Model-Checking CSP-Z

Alexandre Mota and Augusto Sampaio
Federal University of Pernambuco
P. O. BOX 7851 Cidade Universitária
50740-540 Recife - PE Brazil
{acm,acas}@di.ufpe.br

Abstract. Model-checking is now widely recognised as an efficient method for analysing computer system properties, such as deadlock-freedom. Its practical applicability is due to existing automatic tools which deal with tedious proofs. Another increasingly research area is formal language integration where the capabilities of each language are used to capture precisely some aspects of a system. In this paper we describe a formal strategy for deadlock analysis of specifications in CSP-Z (a language which integrates CSP and Z). We also show how FDR (a model-checker originally developed for CSP) can be adapted for CSP-Z. Finally, we present a subset of a CSP-Z formal specification of a real Brazilian artificial microsatellite, and use FDR to check that the specification is deadlock-free.

1 Introduction

There is an increasing interest, among the Computer Science community, in model-checkers. These are programs that work by checking every possible state of a system to verify some specified property such as *deadlock-freedom*. Although model-checking is limited to certain problems, those that have not the *exponential state explosion problem*, it has a great advantage over, for example, general theorem proving because it is fully automatic whereas the latter is not.

Linking theories is also a recent trend in the area of formal methods. The main advantage of these is to capture more than one aspect of a system using a uniform notation. For example, concurrent specification languages, such as CSP [12] or CCS [17], can characterise precisely the behaviour aspects of a system meanwhile they are not suitable for stating concisely (and abstractly) the system data structures. This is because the data structures in these languages are similar to those of a programming language. On the other hand, languages such as Z [20], VDM [14] and OBJ [11] have great expressive power to describe abstract data structures but lack the notion of operation evaluation order. Currently, there are a lot of language integration proposals. Some examples are LOTOS [2], Temporal Logic and CSP [16], LOTOS and Z [6] and CSP-Z [7, 8].

In this paper we use CSP-Z, a language which integrates CSP and Z both sintactically and semantically. CSP-Z was defined such that apart from enabling one to deal with the behaviour and the data structure aspects of a system independently, the resulting specification can also be refined independently, i.e., the

approach to refinement is compositional in the sense that refining the CSP or the Z part (with some constraints) leads to the refinement of the entire CSP-Z specification. The main contribution of this paper is a strategy for deadlock analysis of CSP-Z based on [4] and its mechanisation by adapting the FDR model-checker [9], which was originally developed to deal exclusively with CSP specifications. Finally, we present a case study of a Brazilian artificial microsatellite (SACI-1) being developed by the Brazilian Space Research Institute (INPE), where we apply our strategy for deadlock analysis with the aid of FDR. This case study is a small subset of a detailed formalisation and analysis of the SACI-1, described in [1].

The rest of this paper is organised as follows. Section 2 introduces the CSP-Z language through an example, and briefly describes its syntax and semantics. In Section 3 we present a technique developed by Brookes and Roscoe [4] to analyse the deadlock-freedom property of a CSP specification, and explain how FDR implements this technique. Based on this technique we develop a deadlock analysis strategy for CSP-Z specifications and show how to adapt FDR to work for CSP-Z; this is presented in Section 4. Section 5 illustrates this approach through the specification and analysis of the On-Board Computer system (OBC) of the SACI-1 Brazilian microsatellite. Finally, we consider what are the benefits of using an integrated language and the practical advantages and limitations of using FDR in this setting. We assume some familiarity with the languages CSP and Z.

2 CSP-Z

The language CSP-Z [7, 8] is a conservative extension of both CSP and Z in the sense that the syntactical and semantical aspects of CSP is fully preserved while Z operations have a slightly different interpretation. In order to give an overview of CSP-Z we present part of the specification of our case study, fully described in Section 5. In [8] the integration of CSP with an object oriented extension of Z is presented. Here we consider the plain Z notation.

2.1 A simple Example

The Watch-Dog Timer or simply WDT is a process of the SACI-1 microsatellite responsible for waiting a reset signal that comes (periodically) from another SACI-1 process, the Fault-Tolerant Router (FTR). If this reset signal does not come, the WDT sends a recovery signal to the FTR in order to initiate a recovery process to normalise the situation. This procedure occurs three times and, if after that, the FTR does not respond, than the WDT considers the FTR faulty.

A CSP-Z specification is encapsulated into a **spec** and **end_spec** scope, where the name of the specification follows these keywords. The interface is the first part of a CSP-Z specification and is used to declare the external channels (keyword **channel**) and the local (or hidden) ones (keyword **local_channel**). Each list of channels has an associated Z schema type, where the empty schema type ([])

denotes a list of events, i.e., channels which do not communicate values. The concurrent behaviour of the system is introduced by the keyword main, where other equations can be added to obtain a more structured CSP specification.

spec *WDT*

channel *clockWDT*:[clk : CLOCK]
channel *reset, recover, failFTR*:[]
local_channel *timeOut, noTimeOut*: []

The equation introduced below with the keyword main describes a totally independent behaviour between the processes *Signal* and *Verify* using the CSP interleaving operator (|||). *Signal* is simply characterised by waiting for consecutive reset signals, i.e., waiting for a *reset* and then (→) behaving like *Signal* again (i.e., waiting for another signal). *Verify* waits for a clock, then checks whether a reset signal arrived at the right period or not via the choice operator (□). If a *timeOut* occurs then the WDT tries to send a recovery signal to the FDR. If the FTR is not ready to synchronise in this event then the WDT assumes that the FTR is faulty and then finishes its execution (behaving like skip).

