Generating Hierarchical State Based Representation From Event-B Models

Dipak L. Chaudhari [1,2]

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay,
Mumbai, India

Om P. Damani [3]

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay,
Mumbai, India

Abstract

Many properties of a system may not be obvious just by a quick inspection of the corresponding Event-B model. Users typically rely on animation, scenario analysis, and inspection of state transition graphs for discovering certain behavior of the system. We propose a methodology for generating a hierarchical representation of the system for visualising Event-B models. Our representation is succinct and it provides multiple views to aid in better comprehension of the Event-B models.

Keywords: Event-B, Model visualization, Hierarchical state based representation

1 Introduction

In Event-B, desired global properties of the system are specified in the form of *invariants* and the *invariant preservation proofs* ensure that these properties are maintained by the system after execution of any enabled event[1]. However, after execution of an enabled event, it is not obvious which events will be enabled or disabled next.

[1] This work was supported in part by the Ministry of Human Resources Development, Government of India and by the Tata Research Development and Design Center (TRDDC).
[2] Email: dipakc@cse.iitb.ac.in
[3] Email: damani@cse.iitb.ac.in

Users typically rely on animation, scenario analysis, and inspection of state transition graphs to grasp the behavioral aspects of the system. The ProB animator[9], with the aid of a model checker, can generate graphical visualization of the state space of a B machine. However, because of the flat (non-hierarchical) nature of the ProB state space representation, it becomes difficult to reduce the complexity of the state space graphs even after employing the state space reduction techniques[10]. In general, hierarchical state transition diagrams are found to be useful in reducing the complexity of the state transition diagrams [6].

We propose a hierarchical representation, similar to the statechart diagrams, for visualising Event-B models. We present a top-down methodology for constructing an abstract representation of desired granularity directly from the given Event-B model.

2 Hierarchical Abstract State Transition Machine

To represent a discrete event system, we use a Hierarchical Abstract State Transition Machine (HASTM) representation which uses the concepts of hierarchical states and guarded transitions similar to those in statechart diagrams [6]. In HASTM, state-space is arranged in the form of a tree (which we call a state-space partition tree) and the root node of the tree represents all the valid states of the system, i.e., the states defined by the conjunction of all the invariants. The root node is partitioned into substates based on some predicate. The substates are in turn partitioned further using appropriate predicates. Figure 1 shows the state-space partition tree generated by our method (Algorithm 1) for the Lift Event-B model[4] shown in Figure 2, given the predicates ($cf = topFloor$), ($cf = botFloor$), ($doorOpen = T$), and ($dirUp = T$). The algorithm starts constructing the tree from the root node and at each node selects a partitioning predicate that minimizes the number of transitions in the generated HASTM. This reduces the complexity of the generated HASTM. While partitioning the tree, the algorithm also computes the pre-states, transition guards, and the post states (defined in Section 2.1) for the transitions. The final HASTM for the Lift model is shown in Figure 4.

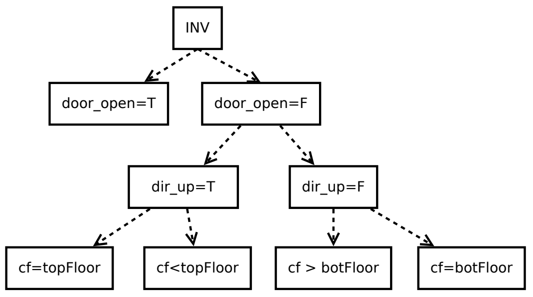

Fig. 1: State-space partition tree for the Lift example. The hierarchy relation is shown by dotted arrow.

2.1 Structure and Semantics of HASTM

If v denotes the variables of a system then the set $\Phi = \{v|\text{True}\}$ is the entire state space of the system. We use the term *abstract state* to represent any subset of Φ and

