Addendum to the MoDELS'15 submission entitled

Extracting Frame Conditions from Operation Contracts

by Philipp Niemann, Frank Hilken, Martin Gogolla, and Robert Wille
Department of Computer Science, University of Bremen, 28359 Bremen, Germany
e-mail: {pniemann,fhilken,gogolla,rwille}@informatik.uni-bremen.de

This document contains detailed descriptions of the models that were used for the evaluation of the proposed approach. The models are presented using the syntax of the *UML Specification Environment* (USE, Gogolla et al., "USE: A UML-based specification environment for validating UML and OCL", *Sci. Comput. Program.*, vol. 69, no. 1-3, pp. 27-34, 2007).

The provisional classifications of properties, which are applied during the analysis of the postconditions (Algorithm 1), are highlighted directly within the postconditions using the following color scheme:

```
property classified variable
property classified ambiguous
property classified unaffected
```

Moreover, the output of our prototype tool, i.e. the hypotheses extracted from postconditions and invariants, are listed below the corresponding pre-/postconditions using the following notation:

```
-- PROPERTY: SCOPE-TERM - CLASSIFICATION; REMARKS
```

Model 1: Smartphone

```
model Phone

class Phone
attributes
      hasNewMessages : Boolean
      credit : Integer
operations
      topup (amount : Integer)
end

class Speaker
attributes
      volume : Integer
      file : Integer
operations
      setVolume (volume : Integer )
end

class Microphone
attributes
      enabled : Boolean
end

class CallingApp
attributes
      inCall : Boolean
operations
      placeCall(number : Integer)
      talk()
      closeCall()
end

class MessagingApp
attributes
      enabled : Boolean
operations
      sendMessage(msg:Message)
      deleteReadMessages()
      emptyTrash()
end

class Message
attributes
      read : Boolean
end

class MusicApp
attributes
      songs : Sequence(Integer)
      currentSong : Integer
operations
      playNextSong()
end
```

```
association SpeakerPhone
between
      Speaker [1] role speaker
      Phone [0..1] role phone
end

association MicrophonePhone
between
      Microphone [1] role microphone
      Phone [0..1] role phone
end

association CallingAppPhone
between
      CallingApp [0..1] role callingapp
      Phone [1] role phone
end

association MessagingAppPhone
between
      MessagingApp [0..1] role messagingapp
      Phone [1] role phone
end

association MusicAppPhone
between
      MusicApp [0..1] role Musicapp
      Phone [1] role phone
end

association MessageInbox
between
      Message [*] role inbox
      MessagingApp [0..1] role inInbox
end

association MessageTrash
between
      Message [*] role trash
      MessagingApp [0..1] role inTrash
end

constraints

context Microphone
  inv microOnWhenInCall: enabled = phone.callingapp.inCall

context Phone
  inv inboxAlert:  hasNewMessages = MessagingApp.allInstances()->exists(mApp |
                   mApp.inbox->exists(m | m.read = false))

context Message
  inv eitherInboxOrTrash: inInbox.oclIsUndefined() or inTrash.oclIsUndefined()
```

```
context Phone::topup(amount: Integer):
  pre:  amount > 0
  post: self.credit = self.credit@pre + amount
-- Phone::credit: self - variable; occurs at @pre and @post

context Speaker::setVolume(volume: Integer):
  pre:  volume >= 0 and volume <= 100
  post: self.volume = volume
-- Speaker::volume: self - variable;


context CallingApp::placeCall(number: Integer):
  pre:  not inCall and phone.credit >= 10
  post: inCall
-- CallingApp::inCall: self - variable;
-- Microphone::enabled: self.phone - variable; // from inv microOnWhenInCall

context CallingApp::talk():
  pre:  inCall and phone.credit >= 10
  post: phone.credit = phone.credit@pre - 10
-- CallingApp::phone: self - ambiguous;
-- Phone::credit: self.phone - variable; occurs at @pre and @post!

