Working Paper Series
ISSN 1170-487X

μ-charts and Z: Extending the Translation

by Greg Reeve and Steve Reeves

Working Paper 00/11
August 2000

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

μ-charts and Z: Extending the Translation

Greg Reeve Steve Reeves

August 23, 2000

Abstract

This paper describes extensions and modifications to the μ-charts as given in earlier papers of Philipps and Scholz. The charts are extended to include a command language, integer-valued signals and local integer variables. The command language is based on the syntax presented in Scholz' thesis and the integer-valued signals and local variables are based loosely on Scholz' earlier work.

After presenting the new semantics we turn to extending the μ-charts-to-Z translation that we developed in previous work. The extensions to the translation process describe both the changes due to the extensions to the μ-charts and a modification to the translation method to more fully capture the beneficial modularisation encouraged by the μ-charts formalism.

We finish by giving three complete translation examples.

The paper should be read as a record of our gradual development of a Z semantics for μ-charts—hence its sometimes exploratory character or laborious explanations as we come to terms (thinking out loud) with the (sometimes very subtle) meaning of μ-charts, especially with regard to pathological and unusual examples of their use.

Contents

1 Introduction

This working paper describes two separate pieces of work which are at once closely related but also markedly different. They are closely related in that they aim to give semantics to the same constructs; they differ in that in subtle but important ways the two versions of the semantics we obtain do not agree. This disagreement is not due to contrariness on our part but flows quite naturally from the two ways we define the two versions of the semantics.

The paper should be read as a record of our gradual development of a Z semantics for μ-charts—hence its sometimes exploratory character or laborious explanations as we come to terms (thinking out loud) with the (sometimes very subtle) meaning of μ-charts, especially with regard to pathological and unusual examples of their use.

That this paper is really a record of *experimentation* in order to understand the language better is made evident by the fact that we have not yet *proved* any relationship between the various semantics we explore here.

We describe extensions and modifications that we have recently made to the μ-chart semantics given in [2]. It must be read in the context of that paper since it owes a lot to it, and borrows and adapts much of that work. In [3] μ-charts were extended to include a command language, integer-valued signals and local integer variables. We have developed a semantics to cover these extensions[1]. The new extended semantics also modifies the original semantics by changing the way in which transitions are denoted and removing the need to use the oracle and copyright signals that were introduced in [2]. The command language that is added to the semantics for transitions is based on the syntax presented in [7] and the integer-valued signals and local variables are based loosely on [8].

The paper also describes how we have made further progress with the μ-charts-to-Z translation presented in [5] (and as was, very briefly, done in [4]). The extensions to the translation process describe both the changes due to the extended μ-charts and a modification to the translation method to more fully capture the beneficial modularisation encouraged by the μ-charts formalism. We finish by giving three complete translation examples.

The Statecharts variant *μ-charts* that is the basis of the translation given in [5] is itself based on a preceding Statecharts variant called Mini-Statecharts. In [8] an extended semantics is given for Mini-Statecharts that includes local variables, integer-valued signals and a command language. These extensions can help to prevent "state explosion", *i.e.* needing an unmanageable or unrealistic number of states in a specification, *e.g.* the example given in [8] gives the specification of a TV set with two channels, represented by a state for each channel, and demonstrates why the extensions are needed by extending this specification to a TV set that has a hundred channels—having a state for each channel in this situation would be cumbersome and unusable.

We also allow more general forms for trigger expressions than in the previous work cited above. There, a trigger expression could only be a Boolean term over the presence or absence of signals. For example the trigger a (where a is a signal) means the transition fires only when the signal a is present in the input and likewise the trigger $a \wedge \neg\ b$ fires when the signal a is in the input but the signal b is not. We have extended these trigger expressions to allow Boolean terms over arithmetic expressions (the grammar for these is given in Section 2.1 below), which include local variables and integer-valued signals. This means that we lose the property, which the original systems had, that we can tell from a static analysis whether or not a chart is nondeterministic. However, since our extension properly includes the original, we can always re-impose use of the more restricted trigger language if the situation requires it. This extension is just what is in [7] (section 2.3.1) with the addition of integer-valued signals.

Along with the changes to the semantics resulting from the extensions described above we also make two other changes. The first is related to the way in which the meaning of transitions is denoted. It is conjectured, though not proven here, that this change makes the formal description of transitions more intuitive while giving the same meaning as the previous semantics. The second change is to introduce extra signals in order to remove the need for the mechanisms referred to as oracle signals, oracle experiments and copyright signals in [8].

When specifying a large complex system the ability to work with systems of related functionality,

[1] We extend integer-valued signals to value-carrying signals where values are allowed from any Z-definable type.

reason about part of the system without needing to think about the overall system and be certain that the part is going to fit seamlessly into the whole is important. This ability to create a specification in a modular fashion allows the overall complexity to be broken into more manageable pieces and can allow several people to more effectively collaborate in the specification process.

μ-Charts are examples of specifications that encourage modular design of a system specification. They encourage parallel composition of charts describing separate concurrent processes, hierarchical decomposition of states of a sequential μ-chart that allows the functionality of an abstract state to be described in more detail in another μ-chart and they provide a communication mechanism, instantaneous broadcast of signals, to allow these separate processes to communicate and interact with one and other. The modification to the translation model allows the translated Z description to maintain the principles of modularity inherent in μ-charts.

2 The New Semantics

This section introduces the new semantics for μ-charts.

First we introduce the structure of signals. Signals are the events to which μ-charts react. They can originate from the external environment, be internally broadcast from within a μ-chart, be value carrying or simply be abstract representations of events. Here the set of signals for a specification also includes a subset that is used purely to aid in defining the correct denotational semantics. The signals of this subset will be referred to as negated signals. The set M^* is used to denote the set of all signal names. M^* is partitioned as two sets with equal cardinality, M the set of all signals of the specification and M' a set that represents the negated signals. M' is assumed to have a unique member pertaining to each member of M. For the semantic description we will assume there is a total bijective function $Neg \in M \rightarrow M'$ that maps each signal in M to its negation in M'. The set M is itself partitioned into two more sets M_p and M_v. The set M_p represents all of the pure signals of the specification and the set M_v represents all of the value carrying signals.

Other definitions used for the semantics include: $States$, a set of all possible state names, $Ident$, a set of unique identifiers for sequential μ-charts and V, a set of local variables that can be associated with a sequential μ-chart. A partial function $\gamma \in M_v \rightarrow \mathbb{Z}$ is used as the environment to hold the current values of the integer-valued signals. $M_v \rightarrow \mathbb{Z}$ is abbreviated to Γ. Another environment ϵ, is defined as a partial function $\epsilon \in V \rightarrow \mathbb{Z}$, which maps local variables used in the chart to values. We make this rather vague statement precise in the sequel. The set of all local variable environments is denoted by $\mathcal{E}$.

The set of all possible μ-charts, $\mathcal{S}$, is defined inductively over four types of μ-charts: a sequential automaton, a parallel composition of two μ-charts, the decomposition of a sequential μ-chart's states by other μ-charts and a μ-chart with feedback. Each of these will be described in the sequel.

The semantics of a μ-chart $S \in \mathcal{S}$ is

$$[\![S]\!] \in \mathbb{P}(M) \rightarrow \Gamma \rightarrow \mathcal{E} \rightarrow \mathbb{P}(\mathbb{P}(M) \times \mathcal{S} \times \mathbb{P}(\mathbb{N}) \times \Gamma \times \mathcal{E})$$

For each input set, integer-valued signal environment and local variable environment the semantics determines the set of possible reactions of a μ-chart. So, a reaction is a quintuple consisting of: a set of output signals, a μ-chart with updated actual state, resulting from any transitions that have taken place, a set of unique transition identifiers (natural numbers here) for naming transitions and environments for integer-valued signals and local variables.

2.1 Command and trigger languages

The command and trigger languages that we introduce in this section are based on [7]. More sets are introduced for the description of the languages: $Aexp$ denotes arithmetic expressions, $Bexp$ denotes Boolean terms, Com denotes commands and $\mathbb{B} = \{tt, ff\}$. Assuming $a \in Aexp$, $b \in Bexp$ and $c \in Com$ the syntax of the command language is given by all productions below and that for the trigger language by just the productions for a and b:

$$a ::= n \mid Y \mid a\ binop\ a \mid s_v$$
$$b ::= true \mid false \mid a = a \mid a < a \mid a \leq a \mid a > a \mid a \geq a \mid s_i \mid \neg\, b \mid b \wedge b \mid b \vee b$$
$$c ::= skip \mid Y := a \mid s_o \mid s_v := a \mid c\ ^*\ c$$

where $n \in \mathbb{Z}$ is an integer value, $Y \in V$ is a local integer variable, $s_v \in M$ is an integer-valued signal. The boolean term s_i is true when the signal $s_i \in M$ is in the input for the current step and false otherwise. The command s_o places the signal $s_o \in M$ in the output set for the current step. The command $c_1\ ^*\ c_2$ stands for parallel composition of two commands. To avoid possible problems with parallel composition the commands must adhere to the *weak Bernstein condition*. This condition is satisfied if no variable common to c_1 and c_2 is assigned to in either command, and it has the appealing and pleasant property that it is a purely syntactic condition, which means that it is easily checked.

Reasons for choosing parallel composition as opposed to sequential composition as in [8] are not discussed further here but the interested reader should consult [7].

Another restriction is that, for any label t/c of any transition, any integer-valued signal must occur positively in the trigger t if it occurs on the right-hand side of an assignment in the command c. This restriction is needed to guarantee that the value associated with such a signal is available when it is to be used. As stated in [8] (section 3.2), this means that s_v occurs positively in t if and only if $t \implies s_v$.

Now the denotational semantics for the two languages are given using the three semantic functions:

$$\lambda\, a.\mathcal{A}[\![a]\!] \in Aexp \to (\mathbb{P}(M) \times \Gamma \times \mathcal{E}) \to (\mathbb{Z} \cup \{\perp\})$$
$$\lambda\, b.\mathcal{B}[\![b]\!] \in Bexp \to (\mathbb{P}(M) \times \Gamma \times \mathcal{E}) \to \mathbb{B}$$
$$\lambda\, c.\mathcal{R}[\![c]\!] \in Com \to (\mathbb{P}(M) \times \Gamma \times \mathcal{E}) \to (\mathbb{P}(M) \times \Gamma \times \mathcal{E})$$

Notice that the meaning of an arithmetic expression can be undefined (represented by the symbol $\perp$) when an integer signal is used that is not input.

The denotation of an arithmetic expression is defined by structural induction:

$$\mathcal{A}[\![n]\!](x, \gamma, \epsilon) = n$$
$$\mathcal{A}[\![Y]\!](x, \gamma, \epsilon) = \epsilon(Y)$$
$$\mathcal{A}[\![s_v]\!](x, \gamma, \epsilon) = \begin{cases} \gamma(s_v) & \textbf{if } s_v \in x, \\ \perp & \textbf{else}. \end{cases}$$
$$\mathcal{A}[\![a_1\ binop\ a_2]\!](x, \gamma, \epsilon) = \mathcal{A}[\![a_1]\!](x, \gamma, \epsilon)\ binop\ \mathcal{A}[\![a_2]\!](x, \gamma, \epsilon)$$

Note it is assumed that *binop* applied to an undefined argument gives undefined, *i.e.* that *binop* is always strict. Also, since every value carrying signal s_v that occurs in the command of a transition must occur positively in the trigger of the transition, $\gamma(s_v)$ is always defined because by definition γ is defined for each of the value carrying signals in the current input set.

Now the denotation of boolean expressions is defined. Note the definitions here assume that trigger expressions are in negation normal form, *i.e.* all negation occurring in the trigger has the smallest possible scope. It is also useful to remember that these definitions mean that a trigger expression like $s > 10$ means the same as $s \wedge s > 10$ and that this is true when s is in the input set and the value it is carrying is greater than 10.

$$\mathcal{B}[\![true]\!](x,\gamma,\epsilon) = tt$$
$$\mathcal{B}[\![false]\!](x,\gamma,\epsilon) = ff$$
$$\mathcal{B}[\![a_1\ lop\ a_2]\!](x,\gamma,\epsilon) = \begin{cases} ff & \textbf{if } \mathcal{A}[\![a_1]\!](x,\gamma,\epsilon) = \perp \\ ff & \textbf{if } \mathcal{A}[\![a_2]\!](x,\gamma,\epsilon) = \perp \\ \mathcal{A}[\![a_1]\!](x,\gamma,\epsilon)\ lop\ \mathcal{A}[\![a_2]\!](x,\gamma,\epsilon) & \textbf{else}. \end{cases}$$
$$\mathcal{B}[\![s_i]\!](x,\gamma,\epsilon) = s_i \in x$$
$$\mathcal{B}[\![\neg\ b]\!](x,\gamma,\epsilon) = \begin{cases} Neg(b) \in x & \textbf{if } b \in M, \\ \neg\ \mathcal{B}[\![b]\!](x,\gamma,\epsilon) & \textbf{else}. \end{cases}$$
$$\mathcal{B}[\![b_1 \wedge b_2]\!](x,\gamma,\epsilon) = \mathcal{B}[\![b_1]\!](x,\gamma,\epsilon) \wedge \mathcal{B}[\![b_2]\!](x,\gamma,\epsilon)$$
$$\mathcal{B}[\![b_1 \vee b_2]\!](x,\gamma,\epsilon) = \mathcal{B}[\![b_1]\!](x,\gamma,\epsilon) \vee \mathcal{B}[\![b_2]\!](x,\gamma,\epsilon)$$

where *lop* is a place holder for $=$, $<$, $\leq$, $>$ and $\geq$.

Notice that the semantics for Boolean expressions must check that any value carrying signal is in the input (*i.e.* it has a defined value) before evaluating an expression depending on that signal's value. If the signal's value is not defined then the result is false.

In the sequel we will need a function-update or -extension operation. For any function $f \in A \to B$, $m \in A$ and $n \in B$ we have:

$$f[m/n](y) = \begin{cases} n, & y = m \\ f(y), & otherwise \end{cases}$$

So, this is extension when $m \notin \mathrm{dom}\, f$.
Finally the semantic function for commands is given.

$$\mathcal{R}[\![skip]\!](x,\gamma,\epsilon) = (\{\},\gamma,\epsilon)$$
$$\mathcal{R}[\![Y := a]\!](x,\gamma,\epsilon) = \textbf{let}\ n = \mathcal{A}[\![a]\!](x,\gamma,\epsilon)\ \textbf{in}\ (\{\},\gamma,\epsilon[Y/n])$$
$$\mathcal{R}[\![s_o]\!](x,\gamma,\epsilon) = (\{s_o\},\gamma,\epsilon)$$
$$\mathcal{R}[\![s_v := a]\!](x,\gamma,\epsilon) = \textbf{let}\ n = \mathcal{A}[\![a]\!](x,\gamma,\epsilon)\ \textbf{in}\ (\{s_v\},\gamma[s_v/n],\epsilon)$$
$$\mathcal{R}[\![c_1 * c_2]\!](x,\gamma,\epsilon) = \textbf{let}(y,\gamma',\epsilon') = \mathcal{R}[\![c_1]\!](x,\gamma,\epsilon)\ \textbf{in}\ (\textbf{let}(z,\gamma'',\epsilon'') = \mathcal{R}[\![c_2]\!](x,\gamma',\epsilon')\ \textbf{in}\ (y \cup z,\gamma'',\epsilon''))$$

A command cannot update the value of a local variable with the value of an integer-valued signal that is not input because of the restriction, stated above, that any integer-valued signal appearing in an arithmetic expression a must occur positively in the transition trigger. Therefore the function $\lambda\ c.\mathcal{R}[\![c]\!]$ is guaranteed to be defined at all its (allowed) uses.

2.2 Sequential μ-Chart Syntax

The simplest type of μ-chart describes a sequential automaton. Such a sequential μ-chart, as we shall call it, has the syntax

$$Seq(N,\iota,\Sigma,\sigma_d,\sigma,\delta)$$

where this μ-chart is an element of the set of all μ-charts if and only if the following constraints hold:

- $N \in Ident$ is a unique identifier of the chart;
- ι is an initial environment for all the local variables in the chart;
- $\Sigma \in \mathbb{P}\ States$ is a non-empty finite set of all states contained in the chart;

- $\sigma_d, \sigma \in \Sigma$ represent the default state (or start or initial state) and current state of the chart respectively;
- $\delta \in \Sigma \times Bexp \to \mathbb{P}(\Sigma \times Com \times \mathbb{P}\,\mathbb{N})$ is the finite, total state transition function for the μ-chart.

The function δ for each μ-chart is defined by giving a value for each of the transitions in the sequential μ-chart. It is made total by having the result of any other trigger that does not cause a transition in the μ-chart return the empty set. Another change to the definition of the δ-function is that for each transition in the μ-chart the δ-function returns, along with the output and new state, a set containing a unique numeric identifier that distinguishes the transition from other transitions. By doing this neither the copyright signals nor oracle signals described in [2] are needed. This mechanism is explained and utilised in section 2.7.

Having to name each transition with a unique number, where that uniqueness is chart-wide, would seem to destroy the inherent modularity of charts. However, something like a Gödel numbering scheme could, formally, be used to name the transitions, where the name is composed from the Gödel number of the chart name and the information in the δ-function for the transition. This information is local to the chart and, via the numbering, leads to a unique (number) name, hence unique naming is possible while preserving modularity. An alternative, and simpler, device would be to code the chart name and prefix it to the numbers 1, 2, ... and use these to name the transitions. Again, this is a local operation and gives a chart-wide unique name for each transition (remembering the charts have unique names). Either way, once this formal device has been observed we can ignore it subsequently and assume that the unique naming is possible and, as we shall, assume that we lose no generality by simply naming the transitions with numbers starting from one.