[4] The Lift Event-B model is adapted from the B model that comes with the ProB tool [9].

```
Constants:
  $botFloor$, $topFloor$
Variables:
  $cf$, $doorOpen$, $callbtns$, $dirUp$
Axioms:
  $botFloor \in \mathbb{Z}$, $topFloor \in \mathbb{Z}$
  $botFloor < topFloor$
Invariants:
  $doorOpen \in BOOL$
  $callbtns \subseteq$
      $(botFloor..topFloor)$
  $dirUp \in BOOL$
  $(doorOpen = T)$
      $\Rightarrow (cf \in callbtns)$
Init $\widehat{=}$
begin
  $cf := botFloor$
  $doorOpen := F$
  $callbtns := \emptyset$
  $dirUp := T$
end

PushCallBtn $\widehat{=}$
any $f$ where
  $f \in topFloor..botFloor$
  $f \notin callbtns$
  $f \neq cf$
then
  $callbtns := callbtns \cup \{f\}$
end
OpenDoor $\widehat{=}$
when
  $doorOpen = F$
  $cf \in callbtns$
then
  $doorOpen := T$
end
CloseDoor $\widehat{=}$
when
  $doorOpen = T$
then
  $doorOpen := F$
  $callbtns :=$
      $callbtns \setminus \{cf\}$
end

MoveUp $\widehat{=}$
when
  $dirUp = T$
  $doorOpen = F$
  $upRequested$
  $cf < topFloor$
  $cf \notin callbtns$
then
  $cf := cf + 1$
end
MoveDown $\widehat{=}$
when
  $dirUp = F$
  $doorOpen = F$
  $downRequested$
  $cf > botFloor$
  $cf \notin callbtns$
then
  $cf := cf - 1$
end

ReverseUp $\widehat{=}$
when
  $dirUp = F$
  $doorOpen = F$
  $upRequested$
  $\neg downRequested$
  $cf \notin callbtns$
then
  $dirUp := T$
end
ReverseDown $\widehat{=}$
when
  $dirUp = T$
  $doorOpen = F$
  $\neg upRequested$
  $downRequested$
  $cf \notin callbtns$
then
  $dirUp := F$
end
```

Fig. 2. Event-B Model for the Lift Controller. *upRequested* stands for $\exists c.(c \in \mathbb{Z} \wedge c > cf \wedge c \in callbtns)$, *downRequested* stands for $\exists c.(c \in \mathbb{Z} \wedge c < cf \wedge c \in callbtns)$,$T$ stands for $True$, and F stands for $False$.

the term *concrete state* or just *state* to represent a particular element of Φ.[5]

Abstract states are usually specified using predicates. If $Q(v)$ is a predicate with free variables in v then we represent by Q the set of all concrete states satisfying $Q(v)$, i.e., $Q = \{v|Q(v)\}$. If a system is in a concrete state q, and $q \in Q$ where Q is an abstract state then the system is said to be in the abstract state Q.

HASTM is a tuple $\mathcal{H} = \langle v, \mathcal{S}, \succ, \Sigma, T, t_0 \rangle$, where

- v denotes the variables of the system.
- $\mathcal{S}$ is a finite set of abstract states. $\mathcal{S} \subseteq \mathcal{P}(\Phi)$ where Φ is the set of all the states of the system.
- $\succ$ is a hierarchy relation on $\mathcal{S}$ that satisfies the following conditions
  - For any two abstract states X and Y in $\mathcal{S}$, $X \succ Y \Rightarrow X \supseteq Y$.
  - There exists a unique abstract state $r \in \mathcal{S}$, called the root state of $\mathcal{H}$ and denoted as $root(\mathcal{H})$, such that $r \notin ran(\succ)$, where $ran(\succ)$ is the range of the relation $\succ$.
  - For every $X \in \mathcal{S} \setminus \{root(\mathcal{H})\}$, there exists a unique abstract state $Y \in \mathcal{S}$ such that $Y \succ X$. Y is called the *immediate superstate* of X, whereas X is called an *immediate substate* of Y. An abstract state without any immediate substate is called a *basic abstract state*.
  - If X is a non-basic abstract state in $\mathcal{S}$ and $\mathcal{W}$ is a set of all immediate substates of X then $\mathcal{W}$ partitions the set X. i.e., all the immediate substates of X are collectively exhaustive ($\bigcup(\mathcal{W}) = X$) and any two distinct immediate substates of X are mutually exclusive. $(\forall A \forall B).A \in \mathcal{W} \wedge B \in \mathcal{W} \Rightarrow A = B \vee A \cap B = \emptyset)$

[5] The terms *abstract state* and *concrete state* should not be confused with the terms *abstract model* and *concrete model* which are used in context of refinements.

- Σ denotes set of event signatures. An event signature consists of an event name and event parameters.
- T is a 5-ary transition relation. Each element of t of T represents a transition in $\mathcal{H}$. We refer the elements of 5-tuple t as $t.Evt$, $t.Pre$, $t.K$, $t.Act$ and $t.Post$, where
  · $t.Evt \in \Sigma$ is an event signature, $t.Pre \in \mathcal{S}$ is the pre-state (originating abstract state) of the transition, $t.K(v,u)$ is a transition guard predicate (where u are the event parameters of $t.Evt$), $t.Post \in \mathcal{S}$ is the post-state (target abstract state) of the transition, and $t.Act$ is the transition action for the event $t.Evt$. Transition action is a simultaneous assignment to the variables of the system.
  · For any two transitions t_1 and t_2, if $t_1.Evt = t_2.Evt$ then

$$t_1.Pre(v) \wedge t_1.K(v,u) \Rightarrow \neg\,(t_2.Pre(v) \wedge t_2.K(v,u)) \tag{1}$$

  where u are the parameters of event $t_1.Evt$ (and $t_2.Evt$). Motivation for this condition is given at the end of this subsection.
- t_0 is a 5-tuple $\langle t_0.Evt, t_0.Pre, t_0.K, t_0.Act, t_0.Post\rangle$ representing the *init* transition, where $t_0.Evt$ is the *init* event, $t_0.Pre$ is a pseudo-state that represents the pre-state of t_0, the transition guard $t_0.K$ is $True$, Act_0 is the initialization action, and $S_0 \in \mathcal{S}$ is the initial abstract state.

