Generating Hierarchical State Based Representation From Event-B Models

Dipak L. Chaudhari, Om P. Damani

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

dipakc@cse.iitb.ac.in, damani@cse.iitb.ac.in

June 21, 2011

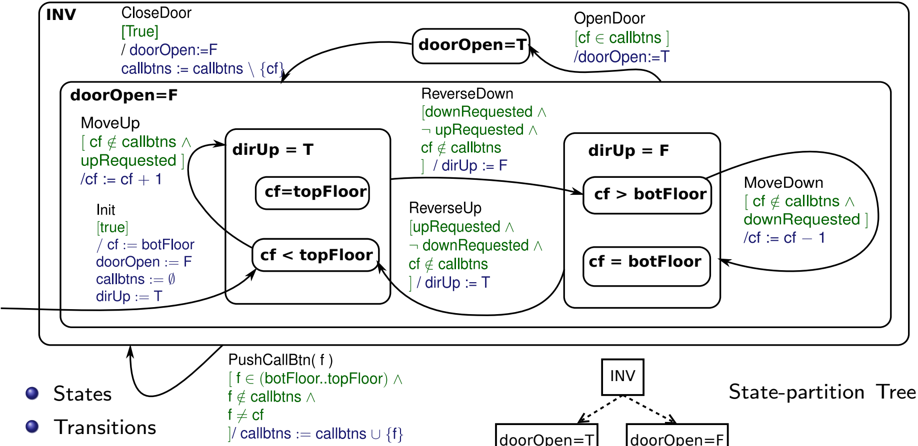
Outline

- Motivation
- Introduction to HASTM
- Algorithm for Generation of HASTM from Event-B model
    - Interactive
    - Automatic
- Multiple Views of the System
- Related and Future Work
- Conclusion

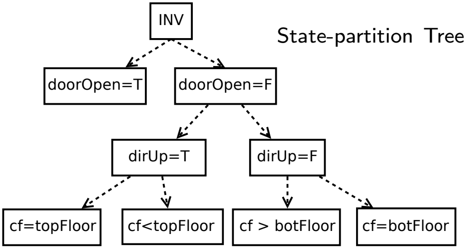
Motivation

- Discovering/Ascertaining properties of the system
  - Example Property: Lift Controller System
    After *OpenDoor*:
    - Allowed events: *CloseDoor* and *PushCallBtn*
    - Disabled events: *MoveUp*, *MoveDown*, etc.
- Animation
- Scenario analysis
- Hierarchical Representation:
  Easier to ascertain properties just by quick visual inspection.

Hierarchical Visual Representation



- States
- Transitions
- Pre-state and Post-state
- Transition Guards
- Transitions originating from a superstate
- Transitions terminating in a superstate
- Our Goal: Generate Hierarchical Representation from Event-B model



State-partition Tree

Lift Controller: Event-B Model

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

Initialisation $\widehat{=}$
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

Lift Controller: Event-B Model - II

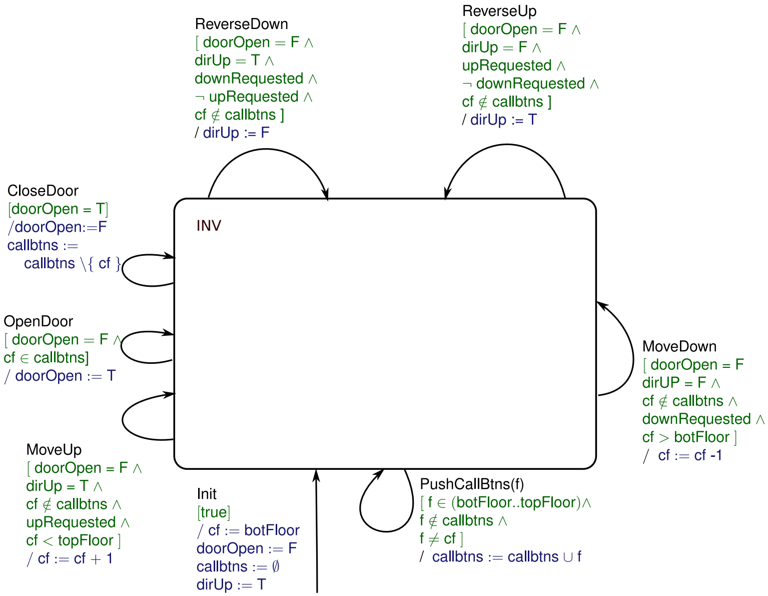
MoveUp $\widehat{=}$
when
$dirUp = T$
$doorOpen = F$
$upRequested$
$cf < topFloor$
$cf \notin callbtns$
then
$cf := cf + 1$
end

