On the expressive power of Z

Peter Baumann and Karl Lermer
Institut für Informatik der Universität Zürich
Winterthurerstr. 190, 8057 Zürich, Switzerland
Email: {baumann,lermer}@ifi.unizh.ch, Tel: ++41 1 257 45 79, Fax: ++41 1 363 00 35

Abstract

The formal specification language Z is successfully used to specify transformational programs. Complex systems can only be adequately modeled by infinite sequences of states. In this paper we develop a systematic approach to use standard Z to specify complex systems by adapting central ideas of TLA to Z.

1 Introduction

A well-known coarse classification [AL88, MP92] of software systems yields two classes of programs. One class consists of programs that can be specified by input/output relations. Manna and Pnueli [MP92] call these programs transformational programs. The other class contains programs whose behavior can only be adequately modeled by infinite sequences of states. These programs are called complex systems in [AL88].

Over the past decade, the formal specification language Z has been proven to be expressive enough to specify non-toy examples of transformational programs in numerous (industrial) application areas. A large number of case studies has been published in relevant journals and conferences. It is therefore natural to ask whether the expressive power of Z is limited to transformational programs or whether it can be used to specify complex software systems. In this paper we investigate how Z can systematically be used to specify complex systems while adhering to existing Z notation and conventions.

This is not just an academic exercise. Experience shows that complex software systems are difficult to specify and develop. We are convinced that formal methods such as Z should form the backbones of software development techniques if we want to make sure that we develop dependable systems. The most powerful tools we have to model complex systems and to solve difficult problems related to them are mathematical formalisms. This is supported by the experience of centuries of successful application of mathematical calculations, uncontested in physics and in all engineering disciplines other than computer science. It is only in computer science that there are doubts that formal methods help to surmount the obstacles of producing reliable systems. This is even more surprising if we think of the chronic software crisis and of the increasing complexity of the systems that are controlled by software.

If today's techniques do not allow the complete formal development of software in any case this does not imply that it will not be possible tomorrow. Thorough studies on the state of the art of formal methods and their use are presented in [BS93, GCR94, Ost94]. It is the topic of this paper to systematically extend the applicability of a successful formal method. Z is a formal specification language whose successful use in a wide variety of situations supports the claim that formal languages are powerful tools in modeling software systems. Since Z has been used to solve real problems people have a lot of experience reading and writing Z specifications. By extending the application area of Z, we can avoid the problem of having to learn different formalisms and new tools. Furthermore, it is to be expected that experience in modeling transformational systems is useful to some extend in modeling complex systems. We regard it an unfortunate tendency in computer science that often a new language is defined when it is not immediately clear how to describe a situation with existing languages. Further practical experience of the industrial use of formal methods is currently hindered by the fact that there too many different formalisms. It is not in general obvious at all how to compare experience with different methods.

Let us start the investigation of whether the application area of Z can be extended by some general remarks on specifying complex systems. Z is a model-oriented specification language. In other words, a Z specification can be understood as essentially giving an abstract-state-machine model of the system to be built. The state space and the operations are described in schemas. Interestingly, in a very illuminating paper [AL88] Abadi and Lamport show that a property specified by a state machine is a safety property. As there are other properties (e.g. liveness properties) [MP92] that we want to specify for complex systems, this seems to imply that Z cannot be used as a specification language for complex systems.

Nevertheless, we claim that Z can be used in a systematic way to specify complex systems. The emphasis here is on the word systematic. We sustain this conjecture by showing how, in Z, one can adopt the approach of TLA [Lam94a] of specifying complex systems. Basically, it is the power of the schema language that gives us the possibility to adapt concepts of TLA to Z. The central idea is to extend standard Z specifications by first describing the underlying abstract state machine of the specification explicitly. We can then specify fairness properties for such machines by introducing operators on sequences of operations, where these operators are given by axiomatic definitions in Z itself. We will also see that this approach allows us to keep the usual proof obligations for operation and data refinement. We also show that our specification technique yields machine closed specifications.

The question of whether Z and TLA can equally well be used for descriptions of complex system is also interesting because Leslie Lamport challenged the Z community at the 1994 Z User Meeting [Lam94b] by proposing an extension of Z, called TLZ, to specify complex systems. In this paper we give a specification of the birthday book example including fairness requirements.

1.1 Organization of the paper

In the next section we show how TLA-like specifications can be written in Z. In the third section we introduce Z machines which make explicit the underlying abstract machine models of Z specifications. We also define a refinement relation between Z machines and show that the standard operation refinement and data refinement techniques are sufficient to guarantee refinement of the corresponding machines. In the fourth section we present a technique of systematically specifying fairness properties in Z itself. Since this paper was triggered by the invited talk of Leslie Lamport at the 1994 Z User Meeting [Lam94b], where he presented a specification of the birthday book example with fairness conditions in TLZ we show in the fifth section how we would specify the same example in our approach. We end the paper with conclusions and an outlook to future work.

We assume that the reader is familiar with the basic concepts of Z [Spi92] and TLA [Lam94a].

2 Z and TLA

To warm up, we demonstrate how Z could in principle be used like TLA. For this purpose we use the same instructive example that Leslie Lamport used to introduce TLA [Lam94a].