context CallingApp::closeCall():
  pre:  inCall
  post: not inCall
-- CallingApp::inCall: self - variable;
-- Microphone::enabled: self.phone - variable; // from inv microOnWhenInCall

context MusicApp::playNextSong():
pre:  currentSong < songs->size() - 1
post: currentSong = currentSong@pre + 1
post: phone.speaker.file = songs->at(currentSong)
-- Speaker::file: self.phone.speaker - variable;
-- MusicApp::songs: self - ambiguous;
-- MusicApp::currentSong: self - variable; occurs at @pre and @post!
-- Phone::speaker: self.phone - ambiguous;
-- MusicApp::phone: self - ambiguous;

context MessagingApp::sendMessage(msg: Message):
  pre: enabled and phone.credit >= 5
  post: phone.credit = phone.credit@pre - 5
-- MessagingApp::phone: self - ambiguous;
-- Phone::credit: self.phone - variable; occurs at @pre and @post!

context MessagingApp::deleteReadMessages():
post: trash = trash@pre->union(inbox@pre->select(m|m.read))
-- MessagingApp::trash: self - variable; occurs at @pre and @post!
-- MessagingApp::inbox: self - ambiguous; occurs only @pre!
-- Message::read: self.inbox@pre - unaffected;
-- Phone::hasNewMessages: Phone.allInstances() - ambiguous //from inv inboxAlert

context MessagingApp::emptyTrash():
  pre: trash->size() > 0
  post: self.trash->isEmpty() and
        Message.allInstances()->excludesAll(self.trash@pre)
-- Message::allInstances(): Message - variable;
-- MessagingApp::trash: self - variable; occurs at @pre and @post!
```

Model 2: TollCollect

```
------------------------------------------------------ model TollCollect
model TollCollect
------------------------------------------------------------ class Truck
class Truck
attributes
  num:String
  trips:Sequence(Point)
  debt:Integer
operations
  init(aNum:String)
  enter(entry:Point)
  move(target:Point)
  pay(amount:Integer)
  bye():Integer
  ------------------------------------------------------------------------
statemachines
  psm TruckLife
  states
    prenatal:initial
    noDebt [current->isEmpty]
    debt   [current->notEmpty]
  transitions
    prenatal -> noDebt { create }
    noDebt   -> debt   { enter() }
    debt     -> debt   { move() }
    debt     -> debt   { pay() }
    debt     -> noDebt { bye() }
  end
end
------------------------------------------------------------ class Point
class Point
attributes
  name:String
operations
  init(aName:String)
  northConnect(aNorth:Point)
  southConnect(aSouth:Point)
  ------------------------------------------------------------------------
statemachines
  psm PointLife
  states
    prenatal:initial
    born    [name=null]
    growing [name<>null]
  transitions
    prenatal -> born    { create }
    born     -> growing { init() }
    growing  -> growing { northConnect() }
    growing  -> growing { southConnect() }
  end
end
------------------------------------------------- association Current
association Current between
  Truck[0..*] role truck
  Point[0..1] role current
end
```

```
---------------------------------------------- association Connection
association Connection between
  Point[0..*] role north
  Point[0..*] role south
end
------------------------------------------------------------ constraints
constraints
------------------------------------------------------------ invariants
context Point
inv nameIsKey:
  Point.allInstances()->forAll(self2|self<>self2 implies self.name<>self2.name)

context Truck
inv numIsKey:
  Truck.allInstances()->forAll(self2|self<>self2 implies self.num<>self2.num)

context Point
inv noCycles:
  not self->closure(north)->includes(self)
------------------------------------------------------------ Truck::init
context Truck::init(aNum:String):
pre freshTruck:
  self.num.oclIsUndefined() and self.trips.oclIsUndefined() and
  self.debt.oclIsUndefined()
post numTripsDebtAssigned:
  aNum=self.num and 0=self.debt
  and oclEmpty(Sequence(Point))=self.trips
-- Truck::debt: self - variable;
-- Truck::num: self - variable;
-- Truck::num: Truck.allInstances() - ambiguous; // from inv numIsKey

------------------------------------------------------------ Truck::enter
context Truck::enter(entry:Point):
pre noDebt:
  0=self.debt
pre currentEmpty:
  self.current.oclIsUndefined()
post debtAssigned:
  1=self.debt
post currentAssigned:
  entry=self.current
-- Truck::point: self - variable;
-- Truck::debt: self - variable;

------------------------------------------------------------ Truck::move
context Truck::move(target:Point):
pre currentExists:
  not self.current.oclIsUndefined()
pre targetReachable:
  current.north->union(current.south)->includes(target)
post currentAssigned:
  target=self.current
post debtIncreased:
  self.debt@pre+1=self.debt
post tripsUpdated:
  self.trips@pre->including(target)=self.trips
-- Truck::current: self - variable;
-- Truck::debt: self - variable; occurs at @pre and @post!