main=*Signal* ||| *Verify*
Signal=(*reset*→*Signal*)
Verify=(*clockWDT*→(*noTimeOut*→*Verify*
□ *timeOut*→(*recover*→*Verify*
□ *failFTR*→skip))

After introducing the behaviour of the WDT, the data structures used are declared. In order to fix a timeout and to know if the clock achieved this maximum we introduce two constants, WDTtOut and WDTP. The system state (State) has simply a declarative part where is recorded the number of cycles that the WDT tries to recover the FTR and the value of the last clock received. The initialisation schema (Init) asserts that the number of cycles is initially zero.

WDTtOut : CLOCK	State $\widehat{=}$ [cycles : LENGTH; time : CLOCK]
WDTP : CLOCK ↔ CLOCK	Init $\widehat{=}$ [State' | cycles' = 0]

The following schemas are standard Z schemas (with a declaration part and a predicate which constrains the values of the declared variables) except that their names are originated from the channel names, prefixing the keyword com_. Informally, the meaning of a CSP-Z specification is that, when a CSP event *c* occurs the respective Z operation com_*c* is executed, possibly changing the data structures. Further, when there is no schema name associated with a given channel, this means that no change of state occurs. An observation is that every external communication has a type, then when no type is explicit CSP-Z assumes the type signal, where the desired behaviour is merely that of a synchronisation and not a value passing. For events with an associated non-empty schema type, the Z schema must have input or output variables with corresponding names in order to exchange communicated values between the CSP and the Z parts. Hence, the input variable clk? receives values communicated through the *clockWDT* channel. For schemas where prime (') variables are omitted, we assume that no modifications occur, i.e., in the schema com_reset below it is implicit that the

time component is not modified (time′ = time).

$$com\_reset \mathrel{\widehat=} [\Delta State \mid cycles' = 0]$$
$$com\_clockWDT \mathrel{\widehat=} [\Delta State; clk? : CLOCK \mid time' = clk?]$$

When a Z schema has a precondition differing from true then it imposes a restriction on the occurrence of a CSP event. It is like a CSP guard, i.e., if the precondition is true then the event is allowed to occur normally, otherwise it is refused and the process behaves like the canonical deadlock process (stop).

Note that the precondition of the schema com_noTimeOut is the predicate $\neg$ WDTP(time, WDTtOut) meaning that the timeout has not yet occurred, whereas the precondition of com_timeOut specifies the occurrence of timeout.

$$com\_noTimeOut \mathrel{\widehat=} [\Xi State \mid \neg WDTP(time, WDTtOut)]$$
$$com\_timeOut \mathrel{\widehat=} [\Delta State \mid WDTP(time, WDTtOut) \wedge cycles' = cycles + 1]$$

As already explained, the recovery process is attempted for 3 times, after which the WDT assumes that the FTR is faulty.

$$com\_recover \mathrel{\widehat=} [\Xi State \mid cycles < 3]$$
$$com\_failFTR \mathrel{\widehat=} [\Xi State \mid cycles = 3]$$

end_spec *WDT*

2.2 Brief Explanation of the Semantics of CSP-Z

The CSP semantical model assumed as standard is the Failures-Divergence model [3]. This means that a specification can be characterised by a set of pairs ($\mathcal{F}$, $\mathcal{D}$) where $\mathcal{F}$ is the failures set and $\mathcal{D}$ is the set of divergences. The failures of a process P is a set of pairs (s, X), of traces (observed events) and refusals, such that after P performs the trace s it cannot engage in any event of the refusal set X. The divergences of a process P are sets of traces such that after P performs any trace of this set it engages in an infinite loop of hidden events. The language CSP-Z is a semantical integration of CSP and Z in that it is given a Failures-Divergence meaning to Z [7, 8]. This interpretation is required in order to allow Z components to be combined using the CSP operators like interleaving (|||) and parallelism (||).

As explained above, a CSP-Z specification is a parallel combination of the CSP and the Z parts via the channel names, such that on the occurrence of a channel c the corresponding Z schema com_c is activated. As the semantics of CSP-Z is also based on the Failures-Divergence model, we should explain what happens when a given event c occurs successfully, when it is refused and when it leads to divergence. These situations are considered below.

Suppose that c is a CSP untyped channel with corresponding schema com_c. If the event c occurs, the guard of the event and the precondition of the schema com_c are satisfied, this characterises a successful execution step. In this case, the state space is subjected to the predicate part of com_c and the CSP part also evolves (where the event c is added to the trace of the process). Now, suppose that the channel c is a typed channel. If c?x is performed and the value v assigned

to x cannot be treated by the input part of com_c, due to a type incompatibility, then c is refused. Similarly, if com_c exhibits a value v from one of its output variables which cannot be communicated through c!v then c is also refused. Finally, suppose that c is not refused by the Z part, according to the above explanation, then if the value communicated falsifies the precondition of com_c then the whole process diverges. A more formal presentation of the semantics is given in Appendix A.1.

