Graphical Design of Reactive Systems[*]

Emil Sekerinski

McMaster University, Department of Computing and Software
Hamilton, Ontario, Canada, L8S 4K1
emil@mcmaster.ca

Abstract. Reactive systems can be designed graphically using statecharts. This paper presents a scheme for the translation of statecharts into the Abstract Machine Notation (AMN) of the B method. By an example of a conveyor system, we illustrate how the design can be initially expressed graphically with statecharts, then translated to AMN and analysed in AMN, and then further refined to executable code.

1 Introduction

Reactive systems are characterised as having to continuously react to stimuli from their environment, rather than having to produce a single outcome. Distributed systems, embedded systems, and real-time systems are examples of reactive systems.

Statecharts are a visual approach to the design of reactive systems [6]. Statecharts extend finite state diagrams, the graphical representation of finite state machines, by three concepts: hierarchy, concurrency, and communication. These three concepts give enough expressiveness for the specification of even complex reactive systems. Because of the appeal of the graphical notation, statecharts have gained some popularity. For example, statecharts are also part of object-oriented modelling techniques [11, 7].

This paper presents a scheme for the translation of statecharts to the Abstract Machine Notation (AMN) of the B method [1]. This translation scheme allows the design of reactive systems to be

1. initially expressed in the graphical notation of statecharts,
2. translated to AMN and analysed in AMN, e.g. for safety properties, and
3. further refined to AMN machines which can be efficiently executed.

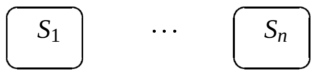
By this translation scheme, statecharts are given a formal semantics in terms of AMN. Several other definitions of the semantics of statecharts were proposed [4]. Although we largely follow the (revised) original definition [8], our goal is a semantic which harmonises well with the refinement calculus in general and with AMN in particular.

Sections 2 – 5 give the translation schemes for state diagrams, hierarchical state diagrams, state diagrams with concurrency, and state diagrams with broadcasting, respectively. Section 6 illustrates the approach by an example of a conveyor system. Section 7 compares this approach with other definitions of statecharts and other approaches to modelling reactive systems with AMN

[*] To appear in D. Bert (Ed.) *2nd International B Conference*, Montpellier, France, Springer-Verlag, 1998.

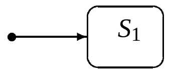
2 State Diagrams

A state diagram is a graphical representation of a finite state machine. State diagrams, the simplest form of statecharts, consists of a finite number of *states* and *transitions* between those state. Upon an *event*, a system (state machine) may evolve from one state into another. In statecharts, states are symbolised by (rounded) boxes. In AMN, the states of a state diagram are represented by a variable of an enumerated set type:



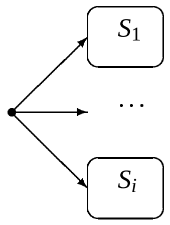
SETS
  $S = \{S_1, \dots, S_n\}$
VARIABLES
  s
INVARIANT
  $s \in S$

A system must have an initial state. In statecharts, an arrow with a fat dot points to the initial state. In AMN, this corresponds to initialising the state variable with that state:



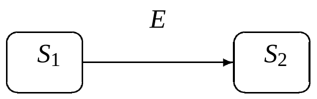
INITIALISATION
  $s := S_1$

A system can have several initial states, which leads to a nondeterministic initialisation:



INITIALISATION
  $s :\in \{S_1, \dots, S_i\}$

Events cause transitions from one state to another. Events may be *generated* by the environment. In statecharts, transitions are visualised by an arrow between two states, where the arrow is labelled with the event which triggers this transition. In AMN, events are represented by operations, which may be called by the environment. Upon an event, a transition takes only place if in the current state there is a transition on this event. Otherwise, the event is ignored. Suppose only one transition in the system for event E exists:



OPERATIONS
  $E \ \widehat{=}$ **IF** $s = S_1$ **THEN** $s := S_2$ **END**

In case there are several transitions labelled with E, the one starting from the current state is taken, if any transition is taken at all. Let $\{S'_1,\ldots,S'_i\} \subseteq S$, where $S'_1,\ldots,S'_i$ do not have to be distinct:

$$\begin{tikzcd}
\boxed{S_1} \arrow[r, "E"] & \boxed{S'_1} \\
\ldots & \ldots \\
\boxed{S_i} \arrow[r, "E"] & \boxed{S'_i}
\end{tikzcd}$$

```
OPERATIONS
  E ≙
    CASE s OF
      EITHER S_1 THEN s := S'_1
      ...
      OR S_i THEN s := S'_i
    END
```

For simplicity, we assume from now on that there is only one transition for any event. In case there are several, the above scheme has to be applied.