-- Truck::trips: self - variable; occurs at @pre and @post!
```

```
------------------------------------------------------------ Truck::pay
context Truck::pay(amount:Integer):
pre amountPositive:
  amount>0
pre currentExists:
  not self.current.oclIsUndefined()
post debtReduced:
  self.debt@pre-amount=self.debt
-- Truck::debt: self - variable; occurs at @pre and @post!

------------------------------------------------------------ Truck::bye
context Truck::bye():Integer
pre currentExists:
  not self.current.oclIsUndefined()
pre noDebt: self.debt<=0
post returnEqualsOverPayment: -self.debt=result
post currentEmpty: self.current.oclIsUndefined()
-- Truck::debt: self - ambiguous; //pattern ``return-value''
-- Truck::current: self - variable;
------------------------------------------------------------ Point::init
context Point::init(aName: String):
pre freshPoint:
  north->isEmpty()  and south->isEmpty()  and  name.oclIsUndefined()
post nameAssigned:
  name=aName
-- Point::name:  self - variable;
-- Point::name: Point.allInstances() - ambiguous; // from inv nameIsKey

------------------------------------------------------ Point::northConnect
context Point::northConnect(aNorth:Point):
pre aNorthDefined:
  not aNorth.oclIsUndefined()
pre freshConnection:
  self.north->includes(aNorth)  and self.south->excludes(aNorth)
pre notSelfLink:
  self<>aNorth
post connectionAssigned: self.north->includes(aNorth)
-- Point::north: self - variable;

------------------------------------------------------ Point::southConnect
context Point::southConnect(aSouth:Point):
pre aSouthDefined:
  not aSouth.oclIsUndefined()
pre freshConnection:
  self.north->includes(aSouth)  and self.south->excludes(aSouth)
pre notSelfLink:
  self<>aSouth
post connectionAssigned: self.south->includes(aSouth)
-- Point::south: self - variable;
```

Model 3: Scheduler

```
model Scheduler

class Scheduler
operations
  Init()
    begin
      for r in self.ready do
        delete (self, r) from ReadyQueue;
      end;
      for w in self.waiting do
        delete (self, w) from WaitingQueue;
      end;
      if not self.active.oclIsUndefined() then
        delete (self, self.active) from Active;
      end;
    end
  New(p:Process)
    begin
      insert (self, p) into WaitingQueue;
    end
  Ready(p:Process)
    begin
      delete (self, p) from WaitingQueue;
      if self.active.oclIsUndefined() then
        insert (self, p) into Active;
      else
        insert (self, p) into ReadyQueue;
      end;
    end
  Swap()
    begin
      insert (self, self.active) into WaitingQueue;
      delete (self, self.active) from Active;
      if not self.ready->isEmpty() then
        declare any : Process;
        any := self.ready->any( true );
        delete (self, any) from ReadyQueue;
        insert (self, any) into Active;
      end;
    end
end

class Process
attributes
  pid : Integer
end

association WaitingQueue
between
  Scheduler [0..1] role schedWaiting
  Process [*] role waiting
end
```

```
association ReadyQueue
between
  Scheduler [0..1] role schedReady
  Process [*] role ready
end

association Active
between
  Scheduler [0..1] role schedActive
  Process [0..1] role active
end

constraints

context Scheduler
inv:
  (ready->intersection(waiting))->isEmpty() and
  not ((ready->union(waiting))->includes(active)) and
  (active.oclIsUndefined() implies ready->isEmpty())

context Scheduler::Init():
post: (ready->union(waiting))->isEmpty() and active.oclIsUndefined()
-- Scheduler::waiting: self - variable;
-- Scheduler::active: self - variable;
-- Scheduler::ready: self - variable;

context Scheduler::New(p:Process):
pre:   active <> p and not((ready->union(waiting))->includes(p))
post:  waiting = waiting@pre->including(p) and
       ready = ready@pre and active = active@pre
-- Scheduler::waiting: self - variable; occurs at @pre and @post!
