Working Paper Series
ISSN 1170-487X

μ-Charts and Z: hows, whys and wherefores

by Greg Reeve and Steve Reeves

Working Paper 00/6
March 2000

μ-Charts and Z: hows, whys and wherefores

Greg Reeve and Steve Reeves

Department of Computer Science,
University of Waikato, Hamilton,
New Zealand

Abstract. In this paper we show, by a series of examples, how the μ-chart formalism can be translated into Z. We give reasons for why this an interesting and sensible thing to do and what it might be used for.

1 Introduction

In this paper we show, by a series of examples, how the μ-chart formalism (as given in [9]) can be translated into Z. We also discuss why this is a useful and interesting thing to do and give some examples of work that might be done in the future in this area which combines Z and μ-charts.

It might seem obvious that we should simply express the denotational semantics given in [9] directly in Z and then do our proofs. After all, the semantics is given in set theory and so Z would be adequate for the task. However, our aim is to produce versions of μ-charts that are recognisably Z models, *i.e.* using the usual state and operation schema constructs and some schema calculus in natural ways—μ-chart states and transitions appear as Z state and operation schemas respectively. Thus we aim for a Z model in the conventionally accepted form. We want this so that the Z model is readily understandable in its own right and also comparable with alternative models in Z, *e.g.* those of Weber [17] (which we will mention briefly again in section 5.3).

To make the paper fairly self-contained we start by giving a brief introduction to the main conceptual apparatus of this paper: the specification language Z, the reactive system modelling formalism of μ-charts and the Z proof tool Z/EVES. In each case, though, the references cited should be used by readers who wish to have a full and proper introduction to each topic.

We then explain our translation via a series of examples in sections 2 and 3. In section 4 we give an extended commentary on aspects of [9] in which we seek to show how the concerns of that paper, which uses the technique of oracles, have been dealt with by our translation. We also, somewhat speculatively, consider extending the technique in [9] in a way which appears to combine the semantics in that paper with the alternative semantics given in [14]. Finally, we motivate the current work and look into the future.

1.1 μ-Charts

μ-Charts [9] are a visual representation used for the specification of reactive systems. They extend finite state transition diagrams by adding modularisation through hierarchical decomposition, *i.e.* allowing states to contain other μ-charts, and allow the modelling of separate communicating processes by parallel composition, that is allowing two μ-charts to communicate via broadcast signals. The communication between μ-charts in a specification is modelled using instantaneous feedback of signals.

μ-Charts are a variant of Statecharts [1] that exclude a number of syntactical concepts which cause semantical problems. Some of the excluded items include inter-(hierarchy-)level transitions and transition priorities determined by hierarchy. Also μ-charts have been given a compositional semantics (which allows abstract specification by incorporating *e.g.* non-determinism) and refinement rules have also been given that describe a formal process of making an abstract or non-deterministic μ-chart more concrete.

μ-Charts can model both preemptive and non-preemptive interrupts. The difference between these concerns whether or not a sub-chart in the hierarchy (or *slave*) is allowed to fire a transition at the same time as it loses control, *i.e.* when a transition leaving the slave is triggered in the *master* (the chart which contains the slave as a sub-chart). The preemptive interrupt, or strong preemption, model prevents a slave from making any further transition, *i.e.* implicitly the negation of all trigger expressions of transitions leaving the slave are added to all transitions within the slave. Non-preemptive interrupts, or weak preemption, allows a slave to make any valid transition in conjunction with the transition leaving it. Strong preemption can be modelled in a weak preemption framework by explicitly adding the required negated trigger expressions to the slave's transitions. This also allows a mixture of strong and weak preemption. However weak preemption cannot be modelled in a strong preemption framework.

The compositional semantics for μ-charts considered in this paper is that described in [9], though we do acknowledge there have since been alternatives to this proposed, as in [13] and [14]. The ramifications of these alternatives have not yet been fully investigated, though we make some comments on this in sections 4, 5 and 6. Since we shall be introducing μ-charts via several examples in the sequel, we will say no more about them here.

1.2 Z

The specification language Z is based upon typed set theory and first-order predicate calculus and includes the notion of schemas used to encapsulate mathematical objects and their properties by declaration and constraint. Z is a model-based notation usually used to represent abstract specifications of systems by describing observations of their state and some operations that can change that state (see [7], [18]).

To give an example of Z we specify a schema named *State* : *Exp* with a single integer observation that is constrained to remain within a given range:

$$\begin{array}{|l}\multicolumn{1}{l}{\textit{State}}\\ \hline x : \mathbb{N} \\ \hline x > 10 \\ x \leq 20 \\ \hline \end{array} \qquad \begin{array}{|l}\multicolumn{1}{l}{\textit{InitState}}\\ \hline \textit{State} \\ \hline x = 11 \\ \\ \hline \end{array}$$

Z state schemas are divided into declarations, *i.e.* observation names (or *labels*) and their types, and predicates constraining those observations. This division is represented graphically by a horizontal line. Two other principles of Z demonstrated here are the convention of initialisation and schema inclusion. Z specifications generally include an initialisation schema, *e.g.* *InitState*, that specifies what happens at system startup. Schema inclusion, *e.g.* of *State* : *Exp* included in the declaration of *InitState*, is a notational shorthand for including the respective declaration and predicate of one schema into another (subject to certain consistency requirements between the declarations).

Operation schemas are different from state schemas in that they include two copies of the state. One copy, known as the *unprimed copy*, represents the values of the state observations before the operation and the other, the *primed copy*, represents the values of the observations after the operation.

$$\begin{array}{|l}\multicolumn{1}{l}{\textit{Increment}}\\ \hline \Delta \textit{State} \\ \hline x < 20 \\ x' = x + 1 \\ \hline \end{array} \qquad \begin{array}{|l}\multicolumn{1}{l}{\textit{Restart}}\\ \hline \Delta \textit{State} \\ \hline x \geq 20 \\ x' = 11 \\ \hline \end{array}$$

$T\_Op \mathrel{\widehat{=}} \textit{Increment} \lor \textit{Restart}$

The schema inclusion of $\Delta \textit{State}$ is another convention in Z where Δ schemas are recognised to be defined as including a primed and unprimed copy of the schema, *e.g.* $\Delta \textit{State} \mathrel{\widehat{=}} [\textit{State}, \textit{State}']$. Also operation schemas including Δ-schemas are conventionally recognised to be operations that change the state. All the various forms of schema given above are examples of Z *paragraphs*.

