Formalizing Hybrid Systems with Event-B

Jean-Raymond Abrial[1], Wen Su[2], and Huibiao Zhu[2]

[1] Marseille, France
jrabrial@neuf.fr
[2] Software Engineering Institute, East China Normal University
{wensu,hbzhu}@sei.ecnu.edu.cn

Abstract. This paper[1] contains the development of hybrid systems in Event-B and the Rodin Platform[2]. It follows the seminal approach introduced at the turn of the century in Action Systems. Many examples illustrate our approach.

1 Introduction

Hybrid systems have been studied for many years ([6] and many more). They are very important in the development of embedded systems where a piece of software, *the controller*, is supposed to manage an external situation, *the environment*. The controller works in a discrete fashion in that it is triggered regularly by detecting the status of the environment (using some *sensors*), and then reacts by sending some information to the environment (using some *actuators*). Between two successive controller detections and actions, the environment is evolving in *a continuous way*.

The formal development of such embedded systems has then to take account of two different frameworks: the discrete framework of the controller and the continuous framework of the environment. The formal development of such *closed systems* must be able to deal with these dual frameworks: this is the purpose of hybrid system.

In this paper, we explain how such systems can be developed in Event-B [7] and the Rodin Platform [8]. The paper is organized as follows: in the next section we explain how our approach follows that developed in Action Systems [1]. Section 3 contains many examples and then we conclude.

2 Approaches

2.1 The Approach of Action System to Hybrid Systems

Background. Action System [1] has been introduced in the eighties by R.J. Back and R. Kurki-Suonio. It has been then further developed by the Finnish

[1] This work is supported in part by National Basic Research Program of China (No. 2011CB302904), National High Technology Research and Development Program of China (No. 2011AA010101 and No. 2012AA011205), National Natural Science Foundation of China (No. 61061130541 and No. 61021004).

[2] An extended version of this paper can be found in [9].

J. Derrick et al. (Eds.): ABZ 2012, LNCS 7316, pp. 178–193, 2012.

School of Formal Methods. Event-B [7] [8] is a direct follower of Action System: many concepts in Event-B have been borrowed from it. As a consequence, before attempting to formalize hybrid systems in Event-B, it seems appropriate to investigate what has been done concerning hybrid systems with Action System.

Continuous Action System. In 2000, R.J. Back and colleagues extended Action System in order to support the definition and proofs of Hybrid Systems [2] [3]. They called this extension *continuous* Action System. In this paper, for the sake of clarity, we shall name the non-continuous Action System the *classical* Action System. This extension is very simple and systematic: "a continuous action system is just a non-deterministic way of defining a collection of time dependent functions". In other words, rather than having the state of the Action System being defined by a collection of *time independent* variables ranging over some sets (as is the case in classical Action System), the state of a continuous Action System is now defined as a collection of *time dependent functions* (where time ranges over the set of non-negative reals $\mathbb{R}^+$). A continuous Action System can be related to a corresponding classical Action System as follows: if $x \in S$ is a variable in a classical Action System, then $x\_c \in \mathbb{R}^+ \rightarrow S$ is the "same" variable in the continuous Action System.

Past, Present, and Future. A "technical" variable, named *now*, ranging over $\mathbb{R}^+$ stands for the "present" time. Initially, *now* is supposed to be equal to 0. The time function x_c in a continuous Action System and its relationship to the corresponding variable x in a classical Action System is to be understood as follows:

1. The time function x_c restricted to the set $\{u \,|\, u \in \mathbb{R}^+ \wedge u < now\}$ denotes the *past* of the variable x.
2. The value $x\_c(now)$ denotes the present value of x (at time $t = now$) .
3. The time function x_c restricted to the set $\{u \,|\, u \in \mathbb{R}^+ \wedge u > now\}$ denotes the *future* of x. Of course, we are not sure about this future, it only denotes what we can expect "now" about it.

Discrete Events. As for classical Action Systems, a continuous Action System contains a finite number of *guarded actions*. In each of them, time functions such as x_c can be modified. These modifications however must obey a systematic constraint: the *past cannot be modified*, only the present and future can. Once an action has been "executed", then the variable *now* is updated: this is done by incrementing it to the *smallest value* making at least one action guard becoming true[3]. In between the present *now* and the future one assigned to it, a time function such as x_c is supposed to make progress following the expected future.

