30.11.21 ФМ5 Лекция 2

Метод Event-B. Платформа Rodin

Платформа Rodin: https://www3.hhu.de/stups/handbook/rodin/current/pdf/rodin-doc.pdf
http://www.event-b.org/ - сайт **Event-B.org**

Пример. Управление движением на перекрестке
Пример. Управление движением автомобилей по мосту
Другое решение задачи управления на мосту
Архитектура платформы **Rodin**
Построение спецификации в **Rodin**:
проект, контекст, машина, теория.
Методы доказательства формул корректности.
Анализ поведения машины в аниматоре.

Автоматное программирование на базисе Event-B

Язык спецификаций Event-B используется в качестве базисного языка для автоматного программирования.

Типы BOOL, INT, NAT.

Оператор присваивания <Переменная> := <Выражение>

Вместо $x \in BOOL$ будем записывать в виде x: BOOL

Начальная инициализация переменных реализуется в секции состояния. Например, x: BOOL := false.

Вместо конструкции partition со сложной семантикой используется описание типа перечисления.

Сегмент **M: if (C) A else B #L** преобразуется к виду

M: C → A #L

M: ¬ C → B #L

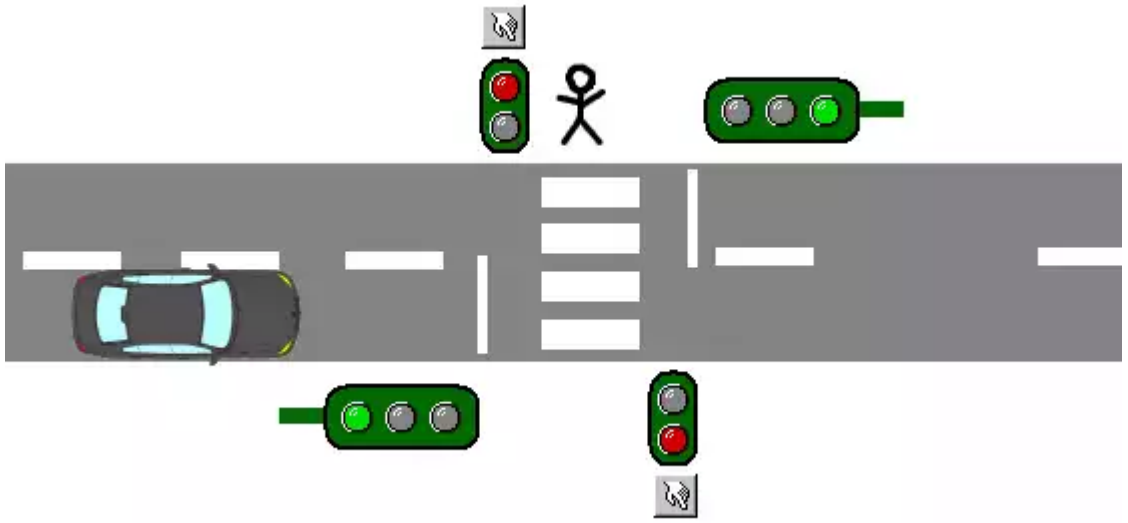
ранее было: **M: C → A #L**

M: B #L

Будем использовать комбинированную нотацию двух языков.

Управление движением на перекрестке

Содержательное описание. На пешеходном переходе через автомобильную магистраль имеется два светофора. Светофор для пешеходов: красный или зеленый. Светофор для автомобилей: красный, желтый или зеленый. Запрещено движение на красный светофор для пешеходов и автомобилей. Контроллер управляет переключениями светофоров. Зеленый светофор для пешеходов сочетается только с красным светофором для автомобилей.



На перекрестке. Анализ требований

Система управления движением не может применяться на практике. В модели должно быть время.

Объекты внешнего окружения: светофор для пешеходов и светофор для автомобилей.

Требования окружения:

RE1: Светофор для пешеходов может быть зеленым или красным.

RE2: Светофор для автомобилей может быть зеленым, желтым или красным.

Требование безопасности:

RS: Если светофор для пешеходов – зеленый, то светофор для автомобилей может быть только красным.