An operation schema has a *precondition* which has explicit and implicit components, taken in conjunction, *e.g.* *Increment* has the components $x < 20$, $x \leq 20$ and $x > 10$; the first explicit, the last two implicit. An operation is said to be *total* when its precondition is true for all possible values of the state. If a specification does not provide total operations then the reaction of the eventual implementation in situations where the precondition of the operation is false will be undefined. This fact can be advantageous, remembering Z is used for abstract specification, when the specifier does not care to define the behaviour in some situations.

1.3 Z/EVES

Z/EVES [11] is a type checking and theorem proving tool for Z specifications. It is an interactive system in which Z specifications can be presented one paragraph at a time or in their entirety. Theorems can be defined and proofs attempted at any time. Z/EVES was developed by ORA [8] and is used here to prove properties about the Z translated from μ-charts.

To give an example of using Z/EVES we evaluate the precondition of the schema T_Op, from section 1.2, to show that this operation is total when applied to any valid observation of the state, *i.e.* to prove that the predicate $State \Rightarrow \mathsf{pre}\ \ T\_Op$ is true:

$try\ State \Rightarrow \mathsf{pre}\ \ T\_Op;$
$invoke;$
$instantiate\ x' == x + 1;$
$prove\ by\ reduce;$
$\longrightarrow$
$\qquad \boldsymbol{true}$

The *try* command (first line of input) allows the user to evaluate predicates about the current specification. In this case the predicate says that assuming the constraints on this state the precondition (calculated by the Z prefix operator pre) of the operation schema T_Op is always true, *i.e.* the operation can be applied to any valid configuration of the state. The Z/EVES commands on lines 2-4 of the input instruct Z/EVES to carry out some evaluation on the given predicate which is true, as expected.

2 The Translation: An Example

In this section we concentrate on giving a series of examples which highlight the main features of our translation procedure which turns μ-charts into Z. The presentation of the procedure in detail (and its formal justification) will be given in a later, fuller paper.

The μ-chart given in figure 1, was chosen as the simplest example that demonstrates the following concepts:

- multiple sequential automata;
- more than one level of hierarchical decomposition;
- instantaneous feedback of signals.

The Z translation is presented by giving a Z description (as a state schema) of each state in the example μ-chart followed by a description of each transition as an operation schema. The correct combination of these transitions then describes the overall behaviour of the μ-chart. Lastly we investigate the resulting Z specification using the tool Z/EVES.

2.1 Describing the States

The first task in the translation is to give a Z state schema, by convention of the same name, for each of the states in the μ-chart. This includes those states that

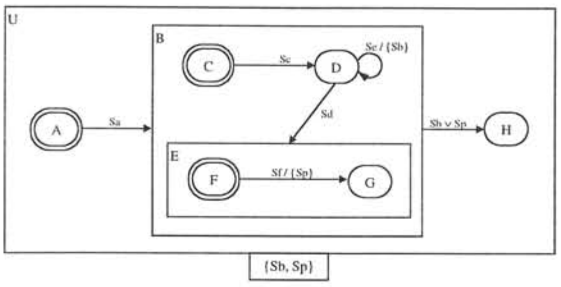

Fig. 1. A simple μ-chart

represent slaves (sub-μ-charts) and those that represent atomic states. We first give values that will be used to name states and signals:

$\mu_{State} ::= a \mid b \mid c \mid d \mid e \mid f \mid g \mid h$
$Signal ::= Sa \mid Sb \mid Sc \mid Sd \mid Se \mid Sf \mid Sp$

Then we need a Z state which represents the whole chart together with its initialisation to the correct μ-chart state:

U
$c_U : \mu_{State}$

$InitU$
U
$c_U = a$

The next states to be described, A and H, are atomic states and therefore are trivially described by the schemas A and H: [1]

A
$c_U : \mu_{State}$
$c_U = a$

H
$c_U : \mu_{State}$
$c_U = h$

More interestingly, B is a slave of U [2]. Therefore, it introduces an observation of the slave μ-chart's state as well as defining what it means for U to be in the state B, *i.e.* $c_U = b$. Each schema that models a μ-chart (as opposed to an atomic state) also has an initialisation operation defined that specifies what state the

[1] Note that in the sequel we shall usually omit explicit mention of such outermost charts as U since their formalisation contributes nothing to our translation.

[2] or—the μ-chart B is a slave of U. This view of B as both a state and a μ-chart (and indeed a schema) needs to be kept in mind, and when which we mean is not clear from the context we make it explicit.

slave is initialised to when it becomes active, that is when the master enters the corresponding state:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{B}} \\ \hline
c_U, c_B : \mu_{State} \\ \hline
c_U = b \\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\textit{InitB}} \\ \hline
B \\ \hline
c_B = c \\ \hline
\end{array}$$

Now the three schemas C, D and E are given describing the states of the μ-chart B. First, for the atomic states C and D:

$$\begin{array}{|l}
\multicolumn{1}{l}{C} \\ \hline
c_B : \mu_{State} \\ \hline
c_B = c \\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{D} \\ \hline
c_B : \mu_{State} \\ \hline
c_B = d \\ \hline
\end{array}$$

Notice that we don't include the schema B in C or D as may seem intuitive. If B were included in D then μ-chart U would be forced to be in state B after any transition in the μ-chart B to state D—but this then blocks modelling in μ-charts what were inter-level transitions in Statecharts. We want to allow such transitions because it allows modular design of μ-charts and hence a more natural and tractable modelling of physical systems.

Schema E models a μ-chart and therefore has an initialisation schema:

$$\begin{array}{|l}
\multicolumn{1}{l}{E} \\ \hline
c_B, c_E : \mu_{State} \\ \hline
c_B = e \\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\textit{InitE}} \\ \hline
E \\ \hline
c_E = f \\ \hline
\end{array}$$

Now schemas F and G are given for the two atomic states of μ-chart E:

$$\begin{array}{|l}
\multicolumn{1}{l}{F} \\ \hline
c_E : \mu_{State} \\ \hline
c_E = f \\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{G} \\ \hline
c_E : \mu_{State} \\ \hline
c_E = g \\ \hline
\end{array}$$

Given all of the state descriptions we can now go on to describe the transitions between these states. The example being described here uses weak preemption between master and slave: however the translation method is easily extended to incorporate strong preemption or preemptive interrupts.