Transition t is represented as $\{t.Pre\} \xrightarrow{t.Evt(u)[t.K(v,u)]/t.Act} \{t.Post\}$ where u are the parameters of Evt. If the system is in the abstract state $t.Pre$ and also satisfies the transition guard $t{\cdot}K(v,u)$, then the transition t is said to be *enabled*. A transition can take place only when it is enabled. If the transition t is enabled and it occurs then after the execution of $t.Act$, the system moves to the abstract state $t.Post$. This behavioral semantics of a transition in HASTM is captured in the following proof obligation.

$$t.Pre(v) \wedge t.K(v,u) \wedge BA(v,u,v') \vdash t.Post(v')$$

where $BA(v,u,v')$ is the *before-after* predicate of the action $t.Act$. The init transition initializes the system to the initial abstract state S_0 and is represented as $\xrightarrow{init[True]/Act_0} \{S_0\}$. Proof obligation for the init transition is $A(v') \vdash S_0(v')$ where A is the *after* predicate of the action Act_0.

Condition in Equation 1 in the HASTM definition ensures that two transitions corresponding to same event are not enabled for a given state and event parameters.

2.2 Representing Event-B Machine as a HASTM

Let M be an Event-B machine, v be its variables, and $I(v)$ be the invariants. Let E_m be the event denoted by "**any** u **where** $G_m(v,u)$ **then** $v \;:\mid BA_m(v,u,v')$ **end**" where $G_m(v,u)$ is the guard and $BA_m(v,u,v')$ is the before-after predicate corresponding to the action Act_m of event E_m. Let $A(v')$ be the after predicate for the init event.(We adapt the notation from [1])

HASTM $\mathcal{H} : \langle v, \mathcal{S}, \succ, \Sigma, T, t_0\rangle$ is a representation of the Event-B machine M if

(i) variables v and event signatures Σ in $\mathcal{H}$ are the same as the variables and event signatures in M.

(ii) root($\mathcal{H}$)$\Leftrightarrow I(v)$.

(iii) For any transition t in $\mathcal{H}$, the action $t.Act$ is the same as the action of event $t.Evt$ in M. The action of the *init* transition in $\mathcal{H}$ is the same as the action of the *init* event in M.

(iv) Let t_1, t_2, …, t_k be the transitions in $\mathcal{H}$ corresponding to an event E_m in M (i.e. $t_i.Evt = E_m$ for $i \in 1..k$) then

$$I(v) \wedge G_m(v,u) \Leftrightarrow \bigvee_{i \in 1..k} (t_i.Pre(v) \wedge t_i.K(v,u)) \tag{2}$$

Condition in Equation 2 along with the condition in Equation 1 imply that event G_m is enabled in M if and only if there is a single enabled transition t in $\mathcal{H}$ with $t.Evt = G_m$.

The HASTM representation is for a specific Event-B model in the Event-B refinement chain. Each Event-B model in the refinement chain will have a separate HASTM representation. In this work, we do not establish a link between HASTM representations corresponding to abstract and concrete Event-B models.

3 Generating HASTM from Event-B Machine

We first explain the process for generating a HASTM interactively and then present an algorithm for automatically generating a HASTM from a given Event-B machine.

Let t be a transition in $\mathcal{H}$ and X be a substate of $t.Pre$. Transition t is said to be *exclusively enabled* in X if $t.Pre(v) \wedge t.K(v,u) \Rightarrow X(v)$ where u are the parameters of $t.Evt$.

Consider a transition $t : \{t.Pre\} \xrightarrow{t.Evt(u)[t.K(v,u)]/t.Act} \{t.Post\}$ in $\mathcal{H}$. Partitioning $t.Pre$ with predicate $p(v)$ generates two immediate substates of $t.Pre$ viz. $X_1 : t.Pre(v) \wedge p(v)$ and $X_2 : t.Pre(v) \wedge \neg p(v)$. If t is exclusively enabled in either X_1 or X_2 then t is said to be *amenable to partitioning* of $t.Pre$ with respect to $p(v)$.

3.1 Interactive Generation of HASTM

We first define a Primitive HASTM representation of an Event-B machine which is a very simple HASTM with a single abstract state I. Consider the Event-B machine M described in Section 2.2. Let r be the number of events in M.
Primitive HASTM of M is a HASTM with the same variables and event signatures as those of M, a single abstract state I, and the following transitions.
$\xrightarrow{init[True]/Act_0} \{I\}$ and $\{I\} \xrightarrow{E_i[G_i]/Act_i} \{I\}$ for i from 1 to r.

