Logic and Refinement for Charts

Greg Reeve **Steve Reeves**

Department of Computer Science,
University of Waikato,
New Zealand,
Email: {gregr,stever}@cs.waikato.ac.nz

Abstract

We introduce a logic for reasoning about and constructing refinements for μ-Charts, a rational simplification and reconstruction of Statecharts. The method of derivation of the logic is that a semantics for the language is constructed in Z and the existing logic and refinement calculus of Z is then used to induce the logic and refinement calculus of μ-Charts, proceeding by a series of definitions and conservative extensions and hence generating a sound logic for μ-Charts, given that the soundness of the Z logic has already been established.

Keywords: Statecharts, reactive systems, Z, $\mathcal{Z}_C$, logic, refinement

1 Introduction

The specification language μ-Charts is a rational simplification and reconstruction of Statecharts (Harel 1987). As such, it can be considered to define the core of the many Statechart-like languages: a family of visual languages that are used for designing reactive systems. It is simpler than the original Statecharts, the simplification being achieved by omitting some of the more complicated and reportedly less-used constructs. It is designed to have a more comprehensible semantics, without losing expressiveness. One important contribution of this work, then, is a semantics and logic for the core of Statecharts, presented independently of any particular tool or other "operational" embodiment of semantics and logic.

In the past a formal semantics has been given to μ-Charts using both a process algebraic, traces approach and denotationally using automata by Scholz (1998), along with a logical treatment (Reeve & Reeves 2000) using the specification language Z (Spivey 1989),(13568 2002). While different aspects, and versions, of μ-Charts have been published (Philipps & Scholz 1997*a*, Philipps & Scholz 1997*b*, Scholz 1998), the definitive account (prior to the development of the Z-based logic) was published by Scholz (1998). Characteristic features of μ-Charts are that transitions are *instantaneous* (and hence input and output signals appear simultaneously); communication using selected signals (feedback) between charts is *local* (pairwise) rather than global and is defined explicitly; and that charts may *nondeterministically* choose between transitions.

The language semantics assumes a chaotic-outside-of-defined-behaviour interpretation. The language is also distinguished from process algebras where the natural interpretation is often one where transitions are triggered by the presence of required signals. μ-Charts also allows transitions to be triggered by the absence of designated signals. This is facilitated by imagining there is a global clock whose tick causes all transitions leaving the state of a chart to evaluate their guards.

The aim of this paper is to present a partial relations-based logic for μ-Charts and and then to show how a refinement theory can be lifted from Z to charts.

In this paper we view μ-Charts as a language for specification, especially since we allow nondeterminism. In a subsequent paper we will show how implementations can be built: the method will be to use the language and notions of refinement presented here to move our specification towards implementation by gradually reducing nondeterminism and adjusting the interfaces as required, and then using the program development work by, *e.g.*, Henson and Reeves (2003) and the more recently by Henson *et al.* (2004) to arrive at an implementation.

The derivation of the logic rules is somewhat simplified due to space constraints: the interested reader can consult work by Reeve (2005) for a more detailed account. For similar reasons the formal semantics given covers just one of the three language constructs.

We do not give examples of reasoning about any particular chart: how to use a logic to reason is, we assume, second nature to our audience and is straightforward. Rather, we use the logic for the far more ambitious and fundamental goal of deriving refinement rules for charts.

Section 2 presents an introduction to the formal treatment of μ-Chart's semantics. This includes describing one of the language operators, giving the general method of deriving a Z model for that operator and the derivation of natural deduction-style logic rules using the Z logic. We divide this section into two: the first part shows how the semantics for charts is given in Z; the second part shows how the Z semantics is given a meaning and then how rules for charts can be derived.

Section 3 shows how we use the existing and well-investigated refinement notion of Z to derive a refinement notion for charts. This is concluded with a discussion of what this refinement notion is in terms of the more traditional process algebraic, trace description of chart behaviour. In Section 4 we consider monotonicity of refinement in general, and in Section 5 we show how the logic we have developed can be used to investigate and prove monotonicity properties of μ-Charts. Finally, we present some conclusions in Section 6.

2 The μ-Charts Logic

This section provides an introduction to μ-Charts via the definition its of semantics in Z. The logic and semantics of Z itself is then used to induce a logic and set-theoretic semantics for μ-Charts.

This process is divided into two natural phases: we first use definitions to express the semantics of μ-Charts in Z. These definitions define what we call the *transition model*, which is essentially a function which maps expressions from μ-Charts to Z, and which we denote by $[\![.]\!]_{Z_t}$. We then use the standard mapping from Z into the underlying, core language $\mathcal{Z}_\mathcal{C}$ (Henson & Reeves 2000). This mapping is denoted by $[\![.]\!]_{\mathcal{Z}_C}$. The composition of these two semantic mappings then gives us the semantics of μ-Charts in $\mathcal{Z}_\mathcal{C}$. We then use the logic of $\mathcal{Z}_\mathcal{C}$ together with the definitions that go to make up $[\![.]\!]_{Z_t}$ to induce a logic for μ-Charts. Since this logic has been constructed from the sound logic for $\mathcal{Z}_\mathcal{C}$ by a series of conservative extensions (via definitions), we know that the logic for μ-Charts is sound too.

A μ-chart is either atomic or a combination of charts using language operators. An atomic chart is essentially a finite-state automaton where a transition in the chart is labelled with a pair, *guard*/*action*. That the transition is taken (or triggered) is conditional on the guard being satisfied by the current input signals and leads to the action happening, meaning that signals are output as required by that action. Each chart has an input interface designating signals that can trigger a transition and an output interface designating signals that the chart can output. A fundamental assumption is that time passes in the control states of a chart, but the transitions between these states occur *instantaneously*. A chart reaction is therefore characterised by the input of a set of signals from the environment and the instantaneous output of a set of signals to the environment. This reaction is called a step or a tick of the clock, but note that this does not mean that the intervals between reactions must be equal.

A chart C has an input interface in_C that typically includes all of the signals that appear in its transition guards and an output interface out_C that typically includes all of the signals that appear in its transition actions.

In order to define large reactive systems, the language has three structuring mechanisms: parallel composition, hierarchic decomposition and input/output interface definition. In a parallel composition, each component chart reacts *synchronously* on a global clock. A feedback mechanism between pairs of charts makes output signals instantaneously available as input signals, which allows component charts to communicate *asynchronously* on signals. A composition chart $C = C_1 \mid \Psi \mid C_2$, which composes charts C_1 and C_2 with feedback on signals in the set Ψ, has an input interface $in_C = in_{C_1} \cup in_{C_2}$ and an output interface $out_C = out_{C_1} \cup out_{C_2}$.

A further structuring mechanism means some of the states of a chart need not be atomic, but rather one chart may be embedded in another ("inside" one of its states) as a sub-chart, using hierarchic decomposition. Finally, the definition of the assumed context of a chart via the explicit definition of the input and output interfaces of a chart (of arbitrary structure) allows signal hiding.

The full definition of the language semantics, via the logic, treats each of these language operators separately. In this paper we will concentrate on just the definition of atomic charts and the parallel composition operator.

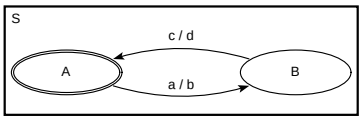

Figure 1: A simple atomic μ-chart

2.1 The transition model

The transition model essentially relates the current configuration of a chart and input to a new configuration and the resulting output. This relation describes every possible step that a chart can take. In contrast, a typical way to give a semantics for such languages is to use sets of observable input/output traces, for example the trace semantics given by Scholz (1998). This abstracts on the control states by defining just its reactions in an assumed context. The link between the logical semantics described here and a *trace* semantics can be constructed by considering a chart making one step after another and recording just the input and resulting output. The resulting trace semantics is considered fully by Reeve (2005).

2.1.1 Atomic charts

An atomic chart has the general textual form (Name, State set, Start state, Feedback signals, transition function). Consider the chart $(S, \{A, B\}, A, \{\}, \delta)$ (where δ is the appropriate transition description) pictured in Figure 1. [1]

Informally the behaviour that this chart captures can be described as: the chart starts in state A; if it is in state A and the signal a is input then signal b is output and the chart changes to state B; and similarly if it is in state B and c is input then d is output and the new state is A.

