The Design and Implementation of the MASTERMIND Toolkit

R. E. Kurt Stirewalt

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

E-mail: kurt@cc.gatech.edu

Technical Report GIT-GVU-97-23
December 1996

1 Introduction

The MASTERMIND project[5, 7, 3] is concerned with the design, integration, and automatic generation of interactive systems from declarative models. One model describes the tasks a user will perform as a protocol of end user actions and their affect on other aspects of the application. The MASTERMIND Dialogue Language (MDL)[6] is a declarative notation for specifying user tasks. The MASTERMIND Toolkit (MMTK) is a run-time infrastructure and collection of reusable C++ components which can be instantiated and aggregated to implement MDL[6] specifications. Code generators implement an MDL expression E by aggregating MMTK components (representing the various operators within E) into a class and connecting these components according to the syntactic structure of E. This approach distributes the control policy of an MDL operator over many independent components, which means components must implement orderings by issuing control commands, announcing activity, and observing the state of connected components. We designed these components around a model of machine execution in which each component is independent and may *message* other components without waiting for them to return. From the code generator's standpoint, MMTK components are concurrent and need only be aggregated and connected in order to implement an MDL behavior expression. The design complexity in MMTK centers around making components compose in this liberal fashion.

2 Syntax of Component Composition

By design, MMTK supports a simple model of composition. Components compose by aggregating them into a tree. We devised a notation called MTREE to express the syntax of an MMTK component tree. MTREE names a component that implements the functionality of an MDL control

Table 1: MTREE component implementations of MDL operators.

MDL Operator	Mealy Machine	Mdl Operator	Mealy Machine
$\mid$	alt	$[>$	dis
$\leftrightarrow$	excl	$\triangle$	int
$e?x,\ e!x$	leaf	$*$	loop
$^{\text{opt}}$	opt	$\|\|, \|\|\|$	par
$\fatsemi, >>$	seq		ceiling

operator (as shown in Table 1). Note that there is no component implementing the **stop** process or the **hide** operator, input and output communications are both implemented by the same component (leaf), and || and ||| are both implemented by the same component (par). Instances of these components are connected using the binary operator $\triangleleft\_\triangleright$ and the unary operator $\_\triangleright$. The $\_$ in these operators will contain one of the names in Table 1. So, for example, $\triangleleft$seq$\triangleright$ is a binary component and loop$\triangleright$ is a unary component. A simple syntactic transformation maps an arbitrary MDL expression into an MTREE expression.

For example, the MDL expression: $(e?x \fatsemi \textbf{stop}) >> ((f?y \fatsemi \textbf{stop}) \mid (g?z \fatsemi \textbf{stop}))$ corresponds to the following MTREE expression:

$$\text{ceiling} \triangleright (\text{leaf} \triangleleft \text{seq} \triangleright (\text{leaf} \triangleleft \text{alt} \triangleright \text{leaf}))$$

We call instances of MTREE components *occurrences* and use middle-of-the-alphabet letters like m and n to represent them in formal arguments. The above MTREE expression, for example, has three occurrences of the leaf component, and one occurrence each of the ceiling, seq, and alt components. The reader should observe three things about this mapping:

1. the **stop** process maps to nothing because MDL expressions denote processes which announce completion,
2. parentheses are retained under the mapping, and
3. the MTREE representation applies the unary operator ceiling$\triangleright$ to the image of the MDL expression.

Components communicate signaling information to subordinates and parents in a tree hierarchy. This means that all the components that implement MDL control operators will expect to be connected to a parent in the tree hierarchy. The special ceiling component is used to **top** a component tree so that there are no unresolved communications.

3 Semantics of Component Composition

For an MTREE component tree to implement an MDL behavior expression, components must interpret ordering properties as a *protocol* of inter-component commands. Examples of these commands include sending a message to a subordinate component to accept activity, sending a message a subordinate component to no longer accept activity, and sending a message to a parent component to

announce activity. To take an example, consider a component A implementing the $>>$ operator, whose left subcomponent B implements an input communication and a whose right subcomponent C implements the | operator. When A is instructed (by its parent component) to accept user activity, it must interpret this command in a way consistent with its ordering. Since the intent of $>>$ is to sequentially order two processes, A instructs B to accept activity but does not instruct C to do anything yet. Now if user activity causes the input communication associated with B to fire, then B must announce activity to A. At this point, A will propagate this announcement to its parent. When B announces completion, A will instruct C to accept activity. Since C implements an ordering, it will forward this command to subordinates in a way that is consistent with its ordering semantics. Specifically, it will instruct both of its children to accept activity and then, when one announces activity, C will instruct the other to forbid activity. Components maintain internal state which manages these coordinating messages. The internal state changes in response to a *signal* which communicates a change of status from one component to another. Finally, components often react to signals by issuing their own signals to other components. To reason about the correctness of these components, we use a formal model which captures this type of communication.