2.2 Describing the Transitions

One of the properties of μ-charts that must be modelled is instantaneous feedback. The semantics of μ-charts, as given in [9], uses a fixed-point construction to calculate, in an operational fashion, the overall input for a μ-chart step. In the Z translation we use the fact that only one transition can happen per μ-chart (in any hierarchy, i.e. can only happen once in U, once in B and once in E) per step, and the nature of the schema calculus (the effects of schema inclusions), to give a declarative description of the possible inputs for a step.

The feed back signals that are locally visible to each μ-chart in the specification are given by the sets l_U, l_B and l_E defined below. The local sets can be read from any attached feed back boxes in the graphical notation. The sets f_U, f_B and f_E determine the scope (due to the hierarchy) of feed back signals in the μ-chart and therefore give all the visible feedback signals for each μ-chart:

$$\begin{aligned}
&l_U == \{Sb, Sp\}\\
&f_U == l_U\\
&l_B == \{\}\\
&f_B == f_U \cup l_B\\
&l_E == \{\}\\
&f_E == f_B \cup l_E
\end{aligned}$$

In this simple example the fed back signals are common across all of the μ-charts. A schema, *Output*, that describes the allowable output information for a μ-chart is also given:

```
── Output ──────────────────────────
│ $o_U : \mathbb{P}\{Sb, Sp\}$
│ $o_B : \mathbb{P}\{Sb, Sp\}$
│ $o_E : \mathbb{P}\{Sp\}$
│ $o! : \mathbb{P}\{Sb, Sp\}$
├──────────────────
│ $o! = \bigcup\{o_U, o_B, o_E\}$
└──────────────────────────────────
```

The schema *Output* has an observation of the output signals generated by each μ-chart in the specification. Each of these observations represents the set (possibly empty) of output signals from the given μ-chart in a step and the union of these sets gives the overall output. The intersection of the output and the feed back set gives the feed back signals visible for the current step.

Now, given the states of the μ-chart and the output/feedback mechanism, an operation schema is given for each transition. The description of the transitions is organised to follow the hierarchy of the example μ-chart.

Transitions in μ-Chart U: Firstly an operation schema, δ_{ab}, is given describing the transition between states A and B in μ-chart U[3]:

[3] Recall: *Signal* is a set of all external, internal (*i.e.* feedback) and output signals that are available in the system being modelled.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\delta_{ab}} \\
\hline
A \\
InitB' \\
Output \\
input? : \mathbb{P}\, Signal \\
\hline
Sa \in input? \cup (o! \cap f_U) \\
o_U = \{\} \\
\hline
\end{array}
$$

Due to the schema inclusions here, the precondition of δ_{ab} states that the current state of U must be A, *i.e.* $c_U = a$, the overall output set is equal to the union of o_U, o_B and o_E and the signal Sa must be external input or a fed back signal for this transition to happen. Since the transition goes into a state which is also a slave μ-chart the schema included has to be the initialisation schema for that slave.

Note the precondition of this transition is not yet determinable because the values of o_B and o_E are not known: it establishes a *constraint* on the allowable solutions. δ_{ab} stipulates that if the precondition is true then the μ-chart U will be in state B after this transition and the μ-chart B will be in state C.

Similarly, the transition between states B and H is given by δ_{bh}:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\delta_{bh}} \\
\hline
B \\
H' \\
Output \\
input? : \mathbb{P}\, Signal \\
\hline
Sb \in input? \cup (o! \cap f_U) \\
\vee\ Sp \in input? \cup (o! \cap f_U) \\
o_U = \{\} \\
\hline
\end{array}
$$

The precondition allows this transition only if the signals Sb or Sp are present in the input or feed back.

The μ-chart semantics state that in the situation where no transition is triggered then the μ-chart stays in its current state. To model this in Z we provide another operation schema. For the μ-chart U this schema is ϵ_U:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\epsilon_U} \\
\hline
c_U, c_U', c_B : \mu_{State} \\
input? : \mathbb{P}\, Signal \\
Output \\
\hline
\neg\, (A \wedge Sa \in input? \cup (o! \cap f_U)) \\
\neg\, (B \wedge (Sb \in input? \cup (o! \cap f_U) \vee Sp \in input? \cup (o! \cap f_U))) \\
c_U' = c_U \\
o_U = \{\} \\
\hline
\end{array}
$$

The precondition of this schema states that no valid transition in the μ-chart U is triggered. The postcondition states that the μ-chart configuration remains unchanged. The output, given the chart makes no transition, must clearly be empty.

Transitions in μ-Chart B: The transition descriptions for μ-chart B are given by δ_{cd}, δ_{dd} and δ_{de}:

$$\begin{array}{|l}
\multicolumn{1}{l}{\delta_{cd}} \\
\hline
B \\
C \\
D' \\
input? : \mathbb{P}\, Signal \\
Output \\
\hline
Sc \in input? \cup (o! \cap f_B) \\
o_B = \{\} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\delta_{dd}} \\
\hline
B \\
D \\
D' \\
input? : \mathbb{P}\, Signal \\
Output \\
\hline
Se \in input? \cup (o! \cap f_B) \\
o_B = \{Sb\} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\delta_{de}} \\
\hline
B \\
D \\
InitE' \\
input? : \mathbb{P}\, Signal \\
Output \\
\hline
Sd \in input? \cup (o! \cap f_B) \\
o_B = \{\} \\
\hline
\end{array}$$

Notice that the preconditions of these operations say that the transitions can only happen if the current μ-chart is in the right state, *e.g.* $c_B = c$ for δ_{cd}, and any master μ-chart is in the expected state or in other words the current μ-chart is active, *e.g.* $c_U = b$ for δ_{cd}.

As important (as we have said before) is the fact that the operation restricts only the state of the μ-chart in which the transition occurs and not its master's state. Next we give the ϵ schema and another auxiliary schema, $Inactive_B$, for the μ-chart B:

$$\begin{array}{|l}
\multicolumn{1}{l}{\epsilon_B} \\
\hline
c_U, c_B, c'_B : \mu_{State} \\
input? : \mathbb{P}\, Signal \\
Output \\
\hline
B \\
\neg\,(C \wedge Sc \in input? \cup (o! \cap f_B)) \\
\neg\,(D \wedge Sd \in input? \cup (o! \cap f_B)) \\
\neg\,(D \wedge Se \in input? \cup (o! \cap f_B)) \\
o_B = \{\} \\
c'_B = c_B \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{Inactive_B} \\
\hline
c_U, c_B, c'_B : \mu_{State} \\
Output \\
\hline
\neg\, B \\
o_B = \{\} \\
c'_B = c \\
\hline
\end{array}$$

The operation schema, *Inactive*$_B$, defines the action of B when it is not active, *i.e.* its master, U, is not in state B. An inactive μ-chart is to remain in its initial state and not react to any signals.

