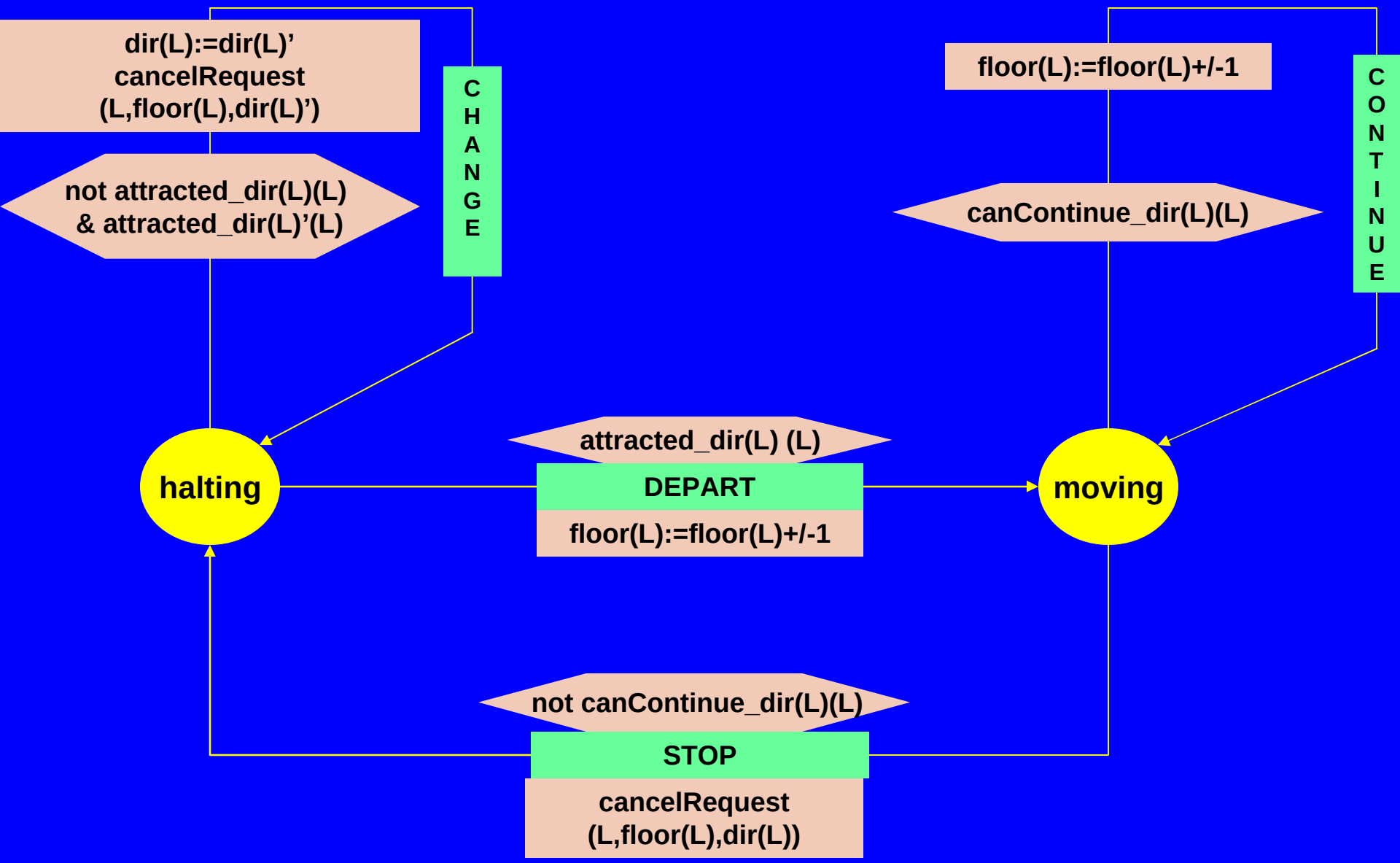
Lift Control (N.Davis, 1984) The Problem

- Design the logic to move n lifts bw m floors, and prove it to be well functioning, where
  - Each lift has for each floor one button which, if pressed, causes the lift to visit (i.e. move to and stop at) that floor.
  - Each floor (except ground and top) has two buttons to request an up-lift and a down-lift. They are cancelled when a lift visits the floor and is either travelling in the desired direction, or visits the floor with no requests outstanding. In the latter case,if both floor request buttons are illuminated, only one should be cancelled.
  - A lift without requests should remain in its final destination and await further requests.
  - Each lift has an emergency button which, if pressed, causes a warning to be sent to the site manager. The lift is then deemed ‘out of service’. Each lift has a mechanism to cancel its ‘out of service’ status.