3.1 Mealy Machines

Mealy machines[4] are finite automata with output. Formally, a Mealy machine M is a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where Q is a set of states, Σ is a set of input symbols, Δ is a set of output symbols, $\delta : Q \times \Sigma \to Q$ is a state transition function, $\lambda : Q \times \Sigma \to \Delta$ is a transition output function, and $q_0 \in Q$ is an initial state of M. A Mealy machine in state s reacts to an input symbol e by:

1. transitioning into state $\delta(s, e)$, and
2. issuing an output symbol $\lambda(s, e)$

all in one step. MMTK components can be modeled using Mealy machines because the components maintain a finite state, and they communicate with other components using a fixed alphabet of signals.

Mealy machines representing MMTK components differ on the set Q of states and the state and output transition functions δ and λ. We denote the specific component to which these differing entities belong by subscripts. So, for example, the set of states that make up the Mealy machine associated with the alt component is denoted Q_{alt}, the state transition function is denoted δ_{alt}, and the output transition function is denoted λ_{alt}. Moreover, as we will see in the next section, the input and output alphabets Σ and Δ differ based on the arity of a component (ceiling, nullary, unary, or binary), and so we represent these distinctions by Σ_{-1}, Σ_0, Σ_1, and Σ_2; and Δ_{-1}, Δ_0, Δ_1, and Δ_2 respectively.

The next task is to define the meaning of $\lhd x \rhd$ for all binary components x and $y \rhd$ for all unary components y. For this, we need to adopt a policy for Mealy machine connectivity. As we will see, this policy affects the design of Σ and Δ. The semantic definition of each component in terms of a Mealy machine is given in Figure 2.

Table 2: Mealy Machine semantics of MTREE components.

Component		Mealy Machine
$\mathfrak{M}[\![\mathsf{alt}]\!]$	==	$(Q_{\mathsf{alt}}, \Sigma_2, \Delta_2, \delta_{\mathsf{alt}}, \lambda_{\mathsf{alt}}, \texttt{notready})$
$\mathfrak{M}[\![\mathsf{ceiling}]\!]$	==	$(Q_{\mathsf{ceiling}}, \Sigma_{-1}, \Delta_{-1}, \delta_{\mathsf{ceiling}}, \lambda_{\mathsf{ceiling}}, \texttt{notready})$
$\mathfrak{M}[\![\mathsf{dis}]\!]$	==	$(Q_{\mathsf{dis}}, \Sigma_2, \Delta_2, \delta_{\mathsf{dis}}, \lambda_{\mathsf{dis}}, \texttt{notready})$
$\mathfrak{M}[\![\mathsf{excl}]\!]$	==	$(Q_{\mathsf{excl}}, \Sigma_2, \Delta_2, \delta_{\mathsf{excl}}, \lambda_{\mathsf{excl}}, \texttt{notready})$
$\mathfrak{M}[\![\mathsf{int}]\!]$	==	$(Q_{\mathsf{int}}, \Sigma_2, \Delta_2, \delta_{\mathsf{int}}, \lambda_{\mathsf{int}}, \texttt{notready})$
$\mathfrak{M}[\![\mathsf{loop}]\!]$	==	$(Q_{\mathsf{loop}}, \Sigma_1, \Delta_1, \delta_{\mathsf{loop}}, \lambda_{\mathsf{loop}}, \texttt{notready})$
$\mathfrak{M}[\![\mathsf{leaf}]\!]$	==	$(Q_{\mathsf{leaf}}, \Sigma_0, \Delta_0, \delta_{\mathsf{leaf}}, \lambda_{\mathsf{leaf}}, \texttt{notready})$
$\mathfrak{M}[\![\mathsf{opt}]\!]$	==	$(Q_{\mathsf{opt}}, \Sigma_1, \Delta_1, \delta_{\mathsf{opt}}, \lambda_{\mathsf{opt}}, \texttt{notready})$
$\mathfrak{M}[\![\mathsf{par}]\!]$	==	$(Q_{\mathsf{par}}, \Sigma_2, \Delta_2, \delta_{\mathsf{par}}, \lambda_{\mathsf{par}}, \texttt{notready})$
$\mathfrak{M}[\![\mathsf{seq}]\!]$	==	$(Q_{\mathsf{seq}}, \Sigma_2, \Delta_2, \delta_{\mathsf{seq}}, \lambda_{\mathsf{seq}}, \texttt{notready})$

