Level Crossing case study - SHARCS development

1 Overview

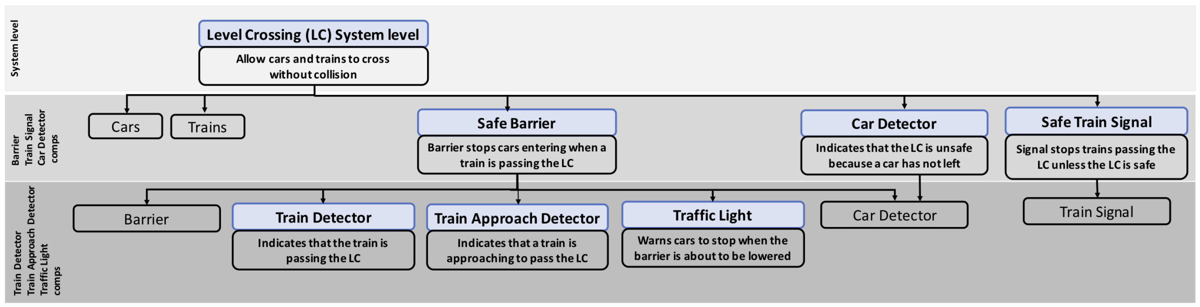

System level
Level Crossing (LC) System level
Allow cars and trains to cross without collision
Barrier Train Signal Car Detector comps
Cars
Trains
Safe Barrier
Barrier stops cars entering when a train is passing the LC
Car Detector
Indicates that the LC is unsafe because a car has not left
Safe Train Signal
Signal stops trains passing the LC unless the LC is safe
Train Detector Train Approach Detector Traffic Light comps
Barrier
Train Detector
Indicates that the train is passing the LC
Train Approach Detector
Indicates that a train is approaching to pass the LC
Traffic Light
Warns cars to stop when the barrier is about to be lowered
Car Detector
Train Signal

Figure 1: Level crossing: hierarchical component design, flow down requirements

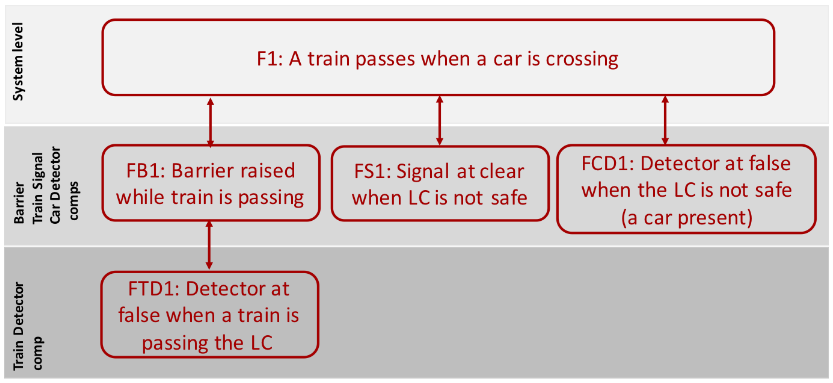

System level
F1: A train passes when a car is crossing
Barrier Train Signal Car Detector comps
FB1: Barrier raised while train is passing
FS1: Signal at clear when LC is not safe
FCD1: Detector at false when the LC is not safe (a car present)
Train Detector comp
FTD1: Detector at false when a train is passing the LC

Figure 2: Level crossing: hierarchical failures

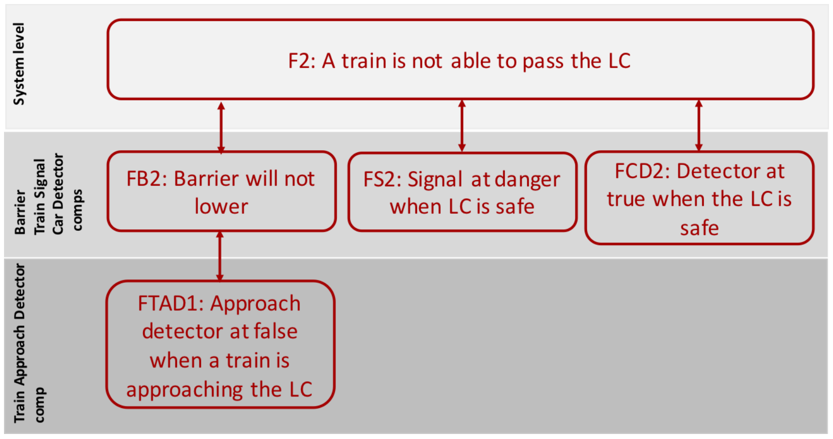

System level
Barrier
Train Signal
Car Detector
comps
Train Approach Detector
comp
F2: A train is not able to pass the LC
FB2: Barrier will not lower
FS2: Signal at danger when LC is safe
FCD2: Detector at true when the LC is safe
FTAD1: Approach detector at false when a train is approaching the LC

Figure 3: Level crossing: hierarchical failures

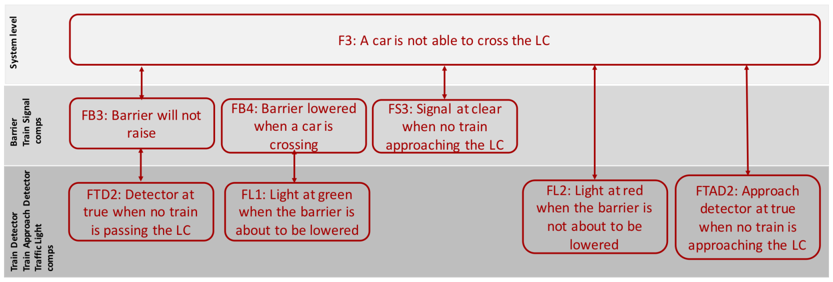

System level
Barrier
Train Signal
comps
Train Detector
Train Approach Detector
Traffic Light
comps
F3: A car is not able to cross the LC
FB3: Barrier will not raise
FB4: Barrier lowered when a car is crossing
FS3: Signal at clear when no train approaching the LC
FTD2: Detector at true when no train is passing the LC
FL1: Light at green when the barrier is about to be lowered
FL2: Light at red when the barrier is not about to be lowered
FTAD2: Approach detector at true when no train is approaching the LC