Given δ defined as above we can now give an auxiliary function that is used by the semantics to determine a chart's reaction to a given set of input signals. The auxiliary function

$$\delta' \in \Sigma \times \mathbb{P}(M) \to \Gamma \to \mathcal{E} \to \mathbb{P}(\Sigma \times Com \times \mathbb{P}\,\mathbb{N})$$

is defined in terms of δ where δ gives a non-empty result, and gives no reaction when there is no defined transition.

$$\delta'(\sigma, x)\,\gamma\,\epsilon = \bigcup\{\delta(\sigma, t) \mid t \in Bexp \wedge \exists \sigma', c, ti.\sigma' \in \Sigma \wedge c \in Com \wedge ti \in \mathbb{N} \wedge (\sigma', c, ti) \in \delta(\sigma, t) \wedge \mathcal{B}[\![t]\!](x, \gamma, \epsilon) = tt\}$$

2.3 Sequential μ-chart Semantics

Given a sequential μ-chart $S \in \mathcal{S}$ defined as $S = Seq(N, \iota, \Sigma, \sigma_d, \sigma, \delta)$ which abides by the constraints given in the previous section, the semantic function for S is,

$$[\![S]\!]\,x\,\gamma\,\epsilon = \{(y, Seq(N, \iota, \Sigma, \sigma_d, \sigma', \delta), t, \gamma', \epsilon') \mid (\sigma', c, t) \in \delta'(\sigma, i)\,\gamma\,\epsilon \wedge i = (x \cup Neg\,(M \setminus x)) \wedge (y, \gamma', \epsilon') = [\![c]\!](x, \gamma, \epsilon)\}$$

where $M \setminus x$ is the set M excluding those signals in x and $Neg(s)$ is assumed to be the application of the function Neg to each of the members of the set s.

To explain how the definition works we break it into parts. Firstly the set $i = (x \cup Neg\,(M \setminus x))$ contains all the signals that were input and a negated signal for all of the remaining signals that were not input. This means a negated trigger expression now informally translates to requiring the appropriate member of M' to be in the input set i on which the function δ' is applied. For each triggered transition, determined from applying the function δ' to the input set i, the predicate $(y, \gamma', \epsilon') = [\![c]\!](x, \gamma, \epsilon)$ translates the command associated with that transition into an output set and appropriately updated environments. The transition's unique identifier and target state obtained from applying δ' also contribute to the resulting tuple output for each non-deterministic possibility given by the semantic function for sequential μ-charts.

2.4 Some auxiliary functions

Before we go on to deal with the main part of the semantics we need three functions that play an auxiliary role later on.

We will need to be able to denote the local variables that occur anywhere in a chart (*i.e.* at any level of any hierarchy that may exist) The function v computes the set of all local variables:[2]

$$v(Seq(N, \iota, \Sigma, \sigma_d, \sigma, \delta)) = \mathrm{dom}\,\iota$$
$$v(And(S_1, S_2)) = v(S_1) \cup v(S_2)$$
$$v(Dec\ Seq(N, \iota, \Sigma, \sigma_d, \sigma, \delta)\ by\ \rho) = \mathrm{dom}\,\iota \cup \{v(\rho(\sigma)) \mid \sigma \in \Sigma\}$$
$$v(Feedback(S, L)) = v(S)$$
$$v(NoDec) = \{\}$$

We will also need to be able to denote the initialised version of any chart. Initialisation involves both placing a chart in its initial (default) state and initialising its local variable environment. The functions $init_s$ and $init_\epsilon$ will be involved in doing this:

$$init_s(Seq(N, \iota, \Sigma, \sigma_d, \sigma, \delta)) = Seq(N, \iota, \Sigma, \sigma_d, \sigma_d, \delta)$$
$$init_s(And(S_1, S_2)) = And(init_s(S_1), init_s(S_2))$$
$$init_s(Dec\ Seq(N, \iota, \Sigma, \sigma_d, \sigma, \delta)\ by\ \rho) =$$
$$\qquad Dec\ Seq(N, \iota, \Sigma, \sigma_d, \sigma_d, \delta)\ by\ \{(\sigma, init_s(\rho(\sigma)) \mid \sigma \in \Sigma\}$$
$$init_s(Feedback(S, L)) = Feedback(init_s(S), L)$$
$$init_s(NoDec) = NoDec$$

$$init_\epsilon(Seq(N, \iota, \Sigma, \sigma_d, \sigma, \delta)) = \iota$$
$$init_\epsilon(And(S_1, S_2) = init_\epsilon(S_1) \cup init_\epsilon(S_2)$$
$$init_\epsilon(Dec\ Seq(N, \iota, \Sigma, \sigma_d, \sigma, \delta)\ by\ \rho) = init_\epsilon(S) \cup init_\epsilon(\rho(\sigma_d))$$
$$init_\epsilon(Feedback(S, L)) = init_\epsilon(S)$$
$$init_\epsilon(NoDec) = \{\}$$

Notice initialising the local variable environment for a decomposed chart entails taking the union of the master's initial environment and the initial environment of any slave that decomposes the master's initial state.

Given this framework we can go on to give the semantic definitions for the three remaining types of μ-chart.

2.5 Parallel Composition

If the μ-charts S_1 and S_2 are elements of $\mathcal{S}$ then their parallel composition denoted by

$$And(S_1, S_2)$$

is also in $\mathcal{S}$. The graphical notation represents parallel composition by splitting a μ-chart box by a dashed line. (See Figure 7 for an example.)

The formal semantics of composition is similar to that of [2]. The function $f \restriction S$ is the restriction of a function f to the domain S. The new definition is as follows.

[2]Note that $\mathrm{dom}\,\rho = \Sigma$ by definition, as we shall see. Also, only sequential charts can occur at the top-level of a hierarchy.

$$
\begin{aligned}
[\![And(S_1,S_2)]\!]\,x\,\gamma\,\epsilon = \{(y_1 \cup y_2, And(S_1',S_2'), t_1 \cup t_2, \gamma_1 \cup \gamma_2, \epsilon_1 \cup \epsilon_2) \mid \\
(y_1,S_1',t_1,\gamma_1,\epsilon_1) \in [\![S_1]\!]\,x\,\gamma\,(\epsilon \restriction v(S_1)) \wedge \\
(y_2,S_2',t_2,\gamma_2,\epsilon_2) \in [\![S_2]\!]\,x\,\gamma\,(\epsilon \restriction v(S_2)) \wedge \\
\gamma_1 \cup \gamma_2 \in \Gamma\} \\
\cup \{(y, And(S_1',S_2), t, \gamma', \epsilon') \mid (y,S_1',t,\gamma',\epsilon') \in [\![S_1]\!]\,x\,\gamma\,(\epsilon \restriction v(S_1)) \wedge \\
[\![S_2]\!]\,x\,\gamma\,(\epsilon \restriction v(S_2)) = \{\}\} \\
\cup \{(y, And(S_1,S_2'), t, \gamma', \epsilon') \mid (y,S_2',t,\gamma',\epsilon') \in [\![S_2]\!]\,x\,\gamma\,(\epsilon \restriction v(S_2)) \wedge \\
[\![S_1]\!]\,x\,\gamma\,(\epsilon \restriction v(S_1)) = \{\}\}
\end{aligned}
$$

The semantic function for parallel composed charts covers the same three cases as that of [2]: both sub-charts make a transition or one or other have an empty reaction set. Also if neither chart makes a transition the reaction set of the parallel composition is empty. There is one new situation to note here which is when both charts have reactions to the input but their reaction gives a conflicting value to an integer-valued signal, *i.e.* more than one chart makes a transition that attempts to output the same integer-valued signal with different values. In any case fitting this description the reaction is not added to the reaction set, *i.e.* if the input causes a transition in both charts that try to give a different value to the same integer-valued signal then neither transition happens. This is enforced formally by the predicate $\gamma_1 \cup \gamma_2 \in \Gamma$, *i.e.* the union of the resulting γ-functions must also be a function. This method was chosen because it is simpler than the conflict resolution method used in [8] in that it does not require the specifier to define a conflict resolution function. However, such a function could obviously be added to the semantics if required.

2.6 Hierarchical Decomposition

Hierarchical decomposition is the mechanism that allows a state of a sequential μ-chart to contain another μ-chart. When a sequential μ-chart is hierarchically decomposed it is termed the master and the μ-chart that is contained in one of its states is referred to as a slave.

Suppose we again have the sequential μ-chart $S = Seq(N, \iota, \Sigma, \sigma_d, \sigma, \delta)$, then the hierarchical decomposition of S is,

$$Dec\ S\ by\ \rho$$

where $\rho \in \Sigma \rightarrow \mathcal{S} \cup \{NoDec\}$.

The semantics for hierarchically decomposed μ-charts is described using three mutually exclusive cases. The first case occurs when the current state of the μ-chart σ contains ("is decomposed by") a slave μ-chart *i.e.* $\rho(\sigma) \neq NoDec$, and the master chart changes state.

The two charts, *i.e.* master and slave, react as if they were composed in parallel, therefore the semantic function uses the equation for composed charts to determine the meaning for the hierarchically decomposed chart. If the state change made by the master ends in another decomposed state then that slave, *i.e.* $\rho(\sigma')$, is initialised and its initialised local variable environment is added to the overall local variable environment. Note the values of $init_s(\rho(\sigma'))$ and $init_\epsilon(\rho(\sigma'))$ (see Section 2.4) will be $NoDec$ and $\{\}$ respectively if $\rho(\sigma') = NoDec$ [3].

$$
\begin{aligned}
[\![Dec\ S\ by\ \rho]\!]\,x\,\gamma\,\epsilon = \{(y, Dec\ S'\ by\ \rho', t, \gamma', (\epsilon' \restriction v(S')) \cup \epsilon'') \mid \exists\, S''.S'' \in \mathcal{S} \wedge \\
(y, And(S', S''), t, \gamma', \epsilon') \in [\![And(S, \rho(\sigma))]\!]\,x\,\gamma\,\epsilon \wedge \sigma' \neq \sigma \wedge \\
\rho' = \rho[\sigma/S'', \sigma'/init_s(\rho(\sigma'))] \wedge \epsilon'' = init_\epsilon(\rho(\sigma'))\}
\end{aligned}
$$

The semantics for hierarchically decomposed μ-charts is different here from that in [2] in that we do not say that a slave chart is initialised at the instant it is left by a transition in the master but rather only when a transition cause the master to change state into the state decomposed by the slave. Therefore a loop transition is a special case that does not cause the slave to initialise. For a more detailed discussion about why this was done and what difference it makes see Section 6.

[3] We assume in this and following definitions that $S = Seq(N, \iota, \Sigma, \sigma_d, \sigma, \delta)$ and $S' = Seq(N', \iota', \Sigma', \sigma_d', \sigma', \delta')$

Obviously the property of parallel composition that says if the master and slave have a reaction that attempts to set the same integer-valued signal to different values then neither transition happens is inherited by hierarchical decomposition. Also a weak preemption model is assumed. That is, if the master leaves the slave any output from any valid transition in the slave is part of the output for the chart's reaction.

The second case describes the meaning of a decomposed chart when the current state of the μ-chart is again decomposed by a slave μ-chart but the master does *not* change state.

$$[\![Dec\ S\ by\ \rho]\!]\, x\, \gamma\, \epsilon = \{(y, Dec\ S'\ by\ \rho', t, \gamma', \epsilon') \mid \exists\, S''.S'' \in \mathcal{S} \wedge (y, And(S', S''), t, \gamma', \epsilon') \in [\![And(S, \rho(\sigma))]\!]\, x\, \gamma\, \epsilon \wedge \sigma' = \sigma \wedge \rho' = \rho[\sigma/S'']\}$$

Obviously there is no initialisation here because the master does not change state. (Note that is not the same as saying the master does not make a transition because it can make loop transitions and so not change state, *i.e.* we still use the compositional definitions to capture the effect of such a transition).

The third case occurs when the current state of the master is atomic ($\rho(\sigma) = NoDec$), *i.e.* it is not decomposed.

$$[\![Dec\ S\ by\ \rho]\!]\, x\, \gamma\, \epsilon = \{(y, Dec\ S'\ by\ \rho', t, \gamma', \epsilon' \cup \epsilon'') \mid (y, S', t, \gamma', \epsilon') \in [\![S]\!]\, x\, \gamma\, \epsilon \wedge \rho' = \rho[\sigma'/init_s(\rho(\sigma'))] \wedge \epsilon'' = init_\epsilon(\rho(\sigma'))\}$$

Again if the master makes a transition to a decomposed state the initialisation described above must happen.

If no transitions occur under either of the three situations above then the result is the empty set of reactions as expected.

2.7 Feedback Operator

Parallel composition is used to specify independent, concurrent components. Hierarchical decomposition is used for the same purpose with the added restriction that a slave can only respond to input when the master is in the state that the slave decomposes. To allow interaction between such concurrent components an explicit feedback operator provides a broadcast communication mechanism. For the same reasons given in [2] we define the feedback operator to model instantaneous feedback.

Assuming the set of signals we wish to broadcast is represented by the set $L \in \mathbb{P}(M)$ then for any μ-chart $S \in \mathcal{S}$ the construct

$$Feedback(S, L)$$

is also in $\mathcal{S}$.

When describing the semantics for μ-charts that have feedback we give meaning to a complex chart (*i.e.* resulting from the combination of several sequential charts) using the meaning of its parts and can be assured that the meaning is well-defined because charts can not be infinitely embedded. Also, and most crucially, we make use of the fact that a given sequential chart can make at most one transition per step. This means that, if the equations are carefully constructed, we can be sure the meaning of a chart is well-defined since no input can lead to anything other than a finite number of transitions in any one step. Because of the way that the scope of fed back signals for a given chart includes all of its component charts, we cannot give a single general equation for all charts, no matter what their form, when considering feedback. The meaning of a composed chart with feedback must be described in terms of the components of the composition with the feedback applied to each of them, thus modelling the scope. The meaning of a sequential chart with feedback can be determined in terms of the chart without feedback, hence this process is well-defined. So, we define the semantics for a μ-chart with feedback via four cases, one for each of the three types of μ-chart plus one which describes the behaviour of the feedback operator when applied to a μ-chart that already has broadcast communication.

One way to describe the idea underlying the development of our semantic functions for feedback charts is something we will refer to as "threads". We imagine that each alternative non-deterministic reaction of a chart throughout a semantic calculation starts a new thread. A thread can be thought of as recording the

transitions that have occurred due to the current input and feedback during the calculation of the possible reactions of a chart. If the output (which also happens to be fed back) from a further transition causes some transitions that have occurred previously (*i.e.* to get us to this point) to be ruled out then this thread does not describe a possible reaction of the chart (we say the thread is broken). This is because some previously taken transition has now been blocked because of subsequent feedback signals (remembering that the absence of some signal can be a trigger and setting the same integer valued signal to different values results in no reaction). So the set of transitions that go to make up a thread must be strictly increasing, *i.e.* none of the transitions that began a thread are ruled out by "re-running" the chart with input plus feedback signals. If some of the transitions disappear from the thread then the thread is broken and disallowed as a meaning of the chart.

The threads are, to some extent, counterparts to the sets built-up due to the iterations taken in computing the least fixed-point solution in the original semantics. However, the subtle difference, in that we consider each of the possible non-deterministic choices separately, means we rule out one of the purposes of the copyright mechanism (a fuller example of this situation is given below by Figure 1). The other purpose for copyright signals is for semantic definitions (such as those presented in [3]) that consider a "chain reaction" of transitions that consider both output and state change, *i.e.* it is possible for more than one transition to occur in a sequential μ-chart without non-determinism. This need for copyright signals does not occur here because a transition's effect on state is not factored into the semantic calculation, therefore the only chance of several transitions happening in a sequential chart arises from non-determinism which is handled by the thread concept.

The first and most trivial case is feedback applied to a sequential chart. The idea for a sequential chart is to firstly obtain the set of solutions, *i.e.* the meaning of the chart, without considering feedback. Then for each non-deterministic reaction of the sequential μ-chart start a thread to ensure it is a valid solution. It may not be valid because of the chart's feedback, *e.g.* if the trigger is $a \wedge \neg\, b$, the input is $\{a\}$ and b is fed back, then this transition should *not* happen. Each thread is followed by again calculating the meaning of the sequential chart, only this time we consider the meaning of the chart when applied to the original input plus any fed back output signals from the first application of the chart for this thread. If the solution originating the thread is again a reaction of the chart then we can be certain that this is indeed a valid reaction. In the case of the trigger $a \wedge \neg\, b$ we can see that the associated transition will now be ruled-out since the trigger is not true once the feedback, b, is taken into account.

Assuming $S = Seq(N, \Sigma, \sigma_d, \sigma, \delta)$ we have,

$$[\![Feedback(S, L)]\!]\, x\, \gamma\, \epsilon = \{(y, Feedback(S', L), \gamma', \epsilon') \mid (y, S', t, \gamma', \epsilon') \in [\![S]\!] x\, \gamma\, \epsilon \wedge (y, S', t, \gamma', \epsilon') \in [\![S]\!]\, (x \cup (y \cap L))\, \gamma\, \epsilon\}$$

Now to demonstrate that this function captures the intended meaning, consider the μ-chart pictured in Figure 1. Assuming the chart is in state 1 and consider an input signal set containing only the signal a, the only valid reaction we expect is that the chart moves to state 2 and outputs signal b. However, because the signal b is instantaneously fed back we must calculate the meaning of the chart a second time using the input set $\{a, b\}$. The reaction now appears to be falsely non-deterministic, *i.e.* the chart seems to be able to make a transition to state 2 or state 3 on input $\{a\}$. However, the rule for a valid solution stipulates that none of the transitions that started the thread are ruled out by feedback. Therefore the solution that allows the transition to state 3 in the second calculation of the thread is ruled out because the initial reaction of that thread (*i.e.* the transition to state 2) has been ruled out (because there is no way from 2 to 3).

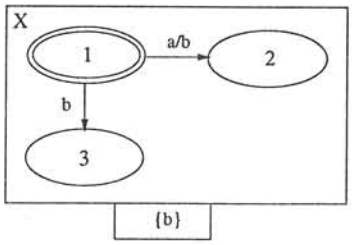

Figure 1: Example μ-Chart

As previously mentioned this example demonstrates one of the purposes of copyright signals in [2], and why they are no longer required.