Transitions in μ-Chart E: The transition descriptions for μ-chart E results in three schemas, δ_{fg}, ϵ_E and *Inactive*$_E$:

$$\begin{array}{|l}
\multicolumn{1}{l}{\delta_{fg}} \\ \hline
B \\
E \\
F \\
G' \\
input? : \mathbb{P}\, Signal \\
Output \\ \hline
Sf \in input? \cup (o! \cap f_E) \\
o_E = \{Sp\} \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\epsilon_E} \\ \hline
c_U, c_B, c_E, c'_E : \mu_{State} \\
input? : \mathbb{P}\, Signal \\
Output \\ \hline
B \\
E \\
\neg\, (F \wedge Sf \in input? \cup (o! \cap f_E)) \\
o_E = \{\} \\
c'_E = c_E \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{Inactive_E} \\ \hline
c_U, c_B, c_E, c'_E : \mu_{State} \\
Output \\ \hline
\neg\, B \vee \neg\, E \\
o_E = \{\} \\
c'_E = f \\ \hline
\end{array}$$

Notice, because a μ-chart is inactive if its master is inactive, the schema *Inactive*$_E$ has the predicate $\neg\, B \vee \neg\, E$ that says the chart is considered inactive if either its master is not active, *i.e.* $\neg\, B$, or it is itself inactive, *i.e.* $\neg\, E$.

2.3 Describing the Step

Now all of the schemas needed to describe the step behaviour of the μ-chart are provided. The predicate of the step-modelling schema *Step* conjoins the possible transitions for each μ-chart and hides each μ-chart's component output observation. The schema *Step* specifies a set of bindings that describes a transition function for the μ-chart with respect to input:

$$\begin{array}{|l}
\multicolumn{1}{l}{Step} \\ \hline
c_U, c'_U, c_B, c'_B, c_E, c'_E : \mu_{State} \\
input?, o! : \mathbb{P}\, Signal \\ \hline
\exists\, o_U, o_B, o_E : \mathbb{P}\, Signal \bullet \\
\qquad ((\delta_{ab} \vee \delta_{bh} \vee \epsilon_U) \wedge \\
\qquad (\delta_{cd} \vee \delta_{de} \vee \delta_{dd} \vee \epsilon_B \vee Inactive_B) \wedge \\
\qquad (\delta_{fg} \vee \epsilon_E \vee Inactive_E)) \\ \hline
\end{array}$$

2.4 Using a Theorem Prover to investigate the Z

This section gives some examples of exploration of the resulting Z specification using the tool Z/EVES.

The first two examples examine the behaviour of the μ-chart when it is in its initial state and the signal Sa is either present or absent.

$$\begin{array}{l} \mathit{try}\ \mathit{Step}[c_U := a, c_B := c, c_E := f, \mathit{input}? := \{Sa\}]; \\ \longrightarrow \\ \qquad \boldsymbol{c'_U = b \wedge c'_B = c \wedge c'_E = f \wedge o! = \{\}} \end{array}$$

$$\begin{array}{l} \mathit{try}\ \mathit{Step}[c_U := a, c_B := c, c_E := f, \mathit{input}? := \{Sp\}]; \\ \longrightarrow \\ \qquad \boldsymbol{c'_U = a \wedge c'_B = c \wedge c'_E = f \wedge o! = \{\}} \end{array}$$

Both of these proofs result in the desired outcome. This method could be used in the same way to test all transition behaviours. For example the more interesting configuration of the μ-chart where U is in state B, B is in state D, E is inactive and Se is input, can be checked as follows:

$$\begin{array}{l} \mathit{try}\ \mathit{Step}[c_U := b, c_B := d, c_E := f, \mathit{input}? := \{Se\}]; \\ \longrightarrow \\ \qquad \boldsymbol{c'_U = h \wedge c'_B = d \wedge c'_E = f \wedge o! = \{Sb\}} \end{array}$$

Other examples of examination of the μ-chart include checking that there is some input signal set and configuration that allows a transition ending in another configuration valid of the system (this may or may not consider the expected output). Or, as valuably, no input and configuration exist that could result in a transition ending in an invalid configuration.

The first example gives the expected post-transition configuration and output and quantifies over the pre-configuration and input.

$$\begin{array}{l} \mathit{try}\ \exists\, \mathit{input}? : \mathbb{P}\, \mathit{Signal};\ c_U, c_B, c_E : \mu_{\mathit{State}} \bullet \\ \qquad\qquad \mathit{Step}[c'_U := h, c'_B := c, c'_E := f, o! := \{\}]; \\ \longrightarrow \\ \qquad \boldsymbol{\mathit{true}} \end{array}$$

This evaluates to true which can be interpreted to mean there does exist some configuration and input that would allow a transition to the given configuration with the expected output.

The next example gives a contradictory configuration and resulting output, *i.e.* we expect that the system can never make a transition ending in this configuration with this output. The expected result is therefore false.

$try\ \exists\, input? : \mathbb{P}\, Signal \bullet Step[c'_U := h, c'_B := c, c'_E := f, o! := \{Sp\}];$

$\longrightarrow$

$\qquad \boldsymbol{false}$

A variation of this type of exploration is obtained by not quantifying over the starting configuration or giving the expected output. The resulting predicate simplifies to constraints on these observations rather than true or false. It is worth noting that as less fixed information is provided the more complicated the proof and resulting predicate become, *e.g.* there are likely to be numerous combinations of possible configurations and outputs on which a transition can give the expected configuration.

$try\ \exists\, input? : \mathbb{P}\, Signal \bullet Step[c'_U := h, c'_B := e, c'_E := g];$

$\longrightarrow$

$$\boldsymbol{c_U = b \wedge e = c_B \wedge c_E = f \wedge o! = \{Sp\}}$$
$$\boldsymbol{\vee\ c_U = b \wedge e = c_B \wedge c_E = f \wedge o! = \{Sp\}}$$

Other important properties of the specification may include the inability of the μ-chart to get from one state to another in one step. For example we can prove that there is no input that will take this μ-chart from state A to state H in one step.