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

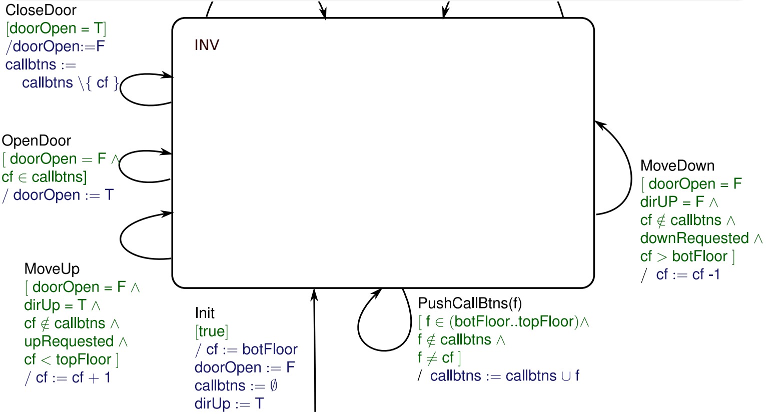
Interactive Generation of HASTM



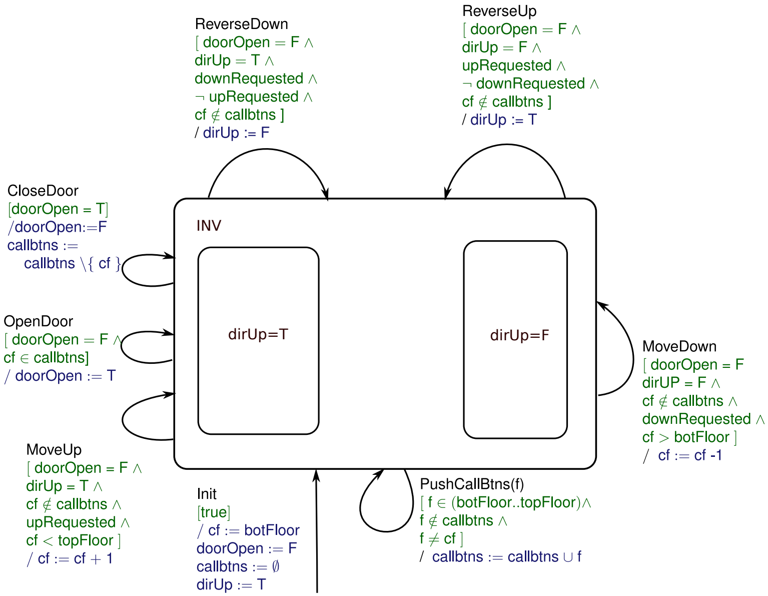
Interactive Generation of HASTM

Partitioning

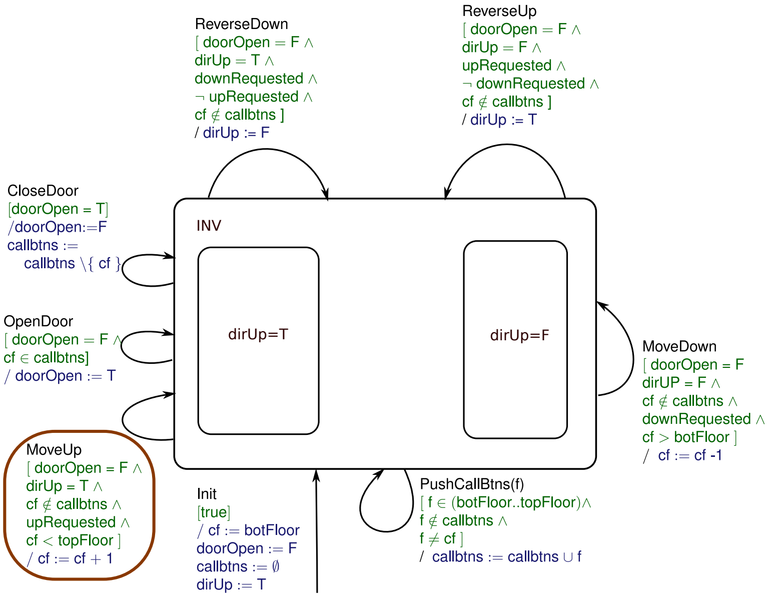
- Partition the state *INV* with predicate $(dirUp = T)$
- Results in two substates
  $INV \wedge (dirUp = T)$ and $INV \wedge (dirUp = F)$