Formalising the above explanation, we can state precisely a refusal or a divergence introduced by the Z part. Let c be a channel and tr be a trace then

- com_c is defined as a standard Z schema operation describing the state effect on the occurrence of c
- enable_c $\widehat{=}$ $\exists$ State'; In?; Out! • com_c is a constraint between the values communicate from the CSP part to the Z part and vice-versa
- pre com_c $\widehat{=}$ $\exists$ State'; Out! • com_c
- com_$\langle\rangle$ $\widehat{=}$ Init, for an empty trace
- com_(c $\frown$ tr) $\widehat{=}$ com_c ; com_tr, where $\frown$ is the concatenation operator and ; is the Z schema composition operator

Thus, a refusal can occur if $\neg$enable_c and a divergence if enable_c$\wedge\neg$ pre com_c.

3 Deadlock Analysis for CSP: Theory and Tools

Concurrent programming is more complex than sequential one mainly because the number of states grows exponentially with the number of processes that compose the system. Describing precisely a concurrent system and analysing its properties is essential to guarantee its expected behaviour. One of the most important properties of a concurrent system is deadlock-freedom, i.e., the system will work normally without an unforeseen and permanent interruption.

In this section we present the two main results of a deadlock analysis technique developed by Brookes and Roscoe[4]. We also show how the FDR model-checker analyses a specification for deadlock-freedom and how the work reported in [4] can guide one in an automatic deadlock analysis of a complex concurrent system using FDR. The theorems presented here are based on some concepts which are informally explained below and defined formally in Appendix A.2. Each of such concepts appears in this section in *slanted* font, to ease the references to the appendix.

The present approach to deadlock analysis considers only CSP processes that do not diverge. This requirement allows a simpler mathematical treatment while it is not too severe in practice, since almost all practical applications are expected to be divergence-free.

Theorem 1 deals with the case where an arbitrary *network* of CSP processes is analysed whereas Theorem 2 is used when one can partition the network into smaller ones. By network we mean a set of parallel processes; it is *busy* if all its processes are deadlock-free. Both theorems use the concept of *vocabulary* which is the set of events containing the synchronisation channels of each pair

of processes of the network. A *request* between processes A and B is just a possibility of A synchronising with B. An *ungranted request* is a request (say, from A to B) that cannot be satisfied, i.e., B cannot offer any event needed by A. A *conflict* occurs when both processes involved in a request have ungranted requests to each other; and a *strong conflict* means that these two processes cannot communicate with a third one. A *cycle* of requests is a sequence of indices (identifying processes) in which each ordered pair of distinct indices form a request.

Theorem 1 *Let* V *be a busy network with vocabulary* A. *If* V *is free of strong* A*-conflict, any deadlock state of the network contains a proper cycle of ungranted requests with respect to* A. *If* V *is conflict-free then any deadlock state contains a proper cycle of ungranted requests* $\langle i_0, \ldots, i_{r-1} \rangle$ *with respect to* A $(r > 2)$, *such that the only requests being made in this state between processes involved in the cycle are the requets recorded in the cycle.*

We can visualise a network using a graph where the processes are the nodes of the graph and the edges are events that synchronise two processes, i.e., events common to two processes. Thus, by *disconnecting edges* we mean those whose removal increase the number of partitions of the graph, and by *essential components* those that stay after removing all disconnecting edges. A pair of processes is *conflict-free* if one cannot find a trace which introduces a reciprocal ungranted request or a strong one between these processes.

Theorem 2 *Suppose* V *is a network with essential components* $V_1, \ldots, V_k$ *where the pair of processes joined by each disconnecting edge are conflict-free with respect to the vocabulary* A. *Then if each of the* V_i *is deadlock-free, so is* V.

The above theorem establishes a connection between deadlock freedom and pairwise conflict freedom of the essential components of a network. The conflict-freedom constraint is necessary because if one essential component blocks then it can infect others if the edge linking two essential components has conflict. This is a very important result because for large networks one can arrange them such that they can be partitioned into simpler ones. Then Theorem 2 tells us that it suffices to check for deadlock-freedom of the essential components.

3.1 FDR

FDR [9] stands for **F**ailures-**D**ivergence **R**efinement and is a model-checker for CSP specifications. Since the specifier uses his knowledge about the theory of communicating processes to overcome the problem of the exponential state explosion, this tool is very efficient to analyse properties such as determinism, deadlock and livelock and to verify some refinement relations among processes.

Differences between CSP and FDR-CSP. FDR adopts a rather different interpretation (defined in [18]) of two elements of the earlier definition of CSP [12]. The first one is the treatment of alphabets that is considered by FDR as a global parameter of the specification. Hence, let $P_1, \ldots, P_n$ be the processes of

the specification then the global alphabet Σ is now denoted as $\alpha P_1 \cup \alpha P_2 \cup \cdots \cup \alpha P_n$. Because of this new view of the alphabet, the parallel operator must have an explicit characterisation of the synchronisation events. In [12], the parallel operator is denoted simply as || because the synchronisation events are precisely determined by the alphabet of the two processes involved, while FDR uses two new (alphabetised) parallel operators: let P and Q be two processes then P[A || C]Q (with $A \subseteq \alpha P$ and $C \subseteq \alpha Q$) is the process that acts as P for events in A, as Q for events in C and as P and Q (synchronisation) for events in $A \cap C$; P[| B |]Q (with $B = A \cap C \subseteq \alpha P \cap \alpha Q$) acts as P for events in $\alpha P - B$, as Q for events in $\alpha Q - B$ and as P and Q for events in B. Regarding notation, FDR-CSP uses a machine-readable version of CSP.