$try\ \exists\, input? : \mathbb{P}\, Signal \bullet Step[c_U := a, c'_U := h];$

$\longrightarrow$

$\qquad \boldsymbol{false}$

We can also compose steps together to more rigorously check reachability of configurations. Due to the grammar used by Z/EVES [4] this requires some definition of temporary schemas such as $Step_2$.

$Step_2 \mathrel{\widehat=} Step[i1/input?, o1/o!] \mathbin{\S} Step[i2/input?, o2/o!]$

$try\ \exists\, i1, i2 : \mathbb{P}\, Signal;\ c_U, c_B, c_E : \mu_{State} \bullet$
$\qquad Step_2[c'_U := h, c'_B := e, c'_E := g];$

$\longrightarrow$

[4] Z/EVES follows the grammar of Spivey [15] which does not class schema expressions as expressions *simpliciter*, hence the definition we require cannot even be expressed (though see the Acknowledgements). The grammar in the proposed standard [16] elides the two categories, so once Z/EVES supports the grammar in the new standard we would be able to straightforwardly define the function we desire.

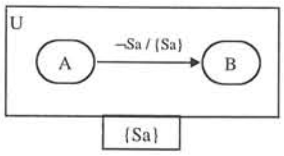

Fig. 2. Example 1: Causality problems

$$o1 = \{\} \wedge o2 = \{Sp\}$$

More interestingly the two cases below check the reachability of a given configuration from the initial configuration of the μ-chart. We need to again define temporary schemas that give compositions of *Step* with itself three times:

$Step_3 \mathrel{\widehat{=}} Step_2 \mathbin{;} Step[i3/input?, o3/o!]$

$try\ \exists\, i1, i2, i3, o1, o2, o3 : \mathbb{P}\, Signal \bullet$
$\quad Step_3[c_U := a, c_B := c, c_E := f, c'_U := h, c'_B := d, c'_E := f];$
$\longrightarrow$
$\quad \boldsymbol{true}$

$try\ \exists\, i1, i2, i3, o1, o2, o3 : \mathbb{P}\, Signal \bullet$
$\quad Step_3[c_U := a, c_B := c, c_E := f, c'_U := b, c'_B := e, c'_E := f];$
$\longrightarrow$
$\quad \boldsymbol{true}$

Given the above examples it is easily seen how we could invent different combinations to investigate more properties of the system. For example we might need to check there is at least one possible transition out of any valid configuration, *i.e.* the system is always ready to react.

3 Avoiding the pitfalls: getting the right semantics

This section gives some pathological examples taken from [9] which are given to show that the Z translation respects the semantics given for μ-charts.

3.1 Causality Problems

The first example is given by the μ-chart in figure 2.

The resulting Z from the translation is:

$$\begin{array}{|l} \multicolumn{1}{l}{A} \\ \hline c_U : \mu_{State} \\ \hline c_U = a \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{B} \\ \hline c_U : \mu_{State} \\ \hline c_U = b \\ \hline \end{array}$$

$l_U == \{Sa\}$
$f_U == l_U$

$$\begin{array}{|l} \multicolumn{1}{l}{Output} \\ \hline o_U : \mathbb{P}\{Sa\} \\ o! : \mathbb{P}\, Signal \\ \hline o! = \bigcup\{o_U\} \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{\delta_{ab}} \\ \hline A \\ B' \\ input? : \mathbb{P}\, Signal \\ Output \\ \hline Sa \notin input? \cup (o! \cap f_U) \\ o_U = \{Sa\} \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\epsilon_U} \\ \hline c_U, c'_U : \mu_{State} \\ input? : \mathbb{P}\, Signal \\ Output \\ \hline \neg\,(A \wedge Sa \notin input? \cup (o! \cap f_U)) \\ o_U = \{\} \\ c'_U = c_U \\ \hline \end{array}$$

The schema δ_{ab} above demonstrates the first translation of a negated trigger expression. Negated trigger expressions are translated by a predicate, *i.e.* $Sa \notin input? \cup (o! \cap f_U)$, that says the signal does not appear in the input (including feedback). This example (and in fact the next example) demonstrates the causality problems that can be introduced by negated trigger expressions. [5] For this example there is no solution when the signal Sa is not input, therefore the predicate of *Step*:

$$\begin{array}{|l} \multicolumn{1}{l}{Step} \\ \hline c_U, c'_U : \mu_{State} \\ input?, o! : \mathbb{P}\, Signal \\ \hline \delta_{ab} \vee \epsilon_U \\ \hline \end{array}$$

is false.

The predicate of the schema δ_{ab} is false in all cases which, as we would expect given the semantics for this μ-chart, means this transition can never occur. If the signal Sa is input the trigger condition on the transition is false causing the μ-chart to stay in the same state and (the operation) ϵ_U to take place. If Sa is not input then no transition is valid and the predicate of *Step* is false. The

[5] The semantics given in [9] introduces the idea of *oracle* signals, which can be used rule out solutions with such causality problems. See section 4 for a fuller discussion of this.

semantics for μ-charts given in [9] states that in the event of empty reactions (which means that the schema *Step* is an empty set of bindings in the translated Z) the μ-chart remains in its current configuration. Hence in both cases, *Sa* present and *Sa* absent, the Z model constrains the after state of the transition to be A, *i.e.* the configuration of the μ-chart remains the same. That this accords with the definition in [9] is evidence of the correctness of our translation.

This can be easily illustrated using Z/EVES as follows:

$\mathit{try}\ \mathit{Step}[c_U := a, \mathit{input?} := \{Sa\}];$
$\longrightarrow$

$$c'_U = a \land o! = \{\}$$

$\mathit{try}\ \mathit{Step}[c_U := a, \mathit{input?} := \{\}];$
$\longrightarrow$

$$\mathit{false}$$

3.2 Causality Problems Continued

A more interesting example is that of figure 3.

The behaviour of this μ-chart is non-deterministic when neither *Sa* nor *Sb* is input. The correct configuration of the μ-chart after a step with no input is either U in state A and V in state D or U in B and V in C.