Also [2, 3] describe the use of oracle signals and oracle tests that are intended to elucidate the meaning of charts with causality problems. We also discuss the use of oracle signals at length in [5]. The need for these signals and tests has been eliminated by the new semantic definitions for the feedback operator. To give an example consider the most basic pathological case of a sequential μ-chart with causality problems which has one transition labelled $\neg\ a/a$ and feedback on the signal a. Applying this chart to an empty input set means the transition is never valid, the reaction should always be empty. This is indeed the meaning denoted by the new semantic function without the need for oracles. This is because the only thread, resulting from the reaction that allows the transition labelled $\neg\ a/a$, is broken when the signal a is fed back and the transition is ruled out.

The second case covers feedback on parallel composed charts and again we apply the concept of threads. For composed charts however we start a thread for each of the possible non-deterministic behaviours of one of the components of the composition (with the same feedback applied). Each of these threads is extended by evaluating the behaviour of the other chart of the composition on the input plus any relevant feedback signals (*i.e.* those resulting from the output of the first chart for this thread). Each of the resulting non-deterministic behaviours of the second chart gives rise to another thread that describes a possible behaviour of the chart. Finally each thread is validated as a solution by again calculating the reaction of the first chart with input and feedback (for this thread) to ensure all of the transitions in the original reaction for the thread still occur. Notice the symmetrical case must also be calculated, *i.e.* starting with a thread for each possible reaction of second chart of the composition to the input.

So given $S_1 \in \mathcal{S}$ and $S_2 \in \mathcal{S}$ the semantic equation is,

$$
\begin{aligned}
&[\![Feedback(And(S_1, S_2), L)]\!]\, x\, \gamma\, \epsilon = \\
&\quad \{(y_1 \cup y_2, Feedback(And(S_1', S_2'), L), t_1 \cup t_2, \gamma_1 \cup \gamma_2, \epsilon_1 \cup \epsilon_2) \mid \\
&\qquad (y_1, Feedback(S_1', L), t_1, \gamma_1, \epsilon_1) \in [\![Feedback(S_1, L)]\!]\, x\, \gamma\, (\epsilon \restriction v(S_1)) \wedge \\
&\qquad (y_2, Feedback(S_2', L), t_2, \gamma_2, \epsilon_2) \in [\![Feedback(S_2, L)]\!]^*\, (x \cup (y_1 \cap L))\, \gamma_1\, (\epsilon \restriction v(S_2)) \wedge \\
&\qquad (y_1, Feedback(S_1', L), t_1, \gamma_3, \epsilon_1) \in [\![Feedback(S_1, L)]\!]\, (x \cup (y_2 \cap L))\, \gamma_2\, (\epsilon \restriction v(S_1)) \wedge \\
&\qquad \gamma_1 \cup \gamma_2 \in \Gamma \wedge \gamma_1 = \gamma_3 \restriction \mathrm{dom}\, \gamma_1\} \\
&\quad \cup \{(y_1 \cup y_2, Feedback(And(S_1', S_2'), L), t_1 \cup t_2, \gamma_1 \cup \gamma_2, \epsilon_1 \cup \epsilon_2) \mid \\
&\qquad (y_2, Feedback(S_2', L), t_2, \gamma_2, \epsilon_2) \in [\![Feedback(S_2, L)]\!]\, x\, \gamma\, (\epsilon \restriction v(S_2)) \wedge \\
&\qquad (y_1, Feedback(S_1', L), t_1, \gamma_1, \epsilon_1) \in [\![Feedback(S_1, L)]\!]^*\, (x \cup (y_2 \cap L))\, \gamma_2\, (\epsilon \restriction v(S_1)) \wedge \\
&\qquad (y_2, Feedback(S_2', L), t_2, \gamma_3, \epsilon_2) \in [\![Feedback(S_2, L)]\!]\, (x \cup (y_1 \cap L))\, \gamma_1\, (\epsilon \restriction v(S_2)) \wedge \\
&\qquad \gamma_2 \cup \gamma_1 \in \Gamma \wedge \gamma_2 = \gamma_3 \restriction \mathrm{dom}\, \gamma_2\}
\end{aligned}
$$

The semantic function $[\![S]\!]^*$ (where $S \in \mathcal{S}$) is a device to allow solutions where only one chart in a composition has a reaction to the input. It says that if one chart of the composition has an empty reaction set then this is synonymous with that chart remaining in the same state and giving no output. It is defined as follows:

$$[\![S]\!]^* x\, \gamma\, \epsilon = \mathit{if}\ [\![S]\!] x\, \gamma\, \epsilon = \{\}\ \mathit{then}\ \{(\{\}, S, \{\}), \gamma, \epsilon\}\ \mathit{else}\ [\![S]\!] x\, \gamma\, \epsilon$$

The third case applies to hierarchically decomposed charts with feedback. The semantic functions for

decomposed charts with feedback, like those without feedback, are defined in three cases with a separate equation for each.

The current state of the master is decomposed by a slave, *i.e.* $\rho(\sigma) \neq NoDec$, and the master changes state.

$$[\![Feedback(Dec\ S\ by\ \rho, L)]\!]\, x\, \gamma\, \epsilon = \{(y, Feedback(Dec\ S'\ by\ \rho'), t, \gamma', (\epsilon' \restriction v(S')) \cup \epsilon'') \mid \exists\, S''.S'' \in \mathcal{S} \wedge (y, Feedback(And(S', S''), L), t, \gamma', \epsilon') \in [\![Feedback(And(S, \rho(\sigma)), L)]\!]\, x\, \gamma\, \epsilon \wedge \sigma' \neq \sigma \wedge \rho' = \rho[\sigma/S'', \sigma'/init_s(\rho(\sigma')) \wedge \epsilon'' = init_\epsilon(\rho(\sigma'))]\}$$

The definition assumes the master changes state and so if the master moves to a decomposed state then that slave is initialised otherwise it is not.

The current state of the master is decomposed by a slave, *i.e.* $\rho(\sigma) \neq NoDec$, and the master does *not* change state.

$$[\![Feedback(Dec\ S\ by\ \rho, L)]\!]\, x\, \gamma\, \epsilon = \{(y, Feedback(Dec\ S'\ by\ \rho'), t, \gamma', \epsilon') \mid \exists\, S''.S'' \in \mathcal{S} \wedge (y, Feedback(And(S', S''), L), t, \gamma', \epsilon') \in [\![Feedback(And(S, \rho(\sigma)), L)]\!]\, x\, \gamma\, \epsilon \wedge \sigma' = \sigma \wedge \rho' = \rho[\sigma/S'']\}$$

The master's current state is not further decomposed, *i.e.* $\rho(\sigma) = NoDec$.

$$[\![Feedback(Dec\ S\ by\rho, L)]\!]\, x\, \gamma\, \epsilon = \{(y, Feedback(Dec\ S'\ by\ \rho', L), t, \gamma', \epsilon' \cup \epsilon'') \mid (y, Feedback(S, L), t, \gamma', \epsilon') \in [\![Feedback(S, L)]\!]\, x\, \gamma\, \epsilon \wedge \rho' = \rho[\sigma'/init_s(\rho(\sigma')) \wedge \epsilon'' = init_\epsilon(\rho(\sigma'))]\}$$

Recall that a condition on μ-charts is that the master chart in a hierarchical decomposition must be a sequential chart. Therefore the reaction of a decomposed chart, with feedback, which is in a state that has no associated slave is equivalent to the reaction of the master treated as a simple sequential chart.

Finally the fourth case deals with charts that are constructed by applying the feedback mechanism to a chart that already has feedback, *e.g.* $A = Feedback(Feedback(S, L_1), L_2)$ where $S \in \mathcal{S}$ and $L_1, L_2 \in \mathbb{P}(M)$. This case is implicitly handled in the semantics given in [2] but needs to be made explicit here because we have split the semantics of feedback charts into cases depending on the type of chart the feedback mechanism has been applied to.

$$[\![Feedback(Feedback(S, L_1), L_2)]\!]\, x\, \gamma\, \epsilon = \{(y, Feedback(S'), t, \gamma', \epsilon') \mid (y, S', t, , \gamma', \epsilon') \in [\![Feedback(S, L_1 \cup L_2)]\!]\, x\, \gamma\, \epsilon\}$$

2.8 Putting the Steps Together: Streams

The semantics above defines what happens to a chart in one step. We finally have to show how those steps are combined so that the chart can react to a sequence of inputs, producing a sequence of outputs. Here we consider input and output to be a set of signals paired with an appropriate environment that gives a value for any integer-valued signals. Following [2] we model such sequences by streams. So, the meaning of a chart together with an environment for local variables is a set of functions. Each function takes a stream of inputs (pairs containing a set of signals and an integer-valued signal environment) to a stream of outputs (which are pairs just like the inputs). The chart denotes a set of functions to take account of non-determinism—each function can be though of as describing one possible strand of behaviour through the non-deterministic state space.

What follows is fairly much verbatim from [2] (augmented by [8]), but we repeat it here for completeness sake.

Let X be a set. Then X^ω is a stream over X and is an infinite sequence of elements of X. Given $x \in X$, $s \in X^\omega$, $x\&s \in X^\omega$ is the stream over X that has x as its first element and continues as s. For any

chart S and local variable environment ϵ we have:

$$[\![(S,\epsilon)]\!]_{io} \in \mathbb{P}((\mathbb{P}(M) \times \Gamma)^\omega \to (\mathbb{P}(M) \times \Gamma)^\omega)$$

i.e. $[\![(S,\epsilon)]\!]_{io}$ is a set of *stream processing* functions.
Further:

$$[\![(S,\epsilon)]\!]_{io} =_{def} \{f \in (\mathbb{P}(M) \times \Gamma)^\omega \to (\mathbb{P}(M) \times \Gamma)^\omega \mid f((x,\gamma)\&s) = y\&g(s)\}$$

where

$$((y,S',t,\gamma',\epsilon') \in [\![S]\!]x\,\gamma\,\epsilon \vee y = \{\} \wedge S' = S \wedge [\![S]\!]x\,\gamma\,\epsilon = \{\}) \wedge g \in [\![(S',\epsilon')]\!]_{io}$$

3 Updating the μ-charts to Z Translation

In this section we examine each of the three extensions in turn.

3.1 Local Variables

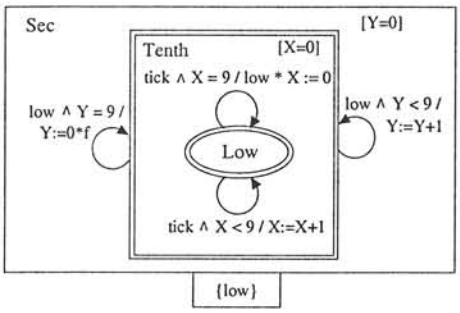

Figure 2: A simple counter

To facilitate modelling the Z translation of a μ-chart that contains local variables we make a finer distinction between describing a μ-chart and describing a state than we did in [5], remembering that a state of a sequential μ-chart may enclose or be replaced by another arbitrary μ-chart, *i.e.* we may have hierarchical decomposition. Therefore when translating a sequential μ-chart there is always one schema that describes the observations of the chart. For example the translation of the μ-chart pictured in Figure 2 gives the four state schemas Sec_c, $Tenth_s$, $Tenth_c$ and Low. The first two schemas are derived from the sequential μ-chart Sec while the second two translate the sequential μ-chart $Tenth$.

Sec_c

$c_{Sec} : \mu_{State}$
$Y : \mathbb{Z}$

$Tenth_s$

Sec_c

$c_{Sec} = tenth$

$Tenth_c$

$c_{Tenth} : \mu_{State}$
$X : \mathbb{Z}$

Low

$Tenth_c$

$c_{Tenth} = low$

The slave chart $Tenth$ now has two Z schemas to describe its role, the first $Tenth_s$ describes its role as a state of its master Sec and the second describes its the observable properties of it viewed as the slave μ-chart.

The local variables named X and Y are now associated with the μ-charts *Tenth* and *Sec* respectively. Initialisation schemas *InitSec* and *InitTenth* will be defined to describe the state that the respective μ-charts are in and what values their local variables have at initialisation. We will say more on all of this in Section 5.

InitSec

Sec_c

$c_{Sec} = tenth$

$Y = 0$

InitTenth

$Tenth_c$

$c_{Tenth} = low$

$X = 0$

The operation schemas that describe transitions now perform the appropriate manipulation on the local variable observations.

δ^1_{Tenth}

$Tenth_c$

$Tenth'_c$

$active\_ : \mathbb{P}\, \mu_{State}$

$i?, o!, o_{Tenth}! : \mathbb{P}\, Signal$

$active(sec)$

$low \in i? \cup (o! \cap f_{Tenth})$

$Y = 9$

$Y' = 0$

$o_{Tenth}! = \{f\}$

The definition of the remaining operations is omitted here for brevity.

The schema *Step* combines all the components of the specification according to the hierarchical structure of the specification.

Step

ΔSec_c

$\Delta Tenth_c$

$i?, o! : \mathbb{P}\, Signal$

$\exists\, o_{Sec}!, o_{Tenth}! : \mathbb{P}\, Signal;\ active\_ : \mathbb{P}\, \mu_{State} \bullet$

$(active(sec) \iff true) \land (active(tenth) \iff true) \land o! = o_{Sec}! \cup o_{Tenth}!$

$(\delta^1_{Tenth} \lor \delta^2_{Tenth} \lor \epsilon_{Sec} \lor Inactive_{Sec}) \land$

$(\delta^1_{Low} \lor \delta^2_{Low} \lor \epsilon_{Tenth} \lor Inactive_{Tenth})$

Notice that this example shows that no initialisation occurs in a slave when a loop transition occurs in the master as we described in Section 2.6.

3.2 Integer-valued Signals

Integer-valued signals have an environment in the semantics for μ-charts which maintains their current values. The translation of this environment however is represented by a constructor function in a Z free type describing the signals of the specification. This free type constructor adds a new signal to the set of signals for each value of the value carrying signal. For example an integer-valued signal called *Va* would be defined by the free type:

$$Signal ::= Va\langle\langle \mathbb{Z} \rangle\rangle$$

Now the expression $Va(10)$ represents a unique signal (of type *Signal*) for the value carrying signal *Va* with value 10. To write predicates involving the value of any given signal we can use the inverse of the constructor function, *i.e.* $Va^{\sim}(Va(10)) = 10$ (where $^{\sim}$ is the Z relational inverse operator).

One other restriction that we must make to ensure that the translation respects the semantics is to ensure that at any step of the system there is at most one value in the output representing any particular value carrying signal, *i.e.* because there is a different signal for each value of a value carrying signal, having more than one of these present in the output would mean having several values for the same integer-valued signal. This restriction is enforced by defining a new, more restricted type for any set representing input and output signals. Assuming a specification has the signals in the set *Signal* then we define a new set IO so that:

$$IO : \mathbb{P}\,\mathbb{P}\,Signal$$
$$IO = \{a : \mathbb{P}\,Signal;\ f : \mathbb{Z} \rightarrow Signal;\ x, y : \mathbb{Z} \mid f\,x \in a \land f\,y \in a \implies x = y \bullet a\}$$

So, IO contains all those subsets of *Signal* which satisfy the required uniqueness condition on value carrying signals, namely that in any given element of IO there is at most one occurrence of any given value carrying signal.

An operation schema takes the form,

δ
$$From$$
$$To'$$
$$active\_ : \mathbb{P}\,\mu_{State}$$
$$i?, o!, o_{Tenth}! : IO$$

$$active(Chart)$$
$$(\exists x : \mathbb{Z} \bullet c(x) \in i? \cup (o! \cap f_{Tenth}) \land x < 9)$$
$$o_{Chart}! = \{c(0), d(10)\}$$

then the *Step* schema is,

Step
$$\Delta Chart$$
$$i?, o! : IO$$

$$\exists\, o_{Chart}!, \ldots : IO;\ active\_ : \mathbb{P}\,\mu_{State} \bullet (active(chart) \iff true) \land o! = o_{Chart}! \cup \ldots \land \delta \lor \ldots\ldots$$

The example of Section 10 gives a fuller demonstration of the treatment of integer-valued signals. Indeed we see that they can be extended to value-carrying signals where the values can come from any Z definable type.

3.3 Command Language

The command language has the syntax given in Section 2.1. The allowable expressions on transitions, *i.e.* members of the set *Com*, translate directly into Z predicates. Recall the syntax for commands is given by the grammar,

$$a ::= n \mid Y \mid a\ binop\ a \mid s_v$$
$$c ::= skip \mid Y := a \mid s_o \mid s_v := a \mid c\ ;\ c$$

Obviously the arithmetic expressions, described by a, translate almost directly into Z arithmetic expressions where $n \in \mathbb{Z}$, Y becomes a label of an observation of the μ-chart's local variable. We assume *binop* to be a proper subset of the binary operators of Z and s_v will become an expression of the form

$zs_v^{\sim}(zs)$ where zs_v is a Z constructor function, $^{\sim}$ is the Z functional inverse operator and zs is some signal with a related value, *i.e.* $zs_v^{\sim}(zs)$ gives the value associated with zs.

Now we translate each of the commands. The command *skip* obviously translates to the predicate *true* which, in the translated Z, requires no extra predicates in the operation schema describing the transition. The command $Y := a$ is translated to the predicate $Y' = a$ where Y' is the observation label Y decorated to represent an after transition value. The command s_o means that the signal s_o is output therefore this command is translated by adding the signal s_o to the component output set in the operation schema *e.g.* $o_{Sec} = \{s_o\}$. Outputting a integer-valued signal $s_v := a$ is similar, *e.g.* $o_{Sec} = \{s_v(a)\}$. The translation of the parallel composition of two commands $c1^*c2$ is translated by applying the given rules (possibly including this one recursively) to each component command $c1$ and $c2$. If $c1$ and $c2$ translate to predicates they are added in conjunction to the operation schema's predicate, *e.g.* $X := X + 1^* Y := s_v$ becomes $X' = X + 1 \land Y' = zs_v^{\sim}(zs)$[4].

Something worth noting is the condition that if an integer-valued signal appears in the predicates generated from the translation then the signal must have appeared positively in the transition trigger, therefore there must also be a predicate in this schema of the form $\exists x : \mathbb{Z} \bullet s_v(x) \in i \cup (o! \cap f)$.

4 Putting it all together in Z: the counterpart to streams

In the Z semantics we do a little more schema calculus as the counterpart to the stream semantics above in Section 2.8.