The essence of the transition model for an atomic μ-chart is the description of each of its transitions using a separate Z operation schema. These operation schemas (one for each transition in the chart) are combined using Z schema disjunction to give one schema that describes the transition behaviour of the chart. The Z state of the model has an observation that indicates the current configuration of the chart. The operation schemas describe each transition, that is, how and when that configuration changes.

For an atomic chart $(C, \Sigma, \sigma_0, \Psi, \delta)$ we introduce Z axiomatic definitions (Figure 2) that model the set of possible chart states, the input interface and the output interface. The sets μ_{State} and $\mu Signal$ contain all allowable state and signal symbols.

The state schema $Chart_C$ records the current state of the chart C using observation c_C. The initial state of the chart is modelled by the schema $Init_C$.

A separate state schema called $C\sigma$ is given for each state in the chart $\sigma \in \Sigma$.

Next we give an operation schema for each chart transition. That is, for each transition $(S_f, S_t, guard/action) \in \delta$ we define an operation schema named $\delta_{S_f S_t}$.

We can see from this definition that each binding in the set has five observations. The meanings of these are:

- c_S—the state of the chart before the transition happens, in this case the state A

[1] We use the piece of text which is the name of the chart to refer to both the chart itself and its name, allowing the context to indicate which we mean. The fact that the name stands for itself as a piece of text is used in the syntactic process of defining the Z that a chart has as its semantics, as we shall see.

$states_C : \mathbb{P}\, \mu_{State}$
$in_C : \mathbb{P}\, \mu Signal$
$out_C : \mathbb{P}\, \mu Signal$
$\Psi : \mathbb{P}\, \mu Signal$

$Chart_C \ == [c_C : states_C]$

Init$_C$
$Chart_C$

$c_C = \sigma_0$

$C\sigma \ == [Chart_C \mid c_C = \sigma]$

$\delta_{S_f S_t}$
CS_f
CS'_t
$i_C : \mathbb{P}\, in_C$
$act : \mathbb{P}\, \mu_{State}$
$o'_C : \mathbb{P}\, out_C$

$C \in act$
$\rho(guard)$
$o'_C = action$

Figure 2: The Z semantics, the transition model, for an atomic chart

- i_S—the set of input signals which are offered by the environment that are in the input interface of the chart
- act—a set that denotes all currently active charts
- c'_S—the state of the chart after the transition happens, in this case B
- o'_S—the output generated by the chart, in this case the set containing the signal b

Note that part of the definition of atomic charts—the observation act—is part of the mechanism that allows for the definition of the hierarchic decomposition operator. As we will not be looking at the details of this operator in this paper, it is sufficient to understand that a chart can be active or inactive, and transitions happen only when an atomic chart is active, which is recorded by having its name contained in act. Hence the predicate $C \in act$ is part of the precondition of the operation schema $\delta_{S_f S_t}$.

The predicate $\rho(guard)$, introduced in schema $\delta_{S_f S_t}$, stands for the Z predicate that models the syntactic guard of a chart transition. If we consider a transition's guard in general as a (possibly empty) list of signal expressions, separated by the conjunction symbol &, then each of the elements in the list can be classified into two categories: either a positive signal expression—simply the name of a signal; or a negative signal expression—the signal name is prefixed with a minus sign. A positive signal expression, say sig where $sig \in in_C$, is denoted by the Z expression $sig \in i_C \cup (o'_C \cap \Psi)$. A negative signal expression, say $-sig$, is denoted by the Z expression $sig \notin i_C \cup (o'_C \cap \Psi)$. The syntactic construction process denoted by ρ determines the appropriate predicate for each signal expression and connects them together using the Z logical conjunction operator $\wedge$. If the list is empty the predicate (produced by the process ρ) would be $true$. So, for the transition labelled a/b in chart S in Fig.1 the predicate produced would be $a \in i_c \cup (o'_c \cap \Psi)$ since the guard of this transition is just a.

This general scheme for giving the Z for a transition defines the semantic function $[\![.]\!]_{Z_t}$. For an arbitrary transition $(S_f, S_t, guard/action)$ we have,

$$[\![(S_f, S_t, guard/action)]\!]_{Z_t} =_{def} \delta_{S_f S_t}$$

The schema $\delta_{S_f S_t}$ provides the Z semantics for the transition.

Along with the schemas for each transition we also need a single schema that models the behaviour of the chart when it is inactive (Figure 3). Again this Z is part of the general transition necessary to model the entire language, in particular, including the decomposition operator. Here it is enough to realise that an inactive chart plays no part in output. For the general atomic chart $(C, \Sigma, \sigma_0, \Psi, \delta)$, we name the inactive schema $Inactive_C$.

Now, the entire transition model for an atomic chart is given by Definition 2.1[2]

Definition 2.1

$$[\![(C, \Sigma, \sigma_0, \Psi, \delta)]\!]_{Z_t} =_{def} (\bigvee \{[\![t]\!]_{Z_t} \mid t \in \delta\}) \vee Inactive_C$$

in a context containing the axiomatic definitions and $Chart_C$, $Init_C$ and $C\sigma$.

The complete transition model for chart S from Figure 1 is defined as the disjunction of each of the individual transition schemas, *i.e.* $\delta_S == \delta_{AB} \vee \delta_{BA} \vee Inactive_S$, and two of these schemas are given as examples in Figure 4.

2.1.2 The composition operator

The composition operator allows us to take two μ-charts C_1 and C_2 and join them together to form a new, more complex chart $C_1 \mid \Psi \mid C_2$ where Ψ is a set of signals on which C_1 and C_2 can communicate. As mentioned, the charts run separately but synchronously, *i.e.* in lock step with one another. Their only medium of communication is asynchronous via the multicast of those signals in the set Ψ. The communication is asynchronous in that output is always enabled—a chart can always broadcast signals. However, there is no guarantee that the other chart in the composition is listening, that is, ready to react on the signals broadcast. Signals persist only during one step of the chart.

The transition model $[\![C]\!]_{Z_t}$ for the composed chart $C = C_1 \mid \Psi \mid C_2$ contains a similar set of Z definitions and schemas (Figure 5) as that for an atomic chart. Here it is assumed that any entity subscripted with C_1 comes from the transition model of the chart C_1 and similarly for C_2.

2.1.3 Partial relations semantics

The step semantics of a chart is no more than the transition model of the entire μ-chart specification

[2] We use the notation $\bigvee X$ to denote the schema disjunction of all the schemas in the set X.

Inactive$_C$
$\Xi Chart_C$
$i_C : \mathbb{P}\, in_C$
$act : \mathbb{P}\, \mu_{State}$
$o'_C : \mathbb{P}\, out_C$

$C \notin act$
$o'_C = \{\}$

Figure 3: Z semantics for Inactive

δ_{AB}

SA
SB'
$i_S : \mathbb{P}\, in_S$
$act : \mathbb{P}\, \mu_{State}$
$o'_S : \mathbb{P}\, out_S$

$S \in act$
$a \in i_S \cup (o'_S \cap \{\})$
$o'_S = \{b\}$

$Inactives_S$

$\Xi Chart_S$
$i_S : \mathbb{P}\, in_S$
$act : \mathbb{P}\, \mu_{State}$
$o'_S : \mathbb{P}\, out_S$

$S \notin act$
$o'_S = \{\}$

Figure 4: Z for the chart in Figure 1

with the active state machinery hidden. Given an arbitrary μ-chart called C, the step behaviour of C is defined by another schema which, by convention, we call $CSys$ (Figure 6).

The schema $CSys$ (right) hides the active state observation and specifies that the top-most chart of any hierarchical structure is active.

So, $[\![]\!]_{Z_t}$ for an arbitrary chart C generates various pieces of Z, depending on the structure of C, defining the transition model. We then have to give a meaning to the Z in order to generate logical rules, which we now do.