Модель упрощенной задачи

в руководстве Rodin: https://www3.hhu.de/stups/handbook/rodin/current/pdf/rodin-doc.pdf

Красный – константа **false**, а зеленый – **true**. Желтого нет.

```
section {
   peds_go: BOOL := false
   cars_go: BOOL := false
   inv  peds_go = false ∨ cars_go = false
}
process PedsCars0 {
Cycle:    [set_peds_go:] cars_go = false → peds_go := true  |
          [set_cars_go:] peds_go = false → cars_go := true  |
          [set_cars_stop:] cars_go:= false  |
          [set_peds_stop:] peds_go := false
        #Cycle
} PedsCars0
```

Модель упрощенной задачи в Event-B

в редакторе Camille

```
machine PedsCars0
variables   peds_go   cars_go
invariants   @inva peds_go ∈ BOOL    @invb cars_go ∈ BOOL
             @invc peds_go = false ∨ cars_go = false
events
 event INITIALISATION  then @act1 peds_go := false  @act2 cars_go := false   end
 event set_peds_go
  where @grd cars_go = false
  then @act1 peds_go := true
 end
 event  set_cars_go
  where @grd peds_go = false
  then @act1 cars_go := true
 end
 event  set_cars_stop then @act1 cars_go:= false end
 event  set_peds_stop then @act1 peds_go := false end
end
```

Упрощенная задача в стандартном редакторе

MACHINE

M0 ›

VARIABLES

- Cars_tl ›
- Peds_tl ›

INVARIANTS

- inv1: Cars_tl∈BOOL not theorem ›
- inv2: Peds_tl∈BOOL not theorem ›
- inv3: ¬(Cars_tl=TRUE ∧ Peds_tl=TRUE) not theorem ›

Упрощенная задача в стандартном редакторе

```
EVENTS
  INITIALISATION:  not extended ordinary ›
  THEN
    act1:   Cars_tl:=FALSE ›
    act2:   Peds_tl:=FALSE ›
  END
```

Упрощенная задача в стандартном редакторе

```
o  set_peds_go:      not extended ordinary ›
   WHERE
   o  grd1:   Cars_tl=FALSE not theorem ›
   THEN
   o  act1:   Peds_tl:=TRUE ›
   END

o  set_peds_stop:    not extended ordinary ›
   THEN
   o  act1:   Peds_tl:=FALSE ›
   END
```

Упрощенная задача в стандартном редакторе

```
o  set_cars:    not extended ordinary ›
   ANY
   o  new_value    ›
   WHERE
   o  grd1:   new_value∈BOOL not theorem ›
   o  grd2:   new_value=TRUE ⇒ Peds_tl=FALSE not theorem ›
   THEN
   o  act1:   Cars_tl≔new_value ›
   END
```

Наша реализация упрощенной задачи

```
section {
   peds_go: BOOL := false
   cars_go: BOOL := false
   inv  peds_go = false ∨ cars_go = false
}
```

Управляющие состояния:

- Cars – машины движутся, пешеходы стоят и ждут;
- Peds – пешеходы переходят магистраль, машины стоят перед красным светофором.

```
process PedsCars1 {
  Cars: inv  peds_go = false & cars_go = true
      #Cars | peds_go := true; cars_go:= false #Peds
  Peds: inv  peds_go = true & cars_go = false
      #Peds | peds_go := false; cars_go:= true #Cars
}
```

Сегмент Cars: #Cars | X эквивалентен Cars: X

Устранение управляющих состояний

```
process PedsCars2 {
    type State = enum { Cars, Peds };
    state:  State;
    state := Cars;
  loop:  switch (state) {
        case Cars:  peds_go := true; cars_go:= false; state:=Peds; break
        case Peds: peds_go := false; cars_go:= true;  state:=Cars
  }  #loop
} PedsCars2
```

```
process PedsCars3 {
Cycle: [Cars_go:] state=Cars → peds_go:=true; cars_go:= false; state:=Peds |
       [Peds_go:] state=Peds → peds_go:=false; cars_go:=true; state:=Cars
       #Cycle
} PedsCars3
```