A transitions may have an *action* associated with it, which is performed when the transition takes places. Actions are assumed to be instantaneous. Typically, actions involve setting actuators or modifying global variables. With the ability to have global variables of arbitrary types, statecharts are not restricted to a finite state space. Here we allow actions to be statements of AMN. In statecharts, an action (or its name) is written by preceding it with a slash:

$$\begin{tikzcd}
\boxed{S_1} \arrow[r, "E/act"] & \boxed{S_2}
\end{tikzcd}$$

```
OPERATIONS
  E ≙ IF s = S_1 THEN s := S_2 || act END
```

An action may also be performed when setting an initial state. Here we give only the rule if there is one initial state:

$$\begin{tikzcd}
\bullet \arrow[r, "/act"] & \boxed{S_1}
\end{tikzcd}$$

```
INITIALISATION
  s := S_1 || act
```

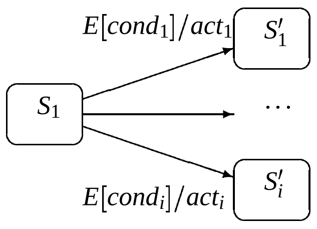
Transitions may be restricted to be taken only when an additional condition holds. Typically, conditions depend on sensor values and global variables. Here we allow conditions to be predicates of AMN. In statecharts, a condition is written in square brackets:

$$\begin{tikzcd}
\boxed{S_1} \arrow[r, "E[cond]/act"] & \boxed{S_2}
\end{tikzcd}$$

```
OPERATIONS
  E ≙
    IF s = S_1 ∧ cond THEN
      s := S_2 || act
    END
```

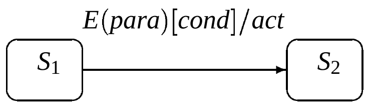
If an action or a condition mentions variable v of type T, then v has to be added to the **VARIABLES** section and $v \in T$ has to be added to the **INVARIANT** section.

So far all transitions on an event are from different states. However, there may also be two or more transition from a single state on the same event. If the conditions are disjoint, then at most one transition can be enabled. If the conditions are overlapping, an enabled transition is chosen nondeterministically:



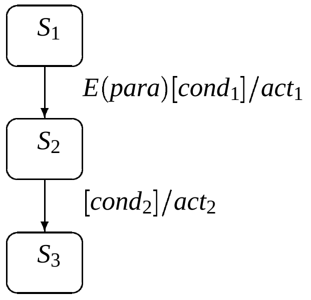
```
**OPERATIONS**
  $E \;\widehat{=}$
    **SELECT** $s = S_1 \wedge cond_1$ **THEN**
      $s := S'_1 \parallel act_1$
    $\dots$
    **WHEN** $s = S_1 \wedge cond_i$ **THEN**
      $s := S'_i \parallel act_i$
    **ELSE** $skip$
    **END**
```

Finally, a transition may have parameters, which are supplied by the environment which generates the event. These parameters may be used in the condition and the action of the transition:



```
**OPERATIONS**
  $E(para) \;\widehat{=}$
    **IF** $s = S_1 \wedge cond$ **THEN**
      $s := S_2 \parallel act$
    **END**
```

As a special case, a transition can be without an event, known as a *spontaneous* transition. In this case the transition leading to a state with a spontaneous transition needs to be continued immediately by this transition, if the condition is true. Spontaneous transitions allow sharing of a condition and an action if several transitions lead to the starting state of a spontaneous transition:

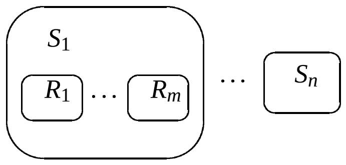


```
**OPERATIONS**
  $E(para) \;\widehat{=}$
    **IF** $s = S_1 \wedge cond_1 \wedge cond_2$ **THEN**
      $s := S_3 \parallel act_1 \parallel act_2$
    **END**
```

For brevity, we continue to consider transitions labelled only by an event. Parameters, conditions, and actions can be added as given above.