Figure 4: Level crossing: hierarchical failures

2 System Level

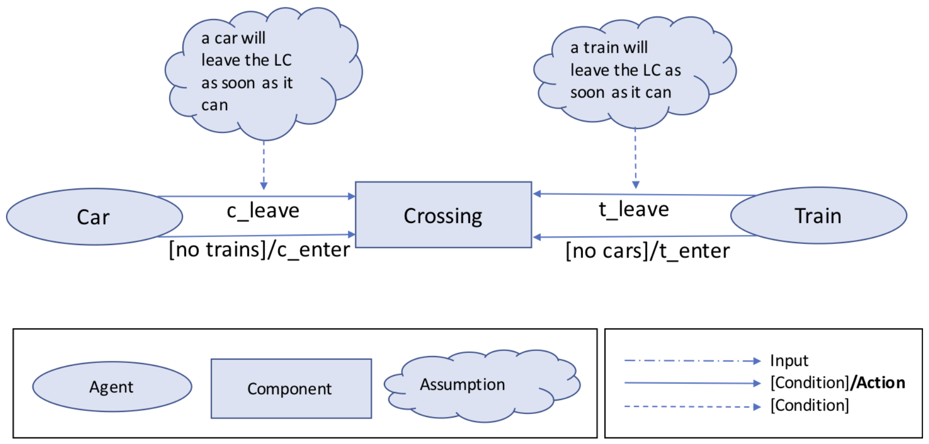
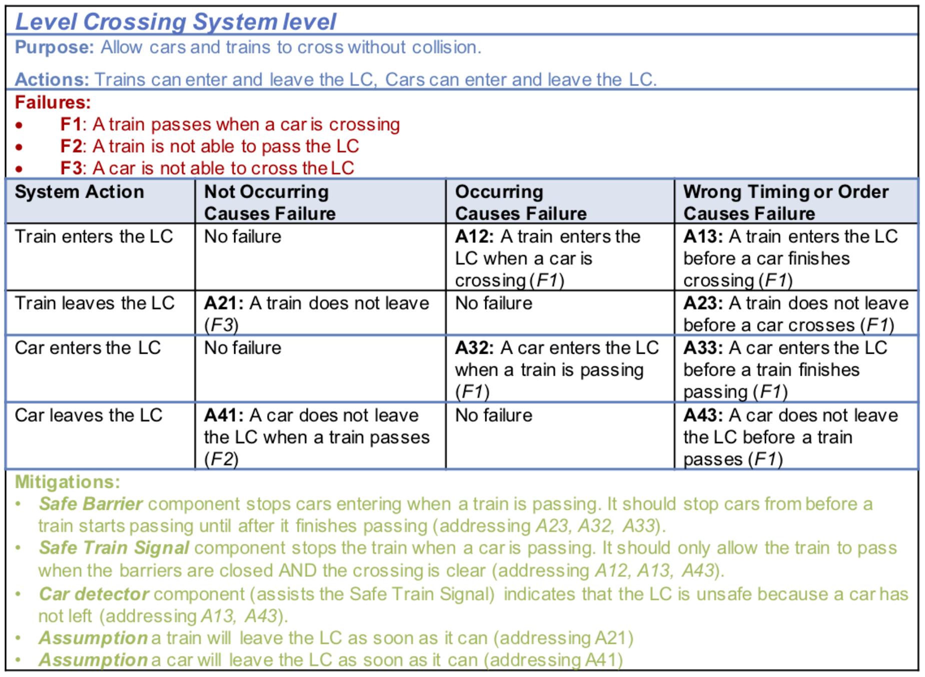


Figure 5: System level, abstraction control diagram



Level Crossing System level

Purpose: Allow cars and trains to cross without collision.

Actions: Trains can enter and leave the LC, Cars can enter and leave the LC.

Failures:

- **F1**: A train passes when a car is crossing
- **F2**: A train is not able to pass the LC
- **F3**: A car is not able to cross the LC

System Action	Not Occurring Causes Failure	Occurring Causes Failure	Wrong Timing or Order Causes Failure
Train enters the LC	No failure	**A12:** A train enters the LC when a car is crossing (*F1*)	**A13:** A train enters the LC before a car finishes crossing (*F1*)
Train leaves the LC	**A21:** A train does not leave (*F3*)	No failure	**A23:** A train does not leave before a car crosses (*F1*)
Car enters the LC	No failure	**A32:** A car enters the LC when a train is passing (*F1*)	**A33:** A car enters the LC before a train finishes passing (*F1*)
Car leaves the LC	**A41:** A car does not leave the LC when a train passes (*F2*)	No failure	**A43:** A car does not leave the LC before a train passes (*F1*)

Mitigations:

- ***Safe Barrier*** component stops cars entering when a train is passing. It should stop cars from before a train starts passing until after it finishes passing (addressing *A23, A32, A33*).
- ***Safe Train Signal*** component stops the train when a car is passing. It should only allow the train to pass when the barriers are closed AND the crossing is clear (addressing *A12, A13, A43*).
- ***Car detector*** component (assists the Safe Train Signal) indicates that the LC is unsafe because a car has not left (addressing *A13, A43*).
- ***Assumption*** a train will leave the LC as soon as it can (addressing A21)
- ***Assumption*** a car will leave the LC as soon as it can (addressing A41)