2.2 The $\mathcal{Z}_\mathcal{C}$ model

From the transition model of a chart as given above we move on to give a logic for charts by modelling the transition model in $\mathcal{Z}_\mathcal{C}$ hence deriving introduction and elimination rules that allow us to prove properties about a chart's transition model and hence about a chart.

2.2.1 Atomic charts

Given the general method for constructing the Z semantics of a chart (*i.e.* the transition model), we can describe the meaning of the chart by describing the meaning of the Z. To do so we rely on a reasonable level of familiarity with the meta-language used in the presentation of the kernel logic $\mathcal{Z}_C$ in Henson and Reeves (2000). Briefly, we: use the binding concatenation operator $\star$; restricted membership $\dot{\in}$; restricted equality $\doteq$; type meta-variables for example T; αT as shorthand for all observations of the schema type T; superscript type meta-variables to denote the types of bindings and schemas; and the type union operator $\curlyvee$. For brevity, we suppress mention of types (indicated by superscripts on terms) in all cases where this is possible and rely on the unique of types that $\mathcal{Z}_\mathcal{C}$ enjoys to assure ourselves of the well-formedness (which includes well-typedness) of our terms.

Returning to the example chart of Figure 1, again, the Z meaning of the transition from configuration A to B is given by the schema δ_{AB}, whose meaning in turn is given in the theory $\mathcal{Z}_C$ as a set of bindings as follows:

$$[\![\delta_{AB}]\!]_{\mathcal{Z}_C} =_{def}$$
$$\{\!|\ c_S \Rrightarrow A,\ i_S \Rrightarrow i,\ act \Rrightarrow active,\ c'_S \Rrightarrow B,\ o'_S \Rrightarrow \{b\}\ |\!\} \mid$$
$$i \subseteq in_S \wedge active \subseteq \mu_{State} \bullet S \in active \wedge a \in i\}$$

$states_C : \mathbb{P}\, \mu_{State}$
$in_C : \mathbb{P}\, \mu Signal$
$out_C : \mathbb{P}\, \mu Signal$
$\Psi : \mathbb{P}\, \mu Signal$

$states_C =$
$\quad states_{C_1} \cup states_{C_2}$
$in_C = in_{C_1} \cup in_{C_2}$
$out_C = out_{C_1} \cup out_{C_2}$

$Chart_C$

$Chart_{C_1}$
$Chart_{C_2}$

$Init_C$

$Init_{C_1}$
$Init_{C_2}$

δ_C

$\Delta Chart_C$
$i_C : \mathbb{P}\, in_C$
$act : \mathbb{P}\, \mu_{State}$
$o'_C : \mathbb{P}\, out_C$

$C_1 \in act \Leftrightarrow C \in act$
$C_2 \in act \Leftrightarrow C \in act$
$\exists\, i_{C_1}, i_{C_2}, {o_{C_1}}', {o_{C_2}}' : \mathbb{P}\, \mu Signal \bullet$
$\quad i_{C_1} = (i_C \cup (o'_C \cap \Psi)) \cap in_{C_1} \wedge$
$\quad i_{C_2} = (i_C \cup (o'_C \cap \Psi)) \cap in_{C_2} \wedge$
$\quad o'_C = {o_{C_1}}' \cup {o_{C_2}}' \wedge \delta_{C_1} \wedge \delta_{C_2}$

Figure 5: Semantics for composition

$CSys$

$\Delta Chart_C$
$i_C : \mathbb{P}\, in_C$
$o'_C : \mathbb{P}\, out_C$

$\exists\, act : \mathbb{P}\, \mu_{State} \bullet C \in act \wedge \delta_C$

Figure 6: Top-level semantics

The second schema $Inactives_S$ describes the behaviour of the chart when it is inactive, and its meaning is given as:

$$[\![Inactives_S]\!]_{\mathcal{Z}_C} =_{def}$$
$$\{\!|\ c_S \Rrightarrow s,\ i_S \Rrightarrow i,\ act \Rrightarrow active,\ c'_S \Rrightarrow s,\ o'_S \Rrightarrow \{\}\ |\!\} \mid$$
$$s \in \{A, B\} \wedge active \subseteq \mu_{State} \wedge i \subseteq in_S \bullet S \notin active\}$$

By definition of schema disjunction the set $[\![\delta_S]\!]_{\mathcal{Z}_C}$, which gives in $\mathcal{Z}_\mathcal{C}$ the meaning of the transition model for the chart, contains all of the bindings from the sets $[\![\delta_{AB}]\!]_{\mathcal{Z}_C}$, $[\![\delta_{BA}]\!]_{\mathcal{Z}_C}$ and $[\![Inactives_S]\!]_{\mathcal{Z}_C}$.

In order to make the logical rules we are working towards slightly more readable, we define the predicate *Trans* which captures what it means for a binding of the transition model to characterise a transition of the chart. Given an arbitrary transition of the form $t = (S_f, S_t, guard/action)$, from the chart C, we have,

$$Trans\ t\ z^T =_{def}$$
$$z.c_C = t.S_f \wedge \rho(t.guard)[\alpha T / z.\alpha T]$$
$$\wedge\ z.c'_C = t.S_t \wedge z.o'_C = t.action$$

The terms $t.S_f$ *etc.* are assumed to be defined in the obvious way such that $t.S_f$ gives the "from state" of a transition, $t.guard$ gives the guard component of a transition, $t.S_t$ gives the "to state" and $t.action$

gives the action component. ρ is as defined above and constructs a predicate from a guard.

Now we give the formal definition of the transition model for charts directly in terms of the meaning of the Z model.

Definition 2.2 For the arbitrary atomic chart $(C, \Sigma, \sigma_0, \Psi, \delta)$, we have:

$$\begin{aligned}&[\![\delta_C]\!]_{\mathcal{Z}_C} =_{def}\\ &\quad \{z \mid C \notin z.act \land z \mathrel{\dot\in} \Xi Chart_C \land z.o'_C = \{\} \ \lor\\ &\qquad C \in z.act \land \exists\, t \in \delta \bullet Trans\ t\ z\}\end{aligned}$$

From this definition we finally derive introduction and elimination rules.

Proposition 2.1 Given the atomic chart $(C, \Sigma, \sigma_0, \Psi, \delta)$, where for arbitrary binding z we have:

$$\frac{z \mathrel{\dot\in} \delta_C \quad actv\ C\ z \quad t \in \delta, Trans\ t\ z \vdash P}{P}\ (Z_t{}^-)$$

$$\frac{actv\ C\ z \quad t \in \delta \quad Trans\ t\ z}{z \mathrel{\dot\in} \delta_C}\ (Z_t{}^+)$$

assuming the usual conditions (due to the elimination of an existential quantifier) for t and P, and where, for an atomic chart C, the predicates $actv\ C\ z$ and $inactv\ C\ z$ are defined as follows:

$$actv\ C\ z =_{def} C \in z.act$$
$$inactv\ C\ z =_{def} \neg\ actv\ C\ z$$

2.2.2 Composition

We give the definition of the $\mathcal{Z}_\mathcal{C}$ model for composed charts in terms of the meaning of the transition model in:

Definition 2.3 Given an arbitrary composition $C = C_1 \mid \Psi \mid C_2$ we have,

$$\begin{aligned}&[\![\delta_C]\!]_{\mathcal{Z}_C} =_{def}\\ &\{z \mid C_1 \in z.act \Leftrightarrow C \in z.act \land\\ &\qquad C_2 \in z.act \Leftrightarrow C \in z.act \land\\ &\qquad \exists\, o_1, o_2 \bullet z.o'_C = o_1 \cup o_2 \land\\ &z \star (\!|\ i_{C_1} \Rrightarrow (z.i_C \cup z.o'_C \cap \Psi) \cap in_{C_1}, o_{C_1}{}' \Rrightarrow o_1\ |\!) \mathrel{\dot\in} \delta_{C_1} \land\\ &z \star (\!|\ i_{C_2} \Rrightarrow (z.i_C \cup z.o'_C \cap \Psi) \cap in_{C_2}, o_{C_2}{}' \Rrightarrow o_2\ |\!) \mathrel{\dot\in} \delta_{C_2}\}\end{aligned}$$

The introduction and elimination rules for composed charts shown in Figure 7 are derived from this definition.