Interactive Generation of HASTM

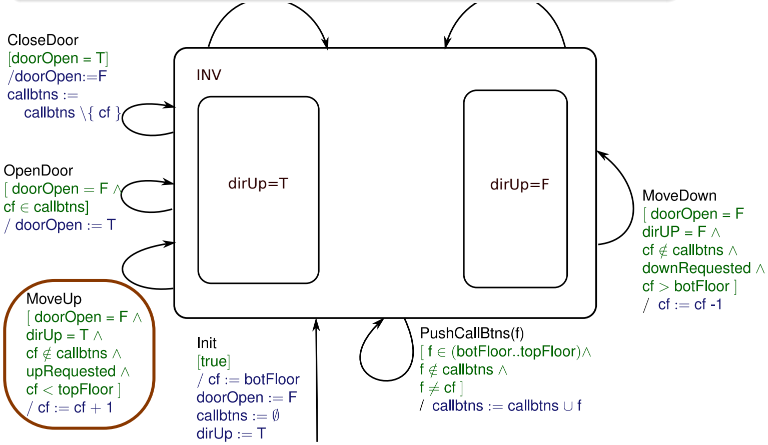


Interactive Generation of HASTM

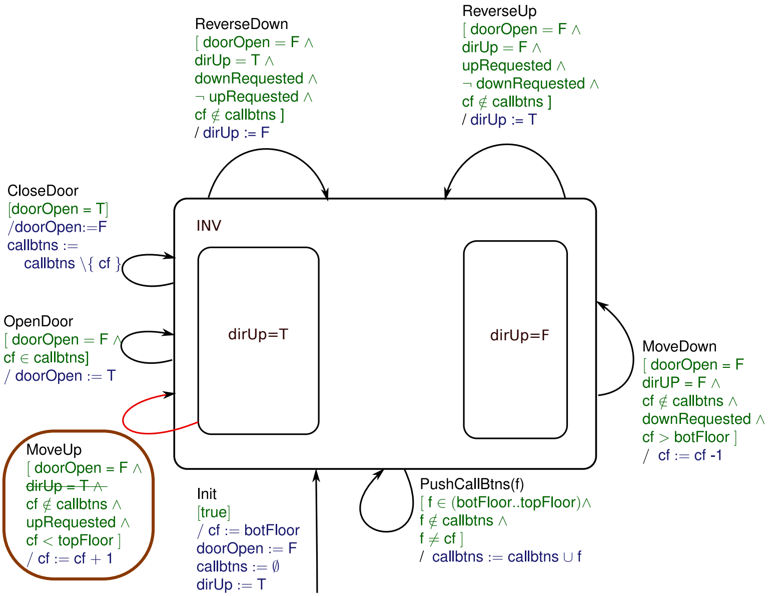


Strengthen the Pre-State of *MoveUp* transition(t)

- $INV \wedge t.K \Rightarrow (dirUp = T)$
- MoveUp transition is disabled for $(dirUp = F)$