Continuous Action Systems as Hybrid Systems. As can be seen, a continuous Action System is indeed a genuine hybrid system: the events correspond to discrete actions situated in the middle of continuous behaviors. Typically, the continuous evolution corresponds to what happens in the external *environment*

[3] If there is no such events then *now* is not updated meaning that the systems evolves *for ever* as prescribed by the future of each time function.

of a system, whereas discrete actions correspond to what a *controller* can do in order to manage the environment. For example, the continuous evolution could be that of a physical train running at a certain speed and a certain acceleration (positive, equal to 0, or negative), whereas the discrete actions are those of the driver (human or automatic) changing the acceleration of the train from time to time depending on the actual speed, the actual acceleration and the actual distance of the train to a necessary stop (at a station or because another train is close to it).

Invariants. As for classical Action Systems, continuous ones must preserve a number of *invariants* to be proved on the past of each time variable. For example, in the train system mentioned above, we might prove that the *past* speed of a train is never greater than a certain maximum speed. We might also prove that no train can hit another one in front of it or not stop at a given station where it should. Such invariants can be stated as follows:

$$\forall t \cdot t < now \;\Rightarrow\; \mathrm{P}(x\_c(t))$$

where P is a predicate denoting the invariant property we want to prove. In order to prove the maintenance of this invariant when *now* is updated to a new value, say *new_now* (greater than *now*), what is to be proved is the following:

$$\forall t \cdot t \geq now \wedge t < new\_now \;\Rightarrow\; \mathrm{P}(x\_c(t))$$

Notice that we do not have to prove the property for $t < now$ since, by definition, the *past is not modified* when updating a time function.

2.2 The Proposed Approach with Event-B

Background. The approach we shall follow with Event-B is very close to that proposed by R.J. Back for continuous Action Systems. However, in studying examples described in [2] and [3], we found a number of difficulties: we had the *subjective feeling* that some proofs of simple invariant properties are more complicated than they should be. In what follows, we propose some simplifications to the original proposals made in continuous Action Systems.

Discrete Variables together with Continuous Variables. In [2] and [3] all variables (except *now*) are time functions. But sometimes these time functions are always constant functions on all considered intervals (different constants however for different intervals). An obvious simplification is to consider that such variables can be better represented as discrete variables as in classical Action Systems. By doing so, we could simplify some of the proofs.

Discrete Systems as an Abstraction of Continuous Ones. We also figured out that the presented approaches for continuous Action System did not take advantage of any refinement steps to be done during the development although this was mentioned in the conclusion as future work in [2].

The main initial steps we propose here before introducing continuous variables is based on our belief that *a discrete system is an abstraction of a continuous* *one.* This is a direct consequence of the observation of what is done traditionally in mathematics since the Greeks (and probably before them): in order to measure the surface of a field you cut it into different rectangles (whose surface is easily determined) and then you *refine* this process by introducing more rectangles incorporating some parts of the field that has not been taken into account previously. This operation is then repeated many times until the remaining part of the field that has not been taken into account becomes *very small.*

In the seventeenth century Newton and Leibnitz formalized this by introducing the Calculus. Later, in the nineteenth century, a considerable effort (Cauchy, Weierstrass) has been done to make this mathematical approach completely rigorous.

Refining a Discrete Systems into a Continuous Ones. When formalizing an hybrid system with Event-B, we can start the development by considering *time independent* variables only, such as x, with the basic invariant $x \in S$. This is done together with some events modifying such discrete variables. This process can be done with different refinement steps, thus making the discrete system more precise.