2.2.3 The step semantics

Now Definition 2.4 defines the step semantics for a chart.

Definition 2.4 For arbitrary chart C with the associated Z description $CSys$,

$$\begin{aligned}&[\![CSys]\!]_{\mathcal{Z}_C} =_{def}\\ &\qquad \{z \mid \exists\, z_1 \bullet \ z_1 \doteq z \land actv\ C\ z_1 \land z_1 \in \delta_C\}\end{aligned}$$

From this definition we derive introduction and elimination rules given in Figure 8.

We often refer to the step semantics as the *partial relations semantics*. This is because the meaning of the schema $CSys$ can be considered as a relation that maps the "before" configuration of a chart and input to its "after" configuration and output. This relation is often partial because a μ-Chart specification describes the reaction to some input events and not others.

3 The μ-Charts Refinement Calculus

In Derrick and Boiten (2001) and Woodcock and Davies (1996), a framework for considering Z specifications and Z refinement in terms of abstract data types (ADTs) is introduced. The idea is to map a "standard" Z specification, *i.e.* state schema, initialisation schema and operation schemas, into a relational ADT setting. Broadly a relational ADT is a tuple of the form (X, xi, xf, Ops) such that: X is a state space; xi is an initialisation relation; xf is a corresponding finalisation relation; and Ops is an indexed set of relational operations. The initialisation and finalisation relations map a global observable state into the ADT's private state and vice versa. A program of an ADT is defined as a particular sequence of the indexed operations upon a data type, preceded by initialisation and ended by finalisation. This mapping is used to derive a data refinement theory for Z specifications from the existing refinement notion for partial relations ADTs.

Given that the partial relation semantics for μ-Charts is defined via Z we can fit charts into the same framework. Recall from Section 2 that the Z model of a chart constitutes a state space, an initialisation schema and one operation schema, this operation schema being the description of every step that the chart can take. If we view the Z model of a chart in the ADT framework we can say that any program allowed by the chart is an example of composing the step operation together with itself again and again. Of course, what we are really interested in is the sequences of inputs and outputs that result from such programs. If we imagine running this program over all possible input sequences and recording the resulting output sequences then we have exactly the trace semantics of the chart. Because we are modelling reactive systems we choose to consider the traces over infinite sequences of input and output. Therefore, we need to imagine composing the step operation with itself indefinitely.

In the following we show how we can generalise the Z/ADT results to charts. In particular we show that the ADT view of a chart can be considered as giving the trace semantics of that chart. Then we derive a notion of partial relations refinement for charts based on an existing notion of partial relation refinement for Z.

We diverge from what may be considered the usual way to give a refinement notion in a reactive systems setting, that is, using the *behavioural approach* (as Derrick and Boiten (2001) calls it) for completing partial relations, and assume chaotic behaviour outside of the preconditions of partial relations (which is the more common-to-Z notion too, as it happens). The resulting notion of refinement is particularly interesting because it allows us to refine both the behaviour of a reactive system and the context, via the system's interface, in which we assume that reactive system will reside. This notion of refining both behaviour and context is not new to this work. The notion of "chaotic refinement" for *specifications* of reactive systems was suggested in the original definition of the language μ-Charts (Scholz 1998). Of course, since a behavioural interpretation is available in the Z framework, we could also derive more traditional rules for a reactive system if we wished to, as we typically would when we move from a specification to an implementation.

3.1 Charts and ADTs

The usual account of ADT refinement makes the simplifying assumption that the types of inputs and outputs associated with the abstract and concrete pro-

Proposition 2.2 Given $C = C_1 \mid \Psi \mid C_2$, for the binding z and arbitrary sets o_3 and o_4, we have:

$$\frac{\begin{array}{l} z.o'_C = o_1 \cup o_2, \\ z \star \{\!|\, i_{C_1} \Rrightarrow (z.i_C \cup z.o'_C \cap \Psi) \cap in_{C_1}, o_{C_1}{}' \Rrightarrow o_1 \,|\!\} \mathrel{\dot\in} \delta_{C_1}, \\ z \star \{\!|\, i_{C_2} \Rrightarrow (z.i_C \cup z.o'_C \cap \Psi) \cap in_{C_2}, o_{C_2}{}' \Rrightarrow o_2 \,|\!\} \mathrel{\dot\in} \delta_{C_2}, \\ actv\ C\ z \lor inactv\ C\ z \qquad\qquad \vdash Q \end{array}}{Q} \; (|\_|^-)$$

where the usual conditions, due to the elimination of existential quantifiers, hold between o_1, o_2 and Q. The predicates $actv\ C\ z$ and $inactv\ C\ z$ are defined for the composed chart $C = C_1 \mid \Psi \mid C_2$ as:

$$actv\ C\ z =_{def} actv\ C_1\ z \land actv\ C_2\ z \land C \in z.act$$
$$inactv\ C\ z =_{def} inactv\ C_1\ z \land inactv\ C_2\ z \land C \notin z.act$$

Figure 7: Rules for composition

Proposition 2.3 For arbitrary chart C and binding z we have,

$$\frac{z \mathrel{\dot\in} CSys \quad z \star z_a \mathrel{\dot\in} \delta_C,\ actv\ C\ z_a \vdash Q}{Q}\ (Z_s^-) \qquad \frac{\exists\, y \bullet z \star y \mathrel{\dot\in} \delta_C \land actv\ C\ y}{z \mathrel{\dot\in} CSys}\ (Z_s^+)$$

where the usual conditions (due to the elimination of an existential quantifier) hold between z_a and Q.

Figure 8: Rules for a step of the system

grams (the two n-operation programs P_a^n and P_c^n) are the same. For our purposes this simplifying assumption is too strict. We allow, under what turn out to be rather strong provisos, the input and output interface of a chart to be changed via refinement. Weakening the assumption of equivalent typed input and output for both abstract and concrete programs is achieved using the respective initialisation and finalisation relations in conjunction with the notion of an observable context for charts. We assume that refinement is a judgement made in the broadest input/output context.

We use the respective initialisation and finalisation relations to make the ADT's global state model the appropriate input/output context. The observable behaviour of the ADT (*i.e.* the global state) is given by the input and output sequences that range over the signals of both charts. The initialisation relation maps the global input sequences into appropriate input sequences for the respective charts. Similarly, the finalisation relation maps the outputs from the respective charts into the global output sequences.

From this we make a link between the semantics for a chart C given by embedding the chart in an ADT framework, denoted by $[\![C]\!]_d^{r\omega}$, and an infinite trace semantic definition of charts, *i.e.* $[\![C]\!]_x^{\omega}$, as follows:[3]

Definition 3.1 For arbitrary chart C and sequences $i \in \mathcal{I}^\omega$ and $o \in \mathcal{O}^\omega$,[4]

$$(i, o) \in [\![C]\!]_d^{r\omega} \;=_{def}\; (i_{\rhd (in_C)}, o_{\rhd (out_C)}) \in [\![C]\!]_x^{\omega}$$

Now we follow the well-known relational ADT approach (for example see Woodcock and Davies (1996) and Derrick and Boiten (2001)) to derive refinement rules for charts in terms of their partial relations semantics. Note that Definition 3.1 allows us to relate the resulting refinement notion back into the infinite trace style semantics for charts as given in Scholz (1998).

[3] We assume that $\mathcal{I}_c^*$ denotes the set of finite sequences ranging over the type $\mathbb{P}\, Input$. Similarly, $\mathcal{O}_c^*$ denotes all finite sequences over the type $\mathbb{P}\, Output$. The infinite sequences $\mathcal{I}_c^\omega$ and $\mathcal{O}_c^\omega$ are similarly defined.

[4] The notation $i_{\rhd (in_C)}$ denotes the pointwise restriction of the elements in the sequence i to the elements in the set in_C and similarly for out_C.