The translation gives:

A
$c_U : \mu_{State}$
$c_U = a$

B
$c_U : \mu_{State}$
$c_U = b$

C
$c_V : \mu_{State}$
$c_V = c$

D
$c_V : \mu_{State}$
$c_V = d$

$l_U == \{Sa, Sb\}$
$f_U == l_U$
$l_V == \{Sa, Sb\}$
$f_V == l_V$

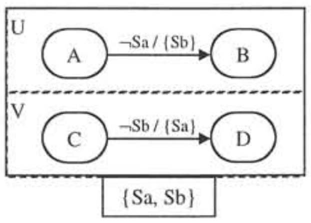

Fig. 3. Example 2: More causality problems

$$\begin{array}{|l}
\hline
\textit{Output} \\
o_U : \mathbb{P}\{Sb\} \\
o_V : \mathbb{P}\{Sa\} \\
o! : \mathbb{P}\,\textit{Signal} \\
\hline
o! = \bigcup\{o_U, o_V\} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\delta_{ab} \\
A \\
B' \\
\textit{input}? : \mathbb{P}\,\textit{Signal} \\
\textit{Output} \\
\hline
Sa \notin \textit{input}? \cup (o! \cap f_U) \\
o_U = \{Sb\} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\epsilon_U \\
c_U, c'_U : \mu_{State} \\
\textit{input}? : \mathbb{P}\,\textit{Signal} \\
\textit{Output} \\
\hline
\neg\,(A \wedge Sa \notin \textit{input}? \cup (o! \cap f_U)) \\
o_U = \{\} \\
c'_U = c_U \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\delta_{cd} \\
C \\
D' \\
\textit{input}? : \mathbb{P}\,\textit{Signal} \\
\textit{Output} \\
\hline
Sb \notin \textit{input}? \cup (o! \cap f_V) \\
o_V = \{Sa\} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\epsilon_V \\
c_V, c'_V : \mu_{State} \\
\textit{input}? : \mathbb{P}\,\textit{Signal} \\
\textit{Output} \\
\hline
\neg\,(C \wedge Sb \notin \textit{input}? \cup (o! \cap f_V)) \\
o_V = \{\} \\
c'_V = c_V \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{Step} \\
c_U, c'_U, c_V, c'_V : \mu_{State} \\
\textit{input}?, o! : \mathbb{P}\,\textit{Signal} \\
\hline
\exists\, o_U, o_V : \mathbb{P}\,\textit{Signal} \bullet \\
\qquad ((\delta_{ab} \vee \epsilon_U) \wedge \\
\qquad (\delta_{cd} \vee \epsilon_V)) \\
\hline
\end{array}$$

Four Z/EVES tests are given for the translation to show that the behaviour of the Z respects the μ-chart meaning.

$\textit{try Step}[c_U := a, c_V := c, \textit{input}? := \{Sa\}];$
$\longrightarrow$

$$\boldsymbol{c'_U = a \wedge c'_V = d \wedge o! = \{Sa\}}$$

$\textit{try Step}[c_U := a, c_V := c, \textit{input}? := \{Sb\}];$
$\longrightarrow$

$$\boldsymbol{c'_U = b \wedge c'_V = c \wedge o! = \{Sb\}}$$

$\textit{try Step}[c_U := a, c_V := c, \textit{input}? := \{\}];$
$\longrightarrow$

$$\begin{array}{l}
\textbf{if } \boldsymbol{c'_U = a} \textbf{ then } \boldsymbol{c'_V = d \wedge o! = \{Sa\}} \\
\quad \textbf{else } \boldsymbol{c'_U = b \wedge c'_V = c \wedge o! = \{Sb\}}
\end{array}$$

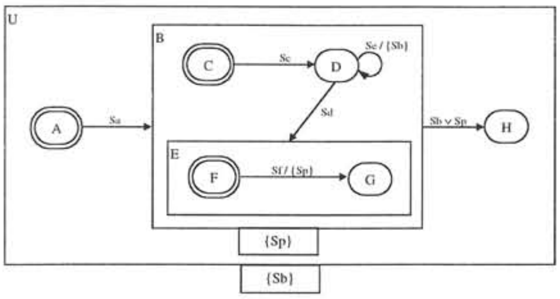

Fig. 4. A μ-chart to demonstrate feedback scoping

$$\mathit{try}\ \mathit{Step}[c_U := a, c_V := c, \mathit{input?} := \{Sa, Sb\}];$$
$$\longrightarrow$$
$$\boldsymbol{c'_U = a \wedge c'_V = c \wedge o! = \{\}}$$

This example also demonstrates how the translated Z deals with non-determinism: it is non-deterministic when neither of the signals *Sa* or *Sb* are input. The resulting *Step* schema allows one behaviour or the other. It is not possible for both to occur and *Step* makes no decision about which should occur.

3.3 Scoping of Feedback

When hierarchically decomposing a μ-chart its states can be replaced by any other μ-chart. The new μ-chart or slave may of course have localised feedback, *i.e.* the feed back signals in the slave chart are not visible to the master. This section presents an example that demonstrates that this situation is handled correctly by the translation process.

The example is a slight modification of that given in section 2 where the signal *Sp* is now only feedback to the μ-chart *B* and its slaves, see figure 4.

The translated Z for this example is almost identical to the original. The only change is the declaration of the constants l_U and l_B which are changed to the following:

$l_U == \{Sb\}$
$f_U == l_U$
$l_B == \{Sp\}$
$f_B == f_U \cup l_B$
$l_E == \{\}$
$f_E == f_B \cup l_E$

In this example the only feed back signal local to the μ-chart *U* is *Sb* and signal *Sp* becomes local to chart *B*. However, the difference between this example and the first can be shown by using Z/EVES to evaluate the outcome of a transition that generates the feed back signal *Sp* from the consecutive examples.

$$try\ Step[c_U := b, c_B := e, c_E := f, input? := \{Sf\}];$$
$$\longrightarrow$$
$$\boldsymbol{c'_U = h \wedge c'_B = e \wedge c'_E = g \wedge o! = \{Sp\}}$$

Given the appropriate configuration the example pictured in figure 1 makes a transition from state F to G with output *Sp* because of the input *Sf*, also a transition from B to H is triggered because the output signal *Sp* is instantaneously feed back to the μ-chart U.

Examining the same situation for the modified example gives:

$$try\ Step[c_U := b, c_B := e, c_E := f, input? := \{Sf\}];$$
$$\longrightarrow$$
$$\boldsymbol{c'_U = b \wedge c'_B = e \wedge c'_E = g \wedge o! = \{Sp\}}$$

The transition from F to G still happens but the transition from B to H doesn't. This is because the output signal *Sp* is no longer visible as feedback to the μ-chart U [6].