Deadlock analysis using FDR. FDR can analyse a CSP specification using one of the three semantical models defined for CSP, namely the Traces model (T), the Failures model (F) and the Failures-Divergence model (FD). With the first model one can prove safety properties of a system, the second can be used to prove safety, liveness and a combination of these properties and, in addition to the previous properties, the last one can be used to check divergence-freedom. Thus, to check deadlock for a divergent-free specification it is sufficient and more efficient to use the Failures model.

We consider how FDR prove deadlock-freedom and how to use the previous results to ease the analysis for complex networks. Initially let DF (Deadlock-Free) be a process such that

$$DF = \sqcap_{a \in \Sigma} a \rightarrow DF$$

Informally, DF can perform any trace, selecting any event a of the alphabet Σ, but may not refuse all events. In FDR, proving that a process P is deadlock-free is simply verifying if P refines DF, i.e., $DF \sqsubseteq_F P$, where F denotes the Failures model. Hence, FDR checks for deadlock based on the Definition 2 (see Appendix A.2), that is, if FDR finds a trace s of P such that after P performs s its refusal set X equals its alphabet αP, then P deadlocks. Further, FDR checks deadlock-freedom through a refinement relation. The relation $DF \sqsubseteq_F P$ is satisfied iff $\mathcal{F}[\![P]\!] \subseteq \mathcal{F}[\![DF]\!]$, that is, $s_P \subseteq s_{DF}$ and $X_P \subseteq X_{DF}$. The first relation is always satisfied because DF can perform any trace (interleaving events) formed by the events of the alphabet of P, but the second will not hold when if P refuses all its events because DF cannot.

The verification of $DF \sqsubseteq_F P$ is done by FDR through a normalisation of the transition system of DF where a transition system equivalent to the original one is built such that there is a one-to-one relation between states and traces. Although the normalisation transition system of any process is smaller than its original one and FDR can also apply compression techniques, one can always get a process that exhibits the exponential state explosion problem. Therefore, it is convenient to apply the decomposition techniques captured by Theorem 2 whenever possible. The deadlock analysis strategy is compositional in the sense that we verify smaller processes and use the theorems to conclude the deadlock-freedom of their parallel compositions. With FDR we can easily

check if a network is busy, verifying its individual components for deadlock. Also we can prove whether two processes are conflict-free, using Theorem 2, simply checking if its parallel composition is deadlock-free.

4 Deadlock Analysis for CSP-Z: Theory and Tools

According to the requirements of the formal strategy for deadlock analysis presented in the previous section, a network can only be investigated if it is divergence-free, triple-disjoint, uses an associative parallel operator such as || (defined in [12]) and has a static topology.

If one can prove that a network has all these properties than all the results of the preceding section can be used. In this section we show what are the conformity obligations for such results to generalise for CSP-Z specifications. We also suggest an approach to adapt the FDR system to work for CSP-Z.

The conformity obligations that must be verified are:

1. When is the parallel operator [|[] used by CSP-Z equivalent to ||?
2. How to guarantee that the Z part does not introduce divergence?
3. How to manage the dynamic aspects introduced by the Z part?

Theorem 3 *(Associativity of* [|[]*)*

Let V *be a triple-disjoint network. Then* [|[] *is associative. Thus, for all processes* P, Q *and* R *of* V *we have* $P \ [|\ X\ |]\ (Q\ [|\ Y\ |]\ R) = (P\ [|\ X'\ |]\ Q)\ [|\ Y'\ |]\ R$ *such that* $X \cup Y = X' \cup Y'$.

Informally, if one finds an event of X which is not in Y (the set $(\alpha Q \cup \alpha R) \setminus Y$) and this event cannot be performed by Q and R (the set $\alpha Q \cap \alpha R$) then [|[] is associative. Hence, the set X' equals $X \cap \alpha Q$ and Y' equals $Y \cup (X \cap \alpha R)$.

Theorem 4 *(Divergence-freedom of Z)*

Let T_c *be the type of a channel* c *and* T_v *be the type of a variable* v *of the state space such that values carried out by* c *are assigned to* v *. If* $T_c \subseteq T_v$ *then the Z part of a CSP-Z specification does not introduce divergence, i.e.,* $(\mathsf{enable\_c} \wedge \neg\, \mathsf{pre\ com\_c}) \equiv \mathsf{false}$. *Actually, under the above assumptions it is possible to prove a stronger result:* $\mathsf{enable\_c} \equiv \mathsf{pre\ com\_c}$.

See [1] for a more detailed consideration about Theorems 3 and 4.

Because of the above theorem we can encapsulate the enable_c schema into the precondition of the com_c one, changing the refusals of the channel c from ¬enable_c to ¬ pre com_c. This simplifies the mathematical treatment of CSP-Z specifications because one does not need to refer to enable_c, only to pre com_c. This is extensively used in what follows.