3.2 Simulation and Corresponding States

Before we derive the refinement rules we briefly introduce and discuss the concept of simulation. When comparing two charts based on input and output traces, that is, checking for or calculating trace refinements, the state information of the charts is already abstracted away. This is not the case, however, when working with the partial relations semantics. We need a way of relating the states of one chart with those of another. This is exactly the task of simulations, sometimes also known as *retrieve relations*, *abstraction relations*, or *coupling invariants* (Derrick & Boiten 2001). Something as simple as changing the names of the states from the abstract chart to the concrete requires that we have a simulation relation that maps the abstract state names into the new concrete state names.

In the standard ADT treatment, a simulation relation encodes the relationship between the states of the abstract specification and the states of the concrete specification. We usually think of the simulation R as completing a series of commuting squares. This allows us to prove the necessary refinement properties for each of the associated operations (in our case there is only one) and use an inductive argument to show that the refinement holds when we compose (in an appropriate order) several operations together into programs. We refer the reader to Derrick and Boiten (2001) for a detailed description of the concepts of data type refinement.

As discussed in Section 3.1, the initialisation and finalisation relations are used to modify the observable input and output sequences to allow refinement to change the context (*i.e.* input/output interfaces) of a chart. Reflecting this, we split the definition of the simulation relation into two separate parts. The first part is the simulation between configurations of the respective abstract and concrete charts. We will refer to this part of the simulation as the *corresponding relation* or $Corr_C^A$ for a simulation between charts A and C.

The Z schema $Corr_C^A$ (Figure 9) gives the general scheme for the corresponding relation. The predicate P defines the simulation relationship between the states of the respective charts A and C, and will, of course, depend on precisely what charts A and C are.

The second part of the simulation relation allows refinements that change the input/output interfaces

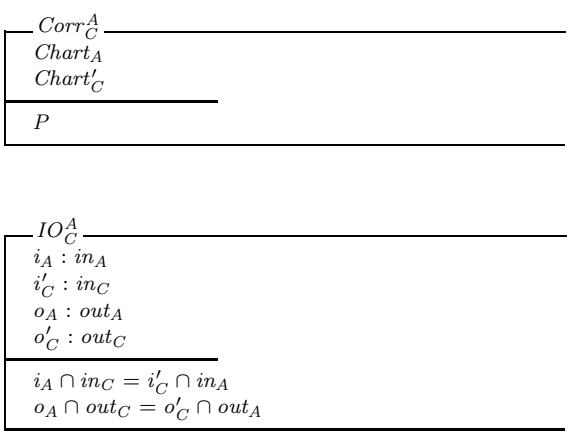

Figure 9: Semantics for simulation relation

of a chart. For arbitrary input interfaces in_A and in_C, and output interfaces out_A and out_C, the schema IO_C^A is constructed so that $[\![IO_C^A]\!]_{\mathcal{Z}_C}$ represents a relation between the inputs and outputs from the abstract chart to the inputs and outputs of the concrete chart. Importantly, when this relation is combined with the corresponding relation we get a schema representing the simulation relation between charts A and C that has type $\mathbb{P}(T_A^{io} \upsilon T_C^{io\prime})$ as follows:

Definition 3.2 For charts A and C we have,

$$R_C^A =_{def} Corr_C^A \wedge IO_C^A$$

where $[\![R_C^A]\!]_{\mathcal{Z}_C}$ has type $\mathbb{P}(T_A^{io} \upsilon T_C^{io\prime})$.

Significantly, when using the refinement theory presented the developer need only define the relationship between states of the "refining" and "refined" charts. The input/output relationship or interface refinement is always constrained by the general relationship identified by the schema IO.

3.3 Partial Relation Refinement

Now we can derive partial relations refinement rules for charts. The derivation of the different sets of rules closely follows a similar treatment by Derrick and Boiten (2001).

We embed the Z-based chart ADT presented so far into a relational data type as follows.

Definition 3.3 For an arbitrary chart C and all sequences si and so, the Z ADT semantics $(Chart_C, Init_C, \{CSys\})$ is embedded in the relational data type $(CState, CInit, \{CStep\}, CFin)$, such that,

$$CState =_{def} \mathcal{I}_C^{\omega} \times \mathcal{O}_C^* \times U_C$$
$$CInit =_{def} \{(si \mapsto (si_{\rhd(in_C)}, \langle\rangle, z)) \mid z \in Init_C\}$$
$$CFin =_{def} \{(si_{\rhd(in_C)}, so_{\rhd(out_C)}, z) \mapsto so \mid z \in Chart_C\}$$
$$CStep =_{def} \{(i \frown si, so, z_1) \mapsto (si, so \frown o, z_2) \mid z_1 \star \langle\!| i_C \Rrightarrow i, o'_C \Rrightarrow o |\!\rangle \star z'_2 \in CSys\}$$

The embedding of the simulation R gives the simulation relation S between the ADTs representing charts A and C.

For arbitrary sequences si and so, and bindings z_1 and z_2 we have,

$$S =_{def} \{(si_{\rhd(in_A)}, so_{\rhd(out_A)}, z_1) \mapsto (si_{\rhd(in_C)}, so_{\rhd(out_C)}, z_2) \mid z_1 \star z'_2 \in Corr_C^A\}$$

Notice that the relational simulation S is defined just in terms of the corresponding relation $Corr_C^A$. This is because the pointwise restriction of the sequences si and so already model the same relationship between input and output as the schema IO_C^A.

3.4 Total Chaos Refinement

3.4.1 Forward Simulation

Here we derive rules for a total chaotic interpretation of charts. Derrick and Boiten (2001) give five refinement conditions that are necessary to show that a relational data type C refines a relational data type A using a forwards simulation S. They begin by lifting (by introducing the special value $\bot$) and totalising the relations of the respective data types. Derrick and Boiten refer to the total chaotic interpretation as the *contract approach*. After giving the necessary lifted totalised relations they show how the five refinement conditions, referred to as *initialisation*, *finalisation*, *finalisation applicability*, *applicability* and *correctness*, can be simplified ("relaxed") to remove any reference to the introduced value $\bot$. We give the five relaxed conditions and refer to Woodcock and Davies (1996) for their derivations.

Definition 3.4 Assuming data types

$$A = (AState, AInit, \{AStep\}, AFin)$$

and

$$C = (CState, CInit, \{CStep\}, CFin)$$

a forwards simulation S is a relation from $AState$ to $CState$ satisfying the following conditions:

$$CInit \subseteq AInit \fatsemi S \quad \text{(init)}$$
$$S \fatsemi CFin \subseteq AFin \quad \text{(fin)}$$
$$\text{ran}((\text{dom } AFin) \lhd S) \subseteq \text{dom } CFin \quad \text{(fin app)}$$
$$\text{ran}((\text{dom } AStep) \lhd S) \subseteq \text{dom } CStep \quad \text{(app)}$$
$$((\text{dom } AStep) \lhd S) \fatsemi CStep \subseteq AStep \fatsemi S \quad \text{(corr)}$$

Now we use each of these conditions along with the relational embedding defined in Definition 3.3 to derive corresponding conditions expressed in Z.

For initialisation we have,

$$CInit \subseteq AInit \fatsemi S$$
$$\Leftrightarrow$$
$$\forall y_c \bullet y_c \mathrel{\dot\in} Init_C \Rightarrow \exists t_1 \bullet t_1 \mathrel{\dot\in} Init_A \wedge t_1 \star y'_c \in R$$

Unlike in the derivation provided by Derrick and Boiten (2001), the finalisation condition does not hold trivially for charts. This difference arises because the derivation for Z refinement makes the assumption that both the abstract and concrete ADTs have equivalently typed input and output, whereas the derivations required here do not.

$$S \fatsemi CFin \subseteq AFin$$
$$\Leftrightarrow$$
$$out_A \subseteq out_C$$

Because the given finalisation relation is total over all output sequences and states of the respective charts, the finalisation applicability condition holds trivially.