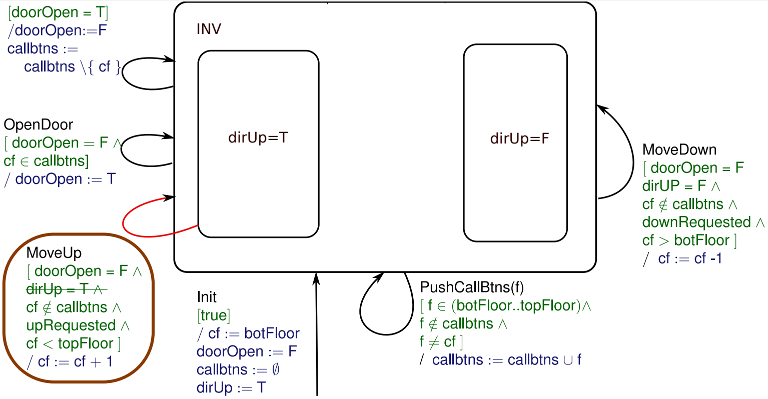
Interactive Generation of HASTM



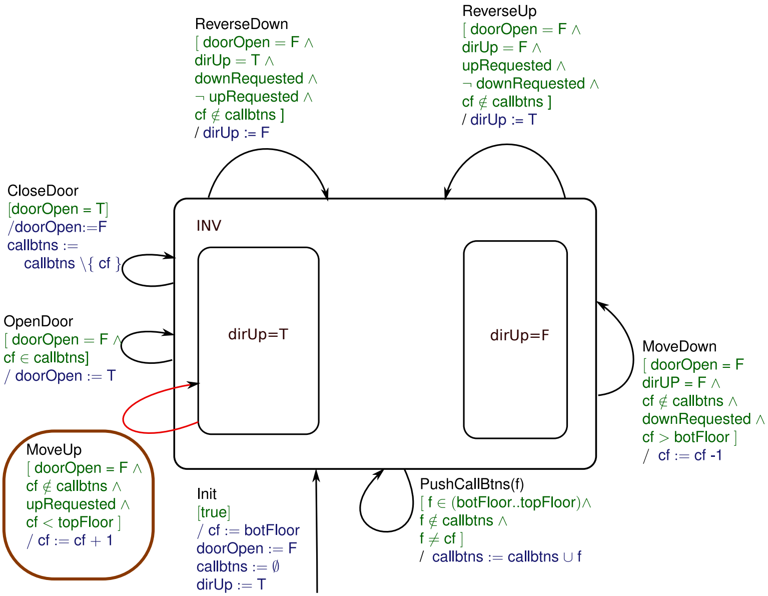
Interactive Generation of HASTM

Strengthen the Post-State of *MoveUp* transition

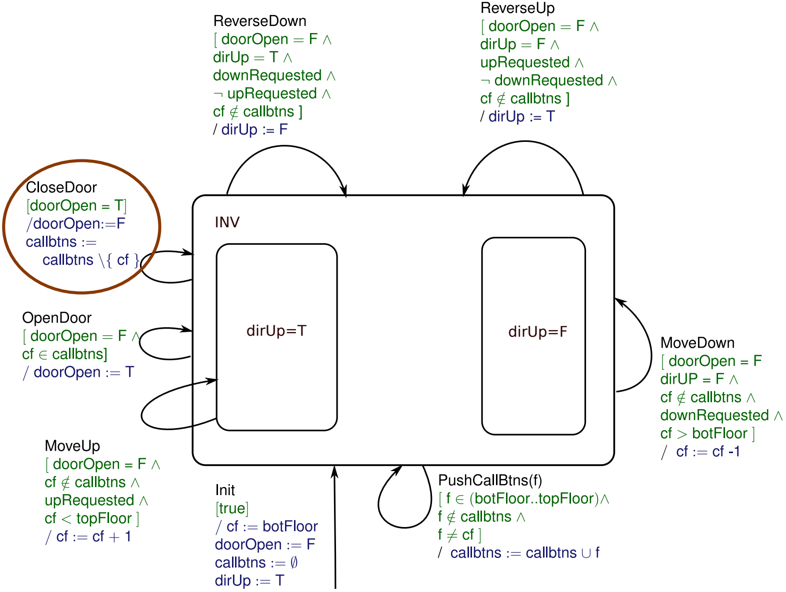
- $t.Pre(v) \wedge t.K(v,u) \wedge BA(v,u,v') \vdash t.Post(v')$
- Post-state $= (dirUp = T)$: Proof Obligation discharged.
- Post-state $= (dirUp = F)$: Proof Obligation NOT discharged.