Простое уточнение

```
section {
   type COLOURS = enum {red,  green}
   peds_col: COLOURS := red ;     cars_col : COLOURS := green
    inv  peds_col = red ∨ peds_col = green & cars_col = red
    inv  peds_col = red ⇔ peds_go = false
    inv  cars_col = red ⇔ cars_go = false
}
process LightControl  refines PedsCars1{
Cars: inv  peds_col = red & cars_col = green
       cars_col:= red; peds_col := green   #Peds
Peds: inv  peds_col = green & cars_col = red
       peds_col := red;  cars_col:= green   #Cars
} LightControl
```

связующие инварианты

Реализация исходной задачи

```
process LightControl {
  Cars: inv  peds_col = red & cars_col = green
        cars_col:= yellowGreen #YellowGreen
  YellowGreen: inv  peds_col = red & cars_col = yellowGreen
        cars_col:= red; peds_col := green #Peds
   Peds: inv  peds_col = green & cars_col = red
        peds_col := red;  cars_col := yellowRed #YellowRed
   YellowRed:  inv  peds_col = red & cars_col = yellowRed
        cars_col:= green #Cars
} LightControl
```

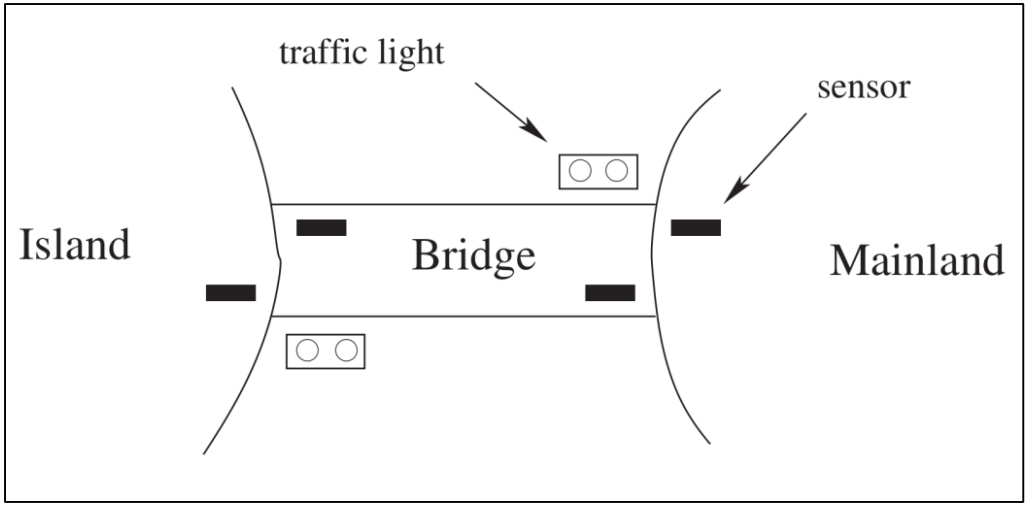
Замена эквивалентной последовательностью:

cars_col:=red; peds_col:=green → cars_col:=yellowGreen; cars_col:=red; peds_col:=green

Редукция сегмента: M: A, B #L → M: A #K
K: B #L

Управление движением автомобилей по мосту

Содержательное описание. Узкий односторонний мост соединяет материк и остров. Два светофора (цвета красный и зеленый) при въезде на мост с материка и с острова. Автомобилям запрещено движение на красный светофор при въезде на мост. Имеются четыре сенсора. Каждый сенсор находится на некотором участке автомобильной трассы и способен фиксировать ситуацию, когда автомобиль находится на этом участке трассы. Первый сенсор находится перед въездом на мост с материка. Второй сенсор на мосту перед съездом на остров. Третий сенсор перед въездом на мост с острова. Четвертый сенсор на мосту перед съездом на материк.



Управление движением автомобилей по мосту

Содержательное описание (прод). Число автомобилей, которые могут находиться на острове, ограничено. Необходимо построить контроллер, который переключает светофоры, используя показания сенсоров, и таким образом обеспечивает безопасное движение автомобилей по мосту.

Детализация и анализ требований. Дадим имена светофорам: mtl – светофор на материке при въезде на мост, itl – светофор на острове при въезде на мост.