-- Scheduler::active: self - unaffected;
-- Scheduler::ready: self – unaffected;

context Scheduler::Ready(p:Process):
pre:  waiting->includes(p)
post: waiting = waiting@pre->excluding(p) and
      if active@pre.oclIsUndefined() then
         (ready = ready@pre and active = p)
      else
       (ready = ready@pre->including(p) and active = active@pre)
      endif
-- Scheduler::waiting: self - variable; occurs at @pre and @post!
-- Scheduler::active: self - variable;
-- Scheduler::ready: self - variable;


context Scheduler::Swap():
pre: not active.oclIsUndefined()
post: if ready@pre->isEmpty() then
      (active.oclIsUndefined() and ready->isEmpty())
      else (
        ready@pre->includes(active) and
        ready = ready@pre->excluding(active))
      endif
      and waiting = waiting@pre->including(active@pre)
-- Scheduler::waiting: self - variable; occurs at @pre and @post!
-- Scheduler::active: self - ambiguous; occurs at @pre and @post!
-- Scheduler::active:  self [if not ready@pre->isEmpty()] - variable;
-- Scheduler::ready: self - variable;
```

Model 4: Library

```
------------------------------------------------------------- Library
model Library
------------------------------------------------------------ class User
class User
attributes
  name:String
  address:String
operations
  init(aName:String, anAddress:String)
    begin
    self.name:=aName;    self.address:=anAddress;
    end
  borrow(aCopy:Copy)
    begin
    insert (self,aCopy) into Borrows;
    end
  return(aCopy:Copy)
    begin
    aCopy.numReturns:=aCopy.numReturns+1;
    delete (self,aCopy) from Borrows;
    end
end
------------------------------------------------------------ class Copy
class Copy
attributes
  signature:String
  numReturns:Integer
operations
  init(aSignature:String, aBook:Book)
    begin
    self.signature:=aSignature;
    self.numReturns:=0;
    insert (self,aBook) into BelongsTo;
    end
  borrow(aUser:User)
    begin
    insert (aUser,self) into Borrows;
    end
  return()
    begin
    self.numReturns:=self.numReturns+1;
    delete (self.user,self) from Borrows;
    end
end
------------------------------------------------------------ class Book
class Book
attributes
  title:String
  authSeq:Sequence(String)
  year:Integer
operations
  init(aTitle:String, anAuthSeq:Sequence(String), aYear:Integer)
    begin
    self.title:=aTitle;    self.authSeq:=anAuthSeq;    self.year:=aYear;
    end
end
```

```
------------------------------------------------- association Borrows
association Borrows between
  User[0..1] role user
  Copy[0..*] role copy
end
-- - - - - - - - - - - - - - - - - - - - - - - - - association BelongsTo
association BelongsTo between
  Copy[0..*] role copy
  Book[1] role book
end
------------------------------------------------------------ constraints
constraints
-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  User
context User
inv nameAddressFormatOk:
  name<>oclUndefined(String) and name<>'' and
  address<>oclUndefined(String) and address<>''
inv nameIsKey:
  User.allInstances()->forAll(u2 | self<>u2 implies self.name<>u2.name)
inv noDoubleBorrowings:
  not(self.copy->exists(c1,c2|c1<>c2 and c1.book=c2.book))