Figure 6: System level, action analysis table

2.1 Event-B System Model

```
context c0
sets
 CAR   // The set of cars
 TRAIN // The set of trains
end
```

```
machine m0
sees c0
variables
 cars // The set of cars in the level crossing
 trains // The set of trains in the level crossing
invariants
 theorem @typeof−cars: cars ⊆ CAR
 theorem @typeof−trains: trains ⊆ TRAIN

 // There should be no collision
 @safety: trains = ∅ ∨ cars = ∅
events
 event INITIALISATION
 begin
  @init−cars: cars := ∅
  @init−trains: trains := ∅
 end

 /**
 * A car @c enters the level crossing.
 * Guards:
 * − @c is not yet in the level crossing
 * − There are no trains in the level crossing
 * Actions:
 * − @c is added to the set of cars in the level crossing.
 */
 event car_enters_LC
 any c where
  @grd1: c ∉ cars
  @grd2: trains = ∅
 then
  @act1: cars := cars ∪ {c}
 end
```

```
/**
 * A car @c leaves the level crossing.
 * Guards:
 * – @c is in the level crossing
 * Actions:
 * – @c is removed from the set of cars in the level crossing.
 */
event car_leaves_LC
any c where
 @grd1: c ∈ cars
then
 @act1: cars := cars \ {c}
end

/**
 * A train @t enters the level crossing.
 * Guards:
 * – There are no trains in the level crossing
 * – There are no cars in the level crossing
 * Actions:
 * – @t is added to the set of trains in the level crossing.
 */
event train_enters_LC
any t where
 @grd1: t ∈ TRAIN
 @grd2: cars = ∅
then
 @act1: trains := trains ∪ {t}
end

/**
 * A train @t leaves the level crossing.
 * Guards:
 * – @t is in the level crossing
 * Actions:
 * – @t is removed from the set of trains in the level crossing.
 */
event train_leaves_LC
any t where
 @grd1: t ∈ trains
then
 @act1: trains := trains \ {t}
end
```

```
    end
```

3 Safe Barrier Component

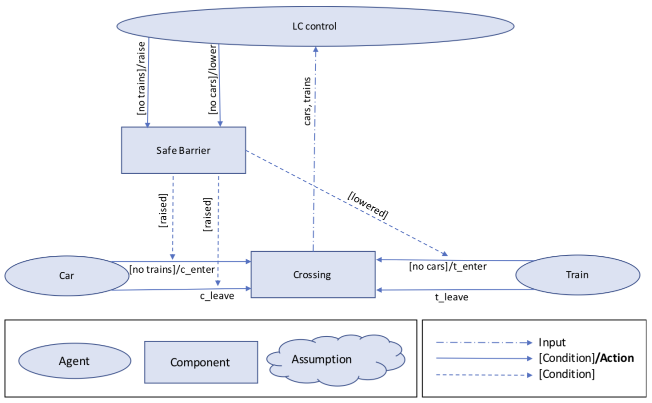
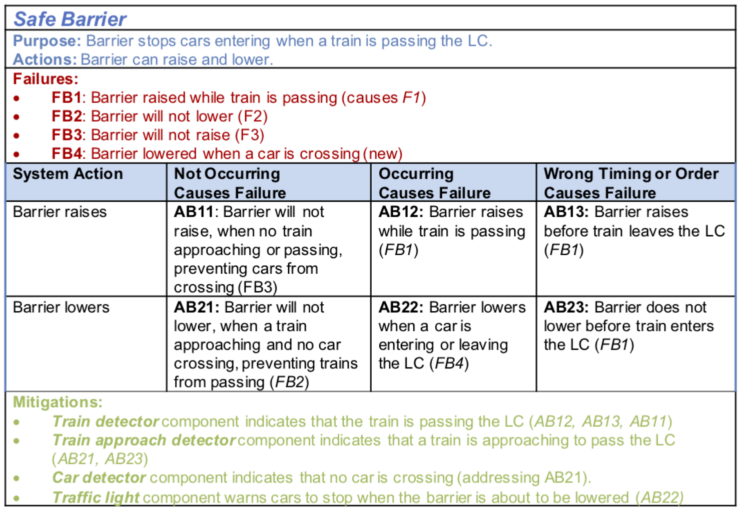


Figure 7: Safe Barrier component, abstraction control diagram



Safe Barrier

Purpose: Barrier stops cars entering when a train is passing the LC.
Actions: Barrier can raise and lower.

Failures:

- **FB1**: Barrier raised while train is passing (causes *F1*)
- **FB2**: Barrier will not lower (F2)
- **FB3**: Barrier will not raise (F3)
- **FB4**: Barrier lowered when a car is crossing (new)

System Action	Not Occurring Causes Failure	Occurring Causes Failure	Wrong Timing or Order Causes Failure
Barrier raises	**AB11**: Barrier will not raise, when no train approaching or passing, preventing cars from crossing (FB3)	**AB12:** Barrier raises while train is passing (*FB1*)	**AB13:** Barrier raises before train leaves the LC (*FB1*)
Barrier lowers	**AB21:** Barrier will not lower, when a train approaching and no car crossing, preventing trains from passing (*FB2*)	**AB22:** Barrier lowers when a car is entering or leaving the LC (*FB4*)	**AB23:** Barrier does not lower before train enters the LC (*FB1*)

Mitigations:

- ***Train detector*** component indicates that the train is passing the LC (*AB12, AB13, AB11*)
- ***Train approach detector*** component indicates that a train is approaching to pass the LC (*AB21, AB23*)
- ***Car detector*** component indicates that no car is crossing (addressing AB21).
- ***Traffic light*** component warns cars to stop when the barrier is about to be lowered (*AB22)*