Lift Control : control state ASM



Parametrized Macros for Lift ASM

- attracted_up (L) iff for some Floor > floor(L)
  - hasToDeliverAt (L,Floor) or
  - existsCallFromTo (Floor,Dir) for some Dir
- attracted_down (L) same with <
    - NB. attracted_dir(L) (L) stands for attracted_Dir(L) & Dir=dir(L)
- canContinue_Dir (L) iff
  - attracted_Dir (L) &
  - neither hasToDeliverAt (L,floor(L))
  - nor existsCallFromTo (floor(L),Dir)
    - NB. Priority to keeping direction of travel
- cancelRequest (L,Floor,Dir) =
  - hasToDeliverAt (L,Floor) := false (cancel request in L for Floor)
  - existsCallFromTo (Floor,Dir) := false

(cancel request from Floor for Dir)

Signature/Constraints for Lift Control State ASM

- shared functions bw users (for input) and lift
  - existsCallFromTo (Floor,Dir) (initially everywhere false)

    formalizes that each floor has 2 buttons to call a lift for going up/down

    constraints: always false for (ground,down), (top,up),
    and for (floor(L),dir(L)) when ctl_state(L)=halting

    formalizes that where a lift is halting no further call can be made for going into its direction of travel
  - hasToDeliverAt (L,Floor) (initially everywhere false)

    formalizes that in every lift there is for every floor a button to request delivery there

    constraint: always false for (L,floor(L)) when ctl_state(L)=halting

    formalizes that no further delivery can be requested for a floor where the lift is halting
- controlled functions
  - ctl_state(L) = halting,moving (initially halting)
  - direction(L) = up, down (initially up)
  - floor(L) current lift position, bw ground and top (initially ground)

Analyzing Possible Runs of Lift Control State ASM

L1. Non-empty runs of any L (from the initial state) have form

(DEPART CONTINUE* STOP) $^{+}$ (CHANGE (DEPART CONTINUE* STOP) *)*

Proof by induction on walks thru the diagram

L2. Moving from any reachable state, every L

moves floor by floor to the farest point of attraction in its direction of travel where, after at most |FLOOR| steps, it STOPs & then either terminates - namely iff L is not attracted in any direction - or it CHANGEs direction & moves into the new direction.

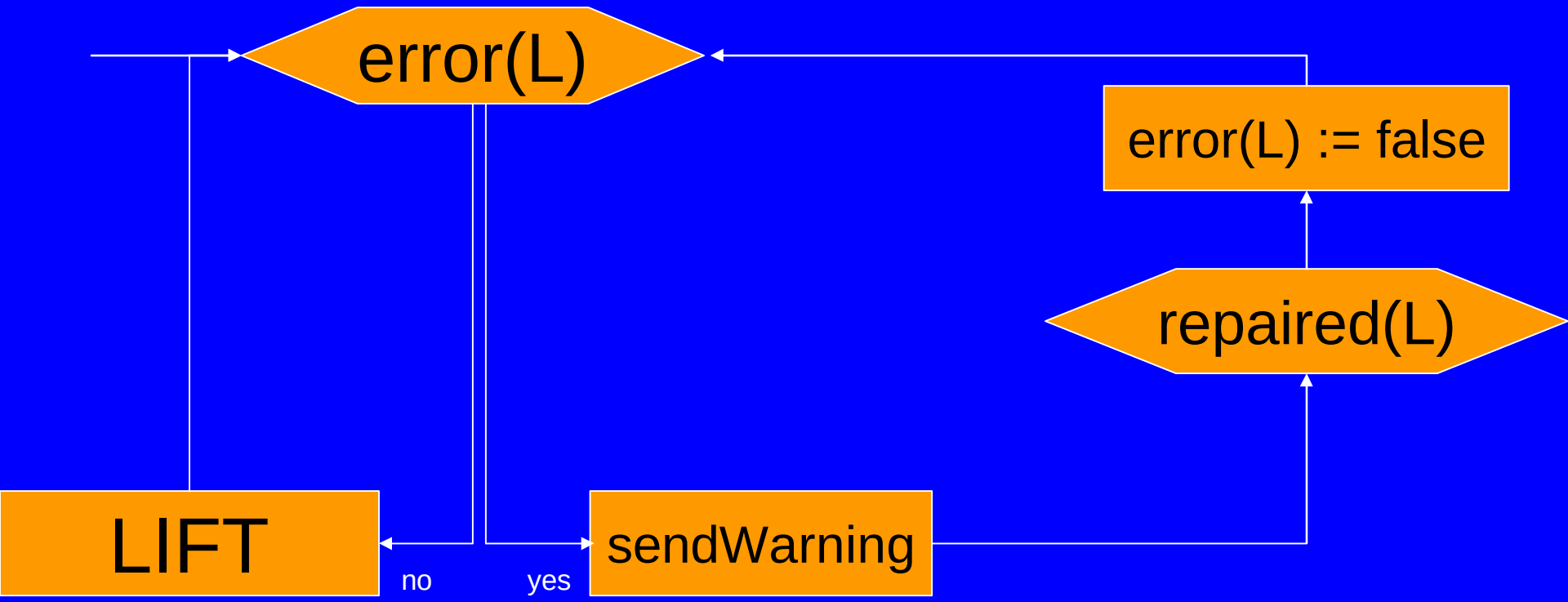
L3. When moving to the farest point of attraction in its direction of travel, every L