-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  Copy
context Copy
inv signatureFormatOk:
  self.signature<>oclUndefined(String) and self.signature<>''
inv signatureIsKey:
  Copy.allInstances()->forAll(c2 |
      self<>c2 implies self.signature<>c2.signature)
-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  Book
context Book
inv titleFormatOk:
  self.title<>oclUndefined(String) and self.title<>''
inv titleIsKey:
  Book.allInstances()->forAll(b2 | self<>b2 implies self.title<>b2.title)
inv authSeqFormatOk:
  Set{1..self.authSeq->size()}->forAll(i|
        authSeq->at(i)<>oclUndefined(String) and authSeq->at(i)<>'')
inv authSeqExistsAndUnique: self.authSeq->size()>0 and
  Set{1..self.authSeq->size()-1}->forAll(i|
    Set{i+1..self.authSeq->size()}->forAll(j|
      authSeq->at(i)<>authSeq->at(j)))
inv yearPlausible:
  1455<=year
------------------------------------------------------------ User::init
context User::init(aName: String, anAddress: String):
pre freshUser:
  self.name.oclIsUndefined() and
  self.address.oclIsUndefined() and self.copy->isEmpty()
post attrsAssigned:
  aName=self.name and anAddress=self.address
-- User::address: self - variable;
-- User::name: self - variable;
-- User::name: User.allInstances() - ambiguous; // from inv nameIsKey
-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  User::borrow
context User::borrow(aCopy:Copy):
pre copyOk:
  aCopy.user->isEmpty()
post linkAssigned:
  self.copy@pre->including(aCopy)=self.copy
-- User::copy: self - variable; occurs at @pre and @post!
```

```
-- - - - - - - - - - - - - - - - - - - - - - - - - - - - -  User::return
context User::return(aCopy:Copy):
pre aCopyOk:
  self.copy->includes(aCopy)
post linkRemoved:
  self.copy@pre->excluding(aCopy)=self.copy
post numReturnsIncreased:
  aCopy.numReturns@pre+1=aCopy.numReturns
-- User::copy: self - variable; occurs at @pre and @post!
-- Copy::numReturns: aCopy - variable; occurs at @pre and @post!
------------------------------------------------------------ Copy::init
context Copy::init(aSignature: String, aBook:Book):
pre freshCopy:
  self.signature.oclIsUndefined() and
  self.numReturns.oclIsUndefined() and
  self.user->isEmpty() and self.book->isEmpty()
pre bookOk:
  aBook<>oclUndefined(Book)
post attrsAndLinkAssigned:
  aSignature=self.signature and 0=self.numReturns and
  aBook=self.book
-- Copy::signature: self - variable;
-- Copy::book: self - variable;
-- Copy::numReturns: self - variable;
-- Copy::signature: Copy.allInstances() - ambiguous; //from inv signatureIsKey
-- - - - - - - - - - - - - - - - - - - - - - - - - - - - -  Copy::borrow
context Copy::borrow(aUser:User):
pre userOk:
  aUser<>oclUndefined(User)
pre notBorrowed:
  self.user->isEmpty()
post linkAssigned:
  aUser=self.user
-- Copy::user: self - variable;
-- - - - - - - - - - - - - - - - - - - - - - - - - - - - -  Copy::return
context Copy::return():
pre copyOk:
  not self.user.oclIsUndefined()
post linkRemoved:
  self.user.oclIsUndefined()
post numReturnsIncreased:
  self.numReturns@pre+1=self.numReturns
-- Copy::user: self - variable;
-- Copy::numReturns: self - variable; occurs at @pre and @post!