Figure 8: Barrier component, action analysis table

3.1 Event-B Safe Barrier Model

```
context c1
sets
 BARRIER
constants
 Lowered
 Raised
axioms
 @def−BARRIER: partition(BARRIER, {Lowered}, {Raised})
end
```

```
machine m1
refines m0
sees c0 c1
variables
 cars // The set of cars in the level crossing
 trains // The set of trains in the level crossing
 barriers // The status of the barriers: Lowered, Raised
invariants
 @safety−barriers_cars: barriers = Lowered ⇒ cars = ∅
 @safety−barriers_trains: barriers = Raised ⇒ trains = ∅
events
 event INITIALISATION extends INITIALISATION
 begin
  @init−barriers: barriers := Raised
 end

 /**
 * A car only can enter if the barriers are not Lowered
 */
 event car_enters_LC refines car_enters_LC
 any c where
  @grd1: c ∉ cars
  @grd2: barriers = Raised
 then
  @act1: cars := cars ∪ {c}
 end

 /**
 * A car only can leave if the barriers are not Lowered
 */
 event car_leaves_LC extends car_leaves_LC
```

```
when
 @grd2: barriers = Raised
end

/**
* A train only can safely enters the level crossing if the barriers are
    Lowered
*/
event train_enters_LC refines train_enters_LC
any t where
 @grd1: t ∈ TRAIN
 @grd2: barriers = Lowered
then
 @act1: trains := trains ∪ {t}
end

event train_leaves_LC extends train_leaves_LC
end

/**
* Lowering the barrier from ”raised” status
* Guards:
* – The barriers are raised
* Actions:
* – The barriers are lowered
*/
event barriers_lowers
when
 @grd1: barriers = Raised
 @grd2: cars = ∅
then
 @act1: barriers := Lowered
end


/**
* Raising the barrier from ”lowered” status
* Guards:
* – The barriers are lowering
* – There are no trains
* Actions:
* – The barriers are raised
*/
event barriers_raises
```

```
  when
   @grd1: barriers = Lowered
   @grd2: trains = ∅
  then
   @act1: barriers := Raised
  end

end
```

4 Safe Train Signal Component

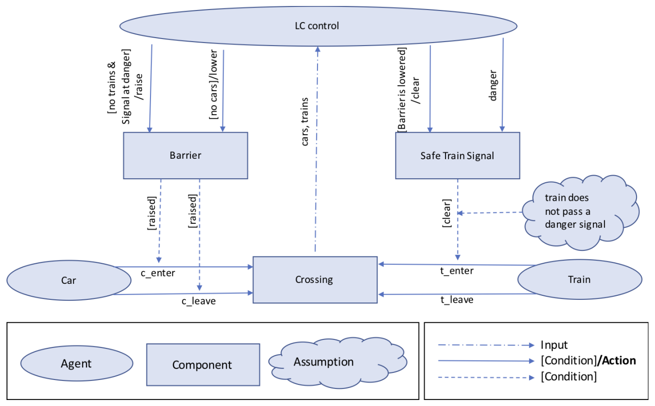
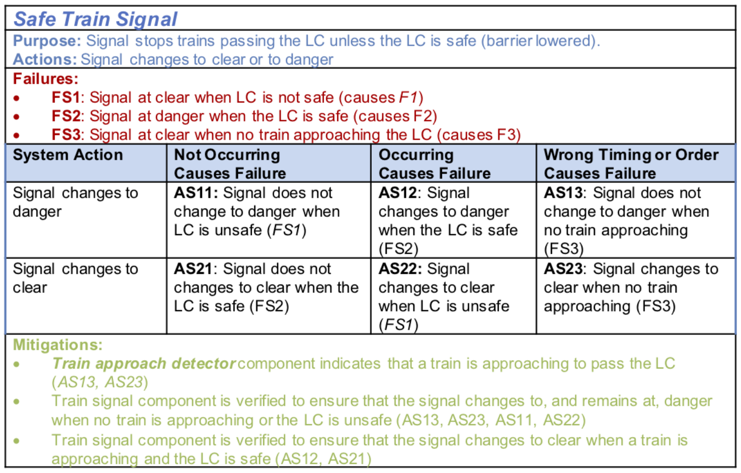


Figure 9: Safe Train Signal component, abstraction control diagram


Safe Train Signal

Purpose: Signal stops trains passing the LC unless the LC is safe (barrier lowered).
Actions: Signal changes to clear or to danger

Failures:
- **FS1**: Signal at clear when LC is not safe (causes *F1*)
- **FS2**: Signal at danger when the LC is safe (causes F2)
- **FS3**: Signal at clear when no train approaching the LC (causes F3)

System Action	**Not Occurring Causes Failure**	**Occurring Causes Failure**	**Wrong Timing or Order Causes Failure**
Signal changes to danger	**AS11:** Signal does not change to danger when LC is unsafe (*FS1*)	**AS12**: Signal changes to danger when the LC is safe (FS2)	**AS13**: Signal does not change to danger when no train approaching (FS3)
Signal changes to clear	**AS21**: Signal does not changes to clear when the LC is safe (FS2)	**AS22:** Signal changes to clear when LC is unsafe (*FS1*)	**AS23**: Signal changes to clear when no train approaching (FS3)

Mitigations:
- ***Train approach detector*** component indicates that a train is approaching to pass the LC (*AS13, AS23*)
- Train signal component is verified to ensure that the signal changes to, and remains at, danger when no train is approaching or the LC is unsafe (AS13, AS23, AS11, AS22)
- Train signal component is verified to ensure that the signal changes to clear when a train is approaching and the LC is safe (AS12, AS21)