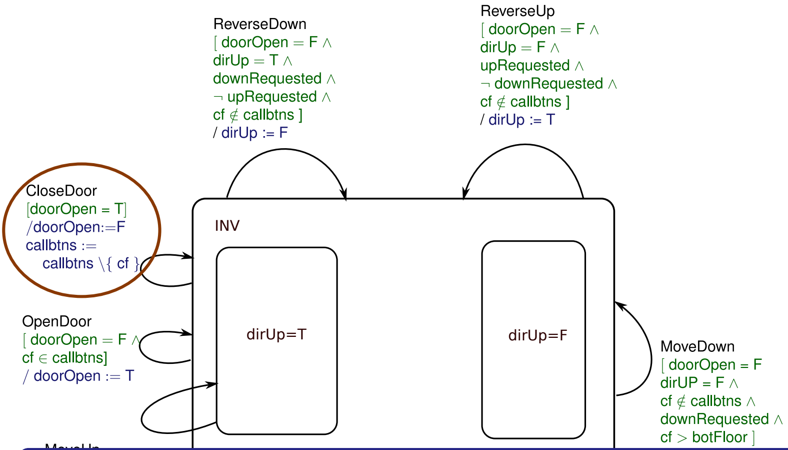
Interactive Generation of HASTM



Interactive Generation of HASTM



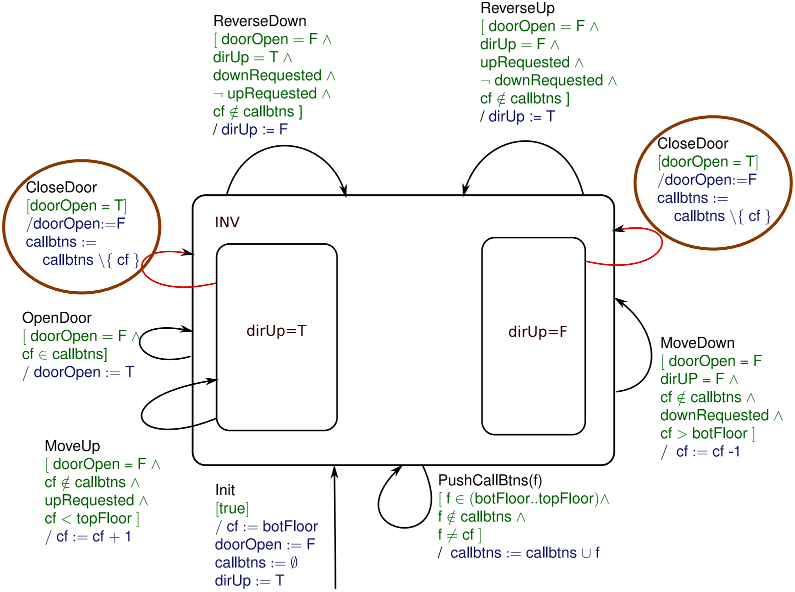
Interactive Generation of HASTM



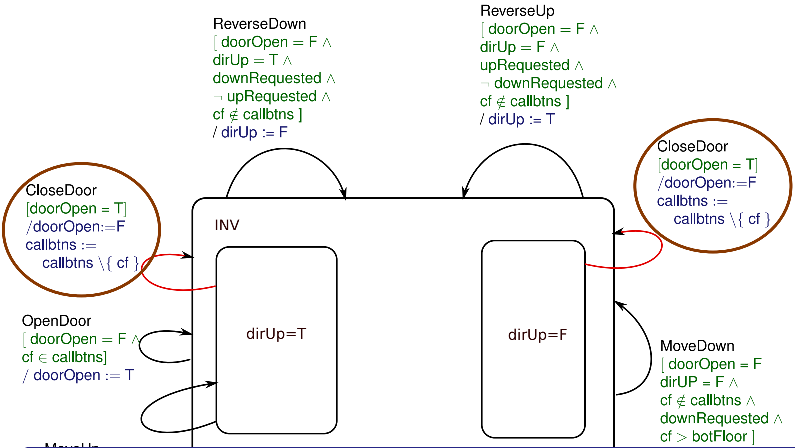
Strengthen the Pre-State of *CloseDoor* transition

- $INV \wedge (OpenDoor = T) \Rightarrow (dirUp = T)$ is not valid
- $INV \wedge (OpenDoor = T) \Rightarrow (dirUp = F)$ is not valid

Interactive Generation of HASTM



Interactive Generation of HASTM

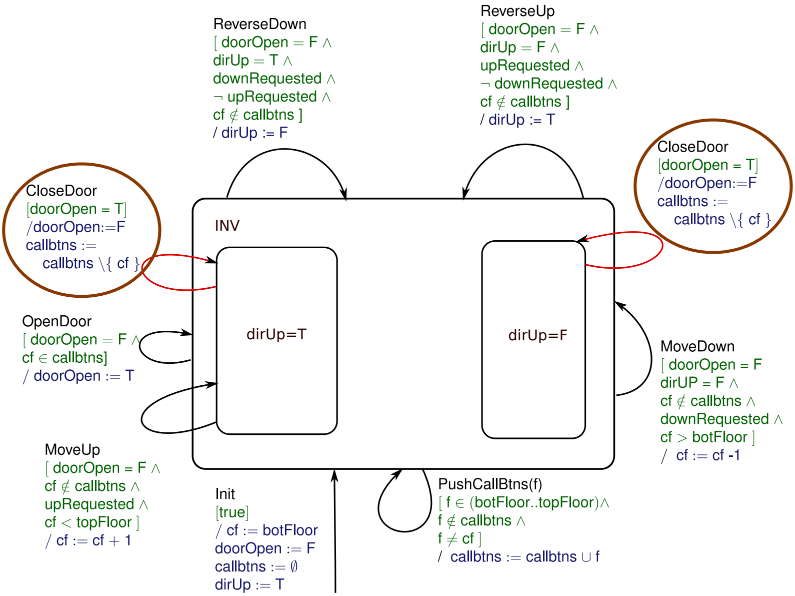


Strengthen the Post-State of *CloseDoor* transition

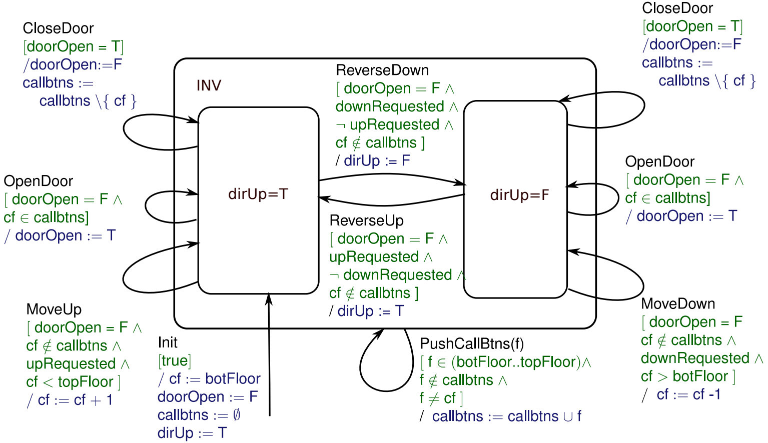
- $t.Pre(v) \wedge t.K(v,u) \wedge BA(v,u,v') \vdash t.Post(v')$