Now for the applicability condition we have:

$$\text{ran}((\text{dom } AStep) \lhd S) \subseteq \text{dom } CStep$$
$$\Leftrightarrow$$
$$\forall y_a, y_c \bullet Pre\ ASys\ y_a \wedge y_a \star y'_c \in R \Rightarrow Pre\ CSys\ y_c$$

And finally, for correctness we have:

$$((\text{dom } AStep) \lhd S) \fatsemi CStep \subseteq AStep \fatsemi S$$
$$\Leftrightarrow$$
$$\forall y_a, y_c, z_c \bullet (Pre\ ASys\ y_a \wedge y_a \star y'_c \in R \wedge y_c \star z'_c \mathrel{\dot\in} CSys) \Rightarrow \exists t \bullet y_a \star t' \mathrel{\dot\in} ASys \wedge t \star z'_c \in R$$

The completion of these derivations gives us the necessary conditions to show that a relation R is a forwards simulation between two charts A and C under the total chaotic interpretation of the partial relations semantics. As we have shown, it follows that chart C refines A in the total chaotic trace interpretation for charts. In line with the natural deduction style presentation that we have adopted, Figure 10 gives introduction and elimination rules for forwards simulation total chaotic refinement.

Notice the rules for forward simulation refinement presented here are, with the exception of the initialisation and finalisation conditions, very similar to the rules presented by Deutsch and Henson in Deutsch and Henson (2003) for *SF-refinement*. A similar method of derivation gives the corresponding rules for the backwards simulation case.

4 Monotonicity results

As with any language that provides operators allowing modular specifications and a refinement calculus for step-wise development, the monotonicity properties of the μ-Charts operators needs to be considered. These monotonicity properties are important for μ-Charts because they show to what extent the language supports modular development. Refinement is considered monotonic with respect to a language operator if a refinement of one part of a composite specification implies a refinement of the specification as a whole, and having this result is clearly important when we turn to using the logic on large specifications.

It turns out that we need quite strong, but very easy to motivate, side-conditions to guarantee that refinement is monotonic with respect to the chart composition operator.

Even though the monotonicity side-conditions described in Proposition 5.1 are presented before the monotonicity result itself, the conditions were formulated and refined from the proof of the monotonicity property (which we omit here due to space constraints). That the process of proving the monotonicity property allows us to state (and prove) these necessary side-conditions is evidence that the method of this paper has met some important goals. That is, the formal framework presented allows us to formulate precise descriptions of general, and typically non-obvious, language properties. In the case of the monotonicity result presented here, the first of the three required side-conditions is particularly non-obvious and at first reading may appear incorrect. However, the proof of monotonicity and careful evaluation of what this condition actually entails, makes clear the significance of the restriction.

5 Monotonicity of the μ-Charts composition operator

We begin by showing that the composition operator of μ-Charts is monotonic with respect to forward simulation refinement only when appropriate side-conditions hold. Like the investigation of Deutsch *et al.* (2003), the monotonicity proof itself is used to establish the necessary side-conditions. After ascertaining the required side-conditions an intuitive (in chart terms) justification for their necessity is given.

Recall that, by definition 3.4, to show that a forward simulation refinement holds between two charts requires that we show that an appropriate simulation exists between the charts. The proof of monotonicity relies heavily on splitting the definition of the simulation into two parts—the simulation between the respective charts' configurations using the *corresponding relation* and the simulation between the allowable input and output signals using the relation IO. This notion of splitting the simulation relation was introduced in Section 3.2 where we define the corresponding relation between two charts A and C as $Corr^A_C$ and the input/output relation as IO^A_C. Where previously we have denoted (total chaotic) forward simulation refinement between two charts C and A as $C \sqsupseteq_{\tau f} A$, here we supplement the relation with an explicit label that names the simulation required for refinement. So, assuming that chart C refines chart A using the simulation S, we will write $C \sqsupseteq^S_{\tau f} A$.

Proposition 5.1 states the monotonicity result for forward simulation refinement.

Proposition 5.1 If, for arbitrary charts A_1, C_2, and signal set Ψ, we have that,

$$\frac{}{\lfloor A_1 \rfloor_\Psi \sqsupseteq^T_{\tau f} \lfloor C_2 \rfloor_\Psi} \; SC_1$$

$$\frac{}{out_{A_1} \cap \Psi = out_{C_2} \cap \Psi} \; SC_2$$

$$\frac{}{out_{A_1} \cap out_B = out_{C_2} \cap out_B} \; SC_3$$

where $T =_{def} Corr^{C_2}_{A_1} \wedge IO^{C_\Psi}_{A_\Psi}$ for $C_\Psi = \lfloor C_2 \rfloor_\Psi$ and $A_\Psi = \lfloor A_1 \rfloor_\Psi$, then for arbitrary chart B, we have the monotonicity result,

$$\frac{C_2 \;\; \sqsupseteq^R_{\tau f} \;\; A_1 \quad SC_1 \quad SC_2 \quad SC_3}{(C_2 \mid \Psi \mid B) \;\; \sqsupseteq^S_{\tau f} \;\; (A_1 \mid \Psi \mid B)}$$

where $S =_{def} Corr^{A_1}_{C_2} \wedge Corr^B_B \wedge IO^A_C$, and $R =_{def} Corr^{A_1}_{C_2} \wedge IO^{A_1}_{C_2}$.

Despite the intricate appearance of the three side-conditions required for monotonic refinement of composed charts, these conditions are not unexpected when described in terms of charts themselves.

First consider the following property that holds in general for arbitrary charts A_1 and C_2, and feedback set Ψ.

Lemma 5.2

$$\frac{C_2 \;\; \sqsupseteq^R_{\tau f} \;\; A_1 \qquad out_{A_1} \cap \Psi = out_{C_2} \cap \Psi}{\lfloor C_2 \rfloor_\Psi \;\; \sqsupseteq^{T'}_{\tau f} \;\; \lfloor A_1 \rfloor_\Psi}$$

where $T' =_{def} Corr^{A_1}_{C_2} \wedge IO^{A_\Psi}_{C_\Psi}$

Given this property holds it follows that, in the context of the monotonicity proof of Proposition 5.1, *i.e.* where SC_1 holds, the charts A_1 and C_2 are output equivalent with respect to the signals in the set Ψ, *i.e.* $\lfloor C_2 \rfloor_\Psi \approx_{\mathcal{O}} \lfloor A_1 \rfloor_\Psi$. In words, an environment that reacts to just those signals in the set Ψ could not tell the difference between the charts A_1 and C_2. Therefore, we see that one of the properties required to guarantee monotonic refinement (with respect to composition) is that refining one part of the composition, say refining chart A_1 into C_2, cannot change the behaviour of A_1 with respect to the signals in Ψ that are used to communicate with the other part of the composition, *e.g.* chart B.

To explain the rôle of the side-condition SC_1 more specifically, with regard to the monotonicity proof, we describe two distinct parts that SC_1 plays in the proof.

Firstly, SC_1 enforces that the precondition of the chart, *i.e.* the set of state/input pairs for which the chart has explicitly defined behaviour, cannot be

Proposition 3.1 For arbitrary charts A and C, and bindings y_a, y_c, and z_c, we have,

$$\frac{\begin{array}{ll} & \vdash out_A \subseteq out_C \\ y_c \mathrel{\dot\in} Init_C & \vdash t_1 \mathrel{\dot\in} Init_A \\ y_c \mathrel{\dot\in} Init_C & \vdash t_1 \star y'_c \in R \\ Pre\ ASys\ y_a,\ y_a \star y'_c \in R & \vdash Pre\ CSys\ y_c \\ Pre\ ASys\ y_a,\ y_a \star y'_c \in R,\ y_c \star z'_c \mathrel{\dot\in} CSys & \vdash y_a \star t'_2 \mathrel{\dot\in} ASys \\ Pre\ ASys\ y_a,\ y_a \star y'_c \in R,\ y_c \star z'_c \mathrel{\dot\in} CSys & \vdash t_2 \star z'_c \in R \end{array}}{C \sqsupseteq_{\tau f} A} \ (\sqsupseteq^+_{\tau f})$$

$$\frac{C \sqsupseteq_{\tau f} A}{out_A \subseteq out_C} \ (\sqsupseteq^-_{\tau f\ I}) \qquad \frac{C \sqsupseteq_{\tau f} A \quad y_c \mathrel{\dot\in} Init_C \quad t_1 \mathrel{\dot\in} Init_A, t_1 \star y'_c \in R \vdash P}{P} \ (\sqsupseteq^-_{\tau f\ II})$$

$$\frac{C \sqsupseteq_{\tau f} A \quad Pre\ ASys\ y_a \quad y_a \star y'_c \in R}{Pre\ CSys\ y_c} \ (\sqsupseteq^-_{\tau f\ III})$$

$$\frac{C \sqsupseteq_{\tau f} A \quad Pre\ ASys\ y_a \quad y_a \star y'_c \in R \quad y_c \star z'_c \mathrel{\dot\in} CSys \quad \begin{array}{l} y_a \star t'_2 \mathrel{\dot\in} ASys, \\ t_2 \star z_c \in R \vdash P \end{array}}{P} \ (\sqsupseteq^-_{\tau f\ IV})$$

where the usual conditions hold, due to elimination of existential quantifiers, between t_1, t_2 and P.