The following TLA formula

$$\Phi \mathrel{\widehat=} \mathit{Init}_\Phi \wedge \Box[\mathcal{M}]_{<x,y>} \wedge \mathit{WF}[\mathcal{M}_1]_{<x,y>} \wedge \mathit{WF}[\mathcal{M}_2]_{<x,y>} \tag{1}$$

where

$$\begin{aligned} \mathit{Init}_\Phi &\mathrel{\widehat=} (x = 0) \wedge (y = 0), \\ \mathcal{M}_1 &\mathrel{\widehat=} (x' = x + 1) \wedge (y' = y), \\ \mathcal{M}_2 &\mathrel{\widehat=} (x' = x) \wedge (y' = y + 1), \\ \mathcal{M} &\mathrel{\widehat=} \mathcal{M}_1 \vee \mathcal{M}_2 \end{aligned}$$

describes fair behaviors of the simple guarded command program:

var natural $x, y = 0$;
do $\langle$ **true** $\rightarrow x := x + 1 \rangle$
  $[\!]$ $\langle$ **true** $\rightarrow y := y + 1 \rangle$
od

Formula (1) is equivalent to the formula (see [Lam94a] for details)

$$\Phi \mathrel{\widehat=} \mathit{Init}_\Phi \wedge \Box[\mathcal{M}]_{<x,y>} \wedge \Box\Diamond\langle\mathcal{M}_1\rangle_{<x,y>} \wedge \Box\Diamond\langle\mathcal{M}_2\rangle_{<x,y>} \tag{2}$$

where the following abbreviations are used:

$$\begin{aligned} [\mathcal{M}]_{<x,y>} &\mathrel{\widehat=} \mathcal{M} \vee (< x', y' > = < x, y >), \\ \langle\mathcal{M}\rangle_{<x,y>} &\mathrel{\widehat=} \mathcal{M} \wedge (< x', y' > \neq < x, y >). \end{aligned}$$

How could we specify this in Z? As usual we start with describing the state space:

$$State \mathrel{\widehat{=}} [x, y : \mathbb{N}]$$

The corresponding Z schemas for $Init_\Phi$, $\mathcal{M}_1$ and $\mathcal{M}_2$ are standard :

$$Init \mathrel{\widehat{=}} [\Delta State \mid x' = 0 \land y' = 0]$$

$$M1 \mathrel{\widehat{=}} [\Delta State \mid x' = x + 1 \land y' = y]$$

$$M2 \mathrel{\widehat{=}} [\Delta State \mid x' = x \land y' = y + 1]$$

Similarly, we can define[1]

$$M \mathrel{\widehat{=}} M1 \lor M2$$

$$[M]\_xy \mathrel{\widehat{=}} M \lor \Xi State$$

$$\langle M1 \rangle\_xy \mathrel{\widehat{=}} M1 \land \neg\, \Xi State$$

$$\langle M2 \rangle\_xy \mathrel{\widehat{=}} M2 \land \neg\, \Xi State$$

It is worth noting that schema negation in the above formulae is correct, since it is used in the context of a conjunction.

In the following, we will often need concatenation of state bindings. This is not a standard operation in Z, but can easily be defined by

$$\begin{array}{|l}
csb : State \times State \rightarrow \Delta State \\
\hline
csb = \{\Delta State \bullet ((\theta State, \theta State'), \theta(\Delta State))\}
\end{array}$$

So far we can express what is called action part of the TLA formula Φ in Z. What is still missing are the temporal operators $\diamond$ (*Sometimes*) and $\Box$ (*Always*). The semantics of these operators is simple and we can define these operators in generic definitions. To this end we first transform operations given by schemas into operations on sequences of states. This is similar to the use of actions as temporal formulas in TLA.

$$\begin{array}{|l}
\textit{Behavior} \\
\hline
\sigma : \mathbb{N} \rightarrow State \\
\hline
\end{array}$$

$$\begin{array}{|l}
_^T : \mathbb{P}\, \Delta State \rightarrow \mathbb{P}(\mathbb{N} \rightarrow State) \\
\hline
\forall B : \mathbb{P}\, \Delta State \bullet B^T = \{\sigma : \mathbb{N} \rightarrow State \mid csb(\sigma(0), \sigma(1)) \in B\}
\end{array}$$

[1]Here we give up the syntactic rules for identifiers in Z for the simple reason of illustrating the correspondence between the TLA and Z specification

As sets of bindings cannot be used in schema expressions, we will sometimes use the following generic schema definition to turn a set of state sequences into corresponding schemas.

$$
\begin{array}{|l}
\hline
S[X] \\
\hline
\sigma : X \\
\hline
\end{array}
$$

In order to become familiar with the notation, it is instructive to check the following rules.

Proposition 2.1 *Let A and B be operation schemas with $A, B \subseteq \Delta State$. Then*

$$
\begin{array}{l}
S[(\neg\ A \wedge \Delta State)^T] = \neg\ S[A^T] \wedge Behavior \\
S[(A \wedge B)^T] = S[A^T] \wedge S[B^T] \\
S[(A \vee B)^T] = S[A^T] \vee S[B^T] \\
S[((A \Rightarrow B) \wedge \Delta State)^T] = (S[A^T] \Rightarrow S[B^T]) \wedge Behavior \\
S[((A \Leftrightarrow B) \wedge \Delta State)^T] = (S[A^T] \Leftrightarrow S[B^T]) \wedge Behavior
\end{array}
$$

Before defining the temporal operators $\diamond$ and $\square$, we need a shift operator:

$$
\begin{array}{|l}
[X] \\
\hline
Sh : \mathbb{N} \times (\mathbb{N} \to X) \to (\mathbb{N} \to X) \\
\hline
\forall\, n, k : \mathbb{N};\ \sigma : \mathbb{N} \to X \bullet Sh(n, \sigma)\ k\ =\ \sigma(k + n) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
[X] \\
\hline
\diamond : \mathbb{P}(\mathbb{N} \to X) \to \mathbb{P}(\mathbb{N} \to X) \\
\hline
\forall\, B : \mathbb{P}(\mathbb{N} \to X) \bullet \diamond B = \{\sigma : \mathbb{N} \to X \mid \exists\, n : \mathbb{N} \bullet Sh(n, \sigma) \in B\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
[X] \\
\hline
\square : \mathbb{P}(\mathbb{N} \to X) \to \mathbb{P}(\mathbb{N} \to X) \\
\hline
\forall\, B : \mathbb{P}(\mathbb{N} \to X) \bullet \square B = \{\sigma : \mathbb{N} \to X \mid \forall\, n : \mathbb{N} \bullet Sh(n, \sigma) \in B\} \\
\hline
\end{array}
$$

It is worth noting that other temporal operators such as the *Next* or the *Until* operators could be defined similarly.