------------------------------------------------------------ Book::init
context Book::
init(aTitle: String, anAuthSeq: Sequence(String), aYear:Integer):
pre freshBook:
  self.title.oclIsUndefined() and
  self.authSeq.oclIsUndefined() and
  self.year.oclIsUndefined() and
  self.copy->isEmpty()
post attrsAssigned:
  aTitle=self.title and anAuthSeq=self.authSeq and aYear=self.year
-- Book::title: self - variable;
-- Book::authSeq: self - variable;
-- Book::year: self - variable;
-- Book::title: Book.allInstances() - ambiguous; //from inv titleIsKey
```

Model 5: Civil Status

```
model CivilStatusWorld

enum CivilStatus {single, married, divorced, widowed}
enum Gender {female, male}

class Person
attributes
  name:String
  civstat:CivilStatus
  gender:Gender
  alive:Boolean
operations
  birth(aName:String, aGender:Gender)
  marry(aSpouse:Person)
  divorce()
  death()
end

association Marriage between
  Person [0..1] role wife
  Person [0..1] role husband
end



constraints



context Person
inv attributesDefined:
  not name.oclIsUndefined() and not civstat.oclIsUndefined() and
  not gender.oclIsUndefined() and alive.oclIsUndefined()
inv nameIsUnique: Person.allInstances()->forAll(self2|
      self<>self2 implies self.name<>self2.name)
inv femaleHasNoWife: gender=#female implies wife.oclIsUndefined()
inv maleHasNoHusband: gender=#male implies husband.oclIsUndefined()



context Person::birth(aName: String, aGender: Gender):
pre freshUnlinkedPerson:
  name.oclIsUndefined() and civstat.oclIsUndefined() and
  gender.oclIsUndefined() and alive.oclIsUndefined() and
  wife.oclIsUndefined() and husband.oclIsUndefined()
post nameAssigned: name=aName
post civstatAssigned: civstat=#single
post genderAssigned: gender=aGender
post isAliveAssigned: alive=true
-- Person::civstat: self - variable;
-- Person::gender: self - variable;
-- Person::name: self - variable;
-- Person::alive: self - variable;
-- Person::name: Person.allInstances() - ambiguous; //from inv nameIsUnique
```

```
context Person::marry(aSpouse: Person):
pre  aSpouseDefined: not aSpouse.oclIsUndefined()
pre  isAlive: alive
pre  aSpouseAlive: aSpouse.alive
pre  isUnmarried: civstat<>#married
pre  aSpouseUnmarried: aSpouse.civstat<>#married
pre  differentGenders: gender<>aSpouse.gender
post isMarried: civstat=#married
post femaleHasMarriedHusband: gender=#female implies
     (husband=aSpouse and husband.civstat=#married)
post maleHasMarriedWife: gender=#male implies
     (wife=aSpouse and wife.civstat=#married)
-- Person::civstat: self.wife - ambiguous;
-- Person::civstat: self.husband - ambiguous;
-- Person::civstat: self - variable;
-- Person::husband: self - ambiguous;
-- Person::wife: self - ambiguous;
-- Person::gender: self - ambiguous;


context Person::divorce():
pre  isMarried: civstat=#married
pre  isAlive: alive
pre  husbandAlive: gender=#female implies husband.alive
pre  wifeAlive: gender=#male implies wife.alive
post isDivorced: civstat=#divorced
post husbandDivorced: gender=#female implies
     (husband.oclIsUndefined() and husband@pre.civstat=#divorced)
post wifeDivorced: gender=#male implies
     (wife.oclIsUndefined() and wife@pre.civstat=#divorced)
-- Person::civstat: self.wife@pre - ambiguous;
-- Person::civstat: self.husband@pre - ambiguous;
-- Person::civstat: self - variable;
-- Person::husband: self - ambiguous; occurs at @pre and @post!
-- Person::wife: self - ambiguous; occurs at @pre and @post!
-- Person::gender: self - ambiguous;



context Person::death():
pre  isAlive: alive
post notAlive: not(alive)
post husbandWidowed:
  gender=#female and (not husband@pre.oclIsUndefined()) implies
    ((not husband@pre.wife.oclIsUndefined()) and husband@pre.civstat=#widowed)
post wifeWidowed:
  gender=#male and (not wife@pre.oclIsUndefined()) implies
    ((not wife@pre.husband.oclIsUndefined()) and wife@pre.civstat=#widowed)
-- Person::civstat: self.wife@pre - ambiguous;
-- Person::civstat: self.husband@pre - ambiguous;
-- Person::husband: self.wife@pre - ambiguous;
-- Person::husband: self - ambiguous; occurs only @pre!