STOPs at each floor where it is attracted, wrt its direction of travel, and turns off the (internal) delivery request and the (external) call from that floor to go into the current direction of travel. When it CHANGEs, it turns off the (external) call from its current floor to go into the new direction of travel .

Justifying the Well Functioning of the LIFT ASM

- To show (Lift Correctness):
  - All requests for floors within lifts must be serviced eventually, with floors being serviced sequentially in the direction of travel
  - All requests for lifts from floors must be serviced eventually, with all floors given equal priority
- Proof follows from the three lemmas above, since every internal request from within L, and every external request, make L being eventually attracted in the requested direction
- NB. The proof does not exclude real-life situations with crowded lifts, where requests may be satisfied logically, but the lack of capacity prevents people from entering the lift. This problem has no logical solution, and should be solved providing more capacity ("larger bandwith")

Adding Error Handling to LIFT Machine



Add new error guard

Add 'out-of-service'entry/exit rules

This stepwise development method is characteristic for ASMs. See for ex. the extension of ASM models for Java and the JVM by exception handling, defined and analysed in

R. Stärk, J. Schmid, E. Börger

Java and the Java Virtual Machine: Definition, Verification, Validation

Introducing Scheduling for LIFT ASM

Problem: all lifts attracted by all external calls (from trav dir) i.e. where existsCallFromTo (Floor,_) is true

Solution: a scheduler hasBeenCalledTo assigns ONE Lift to each external call, from any Floor (for any Dir)

Modular implementation : refine the interface predicates

attracted_up (L) iff for some Floor > floor(L) (< for down)

- hasToDeliverAt (L,Floor) or
- L=hasBeenCalledTo (Floor,Dir) for some Dir

canContinue_Dir (L) iff attracted_Dir (L) & neither hasToDeliverAt (L,floor(L)) nor L=hasBeenCalledTo (floor(L),Dir)

NB. cancelRequest (L,Floor,Dir) refined to

hasToDeliverAt (L,Floor) := false,

If L=hasBeenCalledTo(Floor,_) then existsCallFromTo (Floor,Dir) := false

Optimizing Scheduling for LIFT ASM

- Scheduling only non-crowed lifts: constraining scheduler hasBeenCalledTo to non-crowded(L).
  NB.This scheduling affects the correctness property in case of a continuously crowded lift.
- Reserving L for floor 1 and section [n,m]:
  - constraining the monitored fct hasToDeliverAt(L,_) to those floors
  - restricting the monitored fct existsCallFromTo (Floor,_) for those Floor to calls within that section
  - refining hasBeenCalledTo correspondingly to take into account also the requested target section.
- NB.The correctness property is relativized to the served section.

Exercise

- Extend the LIFT ASM by introducing opening and closing doors
  - as atomic action
  - as durative action
  - with error handling (doors do not open/close)
- For a Petri net solution see section 4, in particular Fig. 26 pg. 87, of
  - W. Reisig: Petri Nets in Software Engineering.
    - In: W. Brauer, W. Reisig, G. Rozenberg (Eds.): Petri Nets: Applications and Relationships to other Models of Concurrecny. Springer LNCS 255 (1987) pp.63-96.

References

- For a solution in B which inspired the ASMs developed here see Section 8.3. of
  - J.-R.Abrial: The B-Book . Cambridge University Press 1996
- For a Petri net solution see section 4, in particular Fig. 26 pg. 87, of
  - W. Reisig: Petri Nets in Software Engineering.
    - In: W. Brauer, W. Reisig, G. Rozenberg (Eds.): Petri Nets: Applications and Relationships to other Models of Concurrecny. Springer LNCS 255 (1987) pp.63-96.
- E. Börger, R. Stärk: Abstract State Machines. A Method for High-Level System Design and Analysis Springer-Verlag 2003, see http://www.di.unipi.it/AsmBook
  - See Chapter 2.3