What is the result of using these operators in conjunction with operation schemas? Applying these new operators on operation schemas, we obtain the following equalities:

Proposition 2.2 *Let A be any operation schema with $A \subseteq \Delta State$. Then*

$$\begin{array}{l}
\Diamond A^T = \{\sigma : \mathbb{N} \to State \mid \exists\, n : \mathbb{N} \bullet csb(\sigma(n), \sigma(n+1)) \in A\} \\
\Box A^T = \{\sigma : \mathbb{N} \to State \mid \forall\, n : \mathbb{N} \bullet csb(\sigma(n), \sigma(n+1)) \in A\} \\
\Box(\Diamond A^T) = \{\sigma : \mathbb{N} \to State \mid \forall\, n : \mathbb{N} \bullet \exists\, k : \mathbb{N} \mid k \geq n \bullet csb(\sigma(k), \sigma(k+1)) \in A\} \\
\Diamond(\Box A^T) = \{\sigma : \mathbb{N} \to State \mid \exists\, n : \mathbb{N} \bullet \forall\, k : \mathbb{N} \mid k \geq n \bullet csb(\sigma(k), \sigma(k+1)) \in A\} \\
\Diamond(\Diamond A^T) = \Diamond A^T \\
\Box(\Box A^T) = \Box A^T \\
\sigma \notin \Diamond A^T \quad \textit{iff} \quad \sigma \in \Box(\neg\, A \wedge \Delta State)^T \\
\sigma \notin \Box A^T \quad \textit{iff} \quad \sigma \in \Diamond(\neg\, A \wedge \Delta State)^T
\end{array}$$

We are now ready to present the desired Z specification of fair executions of the above guarded command program in the form of a schema expression:

$$\Phi_Z \;\widehat{=}\; S[Init^T] \wedge S[\Box[M]\_xy^T] \wedge S[\Box(\Diamond\langle M1\rangle\_xy^T)] \wedge S[\Box(\Diamond\langle M2\rangle\_xy^T)] \tag{3}$$

The equivalence between the two formulae (1) and (2) is due to the fact that the preconditions of the operations $\mathcal{M}_1$ and $\mathcal{M}_2$ are always true. In general, a weak fairness property for an action $\mathcal{A}$ is given by the TLA formula:

$$WF(\mathcal{A}) = \Box\Diamond\mathcal{A} \vee \Box\Diamond\neg\, Enabled(\mathcal{A})$$

The predicate $Enabled(\mathcal{A})$ in TLA corresponds exactly to the precondition $pre\ A$ in Z. Hence, we are now in the position to express the weak fairness condition directly as schema expression in Z:

$$WF_A \;\widehat{=}\; S[\Box(\Diamond A^T)] \vee S[\Box(\Diamond(\neg\, pre\ A \wedge \Delta State)^T)] \tag{4}$$

Similar reflections show that we can also express strong fairness conditions in Z.

It may be argued that the above schema expressions (3) and (4) are not very elegant. Instead of using schema expressions we can also use schemas:

$$\begin{array}{|l}
\hline
\Phi_Z \\
Behavior \\
\hline
\sigma \in Init^T \\
\sigma \in \Box[M]_xy^T \\
\sigma \in \Box(\Diamond\langle M1\rangle_xy^T) \wedge \sigma \in \Box(\Diamond\langle M2\rangle_xy^T) \\
\hline
\end{array}$$

and

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{WF_A} \\
Behavior \\
\hline
\sigma \in \Box(\Diamond A^T) \lor \\
\sigma \in \Box(\Diamond(\neg\ pre\ \ A \land \Delta State)^T) \\
\hline
\end{array}$$

Clearly, the approach presented here can be generalized. TLA formulas can always be written as conjunctions of the form [Lam94a]

$$Init \land \Box[\mathcal{N}]_f \land \ \ F \tag{5}$$

where

$Init$ is a predicate specifying initial values,
$\mathcal{N}$ is the action representing executions of individual operations
f is the n-tuple of flexible variables
F is a conjunction of weak fairness or strong fairness formulas.

$Init$, $\mathcal{N}$ and f have their direct counterparts in Z. Weak fairness or strong fairness formulas can now be expressed in Z with the help of the generic $\Diamond$ and $\Box$ operators defined above.

3 Z machines and refinement

The development in the previous section is not really adequate for Z. The reason is that, in the case of operation composition of nondeterministic operations, Z requires a demonic interpretation. A very clear motivation of demonic interpretations is given in [Woo92].

3.1 Z machines

In this subsection, we show how to obtain demonic interpretations of sequences of operations, thereby making explicit the abstract machine underlying a Z specification.

Given a Z specification with state schema $State$ and operation schemas $Init, Op_1, \ldots, Op_n$. We define OP as the set of the possible operations of the machine, which includes $\Xi State$ in order to model stuttering steps.

$$OP == \{Init, Op_1, \ldots, Op_n\} \cup \{\Xi State\}$$

The next schema describes the corresponding simple machine as sequences of states and sequences of operations with the requirements that the first operation is always $Init$ and that the sequences of states could have been produced by the corresponding operation sequence[2].

[2]For the rest of this paper we assume that csb has been defined for the corresponding state spaces.

$$
\begin{array}{|l}
\hline
\textit{SM} \\
\sigma : \mathbb{N} \to \mathit{State} \\
\delta : \mathbb{N} \to \mathit{OP} \\
\hline
\delta\ 0 = \mathit{Init} \\
\forall\, n : \mathbb{N} \bullet \mathit{csb}(\sigma(n), \sigma(n+1)) \in \delta\ n \\
\hline
\end{array}
$$

As usual [AL88, MP92], terminating computations are modeled by sequences where there exists an index k such that $\delta(i) = \Xi \mathit{State}$ for all $i \geq k$.