Требования окружения:

RE1: Имеются два светофора с именами mtl и itl.

RE2: Каждый светофор может быть зеленым или красным.

Дадим имена сенсорам: mlOut – при въезде на мост с материка, ilIn – при съезде с моста на остров, ilOut – при въезде на мост с острова, mlIn – при съезде с моста на материк.

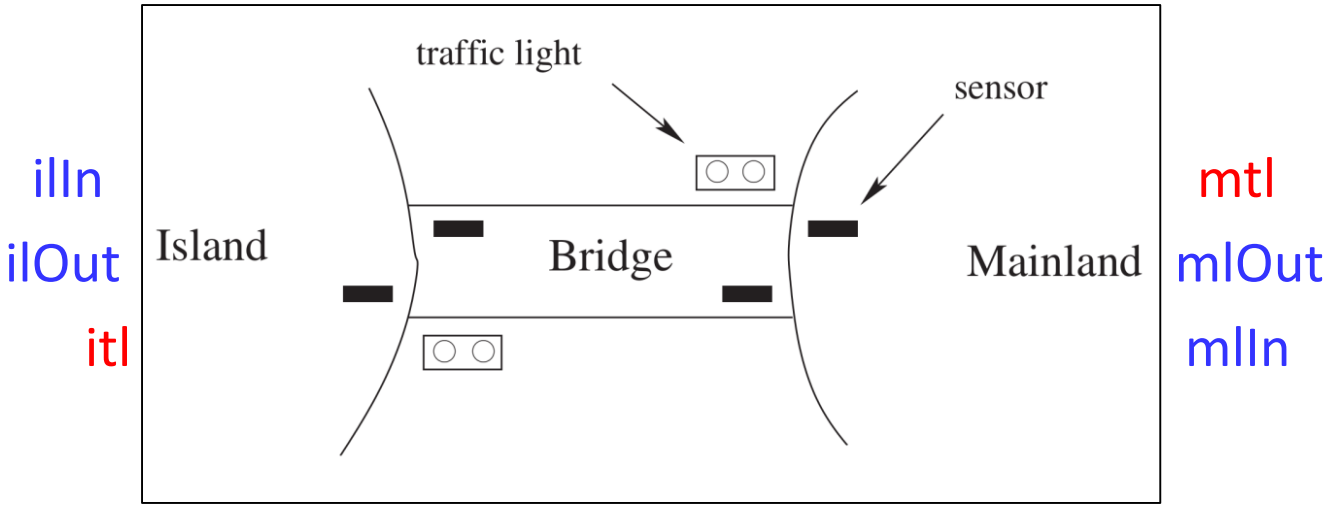
RE3: Имеются четыре сенсора с именами: mlOut, ilIn, ilOut и mlIn.

RE4: Каждый сенсор может быть включен (значение on) или выключен (значение off).

RE5: Сенсор *отключается*, если он изменяет свое значение с on на off.

Сенсоры и светофоры

m – материк, il – остров, tl – traffic light



mlOut – при въезде на мост с материка,

ilIn – при съезде с моста на остров,

ilOut – при въезде на мост с острова,

mlIn – при съезде с моста на материк

Управление движением по мосту

Функциональные требования:

RF1: Отключение сенсора mlOut – очередной автомобиль въехал на мост с материка.

RF2: Отключение сенсора ilIn – очередной автомобиль съехал с моста на остров.

RF3: Отключение сенсора ilOut – очередной автомобиль въехал на мост с острова.

RF4: Отключение сенсора mlIn – очередной автомобиль съехал с моста на материк.

RF5: Число автомобилей на острове ограничено.

RF6: Движение автомобилей по мосту возможно либо с материка на остров, либо с острова на материк. Одновременное движение с материка на остров и с острова на материк невозможно.

Требования безопасности:

RS1: При красном светофоре mtl запрещено движение с материка на мост.

RS2: При красном светофоре itl запрещено движение с острова на мост.

Представленная система управления движением по мосту в целом, и архитектура оборудования в частности, имеют серьезные недостатки и не могут использоваться на практике.