Finally, we arrive at the point to consider how to manage the dynamic aspects introduced by the Z part. This characteristic makes the topology of any network built using CSP-Z dynamic; hence, we cannot carry out only a static analysis such as described in Section 3. According to the CSP-Z semantics, if c is a

channel and pre com_c ≡ false then c is refused even if its environment enables it. Therefore, it is not sufficient to consider only the CSP equations, but one must also consider the state space on every occurrence of a CSP event.

The impact of the refusal sets of the Z part on the theoretical analysis is that CSP maximal refusal sets are not CSP-Z maximal refusal sets. In order to use that strategy for CSP-Z, apart from the CSP maximal refusal sets, one must also consider the pre com_c schema for every event c of every trace tr.

In [4], dynamic networks are not considered. The dynamic aspects, introduced by the Z part, can only be managed keeping track of the network's structure during execution; so, it seems very convenient to use FDR for CSP-Z. Therefore, for analysing a CSP-Z specification it is necessary to consider what happens to the network after its data structures initialisation. Let S be a CSP-Z specification such that the CSP part has a cyclic behaviour as $\langle a, b, c, a, b, c, a, b, c, \ldots \rangle$ then if one can prove that $\forall e : \{a, b, c\} \bullet$ pre com_e ≡ true then the CSP-Z behaviour is equal to the CSP one, otherwise this cyclic trace is broken. The analysis is no more static because for the trace $\langle a, b, c \rangle$, the data structures are affected by the following Z composition com_a ⨾ com_b ⨾ com_c, according to the CSP-Z semantics. Therefore, the next occurrence of a might happen in the context of a state which falsifies pre com_a.

4.1 FDR for CSP-Z

Deadlock analysis is not trivial even if one considers only CSP processes. Hence, it is essential to find out a strategy to mechanise deadlock analysis for CSP-Z. In this section we present how to adapt FDR for analysing CSP-Z specifications.

In order to use FDR to analyse CSP-Z we have to define the following elements in FDR: State (the system state space), Init (the initialisation schema), com_c (schema associated to the channel c), pre com_c (precondition of the schema com_c) and the communication of values between the CSP and the Z parts of the specification. The translation strategy is defined as follows. In general, Z operations are relations between initial and final states, as well as input and output values. However, for simplicity we assume in the following that these relations are functional.

- State: FDR has no means to represent a global state space due to its foundations on CSP. However, FDR processes can have parameters which are commonly used for indexing. Therefore, the system state space can be represented as a parameter of all processes of the specification. When a schema com_c updates the state space the final state produced must be taken as the initial state for the next execution step.
- Init: As FDR cannot represent a state space globally then the Init schema is translated into FDR as a process such that it initialises the data structures used by the main equation. Thus, Init = main(InitialState), where InitialState is a tuple which defines an initial value for each state component.
- com_c: A Z schema can be translated into FDR as a function. The arguments to this function are the (current) state and the values of the input variables; the function result is formed by the final state defined by the schema and the values

of the output variables. This function does not embody the precondition part of the schema, only the effect.

• pre com_c: A precondition is also encoded as an FDR function of type State × Input ⇸ $\mathbb{B}$; it evaluates to true in the states and input values which satisfy the precondition of the com_c schema, and to false otherwise.

• **Communications**: Values communicated in the CSP part of the FDR script must be passed to the Z part, and vice-versa. All conversion patterns below have the form of a CSP guarded command. For an input, the condition of the guard is a prefix choice of a suitable value for the input parameter. The expression a?x : {a.x • x : T, pre_com_a(S, x)} is a set comprehension which generates the set of elements a.x where x ranges over T and satifies the predicate pre_com_a(S, x). For an output we simply pass the result of the Z part to the CSP part.

The following conversion patterns implement the above strategy and ease the encoding of a CSP-Z specification into FDR:

CSP-Z	FDR CSP-Z

P=a→P	`P(S)=pre_com_a(S) &`
	`(let S'=com_a(S)`
	`within a -> P(S'))`

P=a?x→P	`P(S)=a?x:{a.x @ x:T,pre_com_a(S,x)} &`
	`(let S'=com_a(S,x)`
	`within P(S'))`

P=a!e→P	`P(S)=pre_com_a(S) &`
	`(let (S',e)=com_a(S)`
	`within a!e -> P(S'))`

The translation of channel declarations, constants and free types is a straightforward syntactical conversion, as presented in [1].

5 Case Study

In this section we present the CSP-Z specification of two processes which combined in parallel with that introduced in Section 2 results in a final specification that represents the simplified behaviour of the SACI-1 OBC. We also show how to translate the specification into our FDR representation and then we carry out a deadlock analysis using FDR.

The SACI-1 OBC is a fault-tolerant distributed processing system which combines software and hardware components [5]. Its main parts are: its Watch-Dog Timer (WDT) and its Fault-Tolerant Router (FTR). Due to its fault-tolerant aspects, the SACI-1 was designed with redundant components. It has three WDT's, three FTR's, etc. However, for illustrative purposes we consider here a simplification of the real configuration, removing indices and presenting its behaviour.