The demonic interpretation requires to control all possible state sequences for a given operation sequence. For this purpose we borrow a standard trick from the theory of nondeterministic finite state machines:

$$
\begin{array}{|l}
\Delta : (\mathbb{N} \to \mathbb{P}\, \Delta \mathit{State}) \to (\mathbb{N} \to \mathbb{P}\, \mathit{State}) \\
\hline
\forall\, \delta : (\mathbb{N} \to \mathbb{P}\, \Delta \mathit{State}) \bullet \\
\qquad (\Delta\ \delta)\ 0 = \mathit{State} \wedge \forall\, n : \mathbb{N} \bullet (\Delta\ \delta)\ (n+1) = \\
\qquad\qquad\qquad \{t : \mathit{State} \mid (\exists\, s : \mathit{State} \mid s \in (\Delta\ \delta)\ n \bullet \mathit{csb}(s,t) \in \delta\ n)\}
\end{array}
$$

Please observe that if the *nth*-step is a stuttering step, then $(\Delta\ \delta)\ n = (\Delta\ \delta)\ (n+1)$.

In a demonic interpretation, we want to be able to continue computation irrespective of the concrete state which was nondeterministically chosen. This is formalized in the following definition of cones

$$
\mathit{Cones} == \{\delta : \mathbb{N} \to \mathit{OP} \mid \delta(0) = \mathit{Init} \wedge \forall\, n : \mathbb{N}_1 \bullet (\Delta\ \delta)\ n \subseteq \{\mathit{pre}\ (\delta\ n)\}\}
$$

Fig. 1 illustrates the notion of a cone.

Next we define the central concept of a *Z machine* of a Z specification by a schema ZM, which captures the intuitive meaning of executing the operations of the Z specification.

$$
\begin{array}{|l}
\hline
\textit{ZM} \\
\mathit{SM} \\
\hline
\delta \in \mathit{Cones} \\
\hline
\end{array}
$$

Z machines are closed under stuttering, i.e., if $\mathit{zm} \in \mathit{ZM}$ then one can cancel or introduce (except at index 0) stuttering steps and still obtains a binding $\mathit{zm}' \in \mathit{ZM}$.

We say that an infinite sequence $\langle \mathit{zm}_0, \mathit{zm}_1, \ldots, \mathit{zm}_i, \ldots \rangle$ of bindings of a schema type of a Z machine converges to a binding zm of the same schema type iff for all $m \geq 0$ there exists $n \geq 0$ such that $\{1..m\} \lhd \mathit{zm}_i.\sigma = \{1..m\} \lhd \mathit{zm}.\sigma$ and $\{1..m\} \lhd \mathit{zm}_i.\delta = \{1..m\} \lhd \mathit{zm}.\delta$ for all $i \geq n$. In this case we write $\mathit{lim}\ \mathit{zm}_i = \mathit{zm}$.

It is easy to show that a Z machine ZM is closed under the formation of limit points.

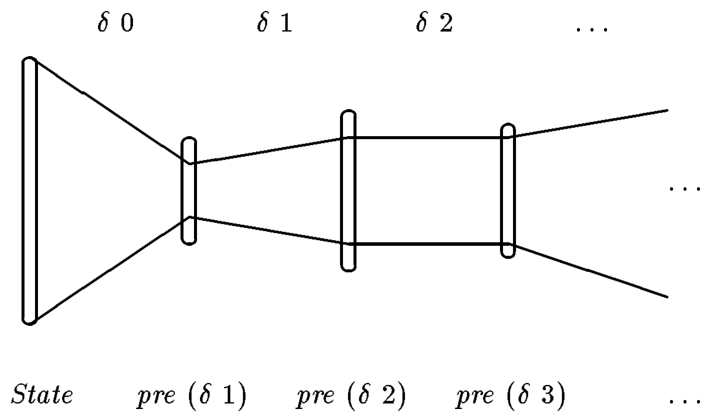

Figure 1: Illustration of a cone

Proposition 3.1 *Let $(zm_i)_i$ be a sequence of bindings $zm_i \in ZM$ and let zm be a binding of the same type. If $lim\ zm_i = zm$ then $zm \in ZM$.*

In other words, the set of bindings given by a Z machine ZM is a closed set. This implies that the property specified by such a machine is a safety property [AL88] with respect to cones.

For each $zm \in ZM$, we can form the set of terminating computations zm^i by defining

$$zm^i.\sigma(k) = \begin{cases} zm.\sigma(k) & \text{if } k \leq i \\ zm.\sigma(i) & \text{if } k > i \end{cases}$$

and

$$zm^i.\delta(k) = \begin{cases} zm.\delta(k) & \text{if } k < i \\ \Xi State & \text{if } k \geq i \end{cases}$$

Clearly, $zm^i \in ZM$ for all i and $lim\ zm^i = zm$.

3.2 Operation refinement

Formal operation and data refinement have become very important notions in the research in formal methods. In this subsection we define a refinement relation between Z machines with the same state spaces and show that the proof obligations of the standard operation refinement is sufficient to guarantee correct refinement of the corresponding machines.

Let us assume we are given two Z specifications $Spec1$ and $Spec2$ with corresponding demonic Z machines:

$$
\begin{array}{|l}
\multicolumn{1}{l}{ZM^A} \\
\hline
\sigma^A : \mathbb{N} \to State \\
\delta^A : \mathbb{N} \to OP^A \\
\hline
\delta^A \in Cones^A \\
\forall\, n : \mathbb{N} \bullet csb(\sigma^A(n), \sigma^A(n+1)) \in \delta^A\ n \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{ZM^C} \\
\hline
\sigma^C : \mathbb{N} \to State \\
\delta^C : \mathbb{N} \to OP^C \\
\hline
\delta^C \in Cones^C \\
\forall\, n : \mathbb{N} \bullet csb(\sigma^C(n), \sigma^C(n+1)) \in \delta^C\ n \\
\hline
\end{array}
$$

Furthermore, we assume that we are given a mapping (or) relating a concrete operation to each abstract operation.

$$
\begin{array}{|l}
or : OP^A \to OP^C
\end{array}
$$

What does it mean to use machine ZM^C instead of machine ZM^A? It simply means that instead of operation sequences with operations $op^A \in OP^A$, we execute the corresponding sequences with operations $op^C = or(op^A) \in OP^C$. Formally, this can be captured in a kind of promotion schema:

$$
\begin{array}{|l}
\multicolumn{1}{l}{UZM_C^A} \\
\hline
ZM^A \\
ZM^C \\
\hline
\delta^C = or \circ \delta^A \\
\hline
\end{array}
$$