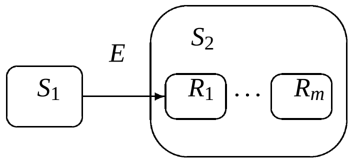
3 Hierarchy

States can have substates. If the system is in a state with substates, it is also in exactly one of those substates. Conversely, if a system is in a substate of a superstate, it is also in that superstate. In statecharts, a superstate with substates is drawn by nesting. In AMN, we model the substates by an extra variable for each superstate containing substates. This generalises to substates which again contain substates:



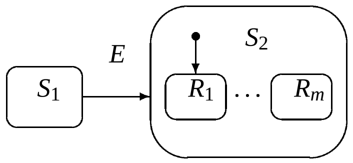
```
SETS
  $S = \{S_1, \dots, S_n\}$ ;
  $R = \{R_1, \dots, R_m\}$
VARIABLES
  $s, r$
INVARIANT
  $s \in S \wedge r \in R$
```

When entering a superstate, the substate to be entered has to be specified as well. In statecharts this is expressed by letting the transition arrow point to a specific substate. In AMN this corresponds to setting the variable for both the superstate and the substate(s) appropriately:



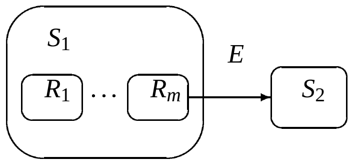
```
OPERATIONS
  $E \;\widehat{=}$
    IF $s = S_1$ THEN
      $s := S_2 \parallel r := R_1$
    END
```

If a transition arrow points only to the contour of a superstate, then an initial substate must be marked as such. This is identical to letting the transition arrow point directly to that initial state. This is useful if the superstate is referred to only by its name and is defined separately:



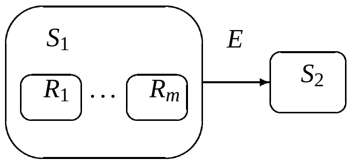
```
OPERATIONS
  $E \;\widehat{=}$
    IF $s = S_1$ THEN
      $s := S_2 \parallel r := R_1$
    END
```

A transition can leave both a state and its superstate. In AMN, this corresponds to simply setting a new value for the variable of the superstate:



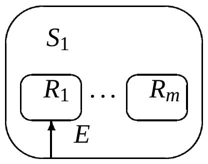
```
OPERATIONS
  $E \;\widehat{=}$
    IF $s = S_1 \wedge r = R_m$ THEN
      $s := S_2$
    END
```

If a transition arrow starts at the contour of a superstate, then this is like having an arrow starting from each of the substates, going to the same state on the same event. This is one of the ways statecharts economise on drawing arrows. In AMN, this corresponds to simply setting a new value of the variable of the superstate, irrespective of the current substate. Note that the basic schema for the transition between two simple states is a special case of this:



OPERATIONS
$E \;\widehat{=}\;$ **IF** $s = S_1$ **THEN** $s := S_2$ **END**

In statecharts, a transition can point from the contour of a superstate inside to one of the substates. This is like having a transition from all of the substates on that event to this substate. It is again a way statecharts economise on drawing arrows. In AMN, this corresponds to going to that substate irrespective of the value of the variable for the substate:

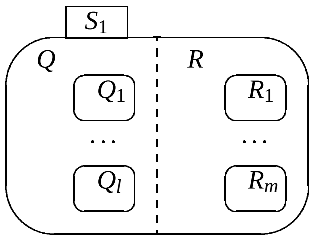


OPERATIONS
$E \;\widehat{=}\;$ **IF** $s = S_1$ **THEN** $r := R_1$ **END**

The schema for multiple transitions on an event and for multiple transitions starting from the same state apply to hierarchical states as well. However, it should be pointed out that if there is a transition on an event between two substates and between the superstate of those substates and another state, then the schema for multiple transitions from the same state applies, which gives rise to nondeterminism.

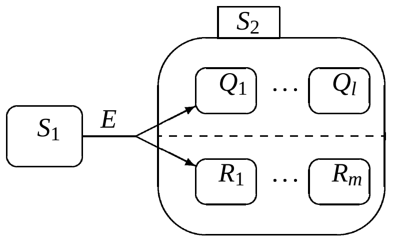
4 Concurrency

Reactive systems are naturally decomposed into concurrent activities. In statecharts, concurrency is expressed by orthogonality: a system can be in two independent states simultaneously. This is drawn by splitting a state with a dashed line into independent substates, each of which consists of a number of states in turn. In AMN, this corresponds to declaring a variable for each of the (two or more) concurrent states. These variables are only relevant if the system is in the corresponding superstate:



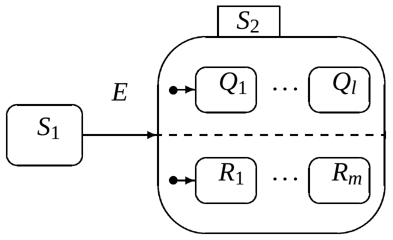
SETS
$S = \{S_1, \dots, S_n\}$;
$Q = \{Q_1, \dots, Q_l\}$;
$R = \{R_1, \dots, S_m\}$
VARIABLES
s, q, r
INVARIANT
$s \in S \;\wedge\; q \in Q \;\wedge\; r \in R$

A state with concurrent substates is entered by a fork into states in each of the concurrent substates. In AMN, this corresponds to setting the variables for all the concurrent states:



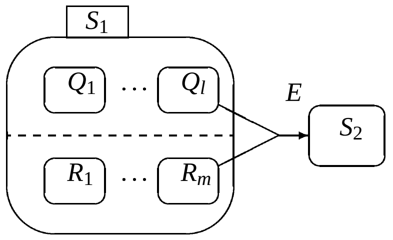
```
OPERATIONS
  E ≙
    IF $s = S_1$ THEN $s := S_2$ ∥
      $q := Q_1 \parallel r := R_1$
    END
```

If a transition arrow points only to the contour of a state with concurrent substates, then an initial state must be marked as such in all of the concurrent states. This is identical to a fork of the transition arrow to all those initial states. This is useful if the state is referred to only by its name and is defined separately:



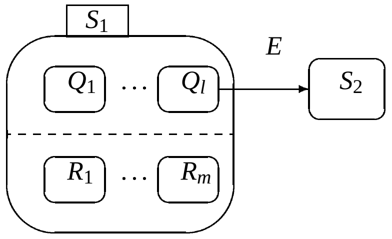
```
OPERATIONS
  E ≙
    IF $s = S_1$ THEN $s := S_2$ ∥
      $q := Q_1 \parallel r := R_1$
    END
```

A state with concurrent substates is exited by a join from all the concurrent states. Such a transition can only be taken if all concurrent states are in the substate from where the transition arrow starts. In AMN, this implies testing all substates before making the transition:



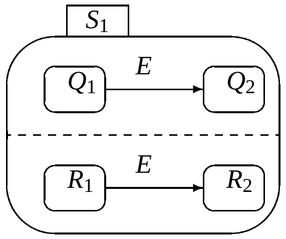
```
OPERATIONS
  E ≙
    IF $s = S_1 \land q = Q_l \land r = R_m$ THEN
      $s := S_2$
    END
```

If a transition arrow for exiting a state with concurrent substates starts only at one of the states in the concurrent substates, then this is like having a join from all other states of the concurrent state. Again, this is one of the ways statecharts economise on drawing arrows. In AMN, this corresponds to simply setting a new value of the variable of the superstate, irrespective of the concurrent substate:



```
OPERATIONS
  $E \mathrel{\widehat{=}}$
    IF $s = S_1 \land q = Q_l$ THEN
      $s := S_2$
    END
```

Two concurrent states may have transitions on the same event. In case this event occurs, these transitions are taken simultaneously. In AMN, this corresponds to the parallel composition of the transitions. Note that this has implications on the global variables which can occur in the conditions and the actions, a variable can only be assigned by one action:



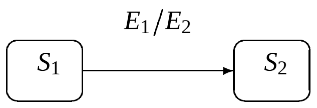
```
OPERATIONS
  $E \mathrel{\widehat{=}}$
    BEGIN
      IF $q = Q_1$ THEN $q := Q_2$ END $\|$
      IF $r = R_1$ THEN $r := R_2$ END
    END
```

5 Communication

Communication between concurrent states is possible in three ways: First, concurrent states can communicate by global variables. These can be set in actions and read in actions and conditions, following the rules for variables given earlier. Secondly, the condition or the action of a transition may depend on the current substate of a concurrent state. The test whether state S is in substate S_i is written *in* S_i. In AMN, this corresponds to comparing the state variable to S_i:

in S_i	$s = S_i$

Thirdly, concurrent states can communicate by broadcasting of events. On a broadcast of an event, all concurrent states react simultaneously. Events are either generated internally through a broadcast or externally by the environment. Broadcasting of events is associated to transitions and is drawn the same way as an action. In AMN, broadcasting an event corresponds to calling the operation for that event:



```
OPERATIONS
  $E \mathrel{\widehat{=}}$ IF $s = S_1$ THEN $s := S_2 \| E_2$ END
```

As calling an operation is a statement, which can appear as an action, the above schema is just a special case of the schema for translating actions. In general, broadcasting of an event may occur anywhere within an action. Care has to be taken since the resulting code may be illegal: an operation of a machine cannot be called from another operation of the same machine. This can be resolved with auxiliary definitions of the bodies of the operations and using those instead of the operations. This way of resolving the dependencies between operations has the benefit that no cycles between operations can be introduced as an effect of broadcasting.