3.2 Mealy Machine Connectivity

MTREE components are connected together in a tree topology. *Connectors* are attributes of MMTK components and represent a communication path between a component and one of its neighbors in the tree topology. There are three categories of connectors: p (for parent), l (for left child), and r (for right child). Binary components, contain all three categories of connector; whereas unary components contain only p and l, nullary components contain only p, and the special ceiling component contains only l. Given an occurrence of a component, the connector can be selected by name using the selection operator '.'. That is, if x is an occurrence of the alt component, then $x.p$ denotes the parent connector, $x.l$ the left child connector, and $x.r$ the right child connector. Components are *connected* according to their relative location in the tree. Connection is specified as the *unification* of connectors. For example, if x and y are components in the tree and y is the left child of x, then $x.l = y.p$ is the unification of connectors that expresses this connection.

The meaning of an MTREE expression like:

$$\mathfrak{C}[\![E_1 \lhd m \rhd E_2]\!] == (m.l = root(E_1).p) \wedge (m.r = root(E_2).p) \wedge \mathfrak{C}[\![E_1]\!] \wedge \mathfrak{C}[\![E_2]\!]$$

leaf $\lhd$ alt $\rhd$ seq is the unification of connectors among the components. That is, the p connector of the leaf component is *connected* to the l connector of the alt component, the p connector of the seq component is connected to the r connector of the alt component, and the p

MTREE components communicate by issuing signals over connectors. A connector is a directed, point-to-point communication path that maps output symbols of one machine, called the *source*, to input symbols of another machine, called the *target*. These signals communicate the control commands of one component to another. When a component m is issued one of these signals, it changes state and may, in turn, issue a signal to another component. The behavior of an entire component tree can, therefore, be expressed in terms of the behavior of individual components and a mapping of input and output symbols to connected Mealy machine communication.

3.3 Control Model

The global state of an interactive system can be thought of as a sequence of *reactive* (or *equilibrium*) states. When in these states, the system awaits a *perturbing* signal from the user or the application functionality. these perturbations cause a flurry of activity among the components. During this flurry the system will not be in an equilibrium state, but rather, will be in a series of *transient* states. Eventually the system settles into another equilibrium state (usually different from s) and awaits the next perturbing signal. Components in our framework obey a *synchrony hypothesis*[2], which asserts that no external perturbations occur during these transient flurries of *internal* signals. The distinction between equilibrium and transient states percolates down to the level of Mealy machines. In fact, the states of each Mealy machine may be strictly partitioned into two sets: *Equilibrium* and *Transient*.

4 Detailed Machine Design

We now describe the equilibrium states in each machine. There are six general categories of state:

notready the machine has either completed or has not been enabled,

active the machine (or one of its subordinates) has observed user activity.

committed the machine has not witnessed activity but is able to witness activity.

completable the machine has witnessed activity, and may complete on its own, or it may be completed implicitly by the activity of other machines.

interrupted the machine has been interrupted.

skippable the machine has not witnessed activity but may be completed implicitly by the activity of other machines.

We use these general categories for initially designing the detailed finite control of our Mealy machines. Actually describing this detailed control requires another notation.

4.1 Alt

The alt machine implements the | ordering. In the MTREE expression $E_1 \lhd \mathsf{alt} \rhd E_2$, the Mealy-machine $\mathfrak{M}[\![\mathsf{alt}]\!]$ must issue signals to the MTREEs E_1 and E_2 so that:

- The user will be presented with a choice, and
- Once a choice is observed, the other choice is no longer available.