4 Oracles

In [9] the authors introduce the idea of *oracles* in order to rule out certain unwanted solutions that their fixed-point construction would otherwise give. The unwanted solutions are those which lead to causality conflicts arising from the presence of negated triggers *i.e.* causality conflicts which arise when we allow the *absence* of signals to be acted on.

We give an example of how the oracle mechanism is used to rule out certain solutions. We then relate those solutions back to the ones we get in our Z model for the same example in order to show one more aspect of the relationship between the two models.

4.1 Oracles and causality

The example we will use is a simple variant of example 2 (which was given in figure 3) where we take the negation off the trigger for the transition in V. (This also gives us an example of a case we have not yet explored, *i.e.* one in which both negated and un-negated triggers occur.)

The method that [9] uses to rule out certain solutions that violate some causality concerns is to first rename any negated trigger $\neg\ a$ as $\neg\ \tilde{a}$ and then conduct experiments to see what the reaction of the chart is when we make

[6] Note that the semantics in [14] differs: there feedback between levels of the hierarchy happens as the default

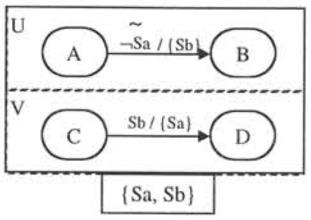

Fig. 5. Illustrating oracle use

assumptions about feed back involving these new signals, which are called *oracles*. So, the oracle signals represent signals that may be fed back during any of a number of instantaneous transitions and they were designed so that we can consider the reactions of the chart when signals are absent and when there are negated triggers in the chart. (We have seen examples of this in sections 3.1 and 3.2.)

The chart we are considering would, therefore, be re-written as in figure 5.

Now, if we want to see what its reactions are when no (external) input is given there are two experiments to perform: firstly we assume that Sa is in the set of signals generated by the chart in this step (the fixed-point for this step), *i.e.* an experiment with $\tilde{Sa}$ being in the fixed-point; secondly we assume that Sa is never output for any reason (and so does not appear in the fixed-point) which means we assume that $\neg\tilde{Sa}$ is in the fixed-point. The reader can see that the resulting fixed point of the first experiment is $\{\tilde{Sa}\}$ and for the second the fixed-point is $\{\neg\tilde{Sa}, Sb, Sa\}$.

The result of the first experiment is not allowed as a possible behaviour of the chart since when we assumed that Sa would appear in the fixed-point it never did. The result of the second experiment is not allowed as a solution either since a signal (Sa) that we assumed would not appear in the fixed-point did. So, neither experimental solution is allowed since they each break certain causality constraints that it seems reasonable to impose.

The first constraint requires that if a signal can appear (because transitions that produce it as output can take place) then it does: this is *self-fulfilment*. The second constraint requires that a signal does not appear as the output of any transition that takes place because of the non-appearance of that signal: this is *consistency*.

So, how is it that we disallow the same causally suspect solutions that the oracles mechanism does, but without using anything like that mechanism? The reason lies in the different ways the semantics are given. In our case, we rely on the usual semantics of set theory and give the semantics in a declarative fashion. In the case of [9] the semantics is given by a fixed-point construction, where each 'iteration' towards the fixed-point models the idea of one transition possibly leading to others once its output has been instantaneously fed back as

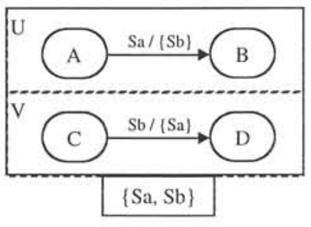

Fig. 6. μ-chart with no negated triggers

input, this process continuing until no further transitions are triggered, when the fixed-point is reached.

In more detail: when computing the fixed-point, the *absence* of a signal has to be recorded in order for the causality test to be meaningfully performed. Otherwise, the reason that at one stage the signal is not in the feedback set (the so-far-computed fixed-point) might be either because it simply hasn't appeared yet or because it must not appear. When we reach the fixed-point we need (for the causality test) to tell these two situations apart. Hence the oracles: $\neg\ \tilde{a}$ appearing means that $\tilde{a}$ must never appear; $\neg\ \tilde{a}$ not appearing entails no such condition. [7]

In our case, to get the same causality condition, we do not need the oracles. This is because our translation imposes global constraints on the possible solutions; constraints which are 'in force' permanently. That is, we declare what constraints a solution must satisfy and then rely on the semantics of Z to ensure that only acceptable sets are allowed as solutions, according to the usual meaning of the existential quantifier that appears outermost in our translation. Thus, if some transition is triggered only when a is absent, but some other transition adds a to the solution set then, since not both of these conditions can be satisfied, such a solution is ruled out. In the oracles case, since the solution set is essentially built iteratively (gradually computing the fixed-point) and since no constraints are passed between the iterations, the 'global' fact that a has been banned by the triggering of one transition must be recorded in order that the causality condition can be checked; simply leaving a out of the set will not do as it does not distinguish finely enough between two possibilities.

4.2 Extending the use of oracles

The semantics given in [14] differs, as we have mentioned a few times, from that in [9]. One way that this difference emerges is on the μ-chart given in figure 6.

In the absence of any external input, the semantics in [9] means that this chart does nothing. In contrast, the semantics in [14] means that chart is non-deterministic: it can either do nothing or both transitions can be triggered. Our

[7] This relationship between declarative and procedural semantics closely mirrors the two ways of giving semantics to Prolog programs.

translation process produces a description of the chart which agrees with the latter interpretation.

When we investigate the reasons for this we see that our constraints on solutions in the translation correspond to replacing not only negated triggers with negated oracle signals but also doing the same with *un-negated* trigger signals. Our conjecture (which we have shown holds for the examples in this paper) is that extending the use of oracles in this way would, in general, extend the solutions given by the semantics in [9] to those given by the semantics in [14] while still ruling out the solutions that offend certain sorts of causality scruples.

5 Uses

Having presented a translation of μ-charts into Z we want in this section to motivate this activity, which we do by giving several positive results of such an activity.

5.1 Alternative views of the solution

Giving two different views, via the different languages of μ-charts and Z, of a solution to a problem can often give us another chance to see if we have the right model of a given system. Different aspects of the model may come to light, given the different expression in the different language, which might lead us to see mistakes in the model.

Some properties of the system being modelled might be more readily apparent in the Z view than in the chart view (and *vice versa*, of course).

5.2 Proof instead of model checking

We have proof available for Z (via, for example, Z/EVES) which means that, via the translation, we can reason about the reactive model of our system. This forms an alternative to the sorts of model checking that staying with the μ-chart model would give.