Figure 10: Rules for chaotic refinement

weakened. Note that here we use the term weakening of the precondition in a very strict sense—side-condition SC_1 restricts any weakening of the precondition within the domain defined by the input interface of the abstract specification. Extending the domain of definition for a chart specification, *i.e.* increasing the input interface and weakening the precondition outside of the original domain, is still permitted in general.

This first aspect of the side-condition SC_1 is required for the part of the monotonicity proof related to the *correctness* property introduced in Section 3.4.1.

Figure 11 presents a counter-example that illustrates why this part of side-condition SC_1 is necessary in terms of charts. Given the charts A and C we clearly have that $C_2 \sqsupseteq_{\tau f} A_1$, yet it is **not** the case that $C \sqsupseteq_{\tau f} A$. That is, even though C_2 refines A_1, the composed chart C is not a valid refinement of A. The defined reaction of chart A given input $\{a\}$ is to output $\{w, t\}$, *i.e.* the two left hand transitions of chart A combine with respect to feedback to create an overall chart transition triggered by just the input $\{a\}$. However, chart C can nondeterministically choose to output $\{w, t\}$ or $\{w, s\}$ given input $\{a\}$, *i.e.* both the respective left hand and right hand transitions combine to give this nondeterministic behaviour. Therefore, C has additional nondeterministic behaviour to A and no valid refinement holds.

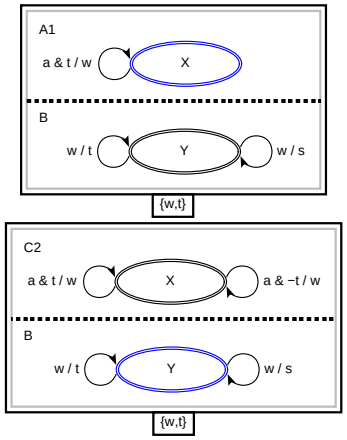

Figure 11: $SC_1, partI$: Charts $A = (A_1 \mid \{w, t\} \mid B)$ and $C = (C_2 \mid \{w, t\} \mid B)$

The second aspect of SC_1 is that it insists that the output behaviour, with respect to feedback, of an abstract specification is not changed via refinement. The property is required to prove the part of the monotonicity result related to the *applicability* condition.

In terms of charts, Figure 12 illustrates another counter-example that demonstrates why this second aspect of SC_1 is a necessary requirement for monotonic refinement. Note that the output interface of the chart C_2 is assumed to contain the signal w, *i.e.* we assume C_2 is a behavioural refinement of A_1 rather than an interface refinement. Again we have that the composed chart C does **not** refine the chart A. This is because A is defined for input $\{a\}$ due to feedback on w where chart C is not. Therefore chart C acts chaotically for input $\{a\}$ and the resulting additional nondeterminism invalidates the refinement relation.

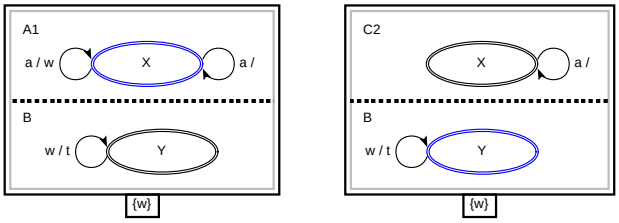

Figure 12: SC_1 and SC_2: Charts $A = (A_1 \mid \{w\} \mid B)$ and $C = (C_2 \mid \{w\} \mid B)$

The same charts from Figure 12 can be used to demonstrate why the side-condition SC_2 is required for monotonicity. In this case, however, we assume that the output interface of chart C_2 is reduced to the empty set of signals, that is, in this case C_2 is an interface refinement of A_1 rather than a behavioural refinement as above. Given this assumption SC_1 holds, that is, $\lfloor A_1 \rfloor_\Psi$ is a valid refinement of $\lfloor C_2 \rfloor_\Psi$. However, from inspection it is obvious that SC_2 does not hold in this case, that is, $out_{A_1} \cap \Psi \neq out_{C_2} \cap \Psi$, specifically, $\{w\} \cap \{w, t\} \neq \{\} \cap \{w, t\}$. The side condition SC_2 is required to prove monotonicity in relation to the *correctness* condition.

Finally, the side-condition SC_3 is required because μ-Charts refinement allows the designer to change the output context of a chart using interface refinement. If an interface refinement of one chart in a composition extends the control that the chart has over the environment using signals that were originally used

just by the other part of the composition, then there is the possibility that this new behaviour, from both charts, will be inconsistent when the charts are recombined in composition. For example, consider the counter example illustrated by the charts of Figure 13.

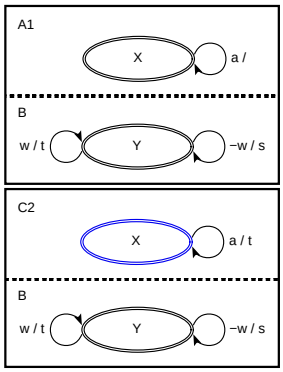

Figure 13: $SC_2(ii)$: Charts $A = (A_1 \parallel B)$ and $C = (C_2 \parallel B)$

Here the valid interface refinement $C_2 \sqsupseteq_{\tau f} A_1$ allows C_2 to control its environment over signals previously dealt with by the chart B, *i.e.* the signal t. The result is that the composed chart C can output $\{s, t\}$ for input $\{a\}$ where chart A could only output $\{s\}$ for input $\{a\}$. Hence, chart C has new behaviour that was not specified by chart A and therefore C is not a valid refinement of A.

Similar arguments can be used to show that the same side-conditions, SC_1, SC_2 and SC_3, are sufficient to guarantee monotonic refinement with respect to the composition operator for charts in the backwards simulation case.

5.1 The firing conditions interpretation of μ-Charts

A requirement for monotonic refinement is that the preconditions remain unchanged over the domain of definition of a chart. This requirement may cause an observant reader to question whether the *total chaotic* and *firing conditions* notions of refinement coincide in the case where refinements adhere to the monotonicity conditions. In particular, the work of Deutsch, Henson and Reeves (2002) shows that refinement based on a *firing conditions* approach can be considered as a notion that insists on the *stability of the precondition*. That is, refinement that allows the reduction of nondeterminism but insists that the precondition is neither strengthened nor weakened.

In fact, we can show that *total chaotic* refinement is both sound and complete with respect to *firing conditions* refinement when we insist that just the first condition SC_1, for monotonic refinement, is met. Any (guaranteed) monotonic refinement that we can prove using the total chaotic rules can also be proved using the rules for firing conditions refinement.

This is expressed by Proposition 5.3 .