To give a meaning in the Z for the chart S we have the *Step* schema as the counterpart to $[\![S]\!]$, so this tells us how the chart moves "in one single instant of time" (as Philipps and Scholz put it).

First, note that *Step* not only contains observations of the state information, but also of the input and output for the step and of the the local variables (if any).

By analogy with the stream semantics (Section 2.8) we use schema composition to model the flow of information concerning states and local variables from step to step. In the stream model these are handled by passing the new state and local variable values around in the correct way via the link between the f and g functions. The inputs and outputs at each step are, respectively, taken from the input stream and placed on the (resultant) output stream. This is what we need to model in the Z.

For each component of the input stream we provide a corresponding use of *Step*. The input set at each place in the input stream is fed to the corresponding use of *Step* by hiding its input observation and equating it to the relevant input value. We also need to hide the output observation since it is not meant to be communicated from step to step (unlike the communicated and so unhidden state and local variable observations). So, given the stream

$$(x_1, \gamma_1)\&(x_2, \gamma_2)\&...$$

we provide the Z

$$(\exists\, i?, o! : \mathbb{P}\, Signal \bullet [Step \mid i? = x_1] \fatsemi (\exists\, i?, o! : \mathbb{P}\, Signal \bullet [Step \mid i? = x_2]) \fatsemi ...$$

Note that the ϵs and γs do not appear here. This is simply because we have modelled these environments directly in the Z—essentially via values in each state for the local variables and via the free type for signals in the case of value-carrying signals: recall Sections 3.1 and 3.2.

In the examples that follow we use a notational extension from Z/EVES [6, 1] which abbreviates $\exists x : T \bullet [S \mid x = e]$ to $S[x := e]$.

5 The Method Change

Recall in Section 3.1 we described the need to make a finer distinction between describing a μ-chart and describing a state. Here we continue this progression and describe each sequential μ-chart independently. Previously the structure of the specification (*i.e.* the composition in the specification) was intricately represented throughout the resulting Z specification. By abstracting on the possibility that a sequential μ-chart,

[4] The restriction of commands by the *weak Bernstein condition* ensures that we will not generate false predicates via the translation.

used as a slave decomposing another chart, can be inactive we can describe each sequential μ-chart separately and then combine them in the required manner at the end of the specification process.

Another advantage of this more modular approach is reuse, that is we may wish to use a particular sequential μ-chart again in subsequent specifications. Because the sequential chart is now translated without regard to its eventual relationship to others in the specification it can easily be slotted into any arbitrary specification that requires its described functionality.

For example consider the μ-chart specification pictured in Figure 3 which describes a simplified counter that counts ten "ticks".

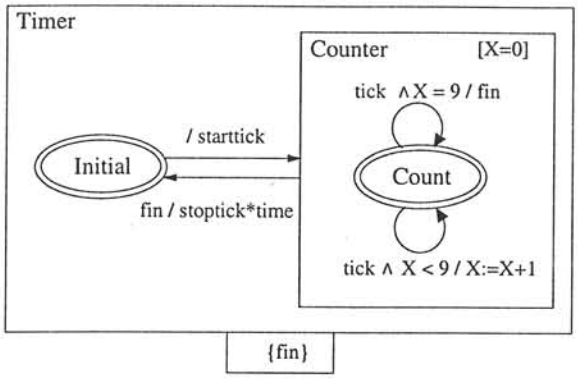

Figure 3: A simple counter

The Z translation for this μ-chart is now as follows.

Z Section *Declarations* imports: *toolkit*

$$\mu_{State} ::= timer \mid initial \mid counter \mid count$$
$$Signal ::= starttick \mid fin \mid stoptick \mid time \mid tick$$

We use the Z section mechanism to distinguish the Z description of individual sequential μ-charts. The type μ_{State} must contain values to represent all states of the eventual specification, and the set *Signal* contains values to represent all the signals. We can imagine that these types (and in fact this Z section) are built up as the specification is formed by joining the required sequential charts.

Firstly we give the translated Z for the μ-chart *Counter*. Notice that we can now, because of the modularity we have introduced, describe the chart by describing each component sequential chart separately and we can do this in any order we like rather than being constrained to a top down approach as previously.

The main innovation follows from acknowledging in the Z translation that states in a chart are of two sorts: atomic states or states which contain charts. So, in the present example *Counter* is a state of the chart *Timer and* itself contains a chart (containing the state *Count*, which is an atomic state). This situation is, arguably, made confusing because there is a convention that the distinction between a state and the chart it contains is often ignored.[5]

This is presented in the Z by having, for each state which is not atomic, two schemas. One, with its name subscripted by 'c', represents the chart contained in the state (via the convention this is viewing the state *qua* chart), hence the 'c'; the other, with its name subscripted by 's', represents the state which contains the chart (via the convention the state *qua* state).

In the former case, the schema just declares the observations we can make of the state *qua* chart, namely what state it is in and what its local variables (if any) are.

[5]So, we often find ourselves thinking of a state having two aspects: sometimes it looks (say "from a suitably abstracting distance") like a state and sometimes ("when we are close enough") it looks like a chart. This somewhat metaphysical duality is just a product of forgetting the distinction above and the fact that we use the stated convention.

In the latter case, the schema for state S includes the schema for its master *qua* chart and fixes the value of the state observation to be s, *i.e.* the Z name for the chart state S under our usual naming convention.

For each state *qua* chart we also have an initialisation schema which includes the schema for the master *qua* chart and fixes the value of the state observation to be the (conventional) name for the chart's initial state and also initialises any local variables.

All this can be seen at work below (and was briefly referred to above in Section 3.1). *Counter* is a state *qua* chart, so there are two schemas: $Counter_c$ (which declares $c_{Counter}$, the state of *Counter qua* chart, and the local variable X), $Init_{Counter}$ (which sets the initial values) and (later on in the section or the *Timer*) $Counter_s$ (which describes *Counter qua* state).

The outcome of all this (which has a certain naturalness once the details have been come to terms with) is, as we have mentioned before, a far more modular style for the translation, and we gain all the usual benefits of modularity from doing it.

Z Section *Counter* imports: *Declarations*

syntax *active prerel*

$Counter_c$

$c_{Counter} : \mu_{State}$
$X : \mathbb{Z}$

$Count_s$

$Counter_c$

$c_{Counter} = count$

$Init_{Counter}$

$Counter_c$

$c_{Counter} = count$
$X = 0$

Before we go on to give the transition descriptions we must define a constant to represent the feedback behaviour of the chart we are describing. The actual value of the feedback set cannot be determined yet because it is dependent on the structure of the eventual specification. Therefore we define the set but do not determine its value.

$| f_{Counter} : \mathbb{P}\, Signal$

$\delta^1\, Count$

$\Delta Count_s$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Counter}!, o! : \mathbb{P}\, Signal$

$active(counter)$
$tick \in i? \cup (o! \cap f_{Counter})$
$X < 9$
$X' = X + 1$
$o_{Counter}! = \{\}$

$\delta^2\, Count$

$\Delta Count_s$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Counter}!, o! : \mathbb{P}\, Signal$

$active(counter)$
$tick \in i? \cup (o! \cap f_{Counter})$
$X = 9$
$o_{Counter}! = \{fin\}$

$\epsilon_{Counter}$

$\Delta Counter_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Counter}!, o! : \mathbb{P}\, Signal$

$active(counter)$
$\neg\, (Count_s \wedge tick \in i? \cup (o! \cap f_{Counter}) \wedge X < 9)$
$\neg\, (Count_s \wedge tick \in i? \cup (o! \cap f_{Counter}) \wedge X = 9)$
$c'_{Counter} = c_{Counter}$
$X' = X$
$o_{Counter}! = \{\}$

The abstraction on the "activeness" of this chart is represented by the function *active* which is a new observation of the μ-chart *Counter*. In Z Boolean functions are modelled using a set and set membership, hence the type of *active* is a set of members of type μ_{State}. Also the underscore is a convention that allows the set membership predicate $counter \in active$ to be written as a unary function application, *i.e.* $active(counter)$. Therefore the μ-chart *Counter* can be assumed to be active when *counter* is in the set active.

It is tempting to want to replace the predicates such as $\neg\ (Count_s \wedge tick \in i? \cup (o! \cap f_{Counter}) \wedge X < 9)$ with the predicate $\neg\ (\mathrm{pre}\ \delta^1{}_{Count})$ because the latter predicate seems to more naturally convey the intended meaning, that is this operation is valid only when no other operations preconditions are true, *i.e.* no valid transition can happen. However, this would be incorrect because of the way in which pre-conditions are calculated in Z, *i.e.* existential quantification of all output and after observations of the operation. Allowing this quantification by using the predicate $\neg\ (\mathrm{pre}\ \delta^1{}_{Count})$ would mean that the output observation $o!$ on which the precondition depends would not be suitably constrained to the output of the specification as it is when using the predicate $\neg\ (Count_s \wedge tick \in i? \cup (o! \cap f_{Counter}) \wedge X < 9)$. This would introduce unintended non-determinism between making no transition and making the correct transition.

$inact_{Counter}$

$\Delta Counter_c$

$active\_ : \mathbb{P}\, \mu_{State}$

$o_{Counter}! : \mathbb{P}\, Signal$

$\neg\ active(counter)$

$o_{Counter}! = \{\}$

The operation schema $inact_{Counter}$ describes what happens when this chart is deemed inactive, *i.e.* its master chart is not in the state that contains this chart or its master chart itself is inactive. A chart is deemed inactive for now when its name is not in the set *active*, *i.e.* $\neg\ active(counter) = true$.

As we discussed in Section 2.6 we have also taken a slightly different approach to specifying what happens to a μ-chart's configuration when it is inactive. In the previous translation method we included the initialisation of the μ-chart into its inactive schema which meant technically it was being initialised at every step while it was inactive. Here, as with the new semantics, we have taken the approach that we don't care what the chart's configuration is when it is inactive and therefore we say nothing about its configuration in its inactive schema. It is still the case however that the chart is initialised each time it is entered.

The overall transition behaviour for the chart *Counter* is now described, therefore we would expect that the precondition of the disjunction of the possible behaviours be true for all configurations of the chart and values of "activeness", input, feedback and the local variable X. This can be checked using Z/EVES.

$try \forall c_{Counter} : \mu_{State};\ active\_ : \mathbb{P}\, \mu_{State};\ i? : \mathbb{P}\, Signal;\ X : \mathbb{Z} \bullet$
$\mathrm{pre}\ \delta^1{}_{Count} \vee \mathrm{pre}\ \delta^2{}_{Count} \vee \mathrm{pre}\ \epsilon_{Counter} \vee \mathrm{pre}\ inact_{Counter};$

prove by reduce; *instantiate* $X\_0' == 0, c'_{Counter} == count$;
prove by reduce; *instantiate* $o! == \{tick\}$; *prove by reduce*;
instantiate $o! == \{tick\}, X' == 9$; *prove by reduce*;
instantiate $o! == \{\}$; *prove by reduce*; *split* $counter \in (active\_)$;
cases; *prove by reduce*; *next*; *prove by reduce*; *next*;

$\longrightarrow$ *true*

Note how the modularity allows us to investigate *Counter* without reference to any chart in which it may be included.

The Z translation for the μ-chart *Timer* is given as follows.

Z Section *Timer* **imports:** *Declarations*

syntax *active prerel*

$Timer_c$

$c_{Timer} : \mu_{State}$

$Initial_s$

$Timer_c$

$c_{Timer} = initial$

$Counter_s$

$Timer_c$

$c_{Timer} = counter$

$Init_{Timer}$

$Timer_c$

$c_{Timer} = initial$

$f_{Timer} : \mathbb{P}\, Signal$

$\delta_{InitialCounter}$

$Initial_s$
$Counter'_s$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$o_{Timer}! = \{starttick\}$

$\delta_{CounterInitial}$

$Counter_s$
$Initial'_s$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$fin \in i? \cup (o! \cap f_{Timer})$
$o_{Timer}! = \{stoptick, time\}$

ϵ_{Timer}

$\Delta Timer_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$\neg\ Initial_s$
$\neg\ (Counter_s \wedge fin \in i? \cup (o! \cap f_{Timer}))$
$c'_{Timer} = c_{Timer}$
$o_{Timer}! = \{\}$

$inact_{Timer}$

$\Delta Timer_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}! : \mathbb{P}\, Signal$

$\neg\ active(timer)$
$o_{Timer}! = \{\}$

Once all charts in the specification are given, we define some machinery to combine the individual parts into a whole specification.

Z Section *Step* **imports:** *Counter*, *Timer*, *Declarations*

$l_{Counter} : \mathbb{P}\, Signal$
$l_{Timer} : \mathbb{P}\, Signal$

$l_{Counter} = \{\}$
$l_{Timer} = \{fin\}$
$f_{Timer} = l_{Timer}$
$f_{Counter} = l_{Counter} \cup f_{Timer}$

The constant $l_{Counter}$ describes the local feedback signals for the sequential μ-chart *Counter*. The value of this constant is read directly from the feedback box attached to the bottom of the graphical representation of the chart where no box is equated to the empty set of local feedback signals.

The set f_{Timer} and $f_{Counter}$ that describe the feedback behaviour of the charts in the specification, as previously mentioned, are made concrete by incorporating the effect of specification structure.

Finally the schema *Step* hides the "activeness" of the specifications charts and leaves the same model of the system that we had originally.

Step

$$\Delta Timer_c$$
$$\Delta Counter_c$$
$$i?, o! : \mathbb{P}\, Signal$$

$$\exists\, o_{Timer}!, o_{Counter}! : \mathbb{P}\, Signal;\ active\_ : \mathbb{P}\, \mu_{State} \bullet$$
$$(active(counter) \iff c_{Timer} = counter) \land$$
$$(active(timer) \iff true) \land$$
$$o! = \bigcup \{o_{Timer}!, o_{Counter}!\} \land$$
$$(\delta_{InitialCounter} \land Init'_{Counter} \lor \delta_{CounterInitial} \lor \epsilon_{Timer} \lor inact_{Timer}) \land$$
$$(\delta^1{}_{Count} \lor \delta^2{}_{Count} \lor \epsilon_{Counter} \lor inact_{Counter})$$

The schema *Step* defines the overall output with regard to the way in which each sequential μ-chart is combined in the specification and hides, via existential quantification, the component outputs. The conditions under which each chart is active are also defined in the predicate of *Step*. Notice the sequential chart *Timer* is always active because in this specification *Timer* is the top level chart. When considering the structure of the overall specification we need to ensure that whenever the transition from state *Initial* to state *Counter* is made the μ-chart *Counter* is initialised hence the predicate $\delta_{InitialCounter} \land Init'_{Counter}$. The schema *Step* here has the same properties as equivalent schemas in [5] and therefore can be investigated in the same fashion.

The next sections go on to demonstrate by example that resulting *Step* schemas from the new translation method are in fact equivalent to the *Step* schema from the previous translation method.

5.1 Sequential μ-Charts

This section uses a simple sequential μ-chart (Figure 4) to demonstrate that the new transition method is equivalent to the old for sequential charts.

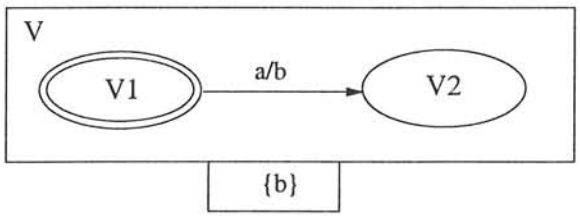

Figure 4: A simple sequential chart

Firstly we give the Z resulting from the old translation method.

$$\mu_{State} ::= v \mid v1 \mid v2$$
$$Signal ::= a \mid b$$

V_c

$$c_V : \mu_{State}$$

InitV

$$V_c$$

$$c_V = v1$$

V1

$$V_c$$

$$c_V = v1$$

$V2$

V_c

$c_V = v2$

$l_V == \{b\}$

$f_V == l_V$

Output

$o_V, o! : \mathbb{P}\{b\}$

$o! = o_V$

δ_{V1V2}

$V1$

$V2'$

Output

$i? : \mathbb{P}\, Signal$

$a \in i? \cup (o! \cap f_V)$

$o_V = \{b\}$

ϵ_V

$c_V, c'_V : \mu_{State}$

$i? : \mathbb{P}\, Signal$

Output

$\neg\, (V1 \wedge a \in i? \cup (o! \cap f_V))$

$c'_V = c_V$

$o_V = \{\}$

Step

$c_V, c'_V : \mu_{State}$

$i?, o! : \mathbb{P}\, Signal$

$\exists\, o_V : \mathbb{P}\, Signal \bullet$
$(\delta_{V1V2} \vee \epsilon_V)$

The Z resulting from the new modular translation is as follows. Notice because the example chart is so simple the Z appears very similar, however we have decorated all of the schema names with an 'n' in the new Z so that we can use the Z/EVES theorem proving tool to ensure the result we require.

Z Section *Declarations* imports: *toolkit*

$\mu_{State} ::= v \mid v1 \mid v2$

$Signal ::= a \mid b$

Z Section V imports: *main*

syntax *active prerel*

V_{cn}

$c_V : \mu_{State}$

$Init V_n$

V_{cn}

$c_V = v1$

$V1_n$

V_{cn}

$c_V = v1$

$V2_n$

V_{cn}

$c_V = v2$

$\mid f_{vn} : \mathbb{P}\, Signal$

δ_{vn}

$V1_n$

$V2'_n$

$i? : \mathbb{P}\, Signal$

$active\_ : \mathbb{P}\, \mu_{State}$

$o_V!, o! : \mathbb{P}\, Signal$

$active(v)$

$a \in i? \cup (o! \cap f_{vn})$

$o_V! = \{b\}$

ϵ_{vn}

ΔV_{cn}

$i? : \mathbb{P}\, Signal$

$active\_ : \mathbb{P}\, \mu_{State}$

$o_V!, o! : \mathbb{P}\, Signal$

$active(v)$

$\neg\, (V1_n \wedge a \in i? \cup (o! \cap f_{vn}))$

$c'_V = c_V$

$o_V! = \{\}$

$Inact_{vn}$

ΔV_{cn}
$active\_ : \mathbb{P}\, \mu_{State}$
$o_V! : \mathbb{P}\, Signal$

$\neg\ active(v)$
$o_V! = \{\}$

Z Section *Step* **imports:** V

$l_{vn} : \mathbb{P}\, Signal$

$l_{vn} = \{b\}$
$f_{vn} = l_{vn}$

$Step_n$

ΔV_{cn}
$i?, o! : \mathbb{P}\, Signal$

$\exists\, o_V! : \mathbb{P}\, Signal;\ active\_ : \mathbb{P}\, \mu_{State} \bullet$
$(active(v) \iff true) \land$
$o! = o_V! \land$
$(\delta_{vn} \lor \epsilon_{vn} \lor Inact_{vn})$

Generally the Z section used here to encapsulate the translation of the chart V would have, as a parent, the section *Declarations*. However in this particular case we want to prove properties between the new and old translation and the Z of the old translation already defines the constants, *i.e.* μ_{State} and *Signal* that would otherwise be included from the section *Declarations*.