callbtns := ∅
dirUp := T

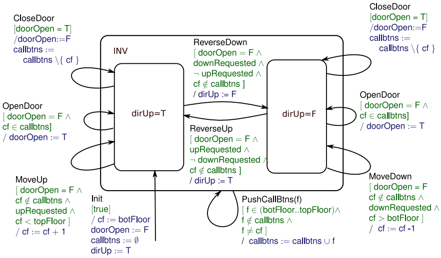
Interactive Generation of HASTM

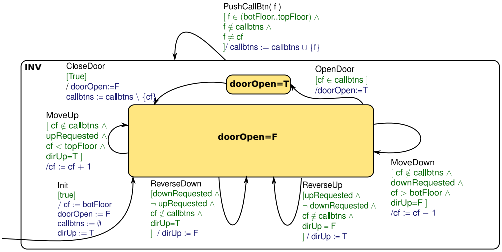


Interactive Generation of HASTM

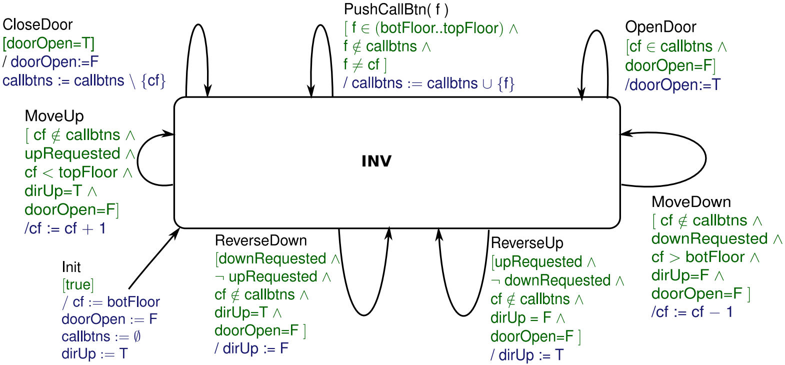


- Selection of right partitioning predicate.
- Different predicate choices result in different diagrams.





Automatic Generation of HASTM



Score: Number of transitions whose pre-state can be strengthened without splitting the transition.