Начальная модель

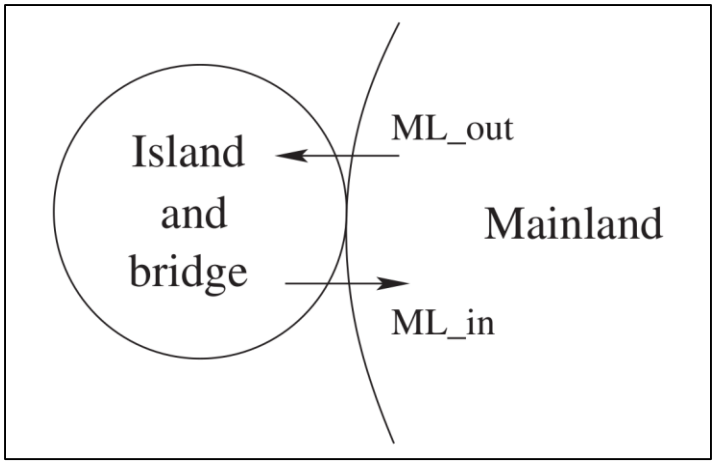
Мост и остров рассматриваются как единое целое.

n – число автомобилей на мосту и на острове.

d – максимальное число автомобилей на острове.

Только два события: уход машины с материка и приход машины на материк.

```
section St0 {
    const d: NAT
    n: NAT
    inv  n <= d
}
process carBridge0er {
   cycle: [ML_out:] n := n + 1 |
          [ML_in:]   n := n - 1
          #cycle
}
```



Начальная модель.Ошибки

```
section St0 {
    const d: NAT
    n: NAT
    inv  n <= d
}
process carBridge0er {
   cycle: [ML_out:] n := n + 1 |
          [ML_in:]   n := n - 1
          #cycle
}
```

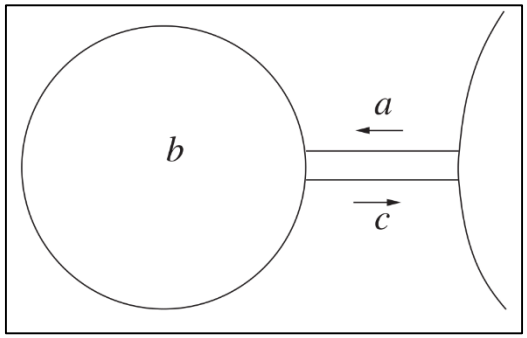
1. После действия n:= n+1 не гарантируется n <= d. n+1<=d
2. Для n := n - 1 генерируется n-1 ∈ NAT
3. В начальный момент не выполняется инвариант n <= d
4. Дедлок. Недоказуема n <= d ⇒ n<d ∨ n>0

Исправленная начальная модель

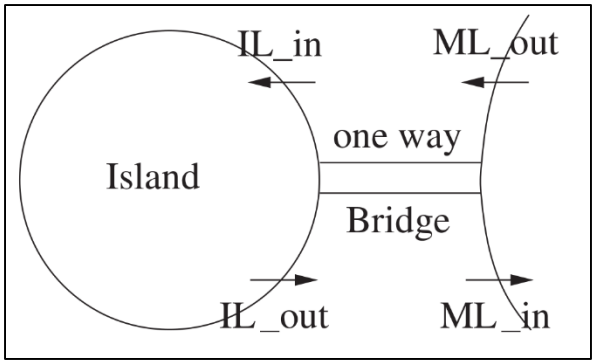
```
section St0 {
    const d: NAT
    axiom d > 0
    n: NAT := 0
    inv  n <= d
}

process carBridge0r {
   cycle: [ML_out:] n < d → n := n + 1 |
          [ML_in:]    n > 0 → n := n - 1
          #cycle
}
```

Уточнение начальной модели




```
section St1 extends St0{
    a: NAT := 0
    b: NAT := 0
    c: NAT := 0
     inv a + b + c = n    //связующий инвариант
     inv a = 0 ∨ c = 0
}
process carBridge1 refines carBridge0 {
    cycle: [ML_out:] a+b<d & c=0 → a := a+1 |
           [ML_in:]    c>0 → c := c-1 |
           [IL_out:]   b>0 & a=0 → b := b-1; c := c+1 |
           [IL_in:]    a>0 →  a := a-1; b := b+1
           #cycle
}
```