Using Z/EVES we can now show that the resulting step schemas from each of the translations (new and old) are equivalent.

try $Step = Step_n$;

apply extensionality; *prove by reduce*; *cases*; *apply* $Step\$member$;
prove by reduce; *instantiate* $(active\_) == \{v\}$; *prove by reduce*;
withnormalizationprovebyreduce; *next*; *apply* '$Step_n\$member$';
prove by reduce; *instantiate* $o_V == \{b\}$; *prove by reduce*;
withnormalizationprovebyreduce; *next*;

$\longrightarrow$ *true*

5.2 Decomposed μ-Charts

In this section we give an example to show that the translation semantics of μ-chart specifications that contain hierarchical decomposition are unchanged by the modularisation of the translation method. The example specification, pictured in Figure 5, is an extension to the sequential specification given in Section 5.1 above. Here we describe a system that carries out the process (admittedly trivial) described by the μ-chart of Figure 4 each time a signal *tick* is present.

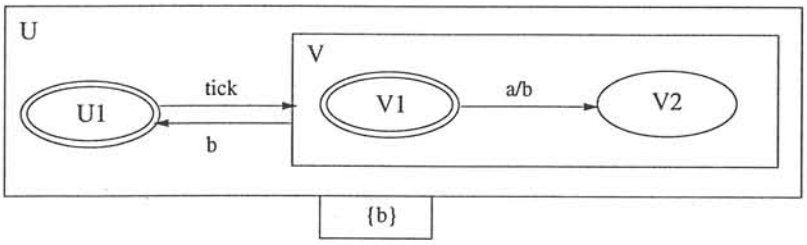

Figure 5: A simple sequential chart

We give the Z for the old translation method in full because the schemas representing transitions change. This demonstrates the most obvious disadvantage of the old translation method.

$\mu_{State} ::= v \mid v1 \mid v2 \mid u \mid u1$
$Signal ::= a \mid b \mid tick$

U_c
$c_u : \mu_{State}$

$InitU$
U_c

$c_u = u1$

$U1$
U_c

$c_u = u1$

V
U_c

$c_u = v$

V_c
$c_V : \mu_{State}$

$InitV$
V_c

$c_V = v1$

$V1$
V_c

$c_V = v1$

$V2$
V_c

$c_V = v2$

$l_u == \{b\}$
$f_u == l_u$
$l_V == \{\}[Signal]$
$f_V == l_V \cup f_u$

$Output$
$o_u, o_V, o! : \mathbb{P}\{b\}$

$o! = o_u \cup o_V$

δ_{u1v}
$U1$
V'
$Output$
$i? : \mathbb{P}\, Signal$

$tick \in i? \cup (o! \cap f_u)$
$o_u = \{\}$

δ_{vu1}
V
$U1'$
$Output$
$i? : \mathbb{P}\, Signal$

$b \in i? \cup (o! \cap f_u)$
$o_u = \{\}$

ϵ_u
$c_u, c'_u : \mu_{State}$
$i? : \mathbb{P}\, Signal$
$Output$

$\neg\, (U1 \wedge tick \in i? \cup (o! \cap f_V))$
$\neg\, (V \wedge b \in i? \cup (o! \cap f_V))$
$c'_u = c_u$
$o_u = \{\}$

δ_{V1V2}
V
$V1$
$V2'$
$Output$
$i? : \mathbb{P}\, Signal$

$a \in i? \cup (o! \cap f_V)$
$o_V = \{b\}$

ϵ_V
V
$c_V, c'_V : \mu_{State}$
$i? : \mathbb{P}\, Signal$
$Output$

$\neg\, (V1 \wedge a \in i? \cup (o! \cap f_V))$
$c'_V = c_V$
$o_V = \{\}$

$inact_V$
$c_u, c_V, c'_V : \mu_{State}$
$InitV'$
$Output$

$\neg\, V$
$o_V = \{\}$

Step

$c_u, c'_u, c_V, c'_V : \mu_{State}$
$i?, o! : \mathbb{P}\, Signal$

$\exists\, o_u, o_V : \mathbb{P}\, Signal \bullet$
$(\delta_{u1v} \lor \delta_{vu1} \lor \epsilon_u) \land (\delta_{V1V2} \lor \epsilon_V \lor inact_V)$

Giving the new translation of the μ-chart of Figure 5 is simpler because we can use the Z section containing the translation for the chart V that has already been given in section 5.1.

The chart U that will be decomposed by V is defined as follows. Again we would generally give a Z section called *Declarations* that would define the constants μ_{State} and *Signal*, only in this case they are already given as part of the old translation.

Z Section U **imports:** *main*

syntax *active prerel*

U_{cn}

$c_u : \mu_{State}$

$InitU_n$

U_{cn}

$c_u = u1$

$U1_n$

U_{cn}

$c_u = u1$

V_n

U_{cn}

$c_u = v$

$| f_{un} : \mathbb{P}\, Signal$

δ_{uv}

$U1_n$
V'_n
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_u!, o! : \mathbb{P}\, Signal$

$active(u)$
$tick \in i? \cup (o! \cap f_{un})$
$o_u! = \{\}$

δ_{vu}

V_n
$U1'_n$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_u!, o! : \mathbb{P}\, Signal$

$active(u)$
$b \in i? \cup (o! \cap f_{un})$
$o_u! = \{\}$

ϵ_{un}

ΔU_{cn}
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_u!, o! : \mathbb{P}\, Signal$

$active(u)$
$\neg\, (U1_n \land tick \in i? \cup (o! \cap f_{un}))$
$\neg\, (V_n \land b \in i? \cup (o! \cap f_{un}))$
$c'_u = c_u$
$o_u! = \{\}$

$Inact_{un}$

ΔU_{cn}
$active\_ : \mathbb{P}\, \mu_{State}$
$o_u! : \mathbb{P}\, Signal$

$\neg\, active(u)$
$o_u! = \{\}$

Z Section *Step* **imports:** U, V

$$
\begin{array}{|l}
l_{un} : \mathbb{P}\, Signal \\
l_{vn} : \mathbb{P}\, Signal \\
\hline
l_{un} = \{b\} \\
l_{vn} = \{\} \\
f_{un} = l_{un} \\
f_{vn} = l_{vn} \cup f_{un}
\end{array}
$$

$Step_n$

$$
\begin{array}{|l}
\Delta U_{cn} \\
\Delta V_{cn} \\
i?, o! : \mathbb{P}\, Signal \\
\hline
\exists\, o_u!, o_V! : \mathbb{P}\, Signal;\ active\_ : \mathbb{P}\, \mu_{State} \bullet \\
\quad (active(u) \iff true) \land (active(v) \iff c_u = v) \land \\
\quad o! = o_u! \cup o_V! \land \\
\quad (\delta_{vn} \lor \epsilon_{vn} \lor Inact_{vn}) \land \\
\quad (\delta_{uv} \land InitV'_n \lor \delta_{vu} \lor \epsilon_{un} \lor Inact_{un})
\end{array}
$$

Now when we attempt to show that both of the translations give an equivalent Z model for the μ-chart that we are describing we find that the predicate $Step = Step_n$ does not hold for the decomposition case. The reason it does not hold is because of the change, as already mentioned, in the way that we have represented an inactive chart in the new modular translation model, *i.e.* we have chosen to say that when a chart is inactive we do not care about its configuration, which can therefore be any value in its associated type, only that the chart gives no output. In fact we can show that $Step \subseteq Step_n$ as follows.

try $Step \subseteq Step_n$;

prove by reduce; *apply inPower*; *apply Step$member*; *prove by reduce*;
split $c_u = v$; *cases*; *prove by reduce*; *instantiate* $(active\_) == \{u, v\}$;
prove by reduce; *split* $c'_V = v1$; *cases*; *prove by reduce*;
with normalization prove by reduce; *next*; *prove by reduce*;
split $c'_u = u1$; *cases*; *prove by reduce*; *next*;
prove by reduce; *next*; *prove by reduce*;
instantiate $(active\_) == \{u\}$; *prove by reduce*; *next*;

$\longrightarrow$ *true*

This implies that any system that implements the machine described by the schema *Step* is an acceptable implementation of the machine described by $Step_n$, *i.e.* is there is a refinement relationship. In fact if we disallow all bindings of $Step_n$ where the chart V is inactive and has a configuration other than its initial we can again prove the sets equal.

$SubStep_n \mathrel{\widehat=} [Step_n \mid c_u \neq v \implies c'_V = v1]$

try $SubStep_n = Step$;

apply extensionality; *prove by reduce*; *cases*; *apply* '$SubStep_n$\$*member*';
prove by reduce; *split* $c_u = u1$; *cases*; *prove by reduce*;
with normalization prove by reduce; *next*; *split* $c_u = v \wedge c'_u = v$; *cases*;
prove by reduce; *next*; *prove by reduce*; *split* $c'_u = u1$; *cases*;
prove by reduce; *with normalization prove by reduce*; *next*;
prove by reduce; *next*; *apply Step*\$*member*; *prove by reduce*;
split $c_u = u1$; *cases*; *prove by reduce*; *instantiate* $(active\_) == \{u\}$;
prove by reduce; *with normalization prove by reduce*; *next*;
split $c_u = v \wedge c'_u = v$; *cases*; *prove by reduce*;
instantiate $(active\_) == \{u, v\}$; *prove by reduce*;
with normalization prove by reduce; *next*; *split* $c'_u = u1$; *cases*;
prove by reduce; *instantiate* $(active\_) == \{u, v\}$; *prove by reduce*;
with normalization prove by reduce; *next*; *prove by reduce*;
instantiate $(active\_) == \{u\}$; *prove by reduce*; *next*;

$\longrightarrow$ *true*

6 General remarks on the various semantics

Here we summarise the major differences between the PS semantics (as we will call the Philipps and Scholz semantics in [2]), our denotational semantics RRd and our Z semantics RRz.

One significant difference is the treatment of loop transitions in a decomposed chart. In particular those attached to a state that is decomposed by a slave. The PS semantics says that this type of loop transition causes the slave to initialise while RRd and RRz say they do not.

Further PS says a slave is initialised when it is left (by any transition including loops) whereas both of our semantics say that a slave is only initialised when the state containing the slave is entered due to a state change in the master, *i.e. not* including loop transitions.

The PS stream semantics gives a meaning to a chart S which is *insensitive* to the idea of initial states. In particular, if the sequential chart S has a decomposed state D that is initially not in state D (so has the form $Dec\ S\rho$ where $\rho(D) = T$, say) then, the first time S changes state into D, the state of T we are then in is completely up to whoever wrote down S in the first place, *i.e.* T can be in any of its states (not just its initial state, T_d). (At all subsequent times, whenever D is entered, T *will* be in its initial state because PS ensures that when a sub-chart is left it is in its initial state.)

Contrast this with RRd, given the same chart S, when it changes state[6] into D for the first time, T will be in its initial state T_d, by definition. Hence, there are no solutions using RRd where T is not in its initial state. (Again, at all subsequent times, whenever D is entered, T will be in its initial state because RRd ensures that when a sub-chart is entered (from another state) it is in its initial state.)

Finally, consider RRz, given the same chart S. We model it moving to D by considering the step predicate specialised so that the primed observation of S's state has value d[7], and the primed observation of T's state has as value the state we want T to be in when it is entered for the first time.

Now, if this value is *not* t_d, which could be the case if we are following PS, then the presence of T's initialisation schema in RRz would ensure that no solutions were possible here since the predicate which determines solutions would be false. On the other hand, if the value for T's primed state observation were t_d then a solution would be possible and it would be the same solution as we get in RRd.

It turns out that RRz and RRd differ when other examples are considered. We mentioned this in [5]. The most striking example is the chart in Figure 6.

[6] We are careful here not to say "makes a transition" because a loop transition is a special case that does not change the state.

[7] Note we're using our usual naming convention for the Z representation here.

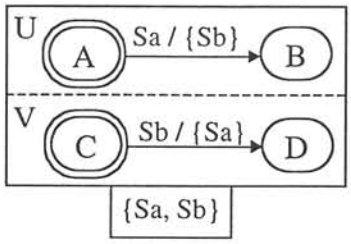

Figure 6: A pathological case

Here, *RRd* tells us that the chart does nothing given no input, whereas *RRz* leaves it non-deterministic as to whether the chart does nothing or performs both transitions similtaneously (and in this it agrees with the semantics in [7]).

7 Conclusions

In this paper we have presented an alternative semantics for the Statecharts variant μ-charts first presented in [2]. The alternative does not require the use of copyright signals or oracles. This new semantics was extended to add local variables, integer-valued signals and a transition command language based on [8] and [7]. One subtle difference in the semantics is the way in which we consider slave charts, in hierarchical decompositions, when the master makes a transition away from that slave. As opposed to the original semantics where the slave is initialised as it is left we have chosen to say that we no longer say anything about the slave after it is left.

The motivation for adding the local variables, integer-valued signals and command language was to both improve the scalability and usability of μ-charts and to allow us to model reactive systems that rely on value carrying input from their environment, *i.e.* make μ-charts applicable to a more diverse range of specification problems .

The μ-chart to Z translation method presented in [5] is also updated to reflect the changed semantics and to more thoroughly utilise the modularisation principles inherent in the μ-chart formalism.

Advantages of following a more modular approach include: the ability to reuse parts of the translated specification in subsequent specifications; that the modular approach is more supportive of collaboration between several specifiers; the scope of local variables is fixed during the translation ensuring they are used correctly, *i.e.* attempting to use local variables that are out of scope causes typing errors in the translated Z.

We have extensively explored μ-charts (some of which is written-up in our referenced papers) and considered different ways of interpreting them. We feel we are now ready to fix on *RRz* as our interpretation.

Acknowledgement

We would like to thank Doug Goldson for his time listening to our thoughts on μ-charts and Z, helping us to see where we were making mistakes and giving us some interesting problems to work on.

As ever, all the mistakes that (no doubt) remain are our own.

References

[1] The ORA web site is `http://www.ora.on.ca`.

[2] 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.

[3] J. Philipps and P. Scholz. Formal verification and hardware design with statecharts. In Bernhard Moller and J. V. Tucker, editors, *Prospects for hardware foundations: ESPRIT Working Group 8533: NADA–new hardware design methods, survey chapters*, volume 1546 of *Lecture Notes in Computer Science*, pages 356–389. Springer-Verlag, 1998.

[4] Greg Reeve and Steve Reeves. μ-Charts and Z: Examples and extensions. In *Proceedings of APSEC2000*. IEEE Computer Society, 2000. To appear.

[5] Greg Reeve and Steve Reeves. μ-Charts and Z: Hows, whys and wherefores. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *Integrated Formal Methods 2000: Proceedings of the 2nd. International Workshop on Integrated Formal Methods*, LNCS 1945. Springer-Verlag, 2000.

[6] 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.

[7] P. Scholz. *Design of Reactive Systems and their Distributed Implementation with Statecharts*. PhD thesis, Institut für Informatik, Technische Universität München, August 1998. TUM-I9821.

[8] Peter Scholz. An extended version of mini-statecharts. Technical Report TUM-I9628, Technische Univerität München, 1996.

8 Appendix A: A central locking system example

In Figure 7 we give an example of a μ-chart taken from [2]. It specifies the central locking system for a car and considers, amongst other things, how such a system should react in the case of a crash. The system as specified should unlock all the doors if a crash happens. This specification allows the possibility of undesirable behaviour when a crash happens while the motors are locking the doors.