At some point (when the discrete system is rich enough) we can start introducing some continuous *time dependent* variables corresponding to some discrete ones. We also introduce the variable *now*. For example, the variable x is refined (and removed) to the variable $x\_c \in \mathbb{R}^+ \mathrel{+\!\!\!\rightarrow} S$. The gluing invariant between x and x_c is clearly the following: $x = x\_c(now)$. The modification of the variable x_c in an event obeys the following pattern:

$$x\_c := \lambda\, t \cdot t \in now \,..\, new\_now \,|\, \mathrm{E}(t)$$

where the value new_now corresponds to the modification of the variable now which is updated together with x_c (that is, $now := new\_now$). As can be seen, we depart here from what was done in continuous Action System. More precisely, we update the time dependent variable x_c to the new continuous value it takes within the time interval $now \,..\, new\_now$ where no discrete action takes place. Notice that new_now is in fact equal to $min(\{t \mid t \geq now \,\wedge\, \mathrm{P}(t)\})$ where P is a predicate corresponding to the disjunction of the guards of the events within which now is replaced by t.

The transformation of the discrete system into a continuous one can be done gradually: we can have intermediate steps where some discrete variables are not yet transformed into continuous ones. Also, as stated above, some discrete variables will so remain because they are constant in all intervals.

3 Examples

3.1 The Saw [2]

Our first example is extracted from [2]. It is a very simple introductory example. The initial state is made of a single discrete variable x taking only two values: 0 and 1.

```
variables:   $x$
```

```
inv0_1:   $x \in \{0, 1\}$
```

The events UP and DN alternatively change the value of x, initialized to 0:

```
INIT
  begin
    $x := 0$
  end
```

```
UP
  when
    $x = 0$
  then
    $x := 1$
  end
```

```
DN
  when
    $x = 1$
  then
    $x := 0$
  end
```

In the refinement, we replace the variable x by a time function x_c as follows (notice the gluing invariant **inv1_3**).The variable x_c is initialized to the constant function $\{0 \mapsto 0\}$, while the variable *now* is initialized to 0 (the beginning of time):

```
variables:   $x\_c$
             $now$
```

```
inv1_1:   $x\_c \in \mathbb{R}^+ \mathrel{+\!\!\!\!\to} \mathbb{R}$

inv1_2:   $now \in \mathrm{dom}(x\_c)$

inv1_3:   $x = x\_c(now)$
```

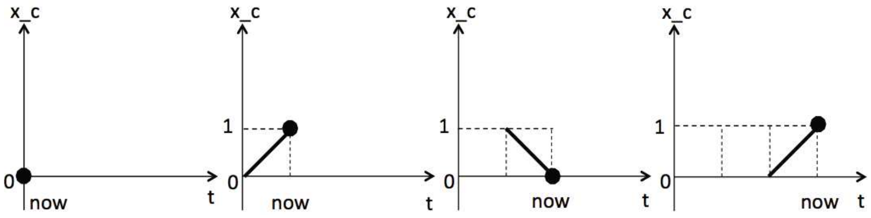
```
INIT
  begin
    $x\_c := \{0 \mapsto 0\}$
    $now := 0$
  end
```

The events UP and DN are refined as follows (notice the updating of the variable *now*):

```
UP
  when
    $x\_c(now) = 0$
  then
    $x\_c :=$
    $\lambda t \cdot t \in now \,..\, now + 1 \mid t - now$
    $now := now + 1$
  end
```

```
DN
  when
    $x\_c(now) = 1$
  then
    $x\_c :=$
    $\lambda t \cdot t \in now \,..\, now + 1 \mid 1 - t + now$
    $now := now + 1$
  end
```

The following figures show the evolution of the variable x_c initially and after various executions of the events UP and DN:



We can add several other invariants. For example we might be interested to prove that the range of the variable x_c is included in the real interval $0 \,..\, 1$:

inv1_4: $\mathrm{ran}(x\_c) \subseteq 0\,..\,1$

The global proof effort for the Rodin Platform [8] on this example is 17 proof obligations, all proved automatically.

3.2 Nuclear Plant Cooling [3]

The nuclear plant cooling example is taken from [3]. Here is an informal description of the problem quoted from [3]:

"The hybrid system is a temperature control system for a heat producing reactor, described by the temperature as a function of time $\theta(t)$. The reactor starts from the initial temperature θ_0 and heats up at a given rate v_r. Whenever it reaches the critical temperature θ_M, it is designed to be cooled down by inserting into the core either of two rods (rod1 or rod2), modeled by the variables $x_1(t)$ and $x_2(t)$, which are in fact clocks measuring the time elapsed between two consecutive insertions of the same rod, respectively. The cooling proceeds at the rate v_1 and v_2 depending on which rod is being used, and the cooling stops when the reactor reaches a given minimum temperature θ_m, by releasing the respective inserted rod. The rod used for cooling is then unavailable for a prescribed time T, after which it is again available for cooling. The object of the modeling is to ascertain that the reactor never reaches the critical temperature θ_M without at least one of the rods available."