The refinement relation between Z machines is defined as follows.

$$ZM^A \sqsubseteq_{or} ZM^C$$

iff

1. $UZM_C^A \setminus (\sigma^A, \delta^C) \subseteq ZM^A[\sigma^C/\sigma^A]$
2. $ZM^A = UZM_C^A \setminus (\sigma^C, \delta^C)$

The first requirement is a correctness criterion, saying that concrete state sequences in UZM_C^A could also have been realized by the abstract machine ZM^A. The second obligation guarantees that the concrete machine allows enough computations. In terms of cones it means that all concrete cones in UZM_C^A are contained in abstract cones and that for each abstract cone there is a concrete cone.

As usual, it is important to check that this relation is reflexive and transitive. This can be done in a straightforward manner for the above defined refinement relation.

Much more interesting is the fact that operation refinement as it is usually defined in Z is sufficient to guarantee that the refinement relation holds between the corresponding Z machines. In the following we call the standard Z refinement conditions *local refinement conditions.* For completeness, we repeat the local refinement conditions for operation refinement here

$$op^A \sqsubseteq op^C$$

iff

1. $pre\ op^A \vdash pre\ op^C$
2. $pre\ op^A \wedge op^C \vdash op^A$

Proposition 3.2 *If* $op^A \sqsubseteq op^C$ *for all* $(op^A, op^C) \in or$ *then* $ZM^A \sqsubseteq_{or} ZM^C$.

Proof
We have to discharge the two obligations

1. $UZM_C^A \setminus (\sigma^A, \delta^C) \subseteq ZM^A[\sigma^C / \sigma^A]$
2. $ZM^A = UZM_C^A \setminus (\sigma^C, \delta^C)$

1. To discharge the first obligation, one shows by induction that for all $n : \mathbb{N} \bullet \sigma^C(n) \in (\Delta^A\ \delta^A)\ n \wedge\ csb(\sigma^C(n), \sigma^C(n+1)) \in \delta^A n$ using the local refinement conditions.
2. Given a binding for σ^A and δ^A in ZM^A. Define $\delta^C : \mathbb{N} \to OP^C$ by $\delta^C = or \circ \delta^A$. Put $\sigma^C(0) = \sigma^A(0)$. Hence, $\sigma^C(0) \in (\Delta^C\ \delta^C)\ 0$ and $(\Delta^C\ \delta^C)\ 0 \subseteq (\Delta^A\ \delta^A)\ 0$.

   Let us assume we have $\sigma^C(n)$ with $\sigma^C(n) \in (\Delta^C\ \delta^C)\ n$ and $(\Delta^C\ \delta^C)\ n \subseteq (\Delta^A\ \delta^A)\ n$. For any $s \in (\Delta^C\ \delta^C)\ n$ we conclude $s \in pre\ (\delta^C n)$ by the first local refinement condition on $\sqsubseteq$. For any t such that $csb(s,t) \in (\delta^C n)$ follows from the second refinement condition on $\sqsubseteq$ that $csb(s,t) \in (\delta^A n)$. Hence, for all such t we know $t \in (\Delta^C\ \delta^C)\ (n+1)$ and $(\Delta^C\ \delta^C)\ (n+1) \subseteq (\Delta^A\ \delta^A)\ (n+1)$ since every element in $(\Delta^C\ \delta^C)\ (n+1)$ corresponds to such a t. Choose any such t as $\sigma^C(n+1)$, where s equals $\sigma^C(n)$.

$\square$

3.3 Data refinement

Let us assume we are given two Z specifications *Spec*1 and *Spec*2 with corresponding demonic Z machines working on different state spaces:

$$
\begin{array}{|l}
\hline
ZM^A \\
\sigma^A : \mathbb{N} \rightarrow State^A \\
\delta^A : \mathbb{N} \rightarrow OP^A \\
\hline
\delta^A \in Cones^A \\
\forall n : \mathbb{N} \bullet csb(\sigma^A(n), \sigma^A(n+1)) \in \delta^A \ n \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
ZM^C \\
\sigma^C : \mathbb{N} \rightarrow State^C \\
\delta^C : \mathbb{N} \rightarrow OP^C \\
\hline
\delta^C \in Cones^C \\
\forall n : \mathbb{N} \bullet csb(\sigma^C(n), \sigma^C(n+1)) \in \delta^C \ n \\
\hline
\end{array}
$$

Additionally, we assume that an abstraction relation between the two state spaces is given.

$$| \ ABS : State^C \leftrightarrow State^A$$

Similarly, there is a mapping relating abstract operations to concrete operations.

$$| \ dr : OP^A \rightarrow OP^C$$

The use of the concrete machine instead of the abstract one can again be described by the following schema.

$$
\begin{array}{|l}
\hline
UZM^A_C \\
ZM^A \\
ZM^C \\
\hline
\delta^C = dr \circ \delta^A \\
\hline
\end{array}
$$

The refinement relation for data refinement is almost identical to the one for operation refinement, except that we have to perform a translation between the state spaces.

$$ZM^A \sqsubseteq_{dr} ZM^C$$

iff

1. $UZM^A_C \setminus (\sigma^A, \delta^C) \subseteq ZM^A_{ABS} \setminus (\sigma^A)$

2. $ZM^A = UZM^A_C \setminus (\sigma^C, \delta^C)$

where

$$ZM^A_{ABS} \mathrel{\widehat=} [ZM^A;\ \sigma^C : \mathbb{N} \to State^C \mid \forall\, n : \mathbb{N}_1 \bullet (\sigma^C(n), \sigma^A(n)) \in ABS]$$

Clearly, $\sqsubseteq_{dr}$ is reflexive. That it is transitive can be seen as follows. Assume we have three machines ZM^0, ZM^1 and ZM^2 such that $ZM^0 \sqsubseteq_{dr_1} ZM^1$ and $ZM^1 \sqsubseteq_{dr_2} ZM^2$. We show $ZM^0 \sqsubseteq_{dr} ZM^2$ with $dr = dr_2 \circ dr_1$ and $ABS = ABS_1 \mathbin{⨾} ABS_2$.