The proof obligations for all the transitions in the primitive HASTM are the same as those of M. It is easy to verify that Primitive HASTM of M satisfies the conditions mentioned in section 2.2, and hence represents M.

To generate a HASTM of desired granularity, we start with a primitive HASTM of the given Event-B machine and then successively partition the basic abstract states and modify the transitions according to the following procedure.

Process for partitioning a basic abstract state X.

(i) **Partitioning**: Select a partitioning predicate $p(v)$ and partition the abstract state X into two substates $X_1 : X(v) \wedge p(v)$ and $X_2 : X(v) \wedge \neg p(v)$.

(ii) **Modify the pre-state**:
For any transition $t{:}\{X\} \xrightarrow{E(u)[K(v,u)]/Act} \{.\}$ originating from X, there are two possibilities.

(a) t is exclusively enabled in X_1: In this case, we strengthen the pre-state of t to X_1. If the transition guard $K(v,u)$ has $p(v)$ as a conjunct, remove it from $K(v,u)$ since it is redundant now as X_1 already has the conjunct. The transition now becomes $t{:}\{X_1\} \xrightarrow{E(u)[K'(v,u)]/Act} \{.\}$ where $K'(v,u)$ is the predicate obtained after removing the conjunct $p(v)$ from $K(v,u)$.

(b) t is exclusively enabled in X_2: This case is symmetric to case a.

(c) t is exclusively enabled in neither X_1 nor in X_2: We have two choices in this case.

- In place of the transition $\{X\} \xrightarrow{E(u)[K(v,u)]/Act} \{.\}$, create two new transitions : $\{X_1\} \xrightarrow{E(u)[K(v,u)]/Act} \{.\}$ and $\{X_2\} \xrightarrow{E(u)[K(v,u)]/Act} \{.\}$
- Keep the transition as it is $\{X\} \xrightarrow{E(u)[K(v,u)]/Act} \{.\}$. This is the default option in the automatic HASTM generation algorithm (to be discussed in Section 3.2). We choose this option in Algorithm 1 to prevent the number of transitions from increasing.

(iii) **Strengthen the post-state**: Strengthen the post-state of all the affected transitions (transitions whose pre-state has changed in step c and the transitions whose post-state has been partitioned in step i).
Consider transition $t : \{.\} \xrightarrow{E(u)[K(v,u)]/Act} \{Y\}$ whose post-state is Y. Let Y' be the immediate substate of Y that already exists in the state-space partition tree. If the proof obligation for $\{.\} \xrightarrow{E(u)[K(v,u)]/Act} \{Y'\}$ is discharged then strengthen the post-state of the transition to Y'. Now with Y' as the new post state, we repeat the above step till we fail to discharge the proof obligation or we reach a basic abstract state. The algorithm is given in function *strengthenPostState* in Algorithm 1.

Example:

Consider the Lift Event-B model given in Figure 2. We start with a primitive HASTM which has a single abstract state I , transitions: $\{I\} \xrightarrow{E_i[G_i]/Act_i} \{I\}$ for i from 1 to 7, and an init transition $\xrightarrow{init[True]/Act_0} \{I\}$. Invariants I and the events are as shown in Figure 2.

We now show how the transitions corresponding to events *MoveUp* and *CloseDoor* are modified after the partitioning of the root abstract state I.

- We use the predicate $p \widehat{=} (dirUp = T)$ for partitioning the abstract state I into two immediate substates $(I \wedge p)$ and $(I \wedge \neg p)$
- Consider the transition in primitive HASTM corresponding to the event *MoveUp*.

  $$\{I\} \xrightarrow{MoveUp \left[\begin{array}{c} (dirUp = T) \wedge (cf \notin callbtn) \wedge upRequested \wedge \\ (cf < topFloor) \wedge (doorOpen = F) \end{array} \right] /cf := cf+1} \{I\}$$

  Since the primitive HASTM has a single abstract state I, pre-state as well as post-state of the transition is I. Pre-state I and the transition guard imply $(dirUp = T)$ and hence $I \wedge (dirUp = T)$. This transition is exclusively enabled in $I \wedge (dirUp = T)$ and hence amenable to partitioning of the abstract state I with predicate $(dirUp = T)$. As per step a, we strengthen the pre-state of the transition to $I \wedge (dirUp = T)$ and weaken the transition guard by removing the conjunct $(dirUp = T)$. The modified transition is

  $$\{I \wedge (dirUp = T)\} \xrightarrow{MoveUp \left[\begin{array}{c} (cf \notin callbtn) \wedge upRequested \wedge \\ (cf < topFloor) \wedge (doorOpen = F) \end{array} \right] /cf := cf+1} \{I\}$$

- Now we try to strengthen the post-state of the transition to one of the sub-states of the current post-state I. The proposed transition is

  $$\{I \wedge (dirUp = T)\} \xrightarrow{MoveUp \left[\begin{array}{c} (cf \notin callbtn) \wedge upRequested \wedge \\ (cf < topFloor) \wedge (doorOpen = F) \end{array} \right] /cf := cf+1} \{I \wedge (dirUp = T)\}$$

  Proof obligation for this transition is discharged successfully. Hence, we strengthen the post-state of the transition to $I \wedge (dirUp = T)$.