The evolution of a system can be studied by considering its *configurations*. A (basic) configuration is a tuple of basic, concurrent states. The basic states determine uniquely the superstates in which the system is and hence a configuration gives the complete information about the current states of a system. We denote the fact that a system makes a transition from configuration C to C' on event E by $C \xrightarrow{E} C'$.

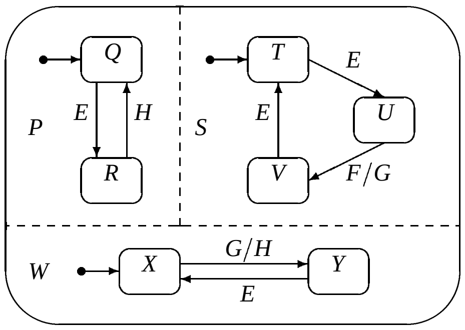


Fig. 1. Statechart with simultaneous transitions on event E and a chain reaction on event F.

In the statechart of Fig. 1, the initial configuration is (Q,T,X). On external event E, states P and S change from Q to R and T to U, respectively, and W remains in X:

$$(Q,T,X) \xrightarrow{E} (R,U,X)$$

If external event F follows, then event G is generated in S on the transition from U to V, which in turn generates event H in W on the transition from Y to Z. This causes the transition from R to Q in P:

$$(R,U,X) \xrightarrow{F} (Q,V,Y)$$

Figure 2 gives the code which results from the translation. It exhibits the above behaviour.

6 Example

We illustrate the technique by an example of a conveyor system. A conveyor belt and an elevating and rotation table are placed next to each other (see Fig. 3). The conveyor belt

```
MACHINE $M$
SETS
  $P = \{Q, R\}$ ;
  $S = \{T, U, V\}$ ;
  $W = \{X, Y\}$
VARIABLES
  $p, s, w$
INVARIANT
  $p \in P \ \wedge \ s \in S \ \wedge \ w \in W$
INITIALISATION
  $p := Q \parallel s := T \parallel w := X$
DEFINITIONS
  $HH == ($IF $p = R$ THEN $p := Q$ END$)$ ;
  $GG == ($IF $w = X$ THEN $w := Y \parallel HH$ END$)$
OPERATIONS
  $E$ =
    BEGIN
      IF $p = Q$ THEN $p := R$ END $\parallel$
      CASE $s$ OF
        EITHER $T$ THEN $s := U$
        OR $V$ THEN $s := T$
        END
      END $\parallel$
      IF $w = Y$ THEN $w := X$ END
    END ;
  $F$ = IF $s = U$ THEN $s := V \parallel GG$ END ;
  $G$ = $GG$ ;
  $H$ = $HH$
END
```

Fig. 2. AMN machine which corresponds to the statechart of Fig. 1

has to transport objects which are placed on the left of the conveyor belt to its right end and then on the table. The table then elevates and rotates the object to make it available for processing by further machines.

The conveyor belt has a photo-electric cell which signals when an object has arrived at the right end or has left the belt (and thus has moved onto the table). The motor for the belt may be switched on and off: it has to be on while waiting for a new object and has to be switched off when an object is at the end of the belt but cannot be delivered onto the table because the table is not in proper position.

The table lifts and rotates an object clockwise to a position for further processing. The table has two reversing electric motors, one for elevating and one for rotating. Mechanical sensors indicate whether the table is at its left, right, upper, and lower end position, respectively. The table must not move beyond its end position. We assume that initially the table is in its lower left position.

Both machines run in parallel, thus allowing several objects to be transported simultaneously. The program has to ensure that all objects are transported properly, i.e. objects leave the conveyor belt only if the table is in lower left position.

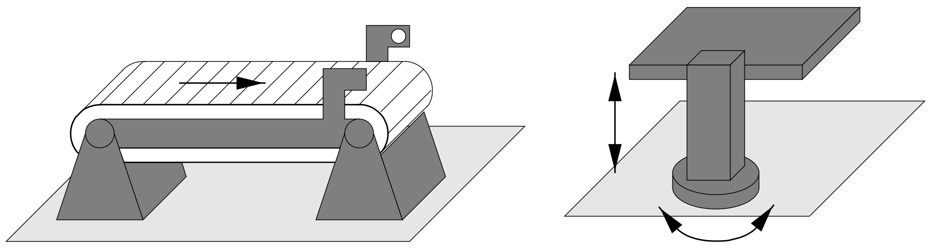


Fig. 3. The conveyor belt on the left and the elevating and rotating table on the right