1. Given a binding for $\sigma^0, \delta^0, \sigma^2$ and δ^2 in UZM^0_2. Because of $ZM^0 \sqsubseteq_{dr_1} ZM^1$ there exists σ^1 and δ^1 forming a cone in ZM^1 with $\delta^1 = dr_1 \circ \delta^0$. Hence, $\delta^2 = dr_2 \circ \delta^1$ implies that there exists σ'^1 and δ^1 with bindings in ZM^1 with $\forall\, n : \mathbb{N}_1 \bullet (\sigma^2(n), \sigma'^1(n)) \in ABS_2$. Therefore, $(\sigma^0, \delta^0, \sigma'^1, \delta^1)$ corresponds to a binding in UZM^0_1. There exists σ'^0 and δ^0 forming a cone in ZM^0 with $\forall\, n : \mathbb{N}_1 \bullet (\sigma'^1(n), \sigma'^0(n)) \in ABS_1$. Together this implies $\forall\, n : \mathbb{N}_1 \bullet (\sigma^2(n), \sigma'^0(n)) \in ABS$.

2. To discharge the second obligation is trivial.

□

We can also show that the standard local data refinement conditions of Z are sufficient to guarantee correct refinement of the machines.

Proposition 3.3 *If the local data refinement conditions hold for all op^A and op^C with respect to ABS then $ZM^A \sqsubseteq_{dr} ZM^C$.*

Proof
The proofs to discharge the two obligations are essentially the same as in the case of operation refinement.

4 Z systems and fairness conditions

It is clear that liveness conditions can be specified in an ad hoc manner in Z. The question is whether such conditions can be described in a systematic way such that the user of Z is guided in the development of the specification. In the following we present such a systematic approach together with the underlying theory.

4.1 Fairness conditions

As pointed out in [Lam94a], allowing arbitrary liveness conditions can lead to inconsistent specifications. The goal must be to develop a framework in which the safety property and the liveness property live in harmony. In this context harmony means machine closed specifications.

It is again the TLA approach that shows us the way of how this can be achieved. The idea is to incorporate conjunctions of weak and strong fairness conditions for operations into our specifications. Therefore, we have to define operators for these two kinds of conditions. The operator for strong fairness is given by

$$
\begin{array}{|l}
SF : OP \rightarrow \mathbb{P}(\mathbb{N} \rightarrow OP) \\
\hline
\forall\, op : OP \bullet SF\ op = \{\delta : \mathbb{N} \rightarrow OP \mid (\forall\, n : \mathbb{N} \bullet \exists\, k : \mathbb{N} \mid k \geq n \bullet \delta\ k = op) \vee \\
\qquad\qquad\qquad\qquad (\exists\, n : \mathbb{N} \bullet \forall\, k : \mathbb{N} \mid k \geq n \bullet \neg\, ((\Delta\ \delta)\ k \subseteq \{pre\ op\}))\}
\end{array}
$$

An operation sequence fulfills the strong fairness condition with respect to an operation if either the operation occurs infinitely often in the sequence or the operation can no more be applied from a certain point on.

The operator for weak fairness is

$$
\begin{array}{|l}
WF : OP \rightarrow \mathbb{P}(\mathbb{N} \rightarrow OP) \\
\hline
\forall\, op : OP \bullet WF\ op = \{\delta : \mathbb{N} \rightarrow OP \mid (\forall\, n : \mathbb{N} \bullet \exists\, k : \mathbb{N} \mid k \geq n \bullet \delta\ k = op) \vee \\
\qquad\qquad\qquad\qquad (\forall\, n : \mathbb{N} \bullet \exists\, k : \mathbb{N} \mid k \geq n \bullet \neg\, ((\Delta\ \delta)\ k \subseteq \{pre\ op\}))\}
\end{array}
$$

An operation sequence fulfills the weak fairness condition with respect to an operation if either the operation occurs infinitely often in the sequence or the operation can infinitely often not be applied.

Proposition 4.1 $SF\ (op) \subseteq WF\ (op)$ *for all* $op \in OP$

4.2 Z Systems

With the help of the above weak fairness and strong fairness operators the notion of *Z systems* can be introduced. Z systems are Z machines with conjunctions of weak and strong fairness conditions for operations in the set OP.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\ulcorner\, ZS\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
ZM \\
\hline
\delta \in WF\ op_j \\
\delta \in SF\ op_i \\
\ldots \\
\hline
\end{array}
$$

An important property of specifications using Z systems is that they are machine closed with respect to our notion of convergence. Machine closure means that the supplementary fairness conditions do not specify any safety property not already described by the underlying Z machine [AL88]. Technically, we have to prove that any $zm \in ZM$ is the limit point of a sequence of bindings $zs^i \in ZS$.

Proposition 4.2 *ZS is machine closed*

Proof
We give the proof for the special case where ZS contains two strong fairness conditions. The general case for more than two strong fairness conditions can be done in similar way. In case of weak fairness conditions one uses the result for strong fairness conditions together with proposition 4.1.