- We now consider the transition in primitive HASTM corresponding to the *CloseDoor* event.

  $$\{I\} \xrightarrow{CloseDoor\ [doorOpen = T]\ /doorOpen := F; callbtns := callbtns \backslash \{cf\}} \{I\}$$

  For this transition, pre-state I along with the transition guard $(doorOpen = T)$ neither imply $(dirUp = T)$ nor imply $(dirUp = F)$. Hence this transition is not amenable to partitioning of I with predicate $(doorOpen = T)$. According to step c, we now have two choices: i) Do not modify the transition ii) Split the transition into two new transitions. Here we opt for the later choice and create two new transitions as follows.

  $$\{I \wedge dirUp = T\} \xrightarrow{CloseDoor\ \ [doorOpen = T]\ / \begin{array}{c} doorOpen := F; \\ callbtns := callbtns \backslash \{cf\} \end{array}} \{I\}$$

  $$\{I \wedge dirUp = F\} \xrightarrow{CloseDoor\ \ [doorOpen = T]\ / \begin{array}{c} doorOpen := F; \\ callbtns := callbtns \backslash \{cf\} \end{array}} \{I\}$$

- The post-states of both these transitions can be strengthened as follows after verifying the corresponding proof obligations.

  $$\{I \wedge dirUp = T\} \xrightarrow{CloseDoor\ \ [doorOpen = T]\ / \begin{array}{c} doorOpen := F; \\ callbtns := callbtns \backslash \{cf\} \end{array}} \{I \wedge dirUp = T\}$$

  $$\{I \wedge dirUp = F\} \xrightarrow{CloseDoor\ \ [doorOpen = T]\ / \begin{array}{c} doorOpen := F; \\ callbtns := callbtns \backslash \{cf\} \end{array}} \{I \wedge dirUp = F\}$$

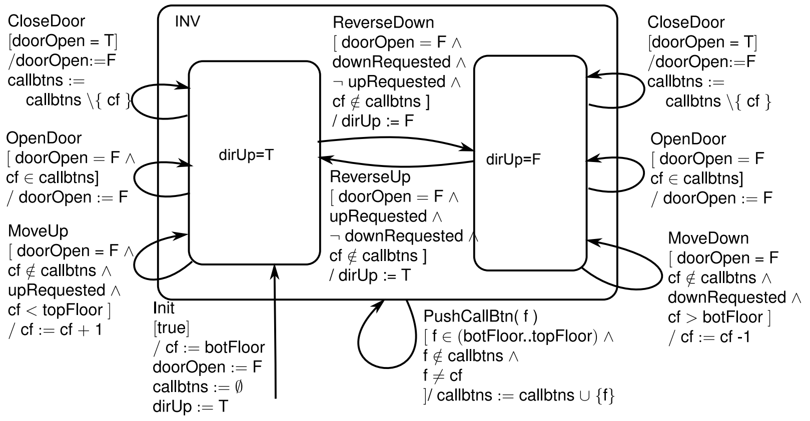

Fig. 3. HASTM representation of the Lift Event-B machine generated interactively by using partitioning predicate $(dirUp = T)$.

Following the above procedure for all the transitions, we get a HASTM as shown in Figure 3. We can further partition a basic abstract state with another predicate and continue the process.

3.2 Automatic Generation of HASTM

Algorithm 1 automatically generates a HASTM from a given Event-B machine and a set of partitioning predicates. Algorithm 1 is similar to the interactive generation algorithm, except that basic abstract states are recursively partitioned and the partitioning predicate is automatically selected from the given set of predicates.

We start with a primitive HASTM of M and then recursively partition the abstract state I. At each basic abstract state in the state-space partition tree, further partitioning predicate is selected to maximize the number of events amenable to partitioning with the selected predicate. This allows us the strengthen pre-state of these events without any increase in the number of transitions in the generated HASTM.

We define a *score* function that assigns each partitioning predicate p with the number of transitions that are amenable to partitioning X with p (Refer function *SelectPredicate* from Algorithm 1). Figure 4 shows a HASTM representation of the Lift Event-B machine generated by Algorithm 1 given the partitioning predicates $(cf = topFloor)$, $(cf = botFloor)$, $(doorOpen = T)$, and $(dirUp = T)$. The score function for the abstract state I is $score = \{(doorOpen = T) \mapsto 6, (dirUp = T) \mapsto 4, (cf = topFloor) \mapsto 2, (cf = botFloor) \mapsto 2\}$. We select the predicate $(doorOpen = T)$ that has maximum score for partitioning the abstract state I. We repeat this process for the sub-states till all the predicates are utilized while creating that abstract state or score of all the given predicates is zero.