Statechart of the Conveyor System

The statechart of the conveyor system is given in Fig. 4. We assume that the events *SensorOff* and *SensorOn* for the conveyor belt and the events *UpReached*, *DownReached*, *RightReached*, *LeftReached*, and *ObjectTaken* for the Table are generated by the environment (hardware). The events *ContinueDelivery* and *ObjectPlaced* serve internally for the communication between the concurrent *ConveyorBelt* and *Table* states. At this stage, we consider only the abstract states of the system, like the table being in loading position, rather than the detailed states of the actuators and sensors. As common in statecharts, some states are not given names.

AMN Specification

An AMN machine of the conveyor system is given in Fig. 5. Note that the names *MovingToUnloading* and *MovingToLoading* for the two unnamed states of *Table* and the

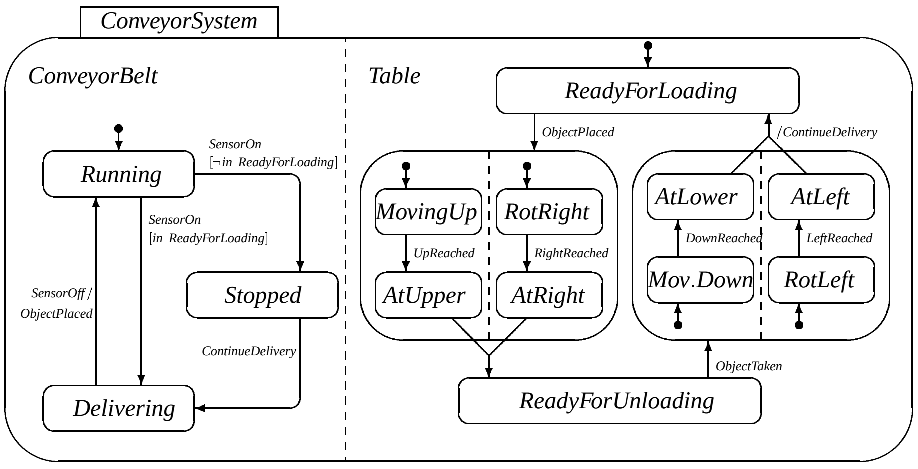


Fig. 4. The statechart of the conveyor system

sets *ToUpper*, *ToRight*, *ToLower*, *ToLeft* for their concurrent substates are introduced. As the events *ObjectPlaced* and *ContinueDelivery* are generated only internally, they are defined only in the **DEFINITIONS** section.

Safety Properties

Given an AMN machine of the conveyor system, safety properties can be expressed as invariants. For example, the condition that the conveyor belt must be *Delivering* only if the table is *ReadyForLoading* can be proved by adding following implication to *ConveyorSystem*:

INVARIANT
$(conveyorBelt = Delivering \Rightarrow table = ReadyForLoading)$

Note that for this safety property to hold, it is essential that the events *ObjectPlaced* and *ContinueDelivery* are considered to be internal events. If they were operations rather than definitions in *ConveyorSystem*, they could be called from the environment at any time and would violate this safety property.

Implementation

An implementation has to set the actuators. Assuming that following types are given,