We need first some auxiliary notation. The embedding of finite sequences into $(\mathbb{N} \rightarrow OP)$ can be done by adding stuttering steps:

$$
\begin{array}{|l}
\_^{E} : (\mathbb{N} \twoheadrightarrow OP) \rightarrow (\mathbb{N} \rightarrow OP) \\
\hline
\forall f : (\mathbb{N} \twoheadrightarrow OP);\ n \in \mathbb{N} \bullet n \in \mathrm{dom}(f) \Rightarrow f^{E}(n) = f(n) \\
\qquad\qquad n \notin \mathrm{dom}(f) \Rightarrow f^{E}(n) = \Xi State
\end{array}
$$

Important is the following restriction of Δ to finite sequences.

$$
\begin{array}{|l}
\Gamma : (\mathbb{N} \twoheadrightarrow OP) \rightarrow (\mathbb{N} \rightarrow \mathbb{P}\, State) \\
\hline
\forall d : (\mathbb{N} \twoheadrightarrow OP) \bullet \Gamma d = \Delta(d^{E})
\end{array}
$$

Let $zm \in ZM$ and $ZS \mathrel{\widehat{=}} [ZM \mid \delta \in SF(op_1) \wedge \delta \in SF(op_2)]$ with $op_1,\ op_2 \in OP$. Then we define a sequence $(zs^i)_i$ in ZS with limit point zm by the following algorithm:

We fix $i \in \mathbb{N}$ and construct sequences $\delta \in (\mathbb{N} \rightarrow OP)$ and $\sigma \in (\mathbb{N} \rightarrow State)$ as follows. We set $\delta(k) := zm.\delta(k)$ for $k \leq i$ and $\sigma(k) := zm.\sigma(k)$ for $k \leq i + 1$. Then we proceed by recursive definition over $n \in \mathbb{N}$ using the following program.

$$
\begin{array}{l}
\textit{if } (\Gamma\delta)(i + n + 1) \subseteq\ \textit{pre } op_1 \wedge \delta(i + n) \neq op_1 \\
\qquad\quad \textit{then } \delta(i + n + 1) := op_1 \\
\qquad \textit{else if } (\Gamma\delta)(i + n + 1) \subseteq\ \textit{pre } op_2 \wedge \delta(i + n) \neq op_2 \\
\qquad\qquad\quad \textit{then } \delta(i + n + 1) := op_2 \\
\qquad\quad \textit{else } \delta(i + n + 1) := \Xi State
\end{array}
$$

It remains to choose $\sigma(i+n+2)$ in the range of $\delta(i+n+1)$ applied to $(\Gamma\delta)(i+n+1)$. Thus we get δ and σ which belong to a binding $zs \in ZS$. We define $zs^i := zs$ and since any finite prefix of the operation and state sequence of zm appears as prefix of some zs^i we have $\textit{lim}\ zs^i = zm$. □

4.3 Refinement

We conclude this section by considering refinement of Z systems. It is no longer true that for all abstract cones δ^A satisfying the fairness conditions (e.g. $SF(op^A)$) of the abstract Z system ZS^A the corresponding cone $\delta^C = or \circ \delta^A$ or $\delta^C = dr \circ \delta^A$ satisfies the corresponding fairness conditions ($SF(op^C)$ with $op^C = or(op^A)$ or $op^C = dr(op^A)$) of the concrete system ZS^C. So, it may be the case that an abstract cone in ZS^A has no concrete representation in ZS^C. The reason is that we loose control over the negation of the preconditions of the concrete operations.

Let us assume, we are given two Z machines ZM^A and ZM^C. Let us further assume we are given corresponding Z systems ZS^A and ZS^C, where all the operations used in the fairness conditions in ZS^A and ZS^C are related by a mapping $dr : OP^A \rightarrow OP^C$resulting from local data refinement. Then the following result holds

Proposition 4.3 $(UZM_C^A \wedge ZS^C) \setminus (\sigma^C, \delta^C) \subseteq ZS^A$

Proof
This results from the simple observation that if the abstract operation is infinitely often enabled then the corresponding concrete operation is infinitely often enabled, too. □

It is clear that a similar result holds for operation refinement, too.

5 Birthday book as a Z system

The well-known birthday book example [Spi92] serves as a standard example of showing essential Z constructions and techniques. Using TLZ Leslie Lamport [Lam94b] explained the main differences between TLA and Z by a temporal specification of the birthday book. In the following, we will transform this specification into our framework in order to have a standard specification example for the above introduced new concepts.

$[Name, Date, Keys]$

$$\begin{array}{|l}
Add, Find, Nil : Keys \\
0 : Date
\end{array}$$

The variables Add, $Find$, Nil serve as control variables for possible state transitions.

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{State} \\
known : \mathbb{P}\, Name \\
birthday : Name \rightarrow\!\!\!\!\!\!+\; Date \\
name_in : Name \\
date_in, date_out : Date \\
keys : Keys \\
\hline
known = \operatorname{dom} birthday \\
\hline
\end{array}$$

The input variables $name\_in$ and $date\_in$ are used for communication and can only be changed from the outside world. They play the same role as the external functions in the evolving algebra approach [Gur94].

$$
\begin{array}{|l}
csb : State \times State \rightarrow \Delta State \\
\hline
csb = \{\Delta State \bullet ((\theta State, \theta State'), \theta(\Delta State))\}
\end{array}
$$

Init

$$
\begin{array}{|l}
\Delta State \\
\hline
keys' = Nil \\
date\_out' = 0 \\
birthday' = \emptyset
\end{array}
$$

The following two operations are used to change the control variables before one of the operations *Add* or *Find* can be performed.

SetControlAdd

$$
\begin{array}{|l}
\Delta State \\
\hline
keys = Nil \\
keys' = Add \\
date\_out' = date\_out \\
birthday' = birthday
\end{array}
$$

SetControlFind

$$
\begin{array}{|l}
\Delta State \\
\hline
keys = Nil \\
keys' = Find \\
date\_out' = date\_out \\
birthday' = birthday
\end{array}
$$

Add

$$
\begin{array}{|l}
\Delta State \\
\hline
keys = Add \\
keys' = Nil \\
name\_in \notin known \Rightarrow \\
\qquad birthday' = birthday \cup \{name\_in \mapsto date\_in\} \\
name\_in \in known \Rightarrow \\
\qquad birthday' = birthday \\
date\_out' = date\_out
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Find} \\
\Delta State \\
\hline
keys = Find \\
keys' = Nil \\
name\_in \in known \Rightarrow date\_out' = birthday(name\_in) \\
name\_in \notin known \Rightarrow date\_out' = 0 \\
birthday' = birthday \\
known' = known \\
\hline
\end{array}
$$