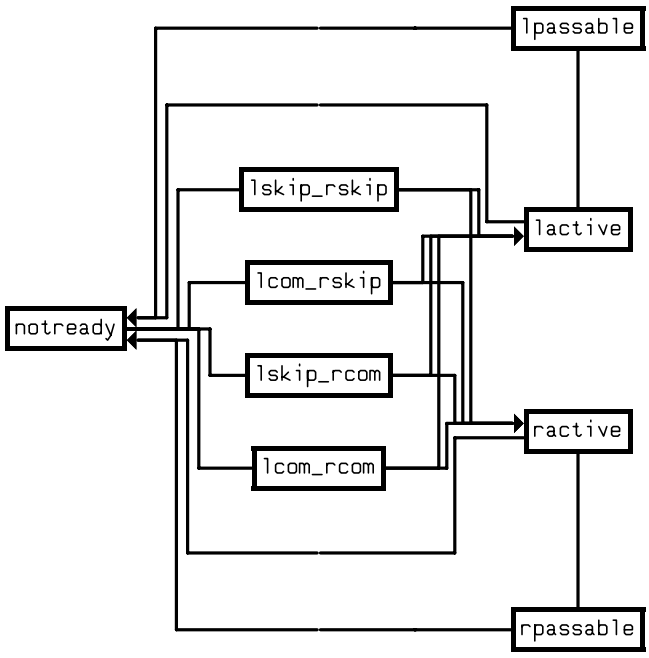

Figure 1: State topology for the alt machine

To implement this, Q_{alt} contains 110 states, of which the following 9 are equilibrium states:

$$\begin{aligned}
\mathit{LeftActive}_{\mathsf{alt}} &== \{\mathit{lactive}, \mathit{lpassable}\} \\
\mathit{RightActive}_{\mathsf{alt}} &== \{\mathit{ractive}, \mathit{rpassable}\} \\
\mathit{Choice}_{\mathsf{alt}} &== \{\mathit{lskip_rskip}, \mathit{lcom_rskip}, \mathit{lskip_rcom}, \mathit{lcom_rcom}\} \\
Q_{\mathsf{alt}} &\supset \{\mathit{notready}\} \\
&\cup \mathit{Choice}_{\mathsf{alt}} \\
&\cup \mathit{LeftActive}_{\mathsf{alt}} \\
&\cup \mathit{RightActive}_{\mathsf{alt}}
\end{aligned}$$

The set $\mathit{LeftActive}_{\mathsf{alt}}$ describes those states that remember activity in the left child, $\mathit{RightActive}_{\mathsf{alt}}$ describes those states that remember activity in the right child, and $\mathit{Choice}_{\mathsf{alt}}$ remembers describes those states that present the user with a choice between the left and right children.

4.2 Ceiling

The ceiling machine is used to top an MTREE so that parent channels will be resolved. In the MTREE expression $\mathsf{ceiling} \rhd E$, the Mealy-machine $\mathfrak{M}[\![\mathsf{ceiling}]\!]$ must issue signals to the MTREE's E so that E will be enabled. The ceiling machine has 20 states, of which the following are equilibrium states:

$$Q_{\mathsf{ceiling}} \supset \{\mathit{notready}, \mathit{ready}, \mathit{active}, \mathit{completable}\}$$

These states merely record the abstract status of the underlying MTREE.