Второе уточнение модели

```
section St2 extends St1{
    type Colour = enum {red, green};
    mtl: Colour := red
     itl: Colour := red
     inv mtl=red ∨ itl=red
     inv mtl=green => a+b<d & c=0  //связующий инвариант
     inv itl=green => b>0 & a=0       //связующий инвариант
}
process carBridge2 refines carBridge1{
  cycle: [ML_out:] mtl=green → a := a + 1 |
          [ML_in:]   c>0 → c:=c-1 |
          [IL_out:]  itl=green →  b:=b-1; c:= c+1  |
          [IL_in:]    a>0 → a := a-1; b:=b+1  |
          [Mtl_green:] mtl=red & c=0 & b<d  →  mtl:=green; itl:=red  |
          [Itl_green:]  itl=red & a=0 & b>0 →  itl:=green; mtl:=red
          #cycle
}
```

Проблемы после второго уточнения

Новые события Mtl_green и Itl_green определяют вроде бы естественные правила переключения светофоров: если мост стал пустым, то оба светофора меняют свет так, чтобы стало возможным движение в противоположную сторону.

Для пустого моста получим быстрое мигание светофоров, что, в частности, может привести к аварии. В книге реализовано следующее решение. Если изменено направление движения, то далее необходимо дождаться появления автомобиля на мосту с противоположной стороны. Данное решение неудовлетворительно. Например. Утром все поехали на остров, где запланирован пикник. Первая партия машин проехала. А остальные окажутся заблокированными до вечера, когда с острова появится первый автомобиль.

Правильное решение следующее. Изменение направления движение возможно, если с противоположной стороны имеется автомобиль, стоящий на сенсоре в ожидании зеленого светофора. Реализация данного решения плохо стыкуется с проведенными уточнениями. Разработку контроллера нужно проводить другим способом.

Другое решение задачи управления на мосту

```
section St0 {
    const d: NAT
    axiom d > 0
    a: NAT := 0        // число автомобилей на мосту в любом из направлений
    b: NAT := 0
    type Colour = enum {red, green};
    mtl: Colour := red
    itl: Colour := red
    ml_out: BOOL := false  //true -  сенсор mlOut включен – на нем машина
    il_out: BOOL := false    //true -  сенсор ilOut включен – на нем машина
    empty: BOOL := false   // true - пустой мост, ждем первый автомобиль
    inv  a + b <= d
    inv mtl=red ∨ itl=red
}
```

Другое решение. Полная программа

Полная программа управления движением автомобилей на мосту:

```
process carBridge {
    LightControl || ML_out || IL_in || IL_out || ML_in
}
```

Программы для сенсоров вычисляют a и b.

Управляющие состояния:

- zero – мост пуст, оба светофора красные, и по значениям переменных ml_out и il_out реализуется соответствующая переустановка светофоров;
- right – реализуется движение слева направо, то есть с острова на материк;
- left – реализуется движение справа налево, то есть с материка на остров.

right0 и left0 - ожидание на мосту первой машины.

Дедлок: первый автомобиль появился на мосту и успел выехать с моста раньше, чем сработает сегмент кода в состояниях right0 или left0. Во избежание блокировки введен признак empty. При истинном значении empty автомобиль не может покинуть мост.

Программа LightControl

```
process LightControl {
    zero: inv a=0 & mtl = red & itl = red;
          if (il_out) {itl := green; empty:=true #right0}
          else if (ml_out & b< d) {mtl := green; empty:=true #left0}
          else #zero
    right0: inv mtl = red & itl = green;
           if (a=0) #right0 else { empty:=false #right}
    right:  inv mtl = red & itl = green;
           if (a=0) {itl:=red #zero} else #right
    left0:  inv mtl = green & itl = red;
          if (a=0) #left0 else { empty:=false #left }
    left:   inv mtl = green & itl = red;
          if (a+b=d) {mtl := red; #leftscan}
          else if (a=0) {mtl := red #zero }
          else #left
    leftscan: inv mtl = red & itl = red;
             if (a=0) #zero else #leftscan                    }
```