-- Person::wife: self.husband@pre - ambiguous;
-- Person::wife: self - ambiguous; occurs only @pre!
-- Person::gender: self - ambiguous;
-- Person::alive: self - variable;
```

Model 6: TrafficLight

```
model TrafficLight

class TrafficLight
attributes
  r:Boolean --    +    +    -    -    R    R    r    r
  y:Boolean --    - ~> + ~> - ~> +    y ~> Y ~> y ~> Y
  g:Boolean --    -    -    +    -    g    g    G    g
operations
  init()
  switch()
end

constraints

context TrafficLight
inv Ryg_RYg_ryG_rYg:
  (r.oclIsUndefined() and y.oclIsUndefined() and g.oclIsUndefined()) or
  (r and not y and not g) or (r and y and not g) or
  (not r and not y and g) or (not r and y and not g)

context TrafficLight::init():
pre:  r.oclIsUndefined() and y.oclIsUndefined() and g.oclIsUndefined()
post: (r and not y and not g) or (r and y and not g) or
      (not r and not y and g) or (not r and y and not g)
-- TrafficLight::g: self - variable;
-- TrafficLight::r: self - variable;
-- TrafficLight::y: self - variable;

context TrafficLight::switch():
post Ryg_2_RYg_2_ryG_2_rYg_2_Ryg:
  (not r.oclIsUndefined() and not y.oclIsUndefined()) and
  (r@pre and not y@pre implies r and y) and
  (r@pre and y@pre implies not r and not y) and
  (not r@pre and not y@pre implies not r and y) and
  (not r@pre and y@pre implies r and not y)
    -- attribute g not mentioned, but determined by invariant
-- TrafficLight::r: self - variable; occurs at @pre and @post!
-- TrafficLight::y: self - variable; occurs at @pre and @post!
-- TrafficLight::g: self - ambiguous; //from inv Ryg_RYg_ryG_rYg
```

Model 7: Airport_CML

```
model Airport

class Aircraft
end

class Airport
attributes
  permission:Set(Aircraft)
  landed:Set(Aircraft)
operations
  permitted(a:Aircraft,perm:Set(Aircraft)):Boolean=perm->includes(a)
  down(a:Aircraft,land:Set(Aircraft)):Boolean=land->includes(a)
  PreRecordLanding(a:Aircraft,p:Set(Aircraft),l:Set(Aircraft)):Boolean=
    permitted(a,p) and not down(a,l)
  NumberWaiting():Integer=(permission-landed)->size
  ----------------------------------------------------------------------
  Init()
  begin self.permission:=Set{}; self.landed:=Set{} end
  -- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  GivePermission(a:Aircraft)
  begin self.permission:=self.permission->including(a) end

  -- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  RecordLanding(a:Aircraft)
  begin self.landed:=self.landed->including(a) end
  -- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  RecordTakeOff(a:Aircraft)
  begin self.landed:=self.landed->excluding(a);
  self.permission:=self.permission->excluding(a) end
end


constraints

context Airport
inv landedSubsetPermission: permission->includesAll(landed)


context Airport::Init():
post permissionEmptyLandedEmpty:  Set{}=permission and Set{}=landed
-- Airport::permission: self - variable;
-- Airport::landed: self - variable;


context Airport::GivePermission(a:Aircraft):
pre notPermitted: not permitted(a,permission)
post addedPermission: permission@pre->union(Set{a})=permission
-- Airport::permission: self - variable;
-- Airport::landed: self - ambiguous; // from inv landedSubsetPermission

context Airport::RecordLanding(a:Aircraft):
pre permittedNotDown: PreRecordLanding(a,permission,landed)
post addedLanded: landed@pre->union(Set{a})=landed
-- Airport::landed: self - variable;
-- Airport::permission: self - ambiguous; // from inv landedSubsetPermission
```

```
context Airport::RecordTakeOff(a:Aircraft):
pre hasLanded: landed->includes(a)
post notLandedNotPermitted:
  landed@pre-Set{a}=landed and permission@pre-Set{a}=permission
-- Airport::permission: self - variable;
-- Airport::landed: self - variable;
```