b}, \ldots)$, and the special "silent" action τ. We let α, β range over *Act*. The set of *labels* contains all actions except τ. We let l range over labels. Complementation extends to all labels: $\overline{\overline{l}} = l$.

Processes are given as *agent expressions* having the following syntax, where A ranges over agent constants, L ranges over sets of labels, and f ranges over functions from actions to actions:

$$P ::= A \;\big|\; \alpha.P \;\big|\; \sum_{i \in I} P_i \;\big|\; P_1 \mid P_2 \;\big|\; P \backslash L \;\big|\; P[f]$$

We write $P_1 + P_2$ as an abbreviation for $\sum_{i\in\{1,2\}} P_i$, and 0 (the *inactive* agent) as an abbreviation for summation over the empty set. Constant definitions are of the form $A \stackrel{\text{def}}{=} P$. We refer to the set of all agents expressions and agent constants as simply *agents*, denoted by $\mathcal{P}$. We let P, Q range over $\mathcal{P}$.

The meaning of agents is given as the transition system $(\mathcal{P}, Act, \{\stackrel{\alpha}{\rightarrow} \mid \alpha \in Act\})$, where $\stackrel{\alpha}{\rightarrow}$ is a labelled transition relation over agents. We write $P \stackrel{\alpha}{\rightarrow} P'$ if $(P, P') \in \stackrel{\alpha}{\rightarrow}$. The relation $\stackrel{\alpha}{\rightarrow}$ is defined as the least relation satisfying the following rules:

$$. : \quad \alpha.P \stackrel{\alpha}{\rightarrow} P$$

$$+ : \frac{P_j \stackrel{\alpha}{\rightarrow} P_j'}{\sum_{i\in I} P_i \stackrel{\alpha}{\rightarrow} P_j'} \; j \in I$$

$$\mid : \frac{P \stackrel{\alpha}{\rightarrow} P'}{P \mid Q \stackrel{\alpha}{\rightarrow} P' \mid Q} \qquad \frac{Q \stackrel{\alpha}{\rightarrow} Q'}{P \mid Q \stackrel{\alpha}{\rightarrow} P \mid Q'} \qquad \frac{P \stackrel{l}{\rightarrow} P' \quad Q \stackrel{\overline{l}}{\rightarrow} Q'}{P \mid Q \stackrel{\tau}{\rightarrow} P' \mid Q'}$$

$$\backslash : \frac{P \stackrel{\alpha}{\rightarrow} P'}{P\backslash L \stackrel{\alpha}{\rightarrow} P'\backslash L} \; \alpha \notin L \cup \overline{L}$$

$$f : \frac{P \stackrel{\alpha}{\rightarrow} P'}{P[f] \stackrel{f(\alpha)}{\longrightarrow} P'[f]}$$

$$\text{def} : \frac{P \stackrel{\alpha}{\rightarrow} P'}{A \stackrel{\alpha}{\rightarrow} P'} \; A \stackrel{\text{def}}{=} P$$

A *hiding* operator, which effectively renames labels to τ actions, can be defined using CCS operators. For simplicity we instead define it directly:

$$\backslash\!\backslash : \frac{P \stackrel{\alpha}{\rightarrow} P'}{P\backslash\!\backslash L \stackrel{\alpha}{\rightarrow} P'\backslash\!\backslash L} \; \alpha \notin L \cup \overline{L} \quad \frac{P \stackrel{\alpha}{\rightarrow} P'}{P\backslash\!\backslash L \stackrel{\tau}{\rightarrow} P'\backslash\!\backslash L} \; \alpha \in L \cup \overline{L}$$

Value-passing CCS adds parametrized actions, parametrized agents, and conditional statements to CCS. These new features are defined by translation to CCS. Assume a fixed set V of data values and a notation for data expressions. Input action $a(x).P$ is defined to be $\sum_{v\in V} a_v.P[v/x]$, where $P[v/x]$ is the agent obtained by substituting v for all free occurrences of x in P. Output action $\overline{a}(e).P$ is defined to be $\overline{a}_v.P$, where v is the value obtained by evaluating data expression e. Parametrized agent definition $A(x_1, x_2, \ldots, x_n) \stackrel{\text{def}}{=} P$ is defined as the set of definitions $\{(A_{v_1,v_2,\ldots,v_n} \stackrel{\text{def}}{=} P[v_1/x_1, \ldots, v_n/x_n]) : (v_1, \ldots, v_n) \in V_{x_1} \times \cdots \times V_{x_n}\}$. Parametrized agent constant $A(e_1, \ldots, e_n)$ is defined to be $A_{v_1,\ldots,v_n}$, where v_i is the value obtained by evaluating data expression e_i. Finally, conditional statement *if* b *then* P *else* Q is defined to be P if the boolean expression b evaluates to true, else P. b

Acknowledgements

We would like to thank Stuart Anderson for helpful discussions and Steve Sims for help with the PAC and North Carolina Concurrency Workbench.

References

1. Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.
2. Michael Barborak, Miroslaw Malek, and Anton Dahbura. The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.
3. S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis. Property preserving simulations. In *Proceedings of CAV '92, LNCS 663*, pages 260–273, 1992.
4. Bard Bloom. Structured operational semantics as a specification language. In *Proceedings of LICS '94*. IEEE Computer Society Press, 1994.
5. Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.
6. Glenn Bruns. A case study in safety-critical design. In G.v. Bochmann and D.K. Probst, editors, *Proceedings of CAV '91, LNCS 575*, pages 220–233, 1991.
7. Glenn Bruns. *Process Abstraction in the Verification of Temporal Properties*. PhD thesis, University of Edinburgh, 1997.
8. R. Cleaveland, E. Madelaine, and S. Sims. Generating front ends for verification tools. In *Proceedings of TACAS '95, LNCS 1019*, pages 153–173, 1995.
9. R. Cleaveland and S. Sims. The NCSU concurrency workbench. In R. Alur and T. Henzinger, editors, *Proceedings of CAV '96*, 1996.
10. Rance Cleaveland, Gerard Lüttgen, V. Natarajan, and Steve Sims. Modeling and verifying distributed systems using priorities: A case study. *Software Concepts and Tools*, 15:50–62, 1996.
11. A.H. Cribbens. Solid-state interlocking (SSI): an integrated electronic signalling system for mainline railways. *IEE Proceedings*, 134(B3), 1987.
12. J. F. Groote and F. W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.
13. D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
14. Parag K. Lala. *Fault Tolerant and Fault Testable Hardware Design*. Prentice Hall, 1985.
15. Patrick Lincoln and John Rushby. The formal verification of an algorithm for interactive consistency under a hybrid fault model. In *Proceedings of CAV '93*, 1993.
16. Nancy A. Lynch. Multivalued possibilities mapping. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, pages 519–543, 1989. LNCS 430.
17. Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–320, 1990.
18. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. of Maths*, 5:285–309, 1955.
19. J. von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. *Annals of Mathematical Studies*, 34:43–98, 1956.

This article was processed using the LaTeX macro package with LLNCS style