For the selected partitioning predicate if there are events not amenable to partitioning then we decide not to strengthen the pre-state. This choice is not required as per the definition of HASTM. However, we choose this option in the automatic generation algorithm to prevent the number of transitions from increasing. The procedure for partitioning and pre-state strengthening is implemented in functions

```
Global Variables:
Input Variables:
 $M$: Event-B machine with $r$ events
 $P$: set of partitioning predicates
Output: HASTM $\mathcal{H} = \langle v, \mathcal{S}, \succ, \Sigma, T, t_0 \rangle$

function Main()
    BuildPrimitiveHASTM()
    PartitionAbstractState($I$)
    for transition $t$ in $T$
        StrengthenPostState($t$)
    StrengthenPostState($t_0$)
function BuildPrimitiveHASTM()
    $v :=$ variables of $M$
    $\Sigma :=$ Event signatures of $M$
    $I :=$ Conjunction of invariants of $M$
    $\mathcal{S} := \{I\}$
    $\succ: \; = \emptyset$
    $T := \{\langle Evt : E_m, Pre : I, K : G_m,$
            $Act : Act_m, Post : I\rangle | m \in 1..r\}$
    $t_0 := \langle Evt : init, Pre : Null,$
            $K : True, Act : Act_0, Post : I\rangle$
function PartitionAbstractState($X$: abstract state)
    $p$ : =SelectPredicate($X$)
    if $p = Null$
        return
    $X_1$:= AddSubState($X, p$)
    $X_2$:= AddSubState($X, \neg p$)
    PartitionAbstractState($X_1$)
    PartitionAbstractState($X_2$)
function AddSubState($X$: abstract state, $q$: predicate)
    $X'(v) := X(v) \wedge q(v)$
    $\mathcal{S} := \mathcal{S} \cup \{X'\}$
    $\succ := \succ \cup \{X \mapsto X'\}$
    for $t$ in $T$ such that $t.Pre = X$
        if $(X(v) \wedge t.K(v, u) \Rightarrow q(v))$
            $t.Pre := X'$
            $t.K = K'$ where $K'$ is the
                predicate obtained
                after removing
                conjuct $q(v)$ from $t.K$
                if it exists.
    return $X'$

function SelectPredicate($X$: abstract state)
    $score := \emptyset$ //$score \in P \to \mathbb{N}$
    for $p$ in $P$
        if $X(v)$ already has conjuct $p(v)$ or $\neg p(v)$
            continue
        $eT := \left\{ t \in T \;\middle|\; \begin{array}{l} t.Pre = X \text{ and } t \text{ is amenable} \\ \text{to partitioning of } X \text{ with } p \end{array} \right\}$
        $score(p) := |eT|$
    if $score = \emptyset$
        return $Null$
    $bestPred$ :$\in \arg\max_p score(p)$
    if $score(bestPred) = 0$
        $bestPred := Null$
    return $bestPred$
function StrengthenPostState(t: transition)
    $\mathcal{Y} = \{Y \in \mathcal{S} | t.Post \succ Y\}$
    if $\mathcal{Y} = \emptyset$  //$t.Post$ is a basic abstract state
        return
    for $Y$ in $\mathcal{Y}$
        if $\left( \begin{array}{l} \text{proof obligation for} \\ \{t.Pre\} \xrightarrow{t.Evt[t.K]/t.Act} \{Y\} \\ \text{is discharged} \end{array} \right)$
            $t.Post := Y$
            strengthenPostState(t)
            break
    return
```

Algorithm 1. Algorithm for generating HASTM representation from an Event-B machine given a set of partitioning predicates.

PartitionAbstractState and *AddSubState* in Algorithm 1. The number of transitions in the HASTM generated by Algorithm 1 always equals the number of events in the Event-B model.

After the complete state-partition tree is ready and pre-state of all the transitions is strengthened to appropriate abstract states, we strengthen the post-states of all the transitions. Refer function *StrengthenPostState* in Algorithm 1 for details.

After the automatic generation of a HASTM, user can further modify the representation by employing the interactive algorithm and partition the basic states with different predicates.