dirUp = T
- ReverseUp
- ReverseDown
- MoveUp
- MoveDown

score = 4

cf = topFloor
-ReverseUp
-MoveUp

score = 2

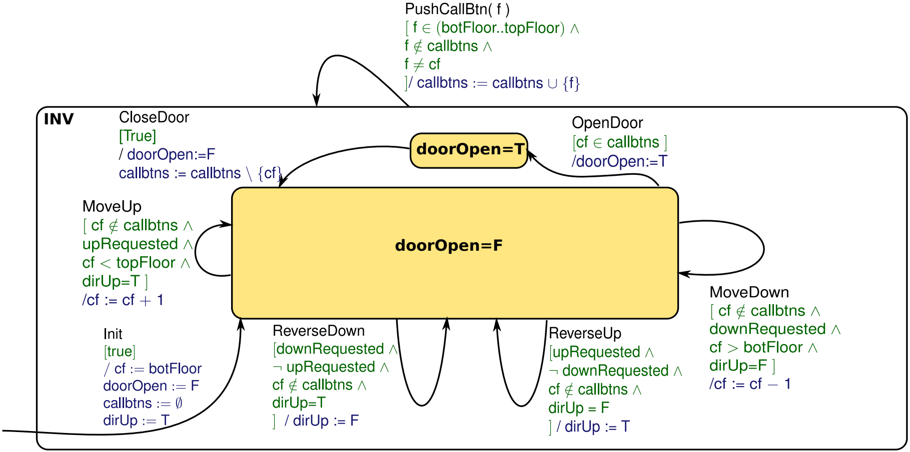
cf = botFloor
-ReverseDown
-MoveDown

score = 2

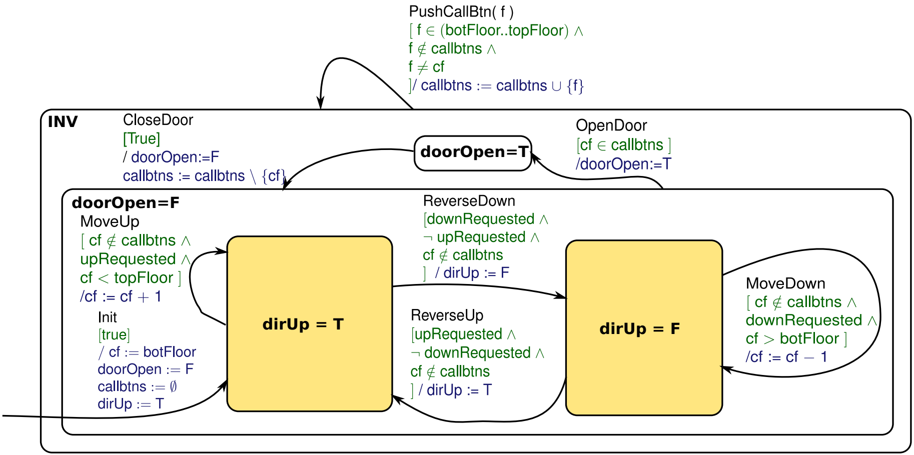
doorOpen = T
- ReverseUp
- ReverseDown
- MoveUp
- MoveDown
- CloseDoor
- OpenDoor

score = 6

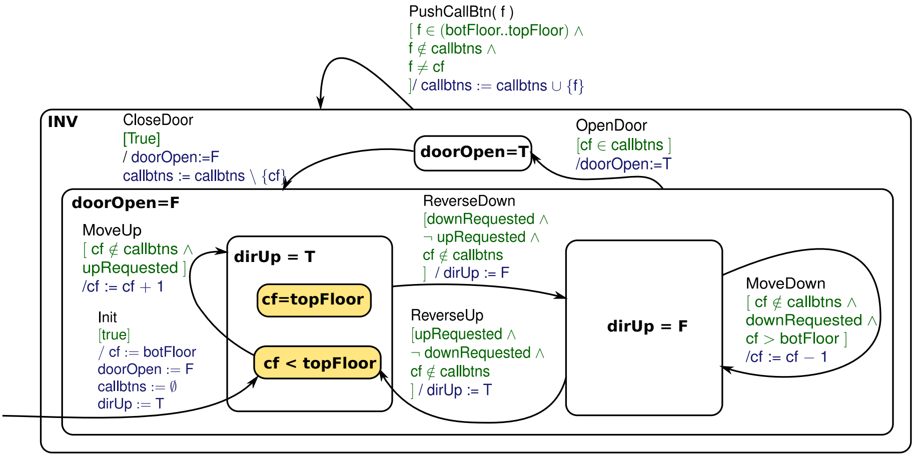
Automatic Generation of HASTM

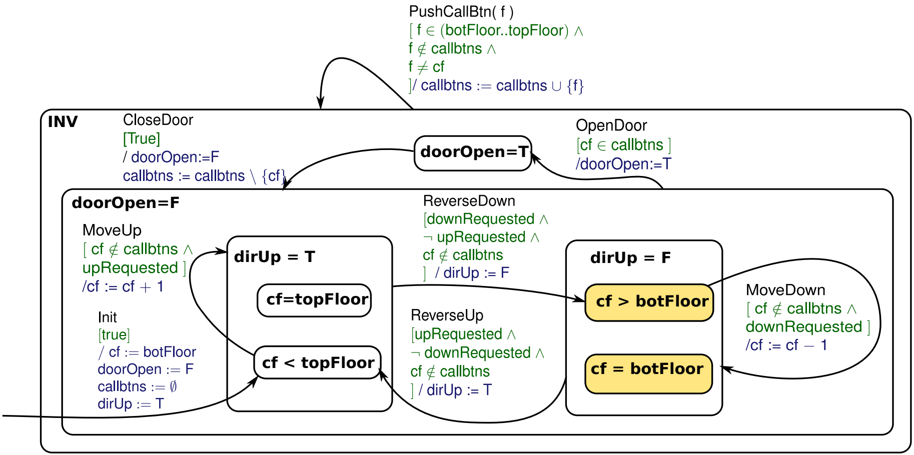


Automatic Generation of HASTM



Automatic Generation of HASTM




PushCallBtn(f)
[f ∈ (botFloor..topFloor) ∧
f ∉ callbtns ∧
f ≠ cf
]/ callbtns := callbtns ∪ {f}
INV
CloseDoor
[True]
/ doorOpen:=F
callbtns := callbtns \ {cf}
doorOpen=T
OpenDoor
[cf ∈ callbtns]
/doorOpen:=T
doorOpen=F
ReverseDown
[downRequested ∧
¬ upRequested ∧
cf ∉ callbtns
] / dirUp := F
MoveUp
[cf ∉ callbtns ∧
upRequested]
/cf := cf + 1
dirUp = T
cf=topFloor
cf < topFloor
dirUp = F
cf > botFloor
cf = botFloor
MoveDown
[cf ∉ callbtns ∧
downRequested]
/cf := cf − 1
Init
[true]
/ cf := botFloor
doorOpen := F
callbtns := ∅
dirUp := T
ReverseUp
[upRequested ∧
¬ downRequested ∧
cf ∉ callbtns
] / dirUp := T