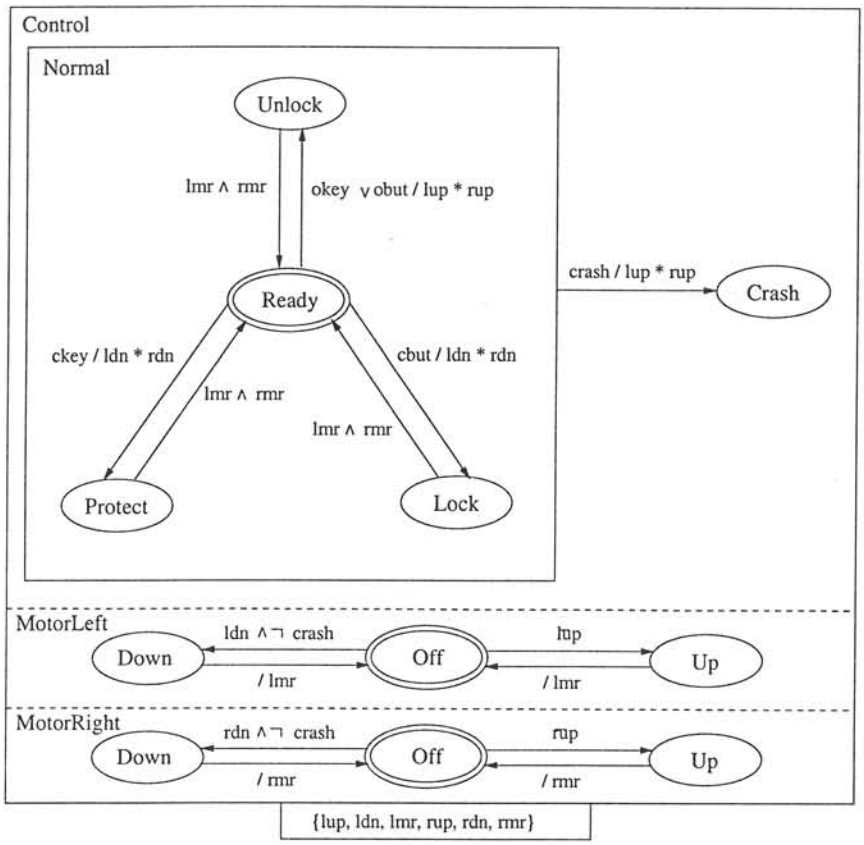

Figure 7: The central locking system

The translated Z is as follows:

Declarations

Z Section *Declarations* **imports:** *toolkit*

$$\mu_{State} ::= control \mid normal \mid crash \mid motorleft \mid down_l \mid off_l \mid up_l \mid motorright \mid down_r \mid off_r \mid up_r \mid ready \mid unlock \mid lock \mid protect$$

$$Signal ::= lmr \mid rmr \mid okey \mid obut \mid ckey \mid cbut \mid lup \mid rup \mid crash_s \mid ldn \mid rdn$$

Control

Z Section *Control* **imports:** *Declarations*

syntax *active prerel*

$Control_c$

$c_{Control} : \mu_{State}$

$InitControl$

$Control_c$

$c_{Control} = normal$

$Normal$

$Control_c$

$c_{Control} = normal$

$Crash$

$Control_c$

$c_{Control} = crash$

$| f_{Control} : \mathbb{P}\, Signal$

$\delta_{NormalCrash}$

$Normal$
$Crash'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Ctrl}!, o! : \mathbb{P}\, Signal$

$active(control)$
$crash_s \in i? \cup (o! \cap f_{Control})$
$o_{Ctrl}! = \{lup, rup\}$

$\epsilon_{Control}$

$\Delta Control_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Ctrl}!, o! : \mathbb{P}\, Signal$

$active(control)$
$\neg\ (Normal \wedge crash_s \in i? \cup (o! \cap f_{Control}))$
$c'_{Control} = c_{Control}$
$o_{Ctrl}! = \{\}$

$Inactive_{Control}$

$\Delta Control_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Ctrl}! : \mathbb{P}\, Signal$

$\neg\ active(control)$
$o_{Ctrl}! = \{\}$

$\delta_{Control} \mathrel{\widehat{=}} \delta_{NormalCrash} \vee \epsilon_{Control} \vee Inactive_{Control}$

Normal

Z Section *Normal* **imports:** *Declarations*

syntax *active prerel*

$Normal_c$

$c_{Normal} : \mu_{State}$

$InitNormal$

$Normal_c$

$c_{Normal} = ready$

$Ready$

$Normal_c$

$c_{Normal} = ready$

$Unlock$

$Normal_c$

$c_{Normal} = unlock$

$Lock$

$Normal_c$

$c_{Normal} = lock$

$Protect$

$Normal_c$

$c_{Normal} = protect$

$| f_{Normal} : \mathbb{P}\, Signal$

$\delta_{ReadyUnlock}$

$Ready$
$Unlock'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(normal)$
$(okey \in i? \cup (o! \cap f_{Normal})$
$\quad \vee\ obut \in i? \cup (o! \cap f_{Normal}))$
$o_{Norm}! = \{lup, rup\}$

$\delta_{UnlockReady}$

$Unlock$
$Ready'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(normal)$
$lmr \in i? \cup (o! \cap f_{Normal})$
$rmr \in i? \cup (o! \cap f_{Normal})$
$o_{Norm}! = \{\}$

$\delta_{ReadyLock}$

$Ready$
$Lock'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(normal)$
$cbut \in i? \cup (o! \cap f_{Normal})$
$o_{Norm}! = \{ldn, rdn\}$

$\delta_{LockReady}$

$Lock$
$Ready'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(normal)$
$lmr \in i? \cup (o! \cap f_{Normal})$
$rmr \in i? \cup (o! \cap f_{Normal})$
$o_{Norm}! = \{\}$

$\delta_{ReadyProtect}$

$Ready$
$Protect'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(normal)$
$ckey \in i? \cup (o! \cap f_{Normal})$
$o_{Norm}! = \{ldn, rdn\}$

$\delta_{ProtectReady}$

$Protect$
$Ready'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(normal)$
$lmr \in i? \cup (o! \cap f_{Normal})$
$rmr \in i? \cup (o! \cap f_{Normal})$
$o_{Norm}! = \{\}$

ϵ_{Normal}

$\Delta Normal_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(normal)$
$\neg\ (Ready \wedge (okey \in i? \cup (o! \cap f_{Normal})$
$\quad \vee\ obut \in i? \cup (o! \cap f_{Normal})))$
$\neg\ (Unlock \wedge lmr \in i? \cup (o! \cap f_{Normal})$
$\quad \wedge\ rmr \in i? \cup (o! \cap f_{Normal}))$
$\neg\ (Ready \wedge cbut \in i? \cup (o! \cap f_{Normal}))$
$\neg\ (Lock \wedge lmr \in i? \cup (o! \cap f_{Normal})$
$\quad \wedge\ rmr \in i? \cup (o! \cap f_{Normal}))$
$\neg\ (Ready \wedge ckey \in i? \cup (o! \cap f_{Normal}))$
$\neg\ (Protect \wedge lmr \in i? \cup (o! \cap f_{Normal})$
$\quad \wedge\ rmr \in i? \cup (o! \cap f_{Normal}))$
$c'_{Normal} = c_{Normal}$
$o_{Norm}! = \{\}$

$Inactive_{Normal}$

$\Delta Normal_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}! : \mathbb{P}\, Signal$

$\neg\ active(normal)$
$o_{Norm}! = \{\}$

$$\delta_{Normal} \mathrel{\widehat{=}} \delta_{ReadyLock} \vee \delta_{LockReady} \vee \delta_{ReadyUnlock} \vee \delta_{UnlockReady} \vee \delta_{ReadyProtect} \vee \delta_{ProtectReady} \vee \epsilon_{Normal} \vee Inactive_{Normal}$$

MotorLeft

Z Section *MotorLeft* **imports:** *Declarations*

syntax *active prerel*

$MotorLeft_c$

$c_{MotorLeft} : \mu_{State}$

$InitMotorLeft$

$MotorLeft_c$

$c_{MotorLeft} = off_l$

Off_l

$MotorLeft_c$

$c_{MotorLeft} = off_l$

Up_l

$MotorLeft_c$

$c_{MotorLeft} = up_l$

$Down_l$

$MotorLeft_c$

$c_{MotorLeft} = down_l$

$| f_{MotorLeft} : \mathbb{P}\, Signal$

$\delta_{OffUpLeft}$

Off_l
Up_l'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(motorleft)$
$lup \in i? \cup (o! \cap f_{MotorLeft})$
$o_{Norm}! = \{\}$

$\delta_{UpOffLeft}$

Up_l
Off_l'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(motorleft)$
$o_{Norm}! = \{lmr\}$

$\delta_{OffDownLeft}$

Off_l
$Down_l'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(motorleft)$
$ldn \in i? \cup (o! \cap f_{MotorLeft})$
$crash_s \notin i? \cup (o! \cap f_{MotorLeft})$
$o_{Norm}! = \{\}$

$\delta_{DownOffLeft}$

$Down_l$
Off_l'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(motorleft)$
$o_{Norm}! = \{lmr\}$

$\epsilon_{MotorLeft}$

$\Delta MotorLeft_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(motorleft)$
$\neg\, (Off_l \wedge lup \in i? \cup (o! \cap f_{MotorLeft}))$
$\neg\, Up_l$
$\neg\, (Off_l \wedge ldn \in i? \cup (o! \cap f_{MotorLeft})$
$\quad \wedge\, crash_s \notin i? \cup (o! \cap f_{MotorLeft}))$
$\neg\, Down_l$
$c'_{MotorLeft} = c_{MotorLeft}$
$o_{Norm}! = \{\}$

$Inactive_{MotorLeft}$

$\Delta MotorLeft_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}! : \mathbb{P}\, Signal$

$\neg\, active(motorleft)$
$o_{Norm}! = \{\}$

$$\delta_{MotorLeft} \mathrel{\widehat=} \delta_{OffUpLeft} \lor \delta_{UpOffLeft} \lor \delta_{OffDownLeft} \lor \delta_{DownOffLeft} \lor \epsilon_{MotorLeft} \lor Inactive_{MotorLeft}$$

MotorRight

Z Section *MotorRight* **imports:** *Declarations*

syntax *active prerel*

$MotorRight_c$

$c_{MotorRight} : \mu_{State}$

$InitMotorRight$

$MotorRight_c$

$c_{MotorRight} = off_r$

Off_r

$MotorRight_c$

$c_{MotorRight} = off_r$

Up_r

$MotorRight_c$

$c_{MotorRight} = up_r$

$Down_r$

$MotorRight_c$

$c_{MotorRight} = down_r$

$| f_{MotorRight} : \mathbb{P}\, Signal$

$\delta_{OffUpRight}$

Off_r
Up'_r
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(motorright)$
$rup \in i? \cup (o! \cap f_{MotorRight})$
$o_{Norm}! = \{\}$

$\delta_{UpOffRight}$

Up_r
Off'_r
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(motorright)$
$o_{Norm}! = \{rmr\}$

$\delta_{OffDownRight}$

Off_r
$Down'_r$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(motorright)$
$rdn \in i? \cup (o! \cap f_{MotorRight})$
$crash_s \notin i? \cup (o! \cap f_{MotorRight})$
$o_{Norm}! = \{\}$

$\delta_{DownOffRight}$

$Down_r$
Off'_r
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(motorright)$
$o_{Norm}! = \{rmr\}$

$\epsilon_{MotorRight}$

$\Delta MotorRight_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}!, o! : \mathbb{P}\, Signal$

$active(motorright)$
$\neg\ (Off_r \land rup \in i? \cup (o! \cap f_{MotorRight}))$
$\neg\ Up_r$
$\neg\ (Off_r \land rdn \in i? \cup (o! \cap f_{MotorRight})$
$\qquad \land\ crash_s \notin i? \cup (o! \cap f_{MotorRight}))$
$\neg\ Down_r$
$c'_{MotorRight} = c_{MotorRight}$
$o_{Norm}! = \{\}$

$Inactive_{MotorRight}$

$\Delta MotorRight_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Norm}! : \mathbb{P}\, Signal$

$\neg\ active(motorright)$
$o_{Norm}! = \{\}$

$$\delta_{MotorRight} \mathrel{\widehat{=}} \delta_{OffUpRight} \lor \delta_{UpOffRight} \lor \delta_{OffDownRight} \lor \delta_{DownOffRight} \lor \epsilon_{MotorRight} \lor Inactive_{MotorRight}$$

Step

Z Section *Step* **imports:** *Control*, *Normal*, *MotorLeft*, *MotorRight*

$l_{Control} : \mathbb{P}\, Signal$
$l_{Normal} : \mathbb{P}\, Signal$
$l_{MotorLeft} : \mathbb{P}\, Signal$
$l_{MotorRight} : \mathbb{P}\, Signal$

$l_{Control} = \{lup, ldn, lmr, rup, rdn, rmr\}$
$l_{Normal} = \{\}$
$l_{MotorLeft} = \{lup, ldn, lmr, rup, rdn, rmr\}$
$l_{MotorRight} = \{lup, ldn, lmr, rup, rdn, rmr\}$
$f_{Control} = l_{Control}$
$f_{Normal} = l_{Normal} \cup f_{Control}$
$f_{MotorLeft} = l_{MotorLeft}$
$f_{MotorRight} = l_{MotorRight}$

$Step$

$\Delta Control_c$
$\Delta Normal_c$
$\Delta MotorLeft_c$
$\Delta MotorRight_c$
$i?, o! : \mathbb{P}\, Signal$

$\exists\, o_{Ctrl}!, o_{Norm}!, o_{Norm}!, o_{Norm}! : \mathbb{P}\, Signal;\ active\_ : \mathbb{P}\, \mu_{State} \bullet$
$\quad (active(control) \iff true) \land$
$\quad (active(normal) \iff c_{Control} = normal) \land$
$\quad (active(motorleft) \iff true) \land$
$\quad (active(motorright) \iff true) \land$
$\quad o! = o_{Ctrl}! \cup o_{Norm}! \cup o_{Norm}! \cup o_{Norm}! \land$
$\quad \delta_{Control} \land \delta_{Normal} \land \delta_{MotorLeft} \land \delta_{MotorRight}$

Tests

Here we give the Z/EVES tests that were carried out on the translation to reassure the author that it is a correct translation.

$$try\ Step[c_{Control} := normal, c_{Normal} := ready, c_{MotorLeft} := off_l, c_{MotorRight} := off_r, i? := \{\}];$$

prove by reduce; *with normalization prove by reduce*;
instantiate $(active\_) == \{control, normal, motorleft, motorright\}$;
prove by reduce;

$$\begin{aligned}\longrightarrow\ & c'_{Control} = normal \\ & \land\ c'_{Normal} = ready \\ & \land\ c'_{MotorLeft} = off_l \\ & \land\ c'_{MotorRight} = off_r \\ & \land\ o! = \{\}\end{aligned}$$

$$try\ Step[c_{Control} := normal, c_{Normal} := ready, c_{MotorLeft} := off_l, c_{MotorRight} := off_r, i? := \{okey\}];$$

prove by reduce; *instantiate* $(active\_) == \{control, normal, motorleft, motorright\}$;
prove by reduce; *with normalization prove by reduce*;

$$\begin{aligned}\longrightarrow\ & c'_{Control} = normal \\ & \land\ c'_{Normal} = unlock \\ & \land\ c'_{MotorLeft} = up_l \\ & \land\ c'_{MotorRight} = up_r \\ & \land\ o! = \{lup\} \cup \{rup\}\end{aligned}$$

$$try\ Step[c_{Control} := normal, c_{Normal} := ready, c_{MotorLeft} := off_l, c_{MotorRight} := off_r, i? := \{crash_s\}];$$

reduce;
instantiate $(active\_) == \{control, normal, motorleft, motorright\}$;
prove by reduce;

$$\begin{aligned}\longrightarrow\ & c'_{Control} = crash \\ & \land\ c'_{Normal} = ready \\ & \land\ c'_{MotorLeft} = up_l \\ & \land\ c'_{MotorRight} = up_r \\ & \land\ o! = \{lup\} \cup \{rup\}\end{aligned}$$

$try\ Step[c_{Control} := normal, c_{Normal} := ready, c_{MotorLeft} := off_l,$
$\quad c_{MotorRight} := off_r, i? := \{crash_s, cbut\}];$

$prove\ by\ reduce;$
$instantiate\ (active\_) == \{control, normal, motorleft, motorright\};$
$prove\ by\ reduce;$
$\longrightarrow\ c'_{Control} = crash$
$\quad \wedge\ c'_{Normal} = lock$
$\quad \wedge\ c'_{MotorLeft} = up_l$
$\quad \wedge\ c'_{MotorRight} = up_r$
$\quad \wedge\ o! = \{lup\} \cup (\{rup\} \cup (\{ldn\} \cup \{rdn\}))$

The following test shows that there is, in fact, still a problem with the specification.

$try\ Step[c_{Control} := normal, c_{Normal} := lock, c_{MotorLeft} := down_l,$
$\quad c_{MotorRight} := down_r, i? := \{crash_s\}];$

$reduce;$
$instantiate\ (active\_) == \{control, normal, motorleft, motorright\};$
$prove\ by\ reduce;$
$\longrightarrow\ c'_{Control} = crash$
$\quad \wedge\ c'_{Normal} = ready$
$\quad \wedge\ c'_{MotorLeft} = off_l$
$\quad \wedge\ c'_{MotorRight} = off_r$
$\quad \wedge\ o! = \{lmr\} \cup (\{rmr\} \cup (\{lup\} \cup \{rup\}))$

9 Appendix B: Specifying a stopwatch

This section gives the translation for the μ-chart pictured in Figure 8. This chart is derived from an example in [7] describing a stopwatch that is assumed to have an external 1 MHz clock.

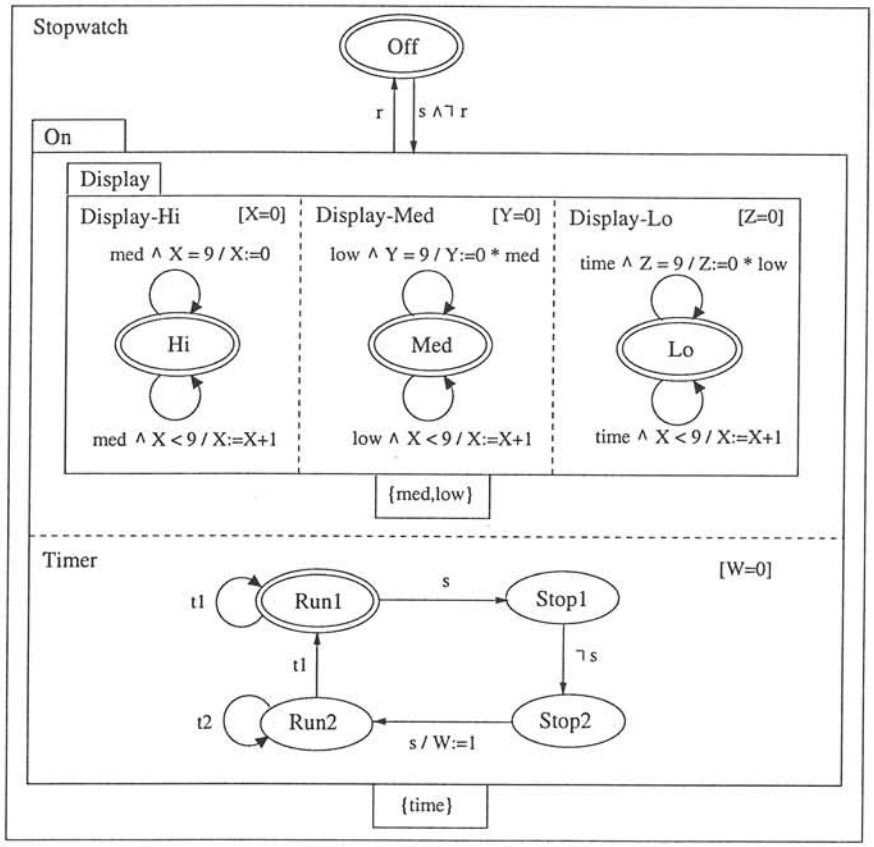

Figure 8: A stopwatch

The transitions labelled $t1$ are in each case an abbreviation for two transitions with the labels $\neg\ s \wedge W < 10^5/W := W + 1$ and $\neg\ s \wedge W \geq 10^5/W := 0 * time$, respectively. Similarly the transition labelled $t2$ represents two transitions with labels $s \wedge W < 10^5/W := W + 1$ and $s \wedge W \geq 10^5/W := 0 * time$.