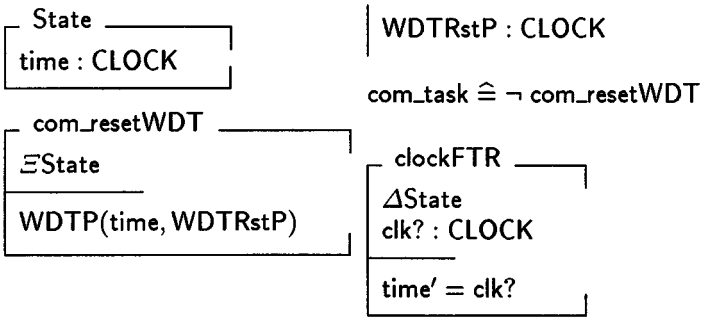
5.1 The SACI-1 Main Components

Fault-Tolerant Router. The FTR is responsible for some tasks and for periodically sending a reset signal to the WDT. In order to model the FTR as close

as possible to its original conception we consider that it can stop temporarily or permanently. In a temporary stop, the FTR can be reanimated through a recover signal. However, in a permanent one the WDT cannot be restarted.

spec *FTR*

channel *clockFTR*:[clk : CLOCK]
channel *reset, recover*:[]
local_channel *resetWDT, task, taskDone, problem*:[]
main=*clockFTR*→*Work*
Work=(Normal ||| *Problem)*
Normal=(resetWDT→ *reset*→main
□ *task*→*((taskDone*→main *)* ^ *(problem*→stop*)))*
Problem=(recover→ main*)*

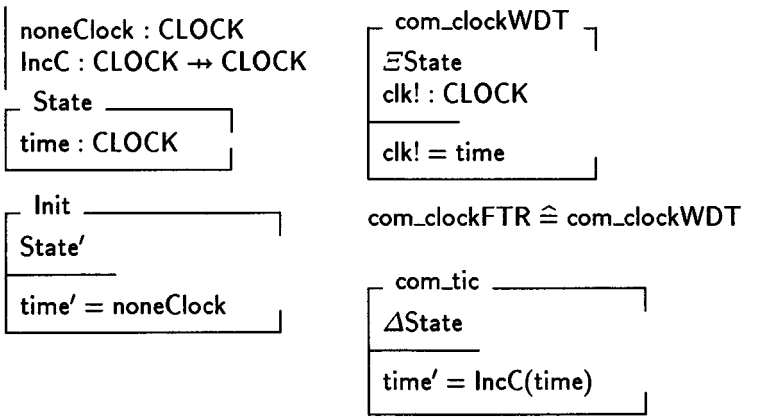


end_spec *FTR*

OBC Clock. As CSP-Z cannot capture precisely temporal aspects of a system, we need some way to characterise the SACI-1 as a system dependent of time. We model a process which exhibits events, carrying clock values, that control the behaviour of the WDT and the FTR.

spec *SCLOCK*

channel *clockWDT, clockFTR* : [clk : CLOCK]
local_channel *tic*: []
main=*(clockWDT*→skip ||| *clockFTR* →skip*)*;*(tic*→main*)*



end_spec *SCLOCK*

SACI-1. The simplified behaviour of the SACI-1 microsatellite is given by an alphabetised parallel composition ([|||]) of the previous three CSP-Z components. In this specification, the elements inside the brackets of the parallel operator are the synchronisation points.

spec *SACI-1*
    main=*(WDT* [| {*reset, recover*} |] *FTR)*
                [| {*clockWDT, clockFTR*} |] *SCLOCK*
end_spec *SACI-1*

5.2 *WDT* in FDR

In this section we present the translation of the *WDT* into FDR. Comments (--) are added to ease the FDR script[1].

```
-- The CLOCK data type (Free type)
datatype CLOCK = noneClock | c1 | c2 | c3 | c4 | c5 | c6
-- spec WDT
channel clockWDT : CLOCK
channel reset, recover, failFTR, timeOut, noTimeOut
-- The main equation and its descendents
main(S)   = Signal(S) ||| Verify(S)
Signal(S) = reset -> Signal(com_reset(S))
Verify(S) = (clockWDT?c ->
          (pre_com_noTimeOut(com_clockWDT(S,c)) &
            noTimeOut -> Verify(com_clockWDT(S,c))
          [] pre_com_timeOut(com_clockWDT(S,c)) &
              timeOut ->
            (pre_com_recover(com_timeOut(com_clockWDT(S,c))) &
              recover -> Verify(com_timeOut(com_clockWDT(S,c)))
            [] pre_com_failFTR(com_timeOut(com_clockWDT(S,c))) &
                failFTR -> SKIP)))
-- The constants
WDTtOut = {c3, c6}
WDTP(time, timeout) = member(time, timeout)
-- Initialisation schema (now a process)
Init = main((0, noneClock)) \ {timeOut, noTimeOut, failFTR}
-- Preconditions
pre_com_noTimeOut((cycles,time)) = not WDTP(time, WDTtOut)
pre_com_timeOut((cycles,time)) = WDTP(time, WDTtOut)
pre_com_recover((cycles,time)) = cycles < 3
pre_com_failFTR((cycles,time)) = cycles == 3
```

[1] This script could be improved using the `let ... within` construct; however, release 2.11 of FDR does not handle this construct correctly. According to Formal Systems (Europe) Ltd, the new version (release 2.20) has solved the problem.

```
-- Schemas
com_reset((cycles,time)) = (0, time)
com_clockWDT((cycles,time),clk) = (cycles, clk)
com_timeOut((cycles,time)) = (cycles + 1, time)
-- end_spec WDT
```

The other two processes are translated into the FDR notation in a similar way. We have done that, loaded into FDR and checked that the *SACI-1* specification is deadlock-free.