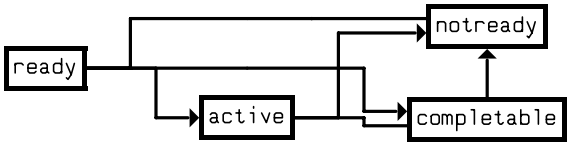

Figure 2: State topology for the ceiling machine

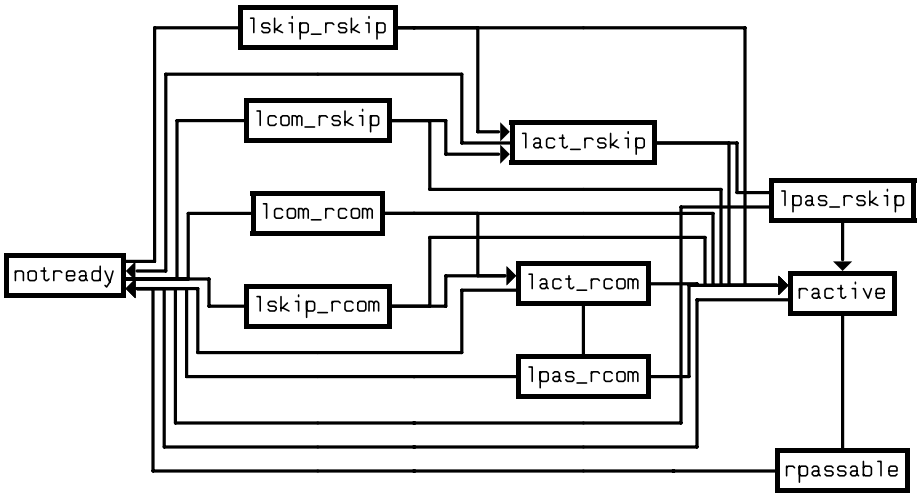

Figure 3: State topology for the dis machine

4.3 Dis

The dis machine implements the $[>$ ordering. In the MTREE expression $E_1 \lhd \mathsf{dis} \rhd E_2$, the Mealy-machine $\mathfrak{M}[\![\mathsf{dis}]\!]$ must issue signals to the MTREEs E_1 and E_2 so that:

1. The user may interact with and complete E_1,
2. At any time the user may interact with E_2, and
3. Any interaction with E_2 disables any further action with E_1.

This machine contains 153 states, of which the following are Equilibrium states:

$$
\begin{array}{lcl}
Normal_{\mathsf{dis}} & == & \{lact\_rcom, lact\_rskip, lcom\_rcom, lcom\_rskip, lpas\_rcom, lpas\_rskip, lskip\_rcom, lskip\_rskip\} \\
Disabled_{\mathsf{dis}} & == & \{ractive, rpassable\} \\
Q_{\mathsf{dis}} & \supset & \{notready\} \\
 & \cup & Normal_{\mathsf{dis}} \\
 & \cup & Disabled_{\mathsf{dis}}
\end{array}
$$

Those states in the *Normal* set represent the enabled activity; whereas those in the *Disabled* set represent activity once the normal activity has been disabled.

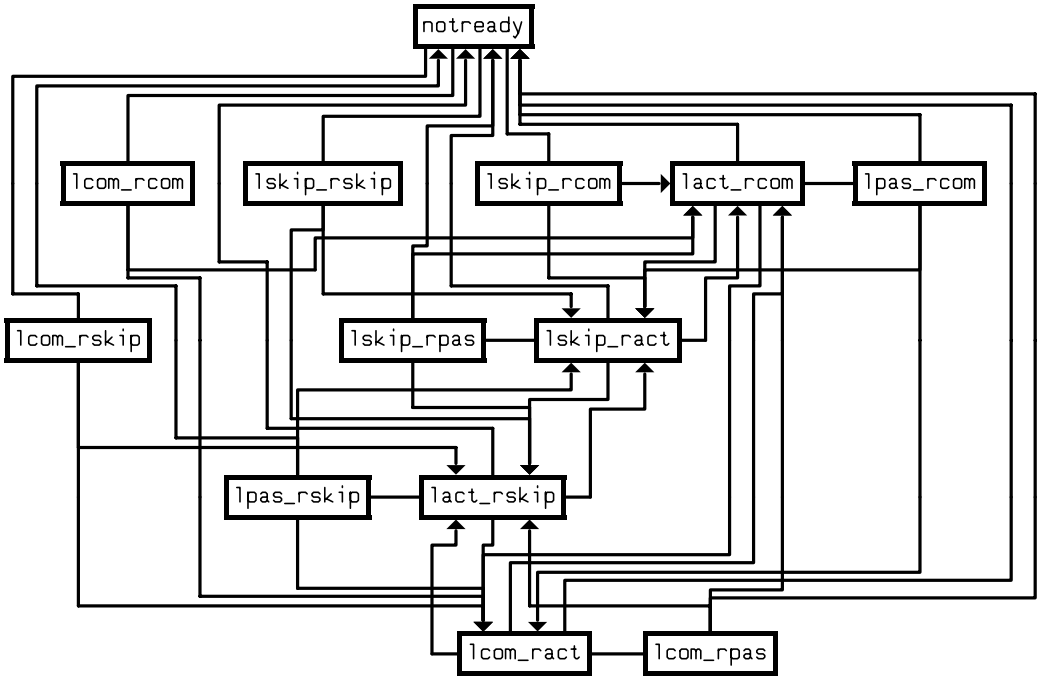

Figure 4: State topology for the excl machine

4.4 Excl

The excl machine implements the mutual disable ordering ($\leftrightarrow$). In the MTREE expression $E_1 \lhd \mathsf{excl} \rhd E_2$, the Mealy-machine $\mathfrak{M}[\![\mathsf{excl}]\!]$ must issue signals to the MTREEs E_1 and E_2 so that:

1. The user *always* has a choice between performing E_1 or E_2,
2. If either E_1 or E_2 is worked to completion, the whole task will be completed.