Figure 10: Train Signal component, action analysis table

4.1 Event-B Safe Train Signal Model

```
context c2
sets
 TRAIN_SIGNAL
constants
 SIGNAL_CLEAR
 SIGNAL_DANGER
axioms
 @def−TRAIN_SIGNAL: partition(TRAIN_SIGNAL, {SIGNAL_CLEAR}, {
     SIGNAL_DANGER})
end
```

```
machine m2
refines m1
sees c0 c1 c2
variables
 cars // The set of cars in the level crossing
 trains // The set of trains in the level crossing
 barriers // The status of the barriers: Lowered, Raised
 train_signal // The train signal
invariants
 @safe_train_signal: train_signal = SIGNAL_CLEAR ⇒ barriers = Lowered
events
 event INITIALISATION extends INITIALISATION
 begin
  @init−train_signal: train_signal := SIGNAL_DANGER
 end

 event car_enters_LC extends car_enters_LC
 end

 event car_leaves_LC extends car_leaves_LC
 end

 /**
 * A train @t enters the level crossing.
 * Guards:
 * − There are no trains in the level crossing
 * − The train signal is "Clear"
 * Actions:
 * − @t is added to the set of trains in the level crossing.
  */
 event train_enters_LC refines train_enters_LC
 any t where
  @grd1: t ∈ TRAIN
  @grd2: train_signal = SIGNAL_CLEAR
 then
  @act1: trains := trains ∪ {t}
 end

 event train_leaves_LC extends train_leaves_LC
 end

 event barriers_lowers extends barriers_lowers
 end


 /**
  * The barriers are raising only when the train signal is "Danger"
  */
 event barriers_raises extends barriers_raises
 when
  @grd3: train_signal = SIGNAL_DANGER
 end

 /**
 * Train signal set to "Clear"
 * Guards:
 * - Train signal is current "Danger"
 * - There are no cars detected
 * - The barriers are Lowered
 * Actions:
 * - Train signal is set to "Clear"
 */
 event train_signal_to_clear
 when
  @grd1: train_signal = SIGNAL_DANGER
  @grd2: barriers = Lowered
 then
  @act1: train_signal := SIGNAL_CLEAR
 end

 /**
 * Train signal set to "Danger"
 * Guards:
   * – Train signal is current "Clear"
   * Actions:
   * – Train signal is set to "Danger"
   */
  event train_signal_to_danger
  when
   @grd1 : train_signal = SIGNAL_CLEAR
  then
   @act1 : train_signal := SIGNAL_DANGER
  end

end
```

5 Car Detection Component

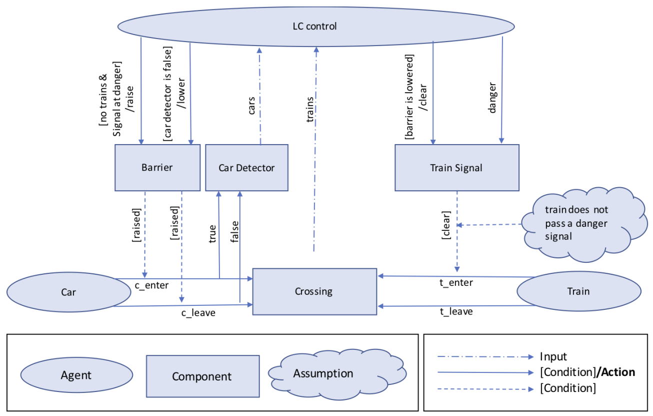
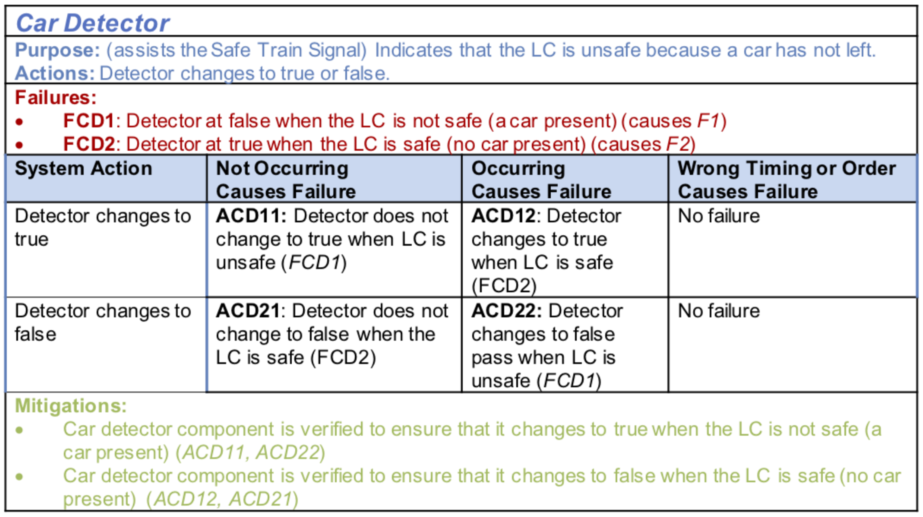


Figure 11: Car detector component, abstraction control diagram



Car Detector

Purpose: (assists the Safe Train Signal) Indicates that the LC is unsafe because a car has not left.
Actions: Detector changes to true or false.

Failures:

- **FCD1**: Detector at false when the LC is not safe (a car present) (causes *F1*)
- **FCD2**: Detector at true when the LC is safe (no car present) (causes *F2*)

System Action	Not Occurring Causes Failure	Occurring Causes Failure	Wrong Timing or Order Causes Failure
Detector changes to true	**ACD11:** Detector does not change to true when LC is unsafe (*FCD1*)	**ACD12**: Detector changes to true when LC is safe (FCD2)	No failure
Detector changes to false	**ACD21**: Detector does not change to false when the LC is safe (FCD2)	**ACD22:** Detector changes to false pass when LC is unsafe (*FCD1*)	No failure

Mitigations:

- Car detector component is verified to ensure that it changes to true when the LC is not safe (a car present) (*ACD11, ACD22*)
- Car detector component is verified to ensure that it changes to false when the LC is safe (no car present) (*ACD12, ACD21*)