6 Conclusion

In this paper we proposed a strategy for model-checking CSP-Z specifications based on previous work for model-checking CSP and on the semantics of the CSP-Z language, verifying its conformity, and adapting the FDR model-checker to work with the state part of CSP-Z specifications. We presented a formal specification in CSP-Z of a subset of the SACI-1 microsatellite OBC as well as a deadlock analysis of this specification using the FDR tool.

The SACI-1 project as developed by the Brazilian Space Research Institute lacked formal documentation, hence our first contribution was to formally define a subset of the SACI-1 [1]: its OBC system. From the very beginning, the goal of the formalisation task was to develop a formal specification free from problems and hence we did not find any deadlocks in our specification, as required. However, some problems in the informal documentation were detected: the informal documentation was found to be ambiguous (difficulting the understanding of the system), and the description of many processes which were supposed to cooperate did not specify synchronisation points. These problems were reported to the members of the SACI-1 project and the specification reported in [1] serves today as a formal reference for the implementation of this project.

One research direction we intend to pursue is the derivation of an implementation in a language like OCCAM [15] from CSP-Z specifications. To this end, we count with an important theoretical result [7, 8]: refinement of the CSP and of the Z part (subject to some constraints) of a CSP-Z specification leads to refinement of the entire CSP-Z specification.

Another topic for further research is the integration of tools to deal with CSP-Z specifications. In [1], we have shown how to use Z-EVES [19] to type-check the Z part of the SACI-1 specification and to refine some of its data structures. Furthermore, the ZANS animator [13] was also used in [1] to analyse the behaviour of the data structures in the Z part of the SACI-1 specification. Ideally, these tools should also be adapted to work for CSP-Z, as we did with FDR. The ultimate goal would be linking all these tools into a uniform development environment for CSP-Z.

A final remark is that although we have based our work on CSP-Z, the results could, in principle, be easily adapted to other approaches to integrate CSP and Z, such as, for example [10].

7 Acknowledgements

We thank people from the Brazilian Space Research Institute (INPE), and in particular Alderico R. Paula Jr. for help in the understanding of the SACI-1. We also thank Clemens Fischer and Paulo Borba for discussions about CSP-Z and FDR, and for suggestions and criticisms which helped us to improve our approach to model-checking CSP-Z.

A Formal Definitions

The whole paper was intended to describe the development of our strategy to model-checking CSP-Z specifications without too much technical details. In this section we present the Failures-Divergence model for Z and the some definitions which were only informally presented in Section 3.

A.1 Failures-Divergence Semantics of Z

Let c be a channel, tr be a trace, Chans(I) be the set of channels of an interface I and *Comm* be the set of communications (pairs (c, v), where c is a channel and v is a communication value), thus:

The Failures-Divergence interpretation for the Z part is given by the following definition:

Definition 1. Let I be an interface and O_Z a list of Z-schemas with exactly one schema enable_c and effect_c for every channel c from Chans(I). Then the semantics of the corresponding CSP-Z specification is defined as follows:

$[\![\textsf{spec } I; \textsf{State; Init; } O_Z \textsf{ end\_spec}]\!] = (\mathcal{F}, \mathcal{D})$

where

$\mathcal{D}$=$\{\textsf{s}^\frown\textsf{t} : \textsf{seq } Comm \mid \exists\, \textsf{State}' \bullet \textsf{effect\_s} \wedge (\textsf{enable\_head(t)})' \wedge \neg(\textsf{pre effect\_head(t)})'\}$

$\mathcal{F}$=$\{(tr, \mathcal{R}) : \textsf{seq } Comm \times \mathbb{P}\ Comm \mid \exists\, \textsf{State}' \bullet (\textsf{effect\_}tr \wedge \mathcal{R} \subseteq REF)\}$
$\quad \cup \{(tr, \mathcal{R}) : \mathcal{D} \times \mathbb{P}\ Comm\} \cup \{(\langle\rangle, \varnothing)\}$

REF==$\{(c, v) \mid \neg(\textsf{enable\_}(c, v))'\}$

In the above definition, $\mathcal{D}$ is the refusal set introduced by the Z part where the predicate enable_c$\wedge\neg$ pre effect_c is verified and $\mathcal{F}$ is the failures set of the Z part, where REF is the set of communications that can be refused, i.e., the predicate $\neg$enable_c is satisfied.

A.2 Definitions for Deadlock Analysis

Definition 2. The process P is *deadlock-free* if $\forall s \in (\alpha P)^* \bullet (s, \alpha P) \notin \mathcal{F}[\![P]\!]$.

CSP operators are compositional in the sense that given two processes P and Q, P $\Box$ Q or P || Q are also processes. Thus, we can see a concurrent system as a composition of parallel processes.

Definition 3. A *network* is a parallel combination of processes. Let V be a network composed of the processes $P_1, \ldots, P_n$ then $V = \langle P_1, \ldots, P_n \rangle$.

In the following definitions, let V be a network such that $V = \langle P_1, \ldots, P_n \rangle$.

Definition 4. A network is *triple-disjoint* iff $\alpha P_i \cap \alpha P_j \cap \alpha P_k = \emptyset$, $i \neq j \neq k$.