This machine contains 204 states, of which the following are Equilibrium states:

$$
\begin{array}{lll}
\mathit{Left\_Active}_{\mathsf{excl}} & == & \{\mathit{lact\_rcom}, \mathit{lact\_rskip}, \mathit{lpas\_rcom}, \mathit{lpas\_rskip}\} \\
\mathit{Right\_Active}_{\mathsf{excl}} & == & \{\mathit{lcom\_ract}, \mathit{lcom\_rpas}, \mathit{lskip\_ract}, \mathit{lskip\_rpas}\} \\
\mathit{No\_Choice}_{\mathsf{excl}} & == & \{\mathit{lcom\_rcom}, \mathit{lcom\_rskip}, \mathit{lskip\_rcom}, \mathit{lskip\_rskip}\} \\
Q_{\mathsf{excl}} & \supset & \{\mathit{notready}\} \\
 & \cup & \mathit{Left\_Active}_{\mathsf{excl}} \\
 & \cup & \mathit{Right\_Active}_{\mathsf{excl}} \\
 & \cup & \mathit{No\_Choice}_{\mathsf{excl}}
\end{array}
$$

4.5 Int

The int machine implements the interruption ordering ($\triangle$). In the MTREE expression $E_1 \lhd \mathsf{int} \rhd E_2$, the Mealy-machine $\mathfrak{M}[\![\mathsf{int}]\!]$ must issue signals to the MTREEs E_1 and E_2 so that:

1. Activity within E_2 may, at any time, interrupt activity within E_1, and

2. If interrupted, activity in E_1 may resume upon completion of E_2.

This machine contains 184 states, of which the following are Equilibrium:

$$
\begin{aligned}
Normal_{\mathsf{int}} &== \{lactive, lcommit, lpassable, lskippable, lcom\_rdone, lskip\_rdone\} \\
Interrupted_{\mathsf{int}} &== \{lcom\_ract, lcom\_rpas, lpas\_ract, lpas\_rpas, lskip\_ract, lskip\_rpas\} \\
Q_{\mathsf{excl}} &\supset \{notready\} \\
&\cup\ Normal\_int \\
&\cup\ Interrupted_{\mathsf{int}}
\end{aligned}
$$

4.6 Leaf

This machine contains 6 states.

4.7 Loop

The loop machine implements the looping ordering $(*)$. In the MTREE expression $\mathsf{loop} \triangleright E$, the Mealy-machine $\mathfrak{M}[\![\mathsf{loop}]\!]$ must issue signals to the MTREE E so that the user may initiate activity with E as many times as he or she likes while also being able to skip E. This machine contains 42 states, of which the following are Equilibrium:

$$Q_{\mathsf{loop}} == \{notready, ready, initial, active, passable\}$$

4.8 Opt

The opt machine is very similar to loop, except that only one completion of the subordinate task is allowed. This machine contains 29 states, of which the following are Equilibrium:

$$Q_{\mathsf{opt}} == \{notready, ready, active, passable\}$$

4.9 Par

While the largest, this machine is the most intuitive. It contains 276 states.

4.10 Seq

This machine contains 127 states, of which the following are equilibrium:

$$
\begin{aligned}
Left\_Active_{\mathsf{seq}} &== \{lactive, lcommit\} \\
Left\_Or\_Right_{\mathsf{seq}} &== \{ldone\_rskip, lpassable, lpas\_rskip, lskip\_rcom, lskip\_rskip\} \\
Right\_Active_{\mathsf{seq}} &== \{ractive, rcommit\} \\
Q_{\mathsf{seq}} &== \{notready\} \\
&\cup\ Left\_Active_{\mathsf{seq}} \\
&\cup\ Left\_Or\_Right_{\mathsf{seq}} \\
&\cup\ Right\_Active_{\mathsf{seq}}
\end{aligned}
$$