The translated Z is given in the following sub-sections.

Declarations

Z Section *Declarations* **imports:** *toolkit*

$$\mu_{State} ::= stopwatch \mid on \mid off \mid display \mid timer \mid display\text{-}hi \mid display\text{-}med \mid display\text{-}lo \mid hi \mid med \mid lo \mid run1 \mid run2 \mid stop1 \mid stop2$$

$$Signal ::= s \mid r \mid time \mid smed \mid low$$

Stopwatch

Z Section *Stopwatch* **imports:** *Declarations*

syntax *active prerel*

$Stopwatch_c$

$c_{Stopwatch} : \mu_{State}$

$InitStopwatch$

$Stopwatch_c$

$c_{Stopwatch} = off$

Off

$Stopwatch_c$

$c_{Stopwatch} = off$

On

$Stopwatch_c$

$c_{Stopwatch} = on$

$f_{Stopwatch} : \mathbb{P}\, Signal$

δ_{OffOn}

Off
On'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Stopwatch}!, o! : \mathbb{P}\, Signal$

$active(stopwatch)$
$s \in i? \cup (o! \cap f_{Stopwatch})$
$\neg\ r \in i? \cup (o! \cap f_{Stopwatch})$
$o_{Stopwatch}! = \{\}$

δ_{OnOff}

On
Off'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Stopwatch}!, o! : \mathbb{P}\, Signal$

$active(stopwatch)$
$r \in i? \cup (o! \cap f_{Stopwatch})$
$o_{Stopwatch}! = \{\}$

$\epsilon_{Stopwatch}$

$\Delta Stopwatch_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Stopwatch}!, o! : \mathbb{P}\, Signal$

$active(stopwatch)$
$\neg\ (Off \land s \in i? \cup (o! \cap f_{Stopwatch}) \land \neg\ r \in i? \cup (o! \cap f_{Stopwatch}))$
$\neg\ (On \land r \in i? \cup (o! \cap f_{Stopwatch}))$
$c'_{Stopwatch} = c_{Stopwatch}$
$o_{Stopwatch}! = \{\}$

$Inactive_{Stopwatch}$

$\Delta Stopwatch_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Stopwatch}! : \mathbb{P}\, Signal$

$\neg\ active(stopwatch)$
$o_{Stopwatch}! = \{\}$

$\delta_{Stopwatch} \mathrel{\widehat{=}} \delta_{OffOn} \lor \delta_{OnOff} \lor \epsilon_{Stopwatch} \lor Inactive_{Stopwatch}$

Timer

Z Section *Timer* **imports:** *Declarations*

syntax *active prerel*

$Timer_c$

$c_{Timer} : \mu_{State}$
$W : \mathbb{N}$

$InitTimer$

$Timer_c$

$c_{Timer} = run1$
$W = 0$

$Run1$

$Timer_c$

$c_{Timer} = run1$

$Run2$

$Timer_c$

$c_{Timer} = run2$

$Stop1$

$Timer_c$

$c_{Timer} = stop1$

$Stop2$

$Timer_c$

$c_{Timer} = stop2$

$f_{Timer} : \mathbb{P}\, Signal$

$\delta_{Run1Stop1}$

$Run1$
$Stop1'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$s \in i? \cup (o! \cap f_{Timer})$
$W' = W$
$o_{Timer}! = \{\}$

$\delta_{Stop1Stop2}$

$Stop1$
$Stop2'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$\neg\, s \in i? \cup (o! \cap f_{Timer})$
$W' = W$
$o_{Timer}! = \{\}$

$\delta_{Stop2Run2}$

$Stop2$
$Run2'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$s \in i? \cup (o! \cap f_{Timer})$
$W' = 1$
$o_{Timer}! = \{\}$

$\delta^1_{Run2Run1}$

$Run2$
$Run1'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$\neg\, s \in i? \cup (o! \cap f_{Timer})$
$W < 100000$
$W' = W + 1$
$o_{Timer}! = \{\}$

$\delta^2_{Run2Run1}$

$Run2$
$Run1'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$\neg\, s \in i? \cup (o! \cap f_{Timer})$
$W \geq 100000$
$W' = 0$
$o_{Timer}! = \{time\}$

$\delta^1_{Run1Run1}$

$Run1$
$Run1'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$\neg\, s \in i? \cup (o! \cap f_{Timer})$
$W < 100000$
$W' = W + 1$
$o_{Timer}! = \{\}$

$\delta^2_{Run1Run1}$

$Run1$
$Run1'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$\neg\, s \in i? \cup (o! \cap f_{Timer})$
$W \geq 100000$
$W' = 0$
$o_{Timer}! = \{time\}$

$\delta^1_{Run2Run2}$

$Run2$
$Run2'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$s \in i? \cup (o! \cap f_{Timer})$
$W < 100000$
$W' = W + 1$
$o_{Timer}! = \{\}$

$\delta^2_{Run2Run2}$

$Run2$
$Run2'$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$s \in i? \cup (o! \cap f_{Timer})$
$W \geq 100000$
$W' = 0$
$o_{Timer}! = \{time\}$

ϵ_{Timer}

$\Delta Timer_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}!, o! : \mathbb{P}\, Signal$

$active(timer)$
$\neg\ Run1$
$\neg\ Run2$
$\neg\ (Stop1 \land \neg\ s \in i? \cup (o! \cap f_{Timer}))$
$\neg\ (Stop2 \land s \in i? \cup (o! \cap f_{Timer}))$
$c'_{Timer} = c_{Timer}$
$W' = W$
$o_{Timer}! = \{\}$

$Inactive_{Timer}$

$\Delta Timer_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Timer}! : \mathbb{P}\, Signal$

$\neg\ active(timer)$
$o_{Timer}! = \{\}$

$$\delta_{Timer} \mathrel{\widehat{=}} \delta_{Run1Stop1} \lor \delta_{Stop1Stop2} \lor \delta_{Stop2Run2} \lor \delta^1_{Run2Run1} \lor \delta^2_{Run2Run1} \lor \delta^1_{Run1Run1} \lor \delta^2_{Run1Run1} \lor \delta^1_{Run2Run2} \lor \delta^2_{Run2Run2} \lor \epsilon_{Timer} \lor Inactive_{Timer}$$

Display-Lo

Z Section *Display-Lo* **imports:** *Declarations*

syntax *active prerel*

$Display\text{-}Lo_c$

$c_{D\text{-}Lo} : \mu_{State}$
$Z : \mathbb{N}$

$InitDisplay\text{-}Lo$

$Display\text{-}Lo_c$

$c_{D\text{-}Lo} = lo$
$Z = 0$

Lo

$Display\text{-}Lo_c$

$c_{D\text{-}Lo} = lo$

$| f_{D\text{-}Lo} : \mathbb{P}\, Signal$

δ^1_{Lo}

Lo
Lo'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Lo}!, o! : \mathbb{P}\, Signal$

$active(display\text{-}lo)$
$time \in i? \cup (o! \cap f_{D\text{-}Lo})$
$Z = 9$
$Z' = 0$
$o_{D\text{-}Lo}! = \{low\}$

δ^2_{Lo}

Lo
Lo'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Lo}!, o! : \mathbb{P}\, Signal$

$active(display\text{-}lo)$
$time \in i? \cup (o! \cap f_{D\text{-}Lo})$
$Z < 9$
$Z' = Z + 1$
$o_{D\text{-}Lo}! = \{\}$

$\epsilon_{Display\text{-}Lo}$

$\Delta Display\text{-}Lo_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Lo}!, o! : \mathbb{P}\, Signal$

$active(display\text{-}lo)$
$\neg\ (Display\text{-}Lo_c \wedge time \in i? \cup (o! \cap f_{D\text{-}Lo}) \wedge Z = 9)$
$\neg\ (Display\text{-}Lo_c \wedge time \in i? \cup (o! \cap f_{D\text{-}Lo}) \wedge Z < 9)$
$c'_{D\text{-}Lo} = c_{D\text{-}Lo}$
$Z' = Z$
$o_{D\text{-}Lo}! = \{\}$

$Inactive_{Display\text{-}Lo}$

$\Delta Display\text{-}Lo_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Lo}! : \mathbb{P}\, Signal$

$\neg\ active(display\text{-}lo)$
$o_{D\text{-}Lo}! = \{\}$

$\delta_{Display\text{-}Lo} \mathrel{\widehat{=}} \delta^1_{Lo} \vee \delta^2_{Lo} \vee \epsilon_{Display\text{-}Lo} \vee Inactive_{Display\text{-}Lo}$

Display-Med

Z Section *Display-Med* **imports:** *Declarations*

syntax *active prerel*

$Display\text{-}Med_c$

$c_{D\text{-}Med} : \mu_{State}$
$Y : \mathbb{N}$

$InitDisplay\text{-}Med$

$Display\text{-}Med_c$

$c_{D\text{-}Med} = med$
$Y = 0$

Med

$Display\text{-}Med_c$

$c_{D\text{-}Med} = med$

| $f_{D\text{-}Med} : \mathbb{P}\, Signal$

δ^1_{Med}

Med
Med'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Med}!, o! : \mathbb{P}\, Signal$

$active(display\text{-}med)$
$low \in i? \cup (o! \cap f_{D\text{-}Med})$
$Y = 9$
$Y' = 0$
$o_{D\text{-}Med}! = \{smed\}$

δ^2_{Med}

Med
Med'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Med}!, o! : \mathbb{P}\, Signal$

$active(display\text{-}med)$
$low \in i? \cup (o! \cap f_{D\text{-}Med})$
$Y < 9$
$Y' = Y + 1$
$o_{D\text{-}Med}! = \{\}$

$\epsilon_{Display\text{-}Med}$

$\Delta Display\text{-}Med_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Med}!, o! : \mathbb{P}\, Signal$

$active(display\text{-}med)$
$\neg\ (Display\text{-}Med_c \land low \in i? \cup (o! \cap f_{D\text{-}Med}) \land Y = 9)$
$\neg\ (Display\text{-}Med_c \land low \in i? \cup (o! \cap f_{D\text{-}Med}) \land Y < 9)$
$c'_{D\text{-}Med} = c_{D\text{-}Med}$
$Y' = Y$
$o_{D\text{-}Med}! = \{\}$

$Inactive_{Display\text{-}Med}$

$\Delta Display\text{-}Med_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Med}! : \mathbb{P}\, Signal$

$\neg\ active(display\text{-}med)$
$o_{D\text{-}Med}! = \{\}$

$\delta_{Display\text{-}Med} \mathrel{\widehat{=}} \delta^1_{Med} \lor \delta^2_{Med} \lor \epsilon_{Display\text{-}Med} \lor Inactive_{Display\text{-}Med}$

Display-Hi

Z Section *Display-Hi* **imports:** *Declarations*

syntax *active prerel*

$Display\text{-}Hi_c$

$c_{D\text{-}Hi} : \mu_{State}$
$X : \mathbb{N}$

$InitDisplay\text{-}Hi$

$Display\text{-}Hi_c$

$c_{D\text{-}Hi} = hi$
$X = 0$

Hi

$Display\text{-}Hi_c$

$c_{D\text{-}Hi} = hi$

$| f_{D\text{-}Hi} : \mathbb{P}\, Signal$

δ^1_{Hi}

Hi
Hi'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Hi}!, o! : \mathbb{P}\, Signal$

$active(display\text{-}hi)$
$smed \in i? \cup (o! \cap f_{D\text{-}Hi})$
$X = 9$
$X' = 0$
$o_{D\text{-}Hi}! = \{\}$

δ^2_{Hi}

Hi
Hi'
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Hi}!, o! : \mathbb{P}\, Signal$

$active(display\text{-}hi)$
$smed \in i? \cup (o! \cap f_{D\text{-}Hi})$
$X < 9$
$X' = X + 1$
$o_{D\text{-}Hi}! = \{\}$

$\epsilon_{Display\text{-}Hi}$

$\Delta Display\text{-}Hi_c$
$i? : \mathbb{P}\, Signal$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Hi}!, o! : \mathbb{P}\, Signal$

$active(display\text{-}hi)$
$\neg\ (Display\text{-}Hi_c \wedge low \in i? \cup (o! \cap f_{D\text{-}Hi}) \wedge X = 9)$
$\neg\ (Display\text{-}Hi_c \wedge low \in i? \cup (o! \cap f_{D\text{-}Hi}) \wedge X < 9)$
$c'_{D\text{-}Hi} = c_{D\text{-}Hi}$
$X' = X$
$o_{D\text{-}Hi}! = \{\}$

$Inactive_{Display\text{-}Hi}$

$\Delta Display\text{-}Hi_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{D\text{-}Hi}! : \mathbb{P}\, Signal$

$\neg\ active(display\text{-}hi)$
$o_{D\text{-}Hi}! = \{\}$

$\delta_{Display\text{-}Hi} \mathrel{\widehat{=}} \delta^1_{Hi} \vee \delta^2_{Hi} \vee \epsilon_{Display\text{-}Hi} \vee Inactive_{Display\text{-}Hi}$

Step

Z Section *Step* **imports:** *Stopwatch*, *Timer*, *Display-Lo*, *Display-Med*, *Display-Hi*

$l_{Stopwatch} : \mathbb{P}\, Signal$
$l_{On} : \mathbb{P}\, Signal$
$l_{Timer} : \mathbb{P}\, Signal$
$l_{Display} : \mathbb{P}\, Signal$
$l_{D\text{-}Lo} : \mathbb{P}\, Signal$
$l_{D\text{-}Med} : \mathbb{P}\, Signal$
$l_{D\text{-}Hi} : \mathbb{P}\, Signal$

$l_{Stopwatch} = \{\}$
$l_{On} = \{time\}$
$l_{Display} = \{smed, low\}$
$l_{Timer} = \{\}$
$l_{D\text{-}Lo} = \{\}$
$l_{D\text{-}Med} = \{\}$
$l_{D\text{-}Hi} = \{\}$
$f_{Stopwatch} = l_{Stopwatch}$
$f_{Timer} = l_{Timer} \cup l_{On} \cup f_{Stopwatch}$
$f_{D\text{-}Lo} = l_{D\text{-}Lo} \cup l_{Display} \cup l_{On} \cup f_{Stopwatch}$
$f_{D\text{-}Med} = l_{D\text{-}Med} \cup l_{Display} \cup l_{On} \cup f_{Stopwatch}$
$f_{D\text{-}Hi} = l_{D\text{-}Hi} \cup l_{Display} \cup l_{On} \cup f_{Stopwatch}$

Step

$\Delta Stopwatch_c$
$\Delta Timer_c$
$\Delta Display\text{-}Lo_c$
$\Delta Display\text{-}Med_c$
$\Delta Display\text{-}Hi_c$
$i?, o! : \mathbb{P}\, Signal$

$\exists\, o_{Stopwatch}!, o_{Timer}!, o_{D\text{-}Lo}!, o_{D\text{-}Med}!, o_{D\text{-}Hi}! : \mathbb{P}\, Signal;\ active\_ : \mathbb{P}\, \mu_{State} \bullet$
$(active(stopwatch) \iff true) \land$
$(active(timer) \iff c_{Stopwatch} = on) \land$
$(active(display\text{-}lo) \iff c_{Stopwatch} = on) \land$
$(active(display\text{-}med) \iff c_{Stopwatch} = on) \land$
$(active(display\text{-}hi) \iff c_{Stopwatch} = on) \land$
$o! = o_{Stopwatch}! \cup o_{Timer}! \cup o_{D\text{-}Lo}! \cup o_{D\text{-}Med}! \cup o_{D\text{-}Hi}! \land$
$((\delta_{OffOn} \land InitDisplay\text{-}Lo' \land InitDisplay\text{-}Med' \land InitDisplay\text{-}Hi'$
$\land\ InitTimer') \lor \delta_{OnOff} \lor \epsilon_{Stopwatch} \lor Inactive_{Stopwatch}) \land$
$\delta_{Timer} \land \delta_{Display\text{-}Lo} \land \delta_{Display\text{-}Med} \land \delta_{Display\text{-}Hi}$

Tests

Again we include the Z/EVES tests that were used by the author for reassurance of the correctness of the translation.

try $Step[c_{Stopwatch} := off, i? := \{s\}]$;

prove by reduce;
instantiate $(active\_) == \{stopwatch\}$;
prove by reduce;

$\longrightarrow\ c'_{Stopwatch} = on \land c_{Timer} \in \mu_{State} \land W \in \mathbb{Z} \land c'_{Timer} = run1$
$\land\ W' = 0 \land c_{D\text{-}Lo} \in \mu_{State} \land Z \in \mathbb{Z} \land c'_{D\text{-}Lo} = lo$
$\land\ Z' = 0 \land c_{D\text{-}Med} \in \mu_{State} \land Y \in \mathbb{Z} \land c'_{D\text{-}Med} = med$
$\land\ Y' = 0 \land c_{D\text{-}Hi} \in \mu_{State} \land X \in \mathbb{Z} \land c'_{D\text{-}Hi} = hi$
$\land\ X' = 0 \land o! = \{\} \land W \geq 0 \land Z \geq 0$
$\land\ Y \geq 0 \land X \geq 0$

try $Step[c_{Stopwatch} := off, i? := \{s, r\}]$;
prove by reduce;
instantiate $(active\_) == \{stopwatch\}$;
prove by reduce;