The definitions for the Z machine and the Z system are straightforward.

$$OP \mathrel{\widehat{=}} \{Init, \Xi State, SetControlAdd, SetControlFind, Add, Find\}$$

$$
\begin{array}{|l}
\Delta : (\mathbb{N} \rightarrow \mathbb{P}\,\Delta State) \rightarrow (\mathbb{N} \rightarrow \mathbb{P}\,State) \\
\hline
\forall\, \delta : (\mathbb{N} \rightarrow \mathbb{P}\,\Delta State) \bullet \\
\quad (\Delta\ \delta)\ 0 = State \land \forall\, n : \mathbb{N} \bullet (\Delta\ \delta)\ (n+1) = \\
\qquad\qquad \{t : State \mid (\exists\, s : State \mid s \in (\Delta\ \delta)\ n \bullet csb(s,t) \in \delta\ n)\}
\end{array}
$$

$$Cones == \{\delta : \mathbb{N} \rightarrow OP \mid (\delta\ 0\ = Init \land \forall\, n : \mathbb{N}_1 \bullet (\Delta\ \delta)\ n \subseteq \{pre\ (\delta\ n)\})\}$$

$$
\begin{array}{|l}
\hline
\textit{ZM} \\
\sigma : \mathbb{N} \rightarrow State \\
\delta : \mathbb{N} \rightarrow OP \\
\hline
\delta\ 0 = Init \\
\forall\, n : \mathbb{N} \bullet csb(\sigma(n), \sigma(n+1)) \in \delta\ n \\
\delta \in Cones \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
WF : OP \rightarrow \mathbb{P}(\mathbb{N} \rightarrow OP) \\
\hline
\forall\, op : OP \bullet WF\ \ op = \{\delta : \mathbb{N} \rightarrow OP \mid (\forall\, n : \mathbb{N} \bullet \exists\, k : \mathbb{N} \mid k \geq n \bullet \delta\ k = op) \lor \\
\qquad\qquad\qquad (\forall\, n : \mathbb{N} \bullet \exists\, k : \mathbb{N} \mid k \geq n \bullet \neg\ ((\Delta\ \delta)\ k \subseteq \{pre\ op\}))\}
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{ZS} \\
ZM \\
\hline
\delta \in WF(Add) \land \delta \in WF(Find) \\
\hline
\end{array}
$$

6 Conclusions and related work

In this paper, we have shown how Z can be systematically used to specify complex systems. To this end, we have introduced the notion of Z machines which captures the ideas of demonic computations of standard Z specifications. Adding fairness conditions to Z machines yields Z systems which form the main concepts for the specification of complex systems. The expressive power of Z systems is comparable to that of TLA formulae. The use of temporal operators can be avoided in Z because schemas allow to specify predicates on sequences of states and operations.

Our systematic use of standard Z for describing complex systems by including fairness conditions is new. Leslie Lamport proposed a language TLZ [Lam94b] which is Z extended by temporal formulae à la TLA. In [DS89], Z is extended to include temporal operators. This extension is used to describe the alternating bit protocol. Particular solutions to specifying complex systems are reported in [ND90] and [HJ94].

The paper that comes closest to our approach is [SJ90], where Z is used for process modeling in a process algebraic style.

It may be argued that we simply make explicit the model underlying demonic computations and that the fairness conditions are specified by operators rather than formulae. This is certainly true to some extend. But we wanted to develop a simple and yet general approach to systematically use Z for complex systems. We share the doubts of Leslie Lamport [Lam94b] that temporal formulae are easy to understand and work with in non-toy examples. Therefore, we do not want to introduce operators such as $\Diamond$ and $\Box$. Our operators SF and WF are easy to understand and they do not have to be further combined in specifications of complex systems other than by conjunctions. The development of a Z system specification is systematic. This makes live easy. Z systems have similar expressive power as TLA and there is a underlying theory that guarantees important properties of specifications such as being machine closed. Another advantage of our approach is that the standard refinement techniques are still relevant.

We are currently investigating the possibilities of conjoining Z systems with the help of the schema calculus. The goal is to develop a technique for modular specification and refinement.

References

[AL88] M. Abadi and L. Lamport. The existence of refinement mappings. In *Proceedings of the 3rd Annual Symposium on Logic in Computer Science*, pages 165–175. IEEE, 1988.

[BS93] J. Bowen and V. Stavridou. Safety-critical systems, formal methods and standards. *Software Engineering Journal*, 8(4):189–209, July 1993.

[DS89] R. Duke and G. Smith. Temporal logic and Z specifications. *The Australian Computer Journal*, 21(2):62–66, 1989.

[GCR94] S. Gerhart, D. Craigen, and T. Ralston. Experience with formal methods in critical systems. *IEEE Software*, 11(1):21–28, January 1994.

[Gur94] Y. Gurevich. Evolving algebras 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*. Oxford University Press, 1994.

[HJ94] I. Houston and M. Josephs. Specifying distributed CICS in Z; accessing local and remote resources. *Formal Aspects of Computing*, 6(6), 1994.

[Lam94a] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[Lam94b] L. Lamport. TLZ. In J. Bowen and A. Hall, editors, *Proceedings of the 1994 Z User Workshop*, pages 267–268. Springer-Verlag, 1994.

[MP92] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag, 1992.

[ND90] K.T. Narayana and S. Dharap. Invariant properties in a dialog system. *ACM SIGSOFT Software Engineering Notes*, 15(4):67–79, 1990.

[Ost94] J. Ostroff. Formal methods for the specification and design of real-time safety critical systems. Draft 2.0, 1994.

[SJ90] B. Sufrin and He Jifeng. Specification, analysis and refinement of interactive processes. In M. Harrison and H. Thimbleby, editors, *Formal Methods in Human-Computer Interaction*, volume 2, chapter 6, pages 153–200. Cambridge University Press, 1990.

[Spi92] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, second edition, 1992.

[Woo92] J.C.P. Woodcock. The rudiments of algorithm refinement. *The Computer Journal*, 35(5):441–450, 1992.