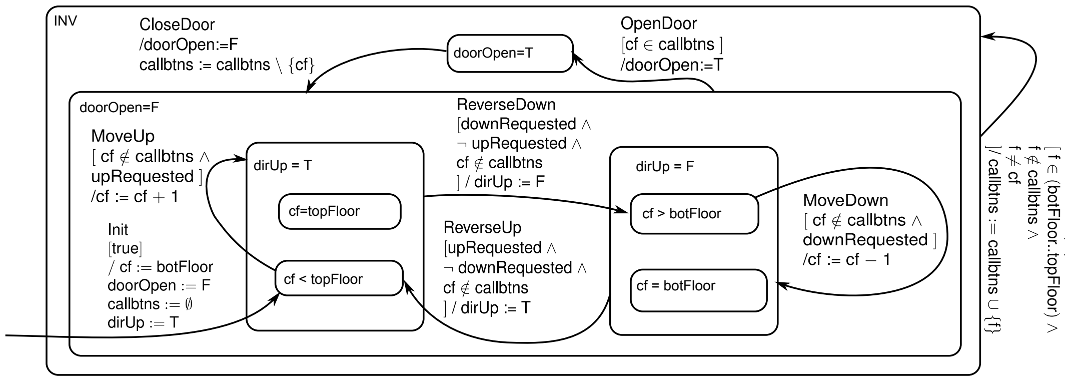

Fig. 4. HASTM representation of the Lift Event-B machine generated by Algorithm 1 given the partitioning predicates $(cf = topFloor)$, $(cf = botFloor)$, $(doorOpen = T)$, and $(dirUp = T)$.

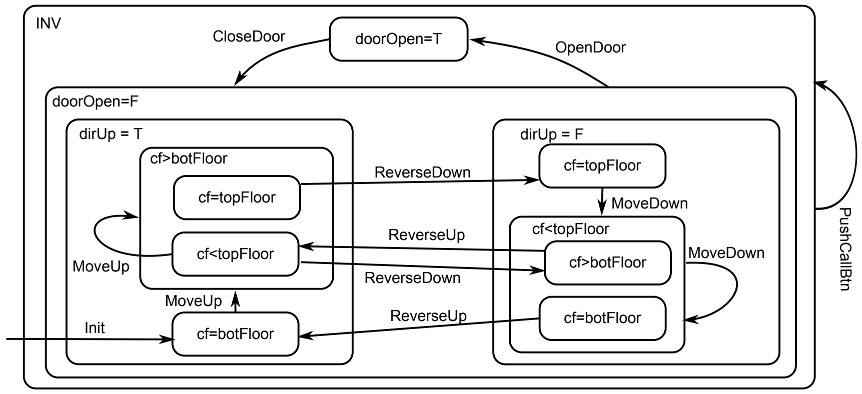

Fig. 5. An alternate HASTM representation of the Lift Event-B machine. At node $(dirUp = T)$, we interactively choose the partitioning predicate $(cf = botFloor)$. Transition guards and actions are not shown in the figure and are the same as in Figure 4 for all the events.

3.3 Multiple Views

In the HASTM shown in Figure 4, the automatic HASTM generation algorithm has not partitioned the node $(doorOpen = T)$ since score of all the predicates is zero. Partitioning with any of the given predicates would not have allowed us to strengthen the pre-state of the *CloseDoor* transition without splitting it.

Although the automatic generation algorithm avoids generating multiple transitions for a single event, user can force selection of different predicate resulting in a different state-space partition tree. For example, at node $(dirUp = T)$ in Figure 4, if we choose the predicate $(cf = botFloor)$ instead of $(cf = topFloor)$, we get a different representation as shown in Figure 5. This partitioning has increased the complexity of the HASTM representation since two transitions are generated for the *MoveUp* event. However, this is sometimes desirable since multiple HASTM representations highlight different aspects of the system.

Note from Figure 4 that after execution of *OpenDoor* event, all other events except *CloseDoor* and *PushCallBtn* are disabled. This property might not be clear from the Event-B machine. Another property that *ReverseUp* event is disabled when $(cf = topFloor)$ is not clear from the Event-B model. Guard of the *ReverseUp* event together with the invariants imply $(cf < topFloor)$. This fact is not clear even in the

HASTM in Figure 4 since the abstract state ($dirUp = F$) is not partitioned with the predicate ($cf = topFloor$). However, if we partition the state-space differently as in the HASTM shown in Figure 5, it becomes clear that *ReverseUp* is only enabled in abstract states with ($cf < topFloor$).

4 Related and Future Work

A lot of work has been done [8,11,12,13] on deriving formal B models from the specifications in visual representations(mostly UML). Our approach is the reverse of this. We start from an existing Event-B model and build multiple visual representations that focus on different behavioral aspects of the system.

In [5], a method for specifying structured models is presented and its use for sequential program development is demonstrated. This approach is especially useful for modeling of problems that require sequential ordering of events. The algorithm presented in this work can be used to extract structure out of the Event-B models in which abstract program counters are used to achieve ordering of events. For such models, predicates involving the program counters are the natural choices for the partitioning predicates. Although, in this paper, we only consider binary partitioning of the state-space, the approach can easily be extended to multi-ary partitioning.