$\longrightarrow\ c'_{Stopwatch} = off \land c_{Timer} \in \mu_{State} \land W \in \mathbb{Z} \land c'_{Timer} \in \mu_{State}$
$\land\ W' \in \mathbb{Z} \land c_{D\text{-}Lo} \in \mu_{State} \land Z \in \mathbb{Z} \land c'_{D\text{-}Lo} \in \mu_{State}$
$\land\ Z' \in \mathbb{Z} \land c_{D\text{-}Med} \in \mu_{State} \land Y \in \mathbb{Z} \land c'_{D\text{-}Med} \in \mu_{State}$
$\land\ Y' \in \mathbb{Z} \land c_{D\text{-}Hi} \in \mu_{State} \land X \in \mathbb{Z} \land c'_{D\text{-}Hi} \in \mu_{State}$
$\land\ X' \in \mathbb{Z} \land o! = \{\} \land W \geq 0 \land W' \geq 0$
$\land\ Z \geq 0 \land Z' \geq 0 \land Y \geq 0 \land Y' \geq 0 \land X \geq 0 \land X' \geq 0$

try Step$[c_{Stopwatch} := on, c_{Timer} := run1, c_{D\text{-}Hi} := hi, c_{D\text{-}Med} := med,$
$c_{D\text{-}Lo} := lo, X := 0, Y := 0, Z := 0, W := 20, i? := \{\}]$;

reduce;
with normalization prove by reduce;
instantiate
$(active\_) == \{stopwatch, display\text{-}hi, display\text{-}med, display\text{-}lo, timer\}$;
prove by reduce;

$\longrightarrow \; c'_{Stopwatch} = on \wedge c'_{Timer} = run1 \wedge W' = 21 \wedge c'_{D\text{-}Lo} = lo$
$\wedge Z' = 0 \wedge c'_{D\text{-}Med} = med \wedge Y' = 0 \wedge c'_{D\text{-}Hi} = hi$
$\wedge X' = 0 \wedge o! = \{\}$

try Step$[c_{Stopwatch} := on, c_{Timer} := run1, c_{D\text{-}Hi} := hi, c_{D\text{-}Med} := med,$
$c_{D\text{-}Lo} := lo, X := 5, Y := 1, Z := 0, W := 100000,$
$i? := \{\}, o! := \{time\}]$;

reduce;
instantiate
$(active\_) == \{stopwatch, display\text{-}hi, display\text{-}med, display\text{-}lo, timer\}$;
prove by reduce;

$\longrightarrow \; c'_{Stopwatch} = on \wedge c'_{Timer} = run1 \wedge W' = 0 \wedge c'_{D\text{-}Lo} = lo$
$\wedge Z' = 1 \wedge c'_{D\text{-}Med} = med \wedge Y' = 1 \wedge c'_{D\text{-}Hi} = hi$
$\wedge X' = 5$

try Step$[c_{Stopwatch} := on, c_{Timer} := run1, c_{D\text{-}Hi} := hi, c_{D\text{-}Med} := med,$
$c_{D\text{-}Lo} := lo, X := 5, Y := 1, Z := 0, W := 100000,$
$i? := \{s\}, o! := \{\}]$;

reduce;
instantiate
$(active\_) == \{stopwatch, display\text{-}hi, display\text{-}med, display\text{-}lo, timer\}$;
prove by reduce;

$\longrightarrow \; c'_{Stopwatch} = on \wedge c'_{Timer} = stop1 \wedge W' = 100000 \wedge c'_{D\text{-}Lo} = lo$
$\wedge Z' = 0 \wedge c'_{D\text{-}Med} = med \wedge Y' = 1 \wedge c'_{D\text{-}Hi} = hi$
$\wedge X' = 5$

$$
\begin{array}{l}
\textit{try Step}[c_{\textit{Stopwatch}} := \textit{on}, c_{\textit{Timer}} := \textit{stop2}, c_{\textit{D-Hi}} := \textit{hi}, c_{\textit{D-Med}} := \textit{med}, \\
\qquad c_{\textit{D-Lo}} := \textit{lo}, X := 5, Y := 1, Z := 0, W := 100000, \\
\qquad i? := \{s\}, o! := \{\}];
\end{array}
$$

reduce;
instantiate
$(active\_) == \{\textit{stopwatch}, \textit{display-hi}, \textit{display-med}, \textit{display-lo}, \textit{timer}\}$;
prove by reduce;

$$
\begin{array}{l}
\longrightarrow \; c'_{\textit{Stopwatch}} = \textit{on} \wedge c'_{\textit{Timer}} = \textit{run2} \wedge W' = 1 \wedge c'_{\textit{D-Lo}} = \textit{lo} \\
\quad \wedge Z' = 0 \wedge c'_{\textit{D-Med}} = \textit{med} \wedge Y' = 1 \wedge c'_{\textit{D-Hi}} = \textit{hi} \\
\quad \wedge X' = 5
\end{array}
$$

This example demonstrates the problem of not initialising sub-charts when they are left.

$$
\begin{array}{l}
\textit{try Step}[c_{\textit{Stopwatch}} := \textit{on}, c_{\textit{Timer}} := \textit{stop2}, c_{\textit{D-Hi}} := \textit{hi}, c_{\textit{D-Med}} := \textit{med}, \\
\qquad c_{\textit{D-Lo}} := \textit{lo}, X := 5, Y := 1, Z := 0, W := 100000, \\
\qquad i? := \{r\}, o! := \{\}];
\end{array}
$$

reduce;
instantiate
$(active\_) == \{\textit{stopwatch}, \textit{display-hi}, \textit{display-med}, \textit{display-lo}, \textit{timer}\}$;
prove by reduce;

$$
\begin{array}{l}
\longrightarrow \; c'_{\textit{Stopwatch}} = \textit{off} \wedge c'_{\textit{Timer}} = \textit{stop2} \wedge W' = 100000 \wedge c'_{\textit{D-Lo}} = \textit{lo} \\
\quad \wedge Z' = 0 \\
\quad \wedge c'_{\textit{D-Med}} = \textit{med} \wedge Y' = 1 \wedge c'_{\textit{D-Hi}} = \textit{hi} \wedge X' = 5
\end{array}
$$

One possible solution to investigate later is to have an operation that happens when the chart is inactive after the operation. This operation would be the disjunct of all possible operations of that chart with all output and after state information hidden except for the output signals, in combination with *Init'* for that chart. This sounds like having the Z looking into the future, *i.e.* mixing preconditions up with post conditions, but this seems to be allowed in μ-charts. More investigation is needed.

$$
\begin{array}{l}
\textit{try Step}[c_{\textit{Stopwatch}} := \textit{off}, c_{\textit{Timer}} := \textit{stop2}, c_{\textit{D-Hi}} := \textit{hi}, c_{\textit{D-Med}} := \textit{med}, \\
\qquad c_{\textit{D-Lo}} := \textit{lo}, X := 5, Y := 1, Z := 0, W := 100000, \\
\qquad i? := \{\}, o! := \{\}];
\end{array}
$$

reduce;
instantiate
$(active\_) == \{\textit{stopwatch}, \textit{display-hi}, \textit{display-med}, \textit{display-lo}, \textit{timer}\}$;
prove by reduce;

$$
\begin{array}{l}
\longrightarrow \; c'_{\textit{Stopwatch}} = \textit{off} \wedge c'_{\textit{Timer}} \in \mu_{\textit{State}} \wedge W' \in \mathbb{Z} \wedge c'_{\textit{D-Lo}} \in \mu_{\textit{State}} \\
\quad \wedge Z' = 0 \wedge c'_{\textit{D-Med}} \in \mu_{\textit{State}} \wedge Y' = 0 \wedge c'_{\textit{D-Hi}} \in \mu_{\textit{State}} \\
\quad \wedge X' = 0 \wedge W' \geq 0
\end{array}
$$

This proof ensures that in the system specified none of the local variables representing the stopwatch's displayed digits can never be greater that 9.

$try\ \forall bind : Step \bullet bind.X \leq 9 \land bind.Y \leq 9 \land bind.Z \leq 9 \implies$
$bind.X' \leq 9 \land bind.Y' \leq 9 \land bind.Z' \leq 9;$

apply Step$member; prove; invoke; prenex;
split $c_{Stopwatch} = off$; *cases; prove; next;*
split $c_{Stopwatch} = on$; *cases; next; prove; next;*
cases; split $X' > 9$; *cases; simplify; next;*
use notgthanlthan$[Y := 9]$; *simplify;*
next; split$Y' > 9$; *cases; simplify; next;*
use notgthanlthan$[X := Y, Y := 9]$; *simplify; next; split*$Z' > 9$;
cases; simplify; next; use notgthanlthan$[X := Z, Y := 9]$;
simplify; next;

$\longrightarrow$ *true*

10 Appendix C: Integer-valued Signals Example

Another important type of system that μ-charts can be used to specify is interactive systems. This section gives a very simple specification of a menu driven system in Figure 9 and its translation into Z.

This example also demonstrates how integer-valued signals are translated.

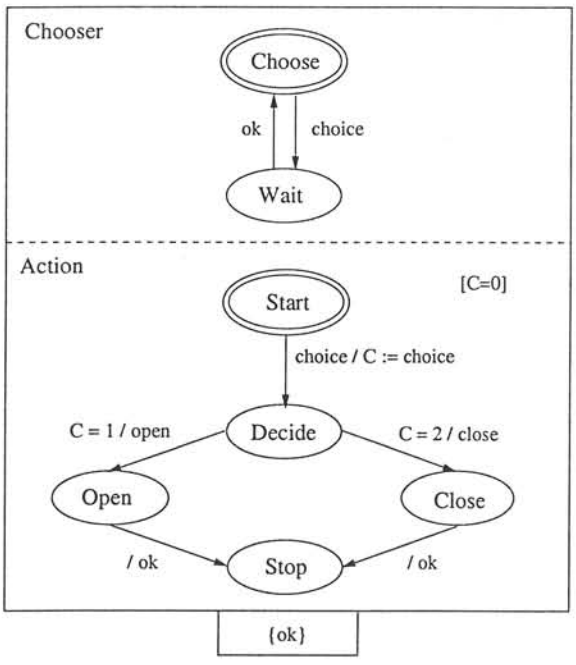

Figure 9: Simple menu driven system

Declarations

Z Section *Declarations* **imports:** *toolkit*

$\mu_{State} ::= chooser \mid choose \mid wait \mid action \mid start \mid decide \mid open \mid close \mid stop$
$Signal ::= ok \mid choice\langle\langle \mathbb{Z} \rangle\rangle \mid open_s \mid close_s$

$$IO : \mathbb{P}(\mathbb{P}\, Signal)$$

$$IO = \{a : \mathbb{P}\, Signal \mid (\forall x, y : \mathbb{Z} \bullet choice(x) \in a \land choice(y) \in a \implies x = y)\}$$

Chooser

Z Section *Chooser* **imports:** *Declarations*

Chooser$_c$

$$c_{Chooser} : \mu_{State}$$

InitChooser

$$Chooser_c$$

$$c_{Chooser} = choose$$

Choose

$$Chooser_c$$

$$c_{Chooser} = choose$$

Wait

$$Chooser_c$$

$$c_{Chooser} = wait$$

$$f_{Chooser} : IO$$

$\delta_{ChooseWait}$

$$Choose$$
$$Wait'$$
$$i? : IO$$
$$active\_ : \mathbb{P}\, \mu_{State}$$
$$o_{Chooser}!, o! : IO$$

$$active(chooser)$$
$$\exists x : \mathbb{N} \bullet choice(x) \in i? \cup (o! \cap f_{Chooser})$$
$$o_{Chooser}! = \{\}$$

$\delta_{WaitChoose}$

$$Wait$$
$$Choose'$$
$$i? : IO$$
$$active\_ : \mathbb{P}\, \mu_{State}$$
$$o_{Chooser}!, o! : IO$$

$$active(chooser)$$
$$ok \in i? \cup (o! \cap f_{Chooser})$$
$$o_{Chooser}! = \{\}$$

$\epsilon_{Chooser}$

$\Delta Chooser_c$
$i? : IO$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Chooser}!, o! : IO$

$active(chooser)$
$\neg\ (Choose \land (\exists x : \mathbb{N} \bullet choice(x) \in i? \cup (o! \cap f_{Chooser})))$
$\neg\ (Wait \land ok \in i? \cup (o! \cap f_{Chooser}))$
$c'_{Chooser} = c_{Chooser}$
$o_{Chooser}! = \{\}$

$Inactive_{Chooser}$

$\Delta Chooser_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Chooser}! : IO$

$\neg\ active(chooser)$
$o_{Chooser}! = \{\}$

Action

Z Section *Action* **imports:** *Declarations*

$Action_c$

$c_{Action} : \mu_{State}$
$C : \mathbb{Z}$

$InitAction$

$Action_c$

$c_{Action} = start$
$C = 0$

$Start$

$Action_c$

$c_{Action} = start$

$Decide$

$Action_c$

$c_{Action} = decide$

$Open$

$Action_c$

$c_{Action} = open$

Close

$Action_c$

$c_{Action} = close$

Stop

$Action_c$

$c_{Action} = stop$

$| f_{Action} : IO$

$\delta_{StartDecide}$

$Start$
$Decide'$
$i? : IO$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Action}!, o! : IO$

$active(action)$
$(\exists x : \mathbb{N} \bullet choice(x) \in i? \cup (o! \cap f_{Action}) \land$
$\quad C' = x)$
$o_{Action}! = \{\}$

$\delta_{DecideOpen}$

$Decide$
$Open'$
$i? : IO$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Action}!, o! : IO$

$active(action)$
$C = 1$
$o_{Action}! = \{open_s\}$

$\delta_{DecideClose}$

$Decide$
$Close'$
$i? : IO$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Action}!, o! : IO$

$active(action)$
$C = 2$
$o_{Action}! = \{close_s\}$

$\delta_{OpenStop}$

$Open$
$Stop'$
$i? : IO$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Action}!, o! : IO$

$active(action)$
$o_{Action}! = \{ok\}$

$\delta_{CloseStop}$

$Close$
$Stop'$
$i? : IO$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Action}!, o! : IO$

$active(action)$
$o_{Action}! = \{ok\}$

ϵ_{Action}

$\Delta Action_c$
$i? : IO$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Action}!, o! : IO$

$active(action)$
$\neg\, (Start \land (\exists\, x : \mathbb{N} \bullet choice(x) \in i? \cup (o! \cap f_{Action})))$
$\neg\, (Decide \land C = 1)$
$\neg\, (Decide \land C = 2)$
$\neg\, Open$
$\neg\, Close$
$c'_{Action} = c_{Action}$
$C' = C$
$o_{Action}! = \{\}$

$Inactive_{Action}$

$\Delta Action_c$
$active\_ : \mathbb{P}\, \mu_{State}$
$o_{Action}! : IO$

$\neg\, active(action)$
$o_{Action}! = \{\}$

Step

Z Section *Step* **imports:** *Chooser*, *Action*

$$
\begin{array}{|l}
l_{Chooser} : IO \\
l_{Action} : IO \\
\hline
l_{Chooser} = \{ok\} \\
f_{Chooser} = l_{Chooser} \\
l_{Action} = \{ok\} \\
f_{Action} = \{ok\}
\end{array}
$$

Step

$$
\begin{array}{|l}
\Delta Chooser_c \\
\Delta Action_c \\
i?, o! : IO \\
\hline
\exists\, o_{Chooser}!, o_{Action}! : IO;\ active\_ : \mathbb{P}\, \mu_{State} \bullet \\
\quad (active(chooser) \iff true) \land \\
\quad (active(action) \iff true) \land \\
\quad o! = o_{Chooser}! \cup o_{Action}! \land \\
\quad (\delta_{ChooseWait} \lor \delta_{WaitChoose} \lor \epsilon_{Chooser} \lor Inactive_{Chooser}) \land \\
\quad (\delta_{StartDecide} \lor \delta_{DecideOpen} \lor \delta_{DecideClose} \lor \delta_{OpenStop} \lor \delta_{CloseStop} \lor \epsilon_{Action} \lor \\
\qquad Inactive_{Action})
\end{array}
$$

$tryStep[c_{Chooser} := choose, c_{Action} := start, C := 0, i? := \{\}];$

prove by reduce;
instantiate $(active\_) == \{chooser, action\};$
prove by reduce;
$\longrightarrow\ c'_{Chooser} = choose$
$\land\ c'_{Action} = start$
$\land\ C' = 0$
$\land\ o! = \{\}$

$tryStep[c_{Chooser} := choose, c_{Action} := start, C := 0, i? := \{choice(2)\}];$

prove by reduce;
instantiate $(active\_) == \{chooser, action\};$
prove by reduce;
$\longrightarrow\ c'_{Chooser} = wait$
$\land\ c'_{Action} = decide$
$\land\ C' = 2$
$\land\ o! = \{\}$

$tryStep[c_{Chooser} := choose, c_{Action} := start, C := 0, i? := \{choice(1), choice(2)\}];$

invoke;
applynotinIO;
prove by reduce;
$\longrightarrow$ *false*

$tryStep[c_{Chooser} := wait, c_{Action} := decide, C := 1, i? := \{\}];$

prove by reduce;
instantiate $(active\_) == \{chooser, action\};$
prove by reduce;
$\longrightarrow \; c'_{Chooser} = wait$
$\quad \wedge \; c'_{Action} = open$
$\quad \wedge \; C' \in \mathbb{Z}$
$\quad \wedge \; o! = \{open_s\}$

$tryStep[c_{Chooser} := wait, c_{Action} := decide, C := 0, i? := \{\}];$

prove by reduce;
instantiate $(active\_) == \{chooser, action\};$
prove by reduce;
$\longrightarrow \; c'_{Chooser} = wait$
$\quad \wedge \; c'_{Action} = decide$
$\quad \wedge \; C' = 0$
$\quad \wedge \; o! = \{\}$

$tryStep[c_{Chooser} := wait, c_{Action} := close, C := 2, i? := \{\}];$

prove by reduce;
instantiate $(active\_) == \{chooser, action\};$
prove by reduce;
$\longrightarrow \; c'_{Chooser} = choose$
$\quad \wedge \; c'_{Action} = stop$
$\quad \wedge \; C' \in \mathbb{Z}$
$\quad \wedge \; o! = \{ok\}$