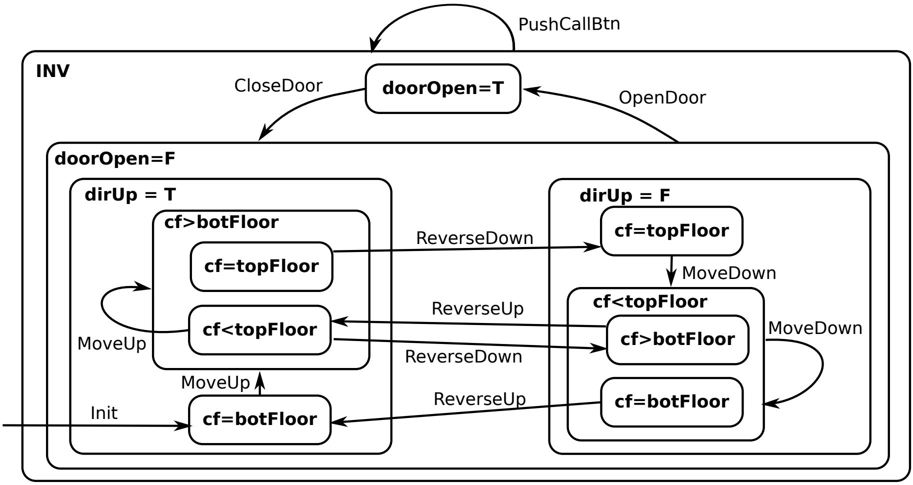
Automatic Generation Algorithm

function **Main()**
BuildPrimitiveHASTM()
PartitionAbstractState(I)
for transition t in T
StrengthenPostState(t)
StrengthenPostState(t_0)
function **BuildPrimitiveHASTM**()
v := variables of M
Σ := 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 $(X(v) \wedge t.K(v, u) \Rightarrow q(v))$
$t.Pre := X'$
$t.K = K'$
return X'

function **SelectPredicate**(X: abstract state)
$score := \emptyset$ //$score \in P \rightarrow \mathbb{N}$
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

Alternate View



Related Work

- From visual specifications to B models.
  - [Ledang and Souquières, 2002, Sekerinski and Zurob, 2002, Snook and Butler, 2006, Snook and Butler, 2008]
- Structured Event-B. [Hallerstede, 2010]
- ProB: [Leuschel and Butler, 2003] Can generate state-space graph of a B machine by traversing the state-space of the machine.
- GeneSyst: [Bert et al., 2010] Builds symbolic labeled transition systems from Event-B specifications
  - Supports refinement
  - Requires the invariants associated with the states in the transition systems to be specified by the user.
- FlowGraph: [Bendisposto and Leuschel, 2011]
  - Useful for uncovering implicit algorithmic structures.

Future Work

- Implementation
- Support for refinement
- Partitioning the local state-space defined by event parameters.

Thank You !!

References I

Bendisposto, J. and Leuschel, M. (2011).
Automatic flow analysis for Event-B.
In Giannakopoulou, D. and Orejas, F., editors, *Fundamental Approaches to Software Engineering*, volume 6603 of *Lecture Notes in Computer Science*, pages 50–64. Springer Berlin / Heidelberg. 10.1007/978-3-642-19811-3_5.

Bert, D., Potet, M., and Stouls, N. (2010).
GeneSyst: a tool to reason about behavioral aspects of B event specifications. application to security properties.
CoRR, abs/1004.1472.

References II

Hallerstede, S. (2010).
Structured Event-B models and proofs.
In Frappier, M., Glässer, U., Khurshid, S., Laleau, R., and Reeves, S., editors, *Abstract State Machines, Alloy, B and Z*, volume 5977 of *Lecture Notes in Computer Science*, pages 273–286. Springer Berlin / Heidelberg.
10.1007/978-3-642-11811-1_21.

Ledang, H. and Souquières, J. (2002).
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. Springer-Verlag.

References III

Leuschel, M. and Butler, M. (2003).
ProB: a model checker for B.
In *FME 2003: FORMAL METHODS, LNCS 2805*, pages 855–874. Springer-Verlag.

Sekerinski, E. and Zurob, R. (2002).
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.

Snook, C. and Butler, M. (2006).
UML-B: formal modeling and design aided by UML.
ACM Transactions on Software Engineering and Methodology, 15(1):92–122.

References IV

Snook, C. and Butler, M. (2008).
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. ACTA Press.