In [7], three techniques based on animation and proof are presented for constructing state transition diagrams. The work proposes to associate the states of the diagram with abstract invariants in order to reduce the number of states to a finite one. The ProB tool [9] can generate state-space graph of a B machine by traversing the state-space of the machine. However, for most models, complete state-space is not explored. Also, for larger state-spaces these graphs become very complex. In [10], two algorithms for reducing the complexity of the state-space graphs are presented making it possible to visualise larger state spaces. However, the transitions in the state-space graphs are not labeled with predicates. In [2], flow graph is derived from an Event-B model which is very useful for uncovering implicit algorithmic structures. Flow graph does not employ hierarchical states and can get very complex as the number of events in the model increase. In HASTM, complexity of the representation can be maintained to a comprehensible level by selective partitioning of hierarchical abstract states. However, sometimes getting the right perspective might need human intervention in selecting the right partitioning predicates.

The work in [3,4] presents a method and a tool (GeneSyst) to build symbolic labeled transition systems from Event-B specifications. GeneSyst system requires the invariants associated with the states in the transition system to be specified by the user. GeneSyst system supports refinements of the models. HASTM has hierarchical states but more work is needed to establish a link between HASTM corresponding to the abstract and concrete Event-B models in the refinement chain.

In this work, we partition the global state-space of the Event-B machine. We would like to explore the partitioning of the local state-space defined by the event parameters. Having developed the basic concept of a HASTM and an algorithm for automatic generation of a HASTM from an Event-B model, we now plan to implement this algorithm and try out this visualisation technique on various Event-B models.

5 Conclusions

In this work, we present a methodology for visualising Event-B models by using Hierarchical State Transition Machines (HASTM). We specify the conditions that a HASTM should satify in order to represent an Event-B machine. We then present an algorithm for automatic generation of a HASTM representation from a given Event-B model and a set of partitioning predicates. With the help of examples, we demonstrate that multiple HASTM representations aid in grasping certain behavioral aspects of the systems.

References

[1] Jean-Raymond Abrial. *Modeling in Event-B : system and software engineering.* Cambridge University Press, Cambridge, New York, 2010.

[2] Jens Bendisposto and Michael Leuschel. Automatic flow analysis for Event-B. In Dimitra Giannakopoulou and Fernando Orejas, editors, *Fundamental Approaches to Software Engineering*, volume 6603 of *Lecture Notes in Computer Science*, pages 50–64. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-19811-3_5.

[3] Didier Bert and Francis Cave. Construction of finite labelled transistion systems from B abstract systems. In *Proceedings of the Second International Conference on Integrated Formal Methods*, IFM '00, pages 235–254, London, UK, 2000. Springer-Verlag.

[4] Didier Bert, Marie-Laure Potet, and Nicolas Stouls. GeneSyst: a tool to reason about behavioral aspects of B event specifications. application to security properties. *CoRR*, abs/1004.1472, 2010.

[5] Stefan Hallerstede. Structured Event-B models and proofs. In Marc Frappier, Uwe Glässer, Sarfraz Khurshid, Régine Laleau, and Steve Reeves, editors, *Abstract State Machines, Alloy, B and Z*, volume 5977 of *Lecture Notes in Computer Science*, pages 273–286. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-11811-1_21.

[6] David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[7] A. Idani and Y. Ledru. Dynamic graphical UML views from formal B specifications. *Information and Software Technology*, 48(3):154–169, 2006.

[8] Hung Ledang and Jeanine Souquières. Contributions for modelling UML State-Charts in B. In *Proceedings of the Third International Conference on Integrated Formal Methods*, IFM '02, pages 109–127, London, UK, UK, 2002. Springer-Verlag.

[9] Michael Leuschel and Michael Butler. ProB: a model checker for B. In *FME 2003: FORMAL METHODS, LNCS 2805*, pages 855–874. Springer-Verlag, 2003.

[10] Michael Leuschel and Edd Turner. Visualising larger state spaces in pro B. In *ZB 2005: Formal Specification and Development in Z and B*, volume 3455 of *Lecture Notes in Computer Science*, pages 6–23. Springer Berlin / Heidelberg, 2005.

[11] Emil Sekerinski and Rafik Zurob. Translating statecharts to B. In *Proc. of the 3rd International Conference on Integrated Formal Methods (IFM'02), volume 2335 of LNCS*, pages 128–144. Springer-Verlag, 2002.

[12] Colin Snook and Michael Butler. UML-B: formal modeling and design aided by UML. *ACM Transactions on Software Engineering and Methodology*, 15(1):92–122, 2006.

[13] Colin Snook and Michael Butler. UML-B and Event-B: an integration of languages and tools. In *Proceedings of the IASTED International Conference on Software Engineering*, SE '08, pages 336–341, Anaheim, CA, USA, 2008. ACTA Press.