Figure 12: Car detector component, action analysis table

5.1 Event-B Car Detector Model

```
machine m3
refines m2
sees c0 c1 c2
variables
 cars // The set of cars in the level crossing
 trains // The set of trains in the level crossing – at most one
 barriers // The status of the barriers: Lowered, Lowering, Raising, Raised
 train_signal // The train signal
 car_detector // car detector mechanism
invariants
 // The car detector mechanism indicating whether or not there are cars
     in the level crossing
 @car_detector: car_detector = TRUE ⇔ cars ≠ ∅
events
 event INITIALISATION extends INITIALISATION
 begin
  @init−car_detector: car_detector := FALSE
 end

 /**
 * Activate the car detector when a car enter the level crossing.
 */
 event car_enters_LC extends car_enters_LC
 begin
  @act2: car_detector := TRUE
 end

 /**
 * Set the car detector accordingly to whether or not @c is the only car in
     the level crossing
 */
 event car_leaves_LC extends car_leaves_LC
 begin
  @act2: car_detector := bool(cars ≠ {c})
 end

 event train_enters_LC extends train_enters_LC
 end

 event train_leaves_LC extends train_leaves_LC
 end
 /**
  * Lowering the barrier from "raised" status
  * Guards:
  * - The barriers are raised
  * - The car detector is FALSE
  * Actions:
  * - The barriers are lowered
  */
 event barriers_lowers refines barriers_lowers
 when
  @grd1: barriers = Raised
  @grd2: car_detector = FALSE
 then
  @act1: barriers := Lowered
 end

 event barriers_raises extends barriers_raises
 end

 event train_signal_to_clear extends train_signal_to_clear
 end

 event train_signal_to_danger extends train_signal_to_danger
 end


end
```

6 Train Detector Component

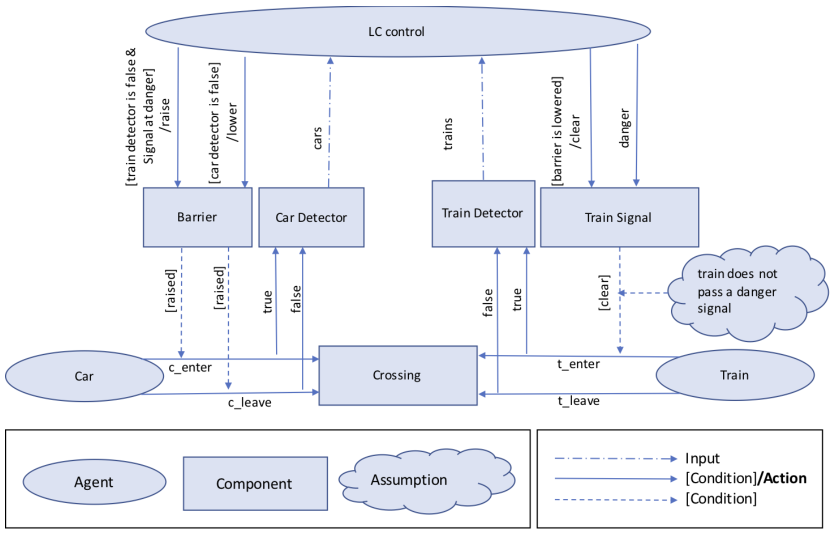
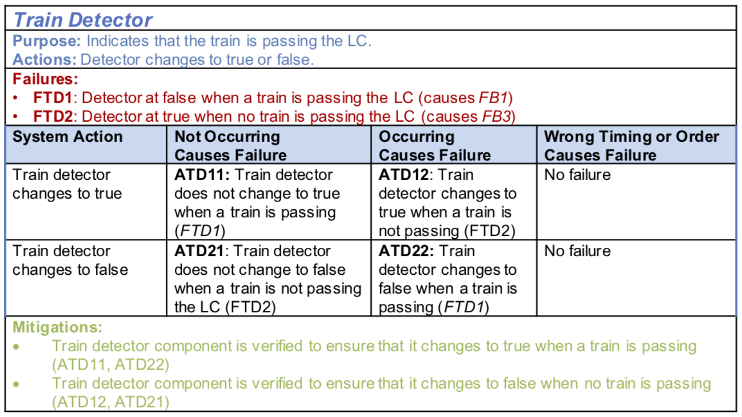


Figure 13: Train detector component, abstraction control diagram



Train Detector

Purpose: Indicates that the train is passing the LC.
Actions: Detector changes to true or false.

Failures:
- **FTD1**: Detector at false when a train is passing the LC (causes *FB1*)
- **FTD2**: Detector at true when no train is passing the LC (causes *FB3*)

System Action	Not Occurring Causes Failure	Occurring Causes Failure	Wrong Timing or Order Causes Failure
Train detector changes to true	**ATD11:** Train detector does not change to true when a train is passing (*FTD1*)	**ATD12**: Train detector changes to true when a train is not passing (FTD2)	No failure
Train detector changes to false	**ATD21**: Train detector does not change to false when a train is not passing the LC (FTD2)	**ATD22:** Train detector changes to false when a train is passing (*FTD1*)	No failure

Mitigations:
- Train detector component is verified to ensure that it changes to true when a train is passing (ATD11, ATD22)
- Train detector component is verified to ensure that it changes to false when no train is passing (ATD12, ATD21)

Figure 14: Train detector component, action analysis table

6.1 Event-B Train Detector Model

```
machine m4
refines m3
sees c0 c1 c2
variables
 cars // The set of cars in the level crossing
 trains // The set of trains in the level crossing — at most one
 barriers // The status of the barriers: Lowered, Lowering, Raising, Raised
 train_signal // The train signal
 car_detector // car detector mechanism
 train_detector // train detector mechanism
invariants
 // The train detector mechanism indicating whether or not there are
     trains in the level crossing
 @train_detector: train_detector = TRUE ⇔ trains ≠ ∅

events
 event INITIALISATION extends INITIALISATION
 begin
  @init−train_detector: train_detector := FALSE
 end

 event car_enters_LC extends car_enters_LC
 end

 event car_leaves_LC extends car_leaves_LC
 end

 event train_enters_LC extends train_enters_LC
 begin
  @act2: train_detector := TRUE
 end

 event train_leaves_LC extends train_leaves_LC
 begin
  @act2: train_detector := bool(trains ≠ {t})
 end

 event barriers_lowers extends barriers_lowers
 end

 /**
 * Raising the barrier from "lowered" status
```

```
    * Guards:
    * – The barriers are lowered
    * – The train detector is FALSE
    * – The train signal is Danger
    * Actions:
    * – The barriers are raised
    */
   event barriers_raises refines barriers_raises
   when
    @grd1: barriers = Lowered
    @grd2: train_detector = FALSE
    @grd3: train_signal = SIGNAL_DANGER
   then
    @act1: barriers := Raised
   end

   event train_signal_to_clear extends train_signal_to_clear
   end

   event train_signal_to_danger extends train_signal_to_danger
   end

  end
```