Программы управления сенсорами

```
process ML_out {
    inv a+b<=d
    off: ml_out:=true #on
    on: mtl=green & a+b<d → a:=a+1; ml_out:=false #off
}
process IL_in {
    inv a+b<=d
    off: a>0 & not empty &  itl = red → #on
    on: a:=a-1; b:=b+1 #off
}
process ML_in {
    off: a>0 & not empty → #on
    on: a:=a-1 #off
}
process IL_out {
    off: b>0 → il_out:=true #on
    on: itl=green → b:=b-1; a=a+1; il_out:=false #off
}
```

Обнаруженные ошибки

1.Сначала была обнаружен дедлок в состояниях right0 и left0, когда первый автомобиль на мосту выезжает с него раньше срабатывания сегмента в состояниях right0 или left0. Был введен признак empty для пустого моста.

2. Следствием исправления данной ошибки стал дедлок в состоянии zero. Для исправления второй ошибки введено дополнительное охранное условие: b < d.

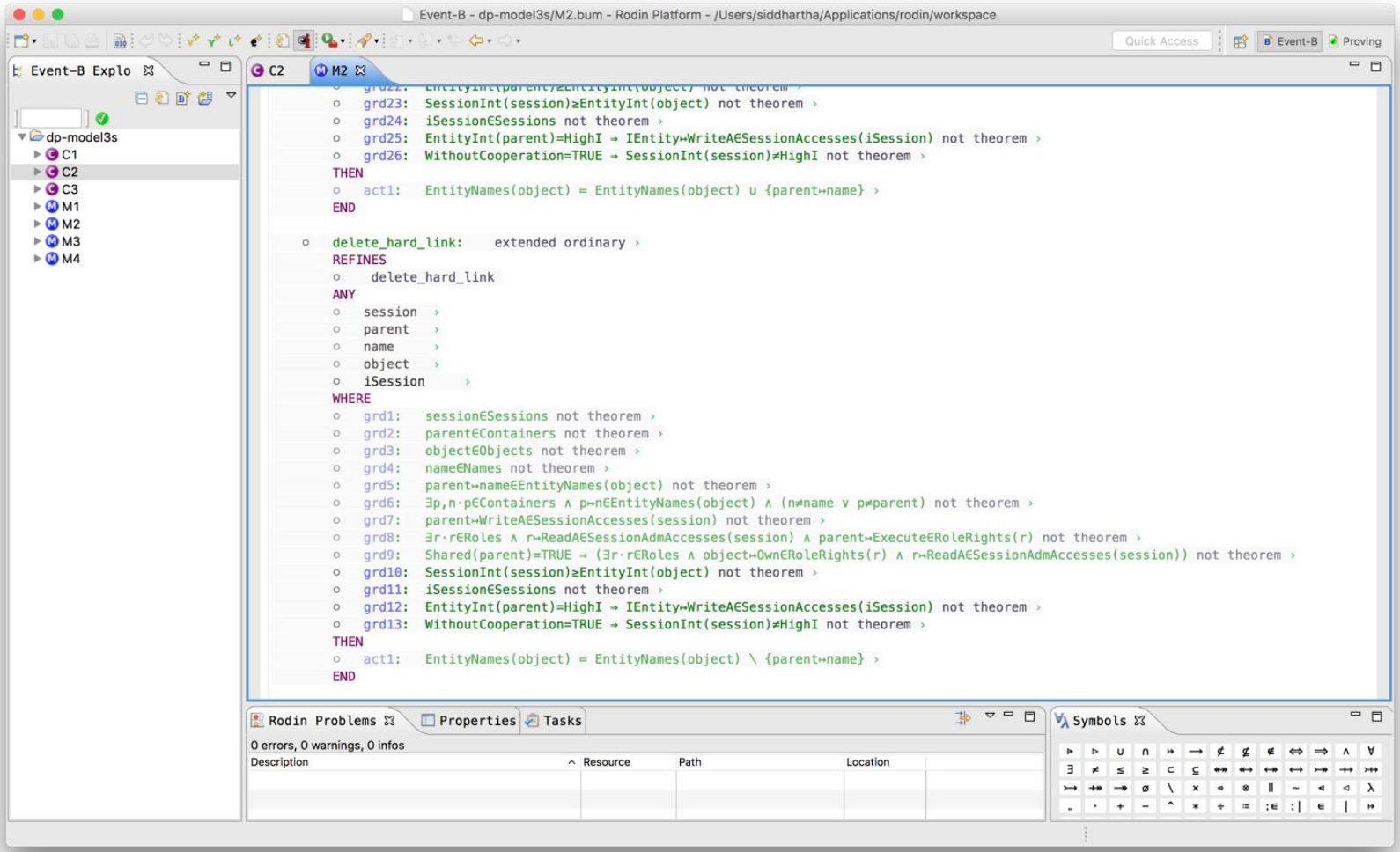
3. Третья ошибка была обнаружена на одном из вариантов анимации. После въезда на мост первого автомобиля с острова этот автомобиль неожиданно возвращался на остров. Здесь оказалось возможным срабатывание сегмента off процесса IL_in. Для исправления ошибки введено дополнительное охранное условие itl = red.