SETS
$MOTOR = \{RUN, HALT\}$;
$REVMOTOR = \{FWD, BACK, STOP\}$

```
MACHINE ConveyorSystem
SETS
  ConveyorBelt = {Running, Stopped, Delivering} ;
  Table = {ReadyForLoading, MovingToUnloading, ReadyForUnloading, MovingToLoading} ;
  ToUpper = {MovingUp, AtUpper} ;
  ToRight = {RotRight, AtRight} ;
  ToLower = {MovingDown, AtLower} ;
  ToLeft = {RotLeft, AtLeft}
VARIABLES
  conveyorBelt, table, toUpper, toRight, toLower, toLeft
INVARIANT
  conveyorBelt ∈ ConveyorBelt ∧ table ∈ Table ∧
  toUpper ∈ ToUpper ∧ toRight ∈ ToRight ∧ toLower ∈ ToLower ∧ toLeft ∈ ToLeft
INITIALISATION
  conveyorBelt := Running ∥ table := ReadyForLoading ∥
  toUpper :∈ ToUpper ∥ toRight :∈ ToRight ∥ toLower :∈ ToLower ∥ toLeft :∈ ToLeft
DEFINITIONS
  ObjectPlaced ==
    (IF table = ReadyForLoading THEN table := MovingToUnloading ∥
      toUpper := MovingUp ∥ toRight := RotRight
    END) ;
  ContinueDelivery ==
    (IF conveyorBelt = Stopped THEN conveyorBelt := Delivering END)
OPERATIONS
  SensorOn ≙
    SELECT conveyorBelt = Running ∧ table = ReadyForLoading THEN
      conveyorBelt := Delivering
    WHEN conveyorBelt = Running ∧ table ≠ ReadyForLoading THEN
      conveyorBelt := Stopped
    ELSE skip
    END ;
  SensorOff ≙
    IF conveyorBelt = Delivering THEN ObjectPlaced ∥ conveyorBelt := Running END ;
  ObjectTaken ≙
    IF table = ReadyForUnloading THEN table := MovingToLoading ∥
      toLower := MovingDown ∥ toLeft := RotLeft
    END ;
  UpReached ≙
    IF table = MovingToUnloading ∧ toUpper = MovingUp THEN toUpper := AtUpper ∥
      IF toRight = AtRight THEN table := ReadyForUnloading END
    END ;
  RightReached ≙
    IF table = MovingToUnloading ∧ toRight = RotRight THEN toRight := AtRight ∥
      IF toUpper = AtUpper THEN table := ReadyForUnloading END
    END ;
  DownReached ≙
    IF table = MovingToLoading ∧ toLower = MovingDown THEN toLower := AtLower ∥
      IF toLeft = AtLeft THEN table := ReadyForLoading ∥ ContinueDelivery END
    END ;
  LeftReached ≙
    IF table = MovingToLoading ∧ toLeft = RotLeft THEN toLeft := AtLeft ∥
      IF toLower = AtLower THEN table := ReadyForLoading ∥ ContinueDelivery END
    END
END
```

Fig. 5. AMN specification of the conveyor system

variables for the conveyor belt motor and the two table motors can be introduced and related to the original states:

VARIABLES
$\quad beltMotor, elevMotor, rotMotor$
INVARIANT
$\quad beltMotor \in MOTOR \wedge elevMotor \in REVMOTOR \wedge rotMotor \in REVMOTOR \wedge$
$\quad (conveyorBelt \in \{Running, Delivering\} \Rightarrow beltMotor = RUN) \wedge$
$\quad (conveyorBelt = Stopped \Rightarrow beltMotor = HALT) \wedge$
$\quad (table = MovingToUnloading \wedge toUpper = MovingUp) \Leftrightarrow (elevMotor = FWD) \wedge$
$\quad (table = MovingToLoading \wedge toLower = MovingDown) \Leftrightarrow (elevMotor = BACK) \wedge$
$\quad (table = MovingToUnloading \wedge toRight = RotRight) \Leftrightarrow (rotMotor = FWD) \wedge$
$\quad (table = MovingToLoading \wedge toLeft = RotLeft) \Leftrightarrow (rotMotor = BACK)$

The implementation then needs to update these variables accordingly. The implementation could also reduce the state space of the specification since not all the information in there is necessary for controlling the conveyor system. Finally, the implementation has to replace parallel composition by compilable constructs like sequential composition. These are all standard refinement steps in AMN but are not common for statecharts.

An alternative to this refinement approach is to provide the information about the actuators and how they are updated in the original statechart itself. We believe this is an unnecessary overloading of the statechart since it is not needed for an abstract understanding of the system. This information can be better presented separately and related to the statechart and checked for consistency via refinement relations.

7 Discussion

We briefly compare our semantics with the (revised) original definition [8]. One difference is that we currently do not give priorities to transitions. If there is the possibility of either a transition between two substates within a superstate or from one of the substates out of the superstate to another state, then we leave this choice nondeterministic. In the original statechart semantics, priority is given to the transitions which leave the superstate.

The other major difference comes from our use of parallel composition for the combination of all actions in concurrent states which are triggered by the same event. Two or more concurrent actions must not modify the same variables. This eliminates any nondeterminism which would be otherwise present. However, it also disallows that any two actions which are taken simultaneously may generate the same event (since this corresponds to a parallel call of the same operation, which inevitably causes a read/write conflict).

A number of useful features of statecharts have been left out [6]:

1. activities which take time (in contrast to actions which don't);
2. entry and exit actions in states for all incoming and outgoing transitions, resp.;
3. internal actions which leave the system in the substates is was;

4. timing, e.g. by an event $timeout(E, d)$ which is generated exactly d time units after event E is generated;
5. transitions with multiple targets depending on a condition;
6. histories, for returning to the same substates from which a superstate was left;
7. overlapping states;
8. boolean expressions for events, e.g. $E \wedge \neg F$.