7 Train Approach Detector Component

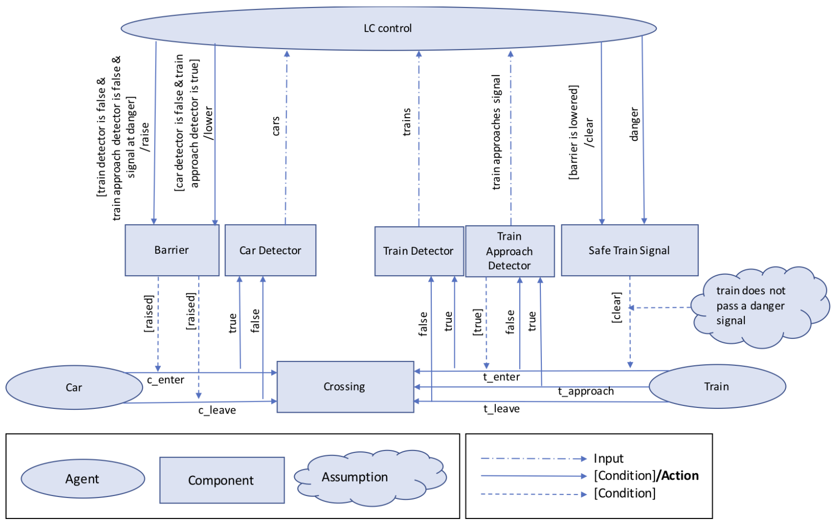
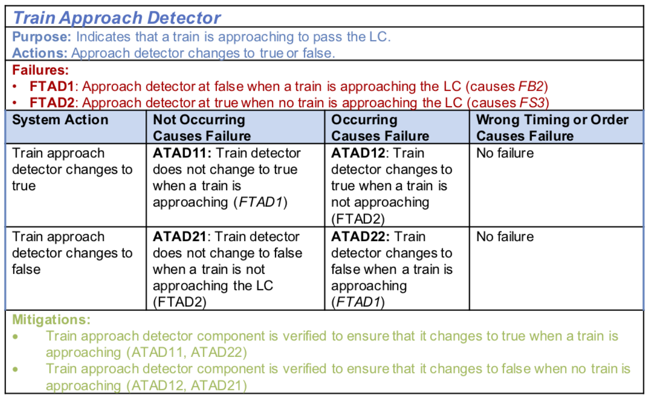


Figure 15: Train approach detector component, abstraction control diagram



Train Approach Detector

Purpose: Indicates that a train is approaching to pass the LC.
Actions: Approach detector changes to true or false.

Failures:
- **FTAD1**: Approach detector at false when a train is approaching the LC (causes *FB2*)
- **FTAD2**: Approach detector at true when no train is approaching the LC (causes *FS3*)

System Action	Not Occurring Causes Failure	Occurring Causes Failure	Wrong Timing or Order Causes Failure
Train approach detector changes to true	**ATAD11:** Train detector does not change to true when a train is approaching (*FTAD1*)	**ATAD12**: Train detector changes to true when a train is not approaching (FTAD2)	No failure
Train approach detector changes to false	**ATAD21**: Train detector does not change to false when a train is not approaching the LC (FTAD2)	**ATAD22:** Train detector changes to false when a train is approaching (*FTAD1*)	No failure

Mitigations:
- Train approach detector component is verified to ensure that it changes to true when a train is approaching (ATAD11, ATAD22)
- Train approach detector component is verified to ensure that it changes to false when no train is approaching (ATAD12, ATAD21)

Figure 16: Train approach detector component, action analysis table

7.1 Event-B Train Approach Detector Model

```
machine m5
refines m4
sees c0 c1 c2
variables
 cars // The set of cars in the level crossing
 trains // The set of trains in the level crossing − at most one
 barriers // The status of the barriers: Lowered, Lowering, Raising, Raised
 train_signal // The train signal
 car_detector // car detector mechanism
 train_detector // train detector mechanism
 train_approach_detector // train approach detector mechanism
invariants
 theorem @typeof−train_approach_detector: train_approach_detector ∈
     BOOL
events
 event INITIALISATION extends INITIALISATION
 begin
  @init−train_approach_detector: train_approach_detector := FALSE
 end

 event car_enters_LC extends car_enters_LC
 end

 event car_leaves_LC extends car_leaves_LC
 end

 event train_enters_LC extends train_enters_LC
 when
  @grd3: train_approach_detector = TRUE
 then
  @act3: train_approach_detector := FALSE
 end

 event train_leaves_LC extends train_leaves_LC
 end

 event barriers_lowers extends barriers_lowers
 when
  @grd3: train_approach_detector = TRUE
 end

 event barriers_raises extends barriers_raises
```

```
   when
    @grd4 : train_approach_detector = FALSE
   end

   event train_signal_to_clear extends train_signal_to_clear
   end

   event train_signal_to_danger extends train_signal_to_danger
   when
    @grd2 : train_approach_detector = FALSE
   end

   event train_approaches_LC
   begin
    @act1 : train_approach_detector := TRUE
   end

  end
```