The importance of proof (or deduction) in contrast to model checking has recently been explored by Pnueli [10], where he makes the following points:

- deduction is based on induction whereas model checking (which he characterises as *exploration*) is based on computing a set of reachable states
- deduction uses a more expressive language, leading to good ways of expressing, especially, parameterised systems
- deduction-based methods have better scalability

Whatever the pros and cons of each method (and whether we accept Pnueli's points or not), it is certainly the case that having both model-checking-based and deduction-based methods at our disposal to investigate systems is advantageous.

5.3 Alternative solutions to the problem

Following Weber's approach in [17], of specifying the problem from a Z perspective *and* a Statechart (for us μ-chart) perspective (and so having two *different* models of the problem, in contrast with the situation in section 5.1 where we had two expressions of the *same* solution), and then performing our translation on the μ-chart model, would allow us to directly compare the two models. We could also more directly check the consistency of the two models if they are both in Z.

6 Future work

6.1 Commands

In line with the work in [14] we will add commands to the transitions. The idea of this is to allow the updating of values associated with the chart. For example, if the chart takes a transition labelled (along with the usual trigger and output signal set) $a := a + 1$ then the value of a (which is accessible throughout the chart) is incremented.

The triggers can also be extended to allow Boolean combinations of expressions involving such values as a, as well as the input signals as currently. Both the extensions are straightforward in Z.

These extensions allow us to model such things as clocks (where the current time, for example, is represented by some value which is incremented by one sub-chart and used by others) or mechanisms which have to wait for certain external values to reach required levels before performing their tasks. Again, the external values (say the level in a tank of liquid) can be updated and used by sub-charts.

6.2 Correctness of the translation

In this paper we have endeavoured to illustrate the translation process, show some of its properties and mention some motivation for it. It still remains for us to show that the translation is correct by showing that, for each construct of the charts, its semantics is the same as the semantics we get going from that chart construct via the translation to Z and the Z semantics. Given that we have a compositional semantics for both formalisms this strategy seems to hold some hope, especially in the light of the transition semantics given in [14]. This is in a form that is closer to our Z form and which has already been shown to be equivalent to the original semantics in [14].

6.3 Refinement

[13] and [14] present a refinement calculus for μ-charts based on a relationship of inclusion, *i.e.* (roughly) S_2 refines S_1 is defined by $[\![S_2]\!] \subseteq [\![S_1]\!]$. This is also

the situation with our notion of refinement in [2], so investigating the relationships between these two notions of refinement would be interesting. In our work schemas are sets of bindings which represent state-and-signal combinations and in the μ-charts work they are sets of state-and-(input/output) signal tuples, so there are close similarities there.

6.4 A logic of μ-charts

We would like to develop a logic for charts, along the line of our logics for Z [3] [4] [5]. We expect to use the Z translation to at least suggest how this logic should look, if not use the translation to induce the chart logic from the Z logic.

6.5 Developing implementations

We would like to use the work on program development from Z [2] and the translation given here to allow program development from reactive specifications as given by charts.

Acknowledgements

Our thanks to Mark Saaltink, who has suggested to us an ingenious, though necessarily complicated, schema iterator which does the job we mentioned at the end of section 2.4 [12].

Thanks also the Peter Scholz for answering some of our basic questions about μ-charts.

Thanks finally to our colleagues on the ISuRF project [6]: Lindsay Groves, Ray Nickson and Mark Utting, and to the Foundation for Research, Science and Technology (FRST) for grant UOW805 and to the University of Waikato, which each provided funding which made this work possible.

References

1. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computing*, pages 231–274, 1987.
2. M. C. Henson and S. Reeves. Program development and specification refinement in the schema calculus. Submitted to ZB2000.
3. M. C. Henson and S. Reeves. Revising Z: I - logic and semantics. *Formal Aspects of Computing Journal*, 11(4):359–380, 1999.
4. M. C. Henson and S. Reeves. Revising Z: II - logical development. *Formal Aspects of Computing Journal*, 11(4):381–401, 1999.
5. M. C. Henson and S. Reeves. Investigating Z. *Journal of Logic and Computation*, 10(1):1–30, 2000.
6. The ISuRF web site is `http://www.cs.waikato.ac.nz/Research/fm/isurf.html`.
7. J. Jacky. *The Way of Z: Practical programming with formal methods.* Cambridge University Press, 1997.
8. The ORA web site is `http://www.ora.on.ca`.

9. J. Philipps and P. Scholz. Compositional specification of embedded systems with statecharts. In M. Bidoit and M. Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development*, number 1214 in LNCS, pages 637–651. Springer-Verlag, 1997.
10. A. Pnueli. Deduction is Forever. Unpublished, invited talk at Formal Methods '99, Toulouse, September 1999.
11. M. Saaltink. The Z/EVES system. In J. Bowen, M. Hinchey, and D. Till, editors, *Proc. 10th Int. Conf. on the Z Formal Method (ZUM)*, volume 1212 of *Lecture Notes in Computer Science*, pages 72–88. Springer-Verlag, Berlin, April 1997.
12. M. Saaltink, March 2000. Private communication.
13. P. Scholz. A refinement calculus for statecharts. In E. Estesiano, editor, *Fundamental approaches to software engineering: First International Conference, FASE'98*, volume 1382 of *Lecture Notes in Computer Science*, pages 285–301. Springer-Verlag, Berlin, 1998.
14. P. Scholz. *Design of Reactive Systems and their Distributed Implementation with Statecharts.* PhD thesis, Institut für Informatik, Technische Universität München, August 1998. TUM-I9821.
15. J. M. Spivey. *The Z notation: A reference manual.* Prentice Hall, 2nd. edition, 1992.
16. I. Toyn, editor. *Z Notation: Final committee draft, CD 13568.2.* Z Standards Panel, 1999. `ftp://ftp.york.ac.uk/hise_reports/cadiz/ZSTAN/fcd.ps.gz`.
17. M. Weber. Combining statecharts and Z for the design of safety-critical control systems. In M.-C. Gaudel and J. Woodcock, editors, *FME '96, industrial benefit and advances in formal methods: Third International Symposium of Formal Methods Europe*, volume 1051 of *Lecture Notes in Computer Science*, pages 307–326, 1996.
18. J. Woodcock and J. Davies. *Using Z: Specification, Refinement and Proof.* Prentice Hall, 1996.