Rodin

Rodin – платформа для разработки и верификации спецификаций на Event-B

Платформа Rodin содержит:

- Текстовый редактор спецификаций на Event-B
- Набор систем автоматического доказательства
- Поддержку интерактивного доказательства


Event-B - dp-model3s/M2.bum - Rodin Platform - /Users/siddhartha/Applications/rodin/workspace
Quick Access
Event-B
Proving
Event-B Explo
dp-model3s
C1
C2
C3
M1
M2
M3
M4
C2
M2
grd23: SessionInt(session)≥EntityInt(object) not theorem ›
grd24: iSession∈Sessions not theorem ›
grd25: EntityInt(parent)=HighI ⇒ IEntity↔WriteA∈SessionAccesses(iSession) not theorem ›
grd26: WithoutCooperation=TRUE ⇒ SessionInt(session)≠HighI not theorem ›
THEN
act1: EntityNames(object) = EntityNames(object) ∪ {parent↦name} ›
END
delete_hard_link: extended ordinary ›
REFINES
delete_hard_link
ANY
session ›
parent ›
name ›
object ›
iSession ›
WHERE
grd1: session∈Sessions not theorem ›
grd2: parent∈Containers not theorem ›
grd3: object∈Objects not theorem ›
grd4: name∈Names not theorem ›
grd5: parent↦name∈EntityNames(object) not theorem ›
grd6: ∃p,n·p∈Containers ∧ p↦n∈EntityNames(object) ∧ (n≠name ∨ p≠parent) not theorem ›
grd7: parent↦WriteA∈SessionAccesses(session) not theorem ›
grd8: ∃r·r∈Roles ∧ r↦ReadA∈SessionAdmAccesses(session) ∧ parent↦ExecuteE∈RoleRights(r) not theorem ›
grd9: Shared(parent)=TRUE ⇒ (∃r·r∈Roles ∧ object↦Own∈RoleRights(r) ∧ r↦ReadA∈SessionAdmAccesses(session)) not theorem ›
grd10: SessionInt(session)≥EntityInt(object) not theorem ›
grd11: iSession∈Sessions not theorem ›
grd12: EntityInt(parent)=HighI ⇒ IEntity↔WriteA∈SessionAccesses(iSession) not theorem ›
grd13: WithoutCooperation=TRUE ⇒ SessionInt(session)≠HighI not theorem ›
THEN
act1: EntityNames(object) = EntityNames(object) \ {parent↦name} ›
END
Rodin Problems
Properties
Tasks
0 errors, 0 warnings, 0 infos
Description
Resource
Path
Location
Symbols

Полезные ссылки

- Основной сайт
  http://www.event-b.org/
- Туториал по платформе Rodin
  https://www3.hhu.de/stups/handbook/rodin/
- Полное описание языка Event-B (4 страницы)
  http://wiki.event-b.org/images/EventB-Summary.pdf

- Книга J.-R. Abrial. Modeling in Event-B. System and Software Engineering