8 Traffic Light Component

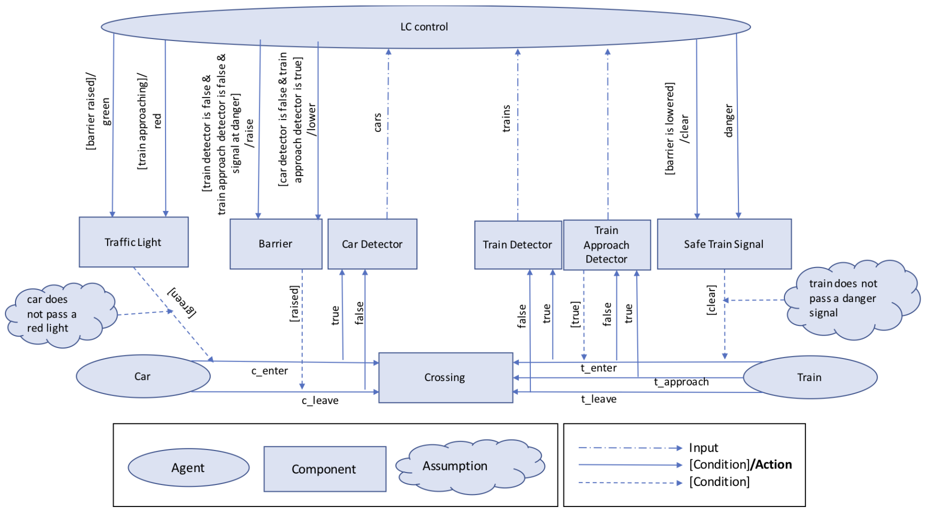
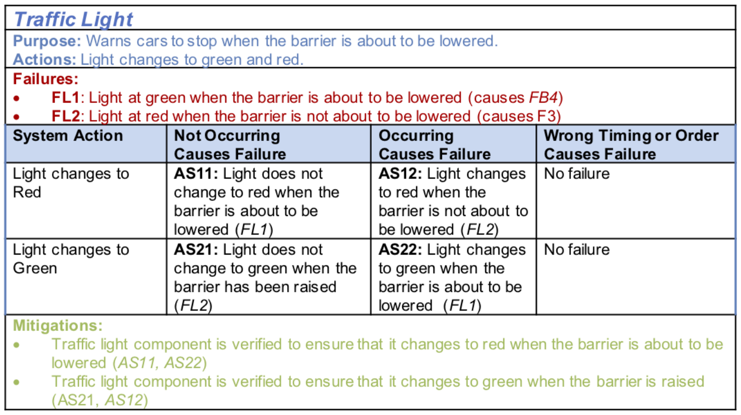


Figure 17: Traffic Light component, abstraction control diagram



Traffic Light

Purpose: Warns cars to stop when the barrier is about to be lowered.
Actions: Light changes to green and red.

Failures:
- **FL1**: Light at green when the barrier is about to be lowered (causes *FB4*)
- **FL2**: Light at red when the barrier is not about to be lowered (causes F3)

System Action	Not Occurring Causes Failure	Occurring Causes Failure	Wrong Timing or Order Causes Failure
Light changes to Red	**AS11:** Light does not change to red when the barrier is about to be lowered (*FL1*)	**AS12:** Light changes to red when the barrier is not about to be lowered (*FL2*)	No failure
Light changes to Green	**AS21:** Light does not change to green when the barrier has been raised (*FL2*)	**AS22:** Light changes to green when the barrier is about to be lowered (*FL1*)	No failure

Mitigations:
- Traffic light component is verified to ensure that it changes to red when the barrier is about to be lowered (*AS11, AS22*)
- Traffic light component is verified to ensure that it changes to green when the barrier is raised (AS21, *AS12*)

Figure 18: Traffic Light component, action analysis table

8.1 Event-B Traffic Light Model

```
context c6
sets
 TRAFFIC_LIGHT
constants
 LIGHT_RED
 LIGHT_GREEN
axioms
 @def−TRAFFIC_LIGHT: partition(TRAFFIC_LIGHT, {LIGHT_RED}, {
     LIGHT_GREEN})
end
```

```
machine m6
refines m5
sees c0 c1 c2 c6
variables
 cars // The set of cars in the level crossing
 trains // The set of trains in the level crossing − at most one
 barriers // The status of the barriers: Lowered, Lowering, Raising, Raised
 train_signal // The train signal
 car_detector // car detector mechanism
 train_detector // train detector mechanism
 train_approach_detector // train approach detector mechanism
 traffic_light // The traffic light
invariants
 // If the traffic light is green then the barriers are not Lowered
 @safety−traffic_light_barriers: traffic_light = LIGHT_GREEN ⇒ barriers =
     Raised
events
 event INITIALISATION extends INITIALISATION
 begin
  @init−traffic_light: traffic_light := LIGHT_RED
 end

 event car_enters_LC refines car_enters_LC
 any c where
  @grd1: c ∉ cars
  @grd2: traffic_light = LIGHT_GREEN
 then
  @act1: cars := cars ∪ {c}
  @act2: car_detector := TRUE
 end

 event car_leaves_LC extends car_leaves_LC
 end

 event train_enters_LC extends train_enters_LC
 end

 event train_leaves_LC extends train_leaves_LC
 end

 event barriers_lowers extends barriers_lowers
 when
  @grd4 : traffic_light = LIGHT_RED
 end

 event barriers_raises extends barriers_raises
 end

 event train_signal_to_clear extends train_signal_to_clear
 end

 event train_signal_to_danger extends train_signal_to_danger
 end

 event train_approaches_LC extends train_approaches_LC
 end

 event traffic_light_to_green
 when
  @grd1 : traffic_light = LIGHT_RED
  @grd2 : barriers = Raised
 then
  @act1 : traffic_light := LIGHT_GREEN
 end

 event traffic_light_to_red
 when
  @grd1 : traffic_light = LIGHT_GREEN
  @grd2 : train_approach_detector = TRUE
 then
  @act1 : traffic_light := LIGHT_RED
 end
end
```