Table 3: Mealy machine (MM) syntax.

```
        State        ==  Equilibrium | Transient
        Equilibrium  ==  Identifier
        Transient    ==  '@'+ { Identifier }
        Identifier   ==  ['a'-'z''A'-'Z']['a'-'z''A'-'Z''0'-'9''_']*
Transaction  ==  Action { Reaction }
Action       ==  { ~ } Connector '?' Signal
Reaction     ==  '[' Connector '!' Signal ']'
Connector    ==  'p'                          /* parent */
              |  'l'                          /* left subordinate */
              |  'r'                          /* right subordinate */
Signal       ==  ['a'-'z']+
```

5 Support infrastructure

5.1 MM: A Notation for Mealy Machine Finite Control

The MM notation graphically describes the transition relationship of a Mealy Machine using a two-dimensional ASCII representation of state machine graphs. The definition of a particular machine often consists of many such descriptions joined together by unifying the names of states. The reader may immediately observe three things about this notation:

1. machine transitions are represented as `s -- c1?e1[c2!e2] --> t`, indicating that a transition takes place from state s to state t upon receipt of signal e_1 on connector c_1, and results in the issuing of signal e_2 down connector c_2,

2. the transitions can be laid out in two dimensions, thus simplifying input and understanding of the state topology, and

3. states are given either mnemonic names like `notready` or they are unnamed and represented with the `@` placeholder. Unnamed states are taken to be transient states; whereas named states are equilibrium states.

Table 3 describes the syntax of MM. Transitions are described as directed edges from one state to another with an intervening *transaction*. Transactions describe the action that stimulates the transition and any resulting reaction. States and transactions are connected using transition edges which appear as ASCII lines in the MM files. A line may be vertical, horizontal, slanted left or right, or bent with the junction symbol `+`. Transitions are always directed with a unique source and target state and a unique transaction. To specify the direction of a transition, one end must be affixed with an arrow point (either `>`, `<`, `^`, or `%`) depending on whether the edge is flowing to the right, to the left, up, or down respectively. Slanted edges cannot be given a direction, and so they must be bent into a non-slanted edge which can then be affixed with an arrow point. For example, `@ --- l?ack[p!ack] --> lcommit`, denotes a transition whose source is the transient state `@`, whose target is the state `lcommit`, and whose transaction is `l?ack[p!ack]`.

5.2 Generating MMTK Components from MM

The ultimate output of a Mealy machine finite control specification is a reusable component in the MASTERMIND toolkit. After having designed and tested the interoperation of the Mealy machines, we invoke a tool called the Mealy Machine Compiler (mmc) to create the C++ implementations. Mmc creates two files–a C++ header (.h) file and a C++ implementation (.cc) file–for each class of Mealy machine.

Consider, for example, the MM definition of the `seq` machine (which constitutes a number of .mm files). The command:

```
$ mmc -c -Mseq seq*.mm
```

creates two files: `MdlSeqOrdering.h` and `MdlSeqOrdering.cc`. The file `MdlSeqOrdering.h` defines the class `MdlSeqOrdering` and declares it to be a subclass of class `BinaryNode`. `MdlSeqOrdering` inherits a method called `getNextState` from `BinaryNode`. This method implements the Mealy machine transition relationship. It is defined in the file `MdlSeqOrdering.cc`.

Since there are ten MDL orderings, there are ten header and ten implementation files that make up the finite control constituent of the MASTERMIND toolkit.

References

[1] L. Bass and C. Unger, editors. *Engineering for Human Computer Interaction*. Chapman & Hall, 1996.

[2] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language; design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.

[3] Thomas Browne, David Davila, Spencer Rugaber, and Kurt Stirewalt. Using declarative descriptions to model user interfaces with MASTERMIND. In Fabio Paternò and Philippe Palanque, editors, *Formal Methods in Human Computer Interaction*. Springer-Verlag, 1997.

[4] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley Publishing Company, 1979.

[5] R. Neches, J. Foley, P. Szekely, P. Sukaviriya, P. Luo, S. Kovacevic, and S. Hudson. Knowledgeable development environments using shared design models. In *Intelligent Interfaces Workshop*, pages 63–70, January 1993.

[6] R. E. Kurt Stirewalt. *Automatic Generation of Interactive Systems from Declarative Models*. PhD thesis, Georgia Institute of Technology, December 1997.

[7] P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher. Declarative interface models for user interface construction tools : The mastermind approach. In L. Bass and C. Unger, editors, *Engineering for Human-Computer Interaction [1]*. Chapman & Hall, 1996.