Proposition 5.3 For arbitrary charts A, C and signal set Ψ we have,

$$\frac{C \sqsupseteq^S_{\tau f} A \qquad \lfloor A \rfloor_\Psi \sqsupseteq^T_{\tau f} \lfloor C \rfloor_\Psi}{C \sqsupseteq^S_{fcf} A} \qquad \frac{C \sqsupseteq^S_{fcf} A}{C \sqsupseteq^S_{\tau f} A}$$

where $S =_{def} Corr^A_C \wedge IO^A_C$ and $T =_{def} Corr^C_A \wedge IO^{C_\Psi}_{A_\Psi}$ for $C_\Psi = \lfloor C \rfloor_\Psi$ and $A_\Psi = \lfloor A \rfloor_\Psi$.

Notice that the second aspect of the side-condition SC_1 and the conditions SC_2 and SC_3 are still a necessary requirement to guarantee that firing conditions-based refinements are monotonic with respect to composition.

Therefore, while it is the case that using $\sqsupseteq_{fcf}$ for chart refinement implies a "more monotonic" refinement calculus, the difference in reality is slight.

The exact difference between the two notions of refinement is that the total chaotic model allows a refinement to weaken the precondition over the abstract domain of definition where the firing conditions model does not. The choice of the appropriate model can only be determined by the context of the refinement application. We do point out, though, that the total chaotic model provides the most general refinement framework.

6 Conclusions

A logic for composition and refinement of μ-Charts has been presented. The presented work has two significant contributions. The first is the presentation of a method for developing a logic for a StateCharts-like specification language—another example of the increasingly popular visual specification languages for reactive systems. The second is an investigation of a chaotic-based notion of refinement for the language μ-Charts.

The logic is developed by modelling the μ-Charts language in the more well-known and investigated language of Z. Given the extensive body of work that gives a logic to Z, we can specialise this logic and thereby induce a logic for charts. (Also, we are able to utilise existing tools for Z to reason about the model of a reactive system, if we wish.)

The notion of refinement that we induce for μ-Charts follows the chaotic-outside-of-defined-behaviour approach that is typically associated with Z-based ADT refinement or data refinement. As with Z-based refinement, the chart refinement defined maintains the principle of substitutivity (Derrick & Boiten 2001). That is, the substitution of an implementation of the specification for an implementation of a refinement of the specification will be indistinguishable in the context of the specification.

The notion of a chaotic semantics for μ-Charts was first introduced by Scholz (1998). The implicit nondeterminism outside of defined behaviour can be considered an abstraction mechanism just as in Z specifications. As the design is refined the nondeterminism is reduced, *i.e.* more decisions are made about undefined behaviour.

The chaotic semantics also facilitates refinements of both a reactive system's specified behaviour, and the specified context of the reactive system. These two types of refinement are both defined in the one notion of refinement presented. Refinement that changes the context of a chart preserves substitutivity because it is assumed that the context for a specified chart is fully defined, *i.e.* the context both controls and is controllable by just the signals in the respective input and output interfaces of the specification. Note that hiding signals from one or other of the interfaces is not in general a refinement.

The refinement rules presented give half (*i.e.* the forward simulation case) of a simulation based refinement calculus for μ-Charts. Unlike other theories of refinement for reactive systems the calculus presented allows the simulations to model a change in possible states from abstract to concrete specification as well as a change in the signals used to interact with the environment, *i.e.* the context, of the specified reactive system.

Of course, using the methods of this paper, as much as required of the whole of StateCharts can have a logic induced for it—once a semantics for a given construct has been defined in Z it is an intricate but conceptually straightforward task to induce

logical rules for the construct from the $\mathcal{Z}_C$ rules and the definition. The same goes, also, for refinement rules.

6.1 Future Work

Given the origins of μ-Charts (it is based on a simplification of the more well-known StateCharts), we typically take for granted that μ-Charts is a useful engineering tool for specifying reactive systems. Not surprisingly, to date most of the uses of the presented logic for μ-Charts have been concerned with investigating and proving properties of the language itself. It remains to be shown whether or not such a logic can be used practically to reason about and develop reactive systems. It is clear however, that using the formal logic for the practical development of reactive systems will require significant tool support. Ideally, this would be proof assistance, based specifically on the logic rules for charts, perhaps using a more general tool developed for the logic $\mathcal{Z}_C$. Other Z-based validation tools such as animation may also provide useful tools for investigating μ-Chart specifications. Limited tool support for μ-Charts already exists including a μ-Charts editor called AMuZed and a model builder (*i.e.* a program that translates a chart into its Z model) called ZooM (*Z-lambda project* 2005).

Another application of the logic that has not been fully investigated is to use the form of the simulations involved in refinement to suggest useful refinements of reactive system specifications. That is, can the form of proofs of refinements be used to indicate useful development strategies for reactive systems? This application of the logic closely follows Dijkstra's notion that "we develop program and correctness proof hand-in-hand" (Dijkstra 1976).

References

13568, I. (2002), *Information Technology—Z Formal Specification Notation—Syntax, Type System and Semantics*, Prentice-Hall International series in computer science, first edn, ISO/IEC.

Derrick, J. & Boiten, E. (2001), *Refinement in Z and Object-Z: Foundations and Advanced Applications*, Formal Approaches to Computing and Information Technology, Springer.
URL: *http://www.cs.ukc.ac.uk/pubs/2001/1200*

Deutsch, M. & Henson, M. C. (2003), An analysis of forward simulation data refinement, *in* D. Bert, J. Bowen, S. King & M. Waldén, eds, 'ZB 2003: Formal Specification and Development in Z and B / Third International Conference of B and Z Users', Vol. 2651 of *Lecture Notes in Computer Science*, Springer-Verlag Heidelberg, pp. 148–167.

Deutsch, M., Henson, M. C. & Reeves, S. (2002), Six theories of operation refinement for partial relation semantics, Technical Report CSM-363, Department of Computer Science Department, University of Essex.

Deutsch, M., Henson, M. C. & Reeves, S. (2003), Operation refinement and monotonicity in the schema calculus, *in* D. Bert, J. P. Bowen, S. King & M. Walden, eds, 'ZB 2003: Formal Specification and Development in Z and B', Vol. 2651 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 103–126.

Dijkstra, E. W. (1976), *A Discipline of Programming*, Prentice Hall.

Harel, D. (1987), 'Statecharts: A visual formalism for complex systems', *Science of Computing* pp. 231–274.

Henson, M. C., Deutsch, M. & Kajtazi, B. (2004), The specification logic νZ, Technical Report CSM-421, Department of Computer Science, University of Essex.

Henson, M. C. & Reeves, S. (2000), 'Investigating Z', *Journal of Logic and Computation* **10**(1), 1–30.

Henson, M. C. & Reeves, S. (2003), 'A logic for schema-based program development', *Formal Aspects of Computing Journal* **15**(1), 48–83.

Philipps, J. & Scholz, P. (1997*a*), Compositional specification of embedded systems with statecharts, *in* M. Bidoit & M. Dauchet, eds, 'TAPSOFT '97: Theory and Practice of Software Development', number 1214 *in* 'LNCS', Springer-Verlag, pp. 637–651.

Philipps, J. & Scholz, P. (1997*b*), Formal verification of statecharts with instantaneous chain reaction, *in* E. Brinksma, ed., 'Tools and Algorithms for the Construction and Analysis of Systems', Vol. 1217 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 224–238.

Reeve, G. (2005), μCharts-Investigating Refinement (To appear), PhD thesis, Department of Computer Science, University of Waikato.

Reeve, G. & Reeves, S. (2000), μ-Charts and Z: Hows, whys and wherefores, *in* W. Grieskamp, T. Santen & B. Stoddart, eds, 'Integrated Formal Methods 2000: Proceedings of the 2nd. International Workshop on Integrated Formal Methods', LNCS 1945, Springer-Verlag, pp. 255–276.

Scholz, P. (1998), A refinement calculus for statecharts, *in* E. Estesiano, ed., 'Fundamental approaches to software engineering: First International Conference, FASE'98', Vol. 1382 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 285–301.

Spivey, J. M. (1989), *The Z notation: A reference manual*, Prentice Hall.

Woodcock, J. & Davies, J. (1996), *Using Z: Specification, Refinement and Proof*, Prentice Hall.

Z-lambda project (2005).
URL: *www.cs.waikato.ac.nz/Research/fm*