Definition 5. A graphical view of a network is a *communication graph* where the processes are the nodes and an arc exists between two nodes iff $\alpha P_i \cap \alpha P_j \neq \emptyset$, $i \neq j$.

Definition 6. The *vocabulary* Λ of a network V is the set $\bigcup\{\alpha P_i \cap \alpha P_j \mid 1 \leq i < j \leq n\}$.

Definition 7. A *state* σ of a network V is a trace s of V and an indexed tuple $\langle X_1, \ldots, X_n \rangle$ of refusal sets X_i such that for each i, $(s \restriction \alpha P_i, X_i) \in \mathcal{F}[\![P_i]\!]$.

Definition 8. Let $\sigma = (s, X)$ be a state and Λ be the vocabulary of the network V. A pair of indices $\langle i, j \rangle$ (with $i \neq j$) is:

* a *request* if $(\alpha P_i - X_i) \cap \alpha P_j \neq \emptyset$ ($P_i \xrightarrow{\sigma} P_j$ or $P_i \xrightarrow{\sigma,\Lambda} P_j$);
* a *strong request* if $\emptyset \neq (\alpha P_i - X_i) \subseteq \alpha P_j$ ($P_i \stackrel{\sigma}{\Rightarrow} P_j$ or $P_i \stackrel{\sigma,\Lambda}{\Rightarrow} P_j$);
* *ungranted* if in addition $\alpha P_i \cap \alpha P_j \subseteq X_i \cup X_j$ ($P_i \xrightarrow{\sigma}_{\bullet} P_j$ or $P_i \xrightarrow{\sigma,\Lambda}_{\bullet} P_j$).

Definition 9. A state σ of the pair $\langle P, Q \rangle$ is a *Γ-conflict* if $P \xrightarrow{\sigma,\Gamma}_{\bullet} Q$ and $Q \xrightarrow{\sigma,\Gamma}_{\bullet} P$ and *strong Γ-conflict* if $P \stackrel{\sigma,\Gamma}{\Rightarrow}_{\bullet} Q$ or $Q \stackrel{\sigma,\Gamma}{\Rightarrow}_{\bullet} P$ (with respect to Γ).

Definition 10. A pair $\langle P, Q \rangle$ is *free of Γ-conflict* if none of its states is a Γ-conflict.

Definition 11. A network V is *(strong) conflict-free* iff for all $i \neq j$ the pair $\langle P_i, P_j \rangle$ is free of strong Λ-conflict.

Definition 12. The edges (nodes) $V_1, \ldots, V_k$ are the *disconnecting edges* of the network V iff they are nodes of the communication graph of V whose removal would increase the number of connected components (partitions).

Definition 13. The *essential components* of V are the connected components of the graph that remains after all disconnecting edges were removed.

References

1. A. C. Mota. Formalisation and Analysis of the SACI-1 Microsatellite in CSP-Z. Master's thesis, Federal University of Pernambuco, 1997.
2. T. Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, January 1987.
3. S. D. Brookes and A. W. Roscoe. An improved failures model for communication processes. In *Lecture Notes on Computer Science*, volume 197, pages 281–305, 1985.
4. S. D. Brookes and A. W. Roscoe. Deadlock analysis in networks of communicating processes. *Distributed Computing*, pages 209–230, 1991.
5. A. R. de Paula Jr. Fault-Tolerance Aspects of the On-Board Computer of the First INPE Microsatellite for Scientific Applications. *VI Brazilian Symposium on Fault-Tolerant Computers*, August 1995.

6. E. Boiten, H. Bowman, J. Derrick and M. Steen. Viewpoint Consistency in Z and LOTOS: A Case Study. In *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods*, pages 644–664. Springer Verlag, 1997.
7. C. Fischer. Combining CSP and Z. Technical report, University of Oldenburg, 1996.
8. C. Fischer. CSP-OZ: A Combination of Object-Z and CSP. In *2nd IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMODDS'97)*. Chapman Hall, 1997.
9. Formal Systems (Europe) Ltd. *FDR: User Manual and Tutorial, version 2.01*, August 1996.
10. G. Smith. A Semantic Integration of Object-Z and CSP for the Specification of Concurrent Systems. In *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods*, number 1313 in Lecture Notes in Computer Science, pages 62–81. Springer Verlag, 1997.
11. J. A. Goguen and T. Winkler. Introducing OBJ3. Technical report, SRI International, SRI-CSL-88-9, August 1988. Revised version to appear with additional authors José Meseguer, Kokichi Futatsugi and Jean-Pierre Jouannaud, in *Applications of Algebraic Specification using OBJ*,.
12. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
13. X. Jia. *A Tutorial of ZANS - A Z Animation System*, 1995.
14. C. B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall International, 1986.
15. G. Jones and M. Goldsmith. *Programming in OCCAM 2.* Prentice-Hall International, 1988.
16. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1991.
17. R. Milner. A Calculus of Communicating Systems. In *Lecture Notes in Computer Science 92.* Springer-Verlag, 1980.
18. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall International, 1997.
19. M. Saaltink. The Z/EVES System. In *ZUM'97: The Z Formal Specification Notation*, pages 72–85. Lecture Notes in Computer Science, 1212, Springer, 1997.
20. M. Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall International, 2nd edition, 1992.