These remain the subject of further research. We believe that most can be treated by straightforward extensions, possibly with the exception of the last one. Also, as soon as the interaction between statechart features gets more involved, a more rigorous definition of the translation to AMN becomes desirable.

The use of statecharts as well as other graphical, object-oriented techniques in the formal program development with AMN is studied in [9]. There, the more general situation is studied when statecharts describe the behaviour of a set of objects, rather than a single object as here. Statechart events are translated to operations in AMN, as here, but with the initial state of a transition being part of the precondition of the corresponding operation. However, this implies that during refinement this precondition can be weakened and more transitions become possible, which is not the case here.

The theory of action systems can also be applied to the refinement of reactive systems in AMN [5, 13]. Although action systems can be naturally expressed in AMN, with each action becoming an AMN operation with a guard (rather than a precondition), an additional proof obligation (for the exit condition) is necessary, which is not part the AMN refinement rule. With the modelling of reactive systems by statecharts in AMN as studied here, the AMN refinement rule is sufficient. However, the reason for this is that action systems describe reactive behaviour in terms of actions, which are not called but become enabled, whereas the reactive behaviour of statecharts in AMN is in terms of operations, which are called. Hence, the operations here correspond to procedures rather than actions [3]. These serve different purposes: AMN operations as procedures describe ‘abstract machines’, whereas AMN operations as actions describe ‘abstract systems’ [2].

Statecharts have also been combined with the Z specification language in various ways, for example for formalising different aspects of a reactive system in Z and statecharts and then checking the consistency of these views [12, 14], or by extending the statechart semantics to include Z data types and operations [10]. By comparison, here we embed statecharts in AMN. In all cases, this gives similar possibilities of specifying the state space with data types and the modification of the state space in actions with operations of Z and AMN, respectively. The embedding of statecharts in AMN offers the further possibility of subsequent refinement: the statechart specification may give a concise, abstract view of the system; implementation details can be introduced gradually in refinement steps.

Acknowledgement. We are grateful to the reviewers for their careful reading and for pointing out related literature.

References

1. J.-R. Abrial. *The B Book: Assigning Programs to Meaning.* Cambridge University Press, 1996.
2. J.-R. Abrial. Extending B without changing it. In H. Habrias, editor, *First B Conference*, pages 169–170, Nantes, France, 1996. Institut de Recherche en Informatique de Nantes.
3. R. J. R. Back and K. Sere. Action systems with synchronous communication. In E.-R. Olderog, editor, *IFIP Working Conference on Programming Concepts, Methods, Calculi*, pages 107–126, San Miniato, Italy, 1994. North-Holland.
4. M. von der Beck. A comparison of statechart variants. In H. Langmaack, W.-P. deRoever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science 863, pages 128–148. Springer Verlag, 1994.
5. M. Butler and M. Walden. Distributed system development in B. In H. Habrias, editor, *First B Conference*, pages 155–168, Nantes, France, 1996. Institut de Recherche en Informatique de Nantes.
6. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
7. D. Harel and E. Gery. Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42, 1996.
8. D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(5):293–333, 1996.
9. K. Lano, H. Haughton, and P. Wheeler. Integrating formal and structured methods in object oriented system development. In S. J. Goldsack and S. J. H. Kent, editors, *Formal Methods and Object Technology*. Springer-Verlag, 1996.
10. C. Petersohn. *Data and Control Flow Diagrams, Statecharts and Z: Their Formalization, Integration, and Real-Time Extension.* Doctoral thesis, Christian-Albrecht Universität, 1997.
11. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddi, and W. Lorensen. *Object-Oriented Modelling and Design.* Prentice-Hall, 1991.
12. F. G. Shi, J. A. McDermid, and J. M. Armstrong. An introduction to ZedCharts and its application. Research Report YCS-96-272, University of York, Department of Computer Science, 1996.
13. M. Walden and K. Sere. Refining action systems within B-Tool. In M.-C. Gaudel and J. Woodcock, editors, *FME'96: Industrial Benefit and Advances in Formal Methods*, Lecture Notes in Computer Science 1051, pages 85–104. Springer-Verlag, 1996.
14. M. Weber. Combining statecharts and Z for the design of safety-critical control systems. In M.-C. Gaudel and J. Woodcock, editors, *FME '96: Industrial Benefits and Advances in Formal Methods*, Lecture Notes in Computer Science 1051, pages 307–326. Springer Verlag, 1996.