In the development done in [3] the main safety proof mentioned at the end of the previous informal description (i.e. "the reactor never reaches the critical temperature θ_M without at least one of the rods available") is done directly on the time dependent variables and results in a rather heavy proof. As for the previous example, we start the Event-B development with a discrete system. We do the safety proof at this level: this results in a simple proof as expected. We then refine the system in several steps to a genuine hybrid system. We first start by defining a number of constants as introduced in the previous informal explanation: θ_m, θ_M, v_1, v_2, v_r, and T. We then define the heating time, a, needed to raise the temperature from θ_m to θ_M in the reactor and also the cooling times, b_1 and b_2, needed to decrease the temperature from θ_M to θ_m with rod1 or rod2. We suppose the following constraints on a, $b1$, $b2$, and T:

constants: a
b_1
b_2

axm0_1: $a v_r = \theta_M - \theta_m$
axm0_2: $b_1 v_1 = \theta_M - \theta_m$
axm0_3: $b_2 v_2 = \theta_M - \theta_m$

axm0_4: $2a + b_1 \geq T$
axm0_5: $2a + b_2 \geq T$
axm0_6: $a < T$

Notice axiom **axm0_6**: should it be $a \geq T$ then the cooling with a rod would always be possible because then the time, a, of temperature increasing in the reactor would be greater than or equal to the the time, T, after which a rod would be made available after being used. Axioms **axm0_4** and **axm0_5** seem a bit strange: they can be discovered as sufficient conditions while doing the proofs.

The initial model is a discrete one. We define some state variables: θ is the temperature of the reactor, $t1$ and $t2$ denote the time elapsed since rod1 or rod2 have been released. The system works with a *phase* variable (with values 0, 1, and 2). When *phase* is 0, it means that the reactor has reached the maximum temperature θ_M (invariant **inv0_5**). When *phase* is 1 or 2, it means that the reactor has reached the minimum temperature θ_m by being cooled either with rod1 or with rod2 (invariant **inv0_6**):

```
variables:  θ
            t_1
            t_2
            phase
```

inv0_1: $\theta \in \mathbb{R}^+$
inv0_2: $t_1 \in \mathbb{R}^+$
inv0_3: $t_2 \in \mathbb{R}^+$
inv0_4: $phase \in \{0, 1, 2\}$
inv0_5: $phase = 0 \;\Rightarrow\; \theta = \theta_M$
inv0_6: $phase \in \{1, 2\} \;\Rightarrow\; \theta = \theta_m$

The main safety invariant is the following. It states that there is always one rod available when the reactor's temperature reaches the maximum temperature θ_M:

inv0_7: $phase = 0 \;\Rightarrow\; t_1 \geq T \;\vee\; t_2 \geq T$

In this initial model, besides the initializing event, we have four events: cool_rod1, cool_rod2, release_rod1, and release_rod2. In order to simplify, we suppose that we start when the reactor has reached the maximum temperature θ_M. Here are some events (the remaining ones for rod2 are similar):

```
INIT
  begin
    $t_1 := T$
    $t_2 := a$
    $\theta := \theta_M$
    $phase := 0$
  end
```

```
cool_rod1
  when
    $phase = 0$
    $t_1 \geq T$
  then
    $phase := 1$
    $t_2 := t_2 + b1$
    $\theta := \theta_m$
  end
```

```
release_rod1
  when
    $phase = 1$
  then
    $phase := 0$
    $t_1 := a$
    $t_2 := t_2 + a$
    $\theta := \theta_M$
  end
```

Remember that $t1$ and $t2$ denote the time elapsed since rod1 or rod2 have been released. In event cool_rod1, the temperature goes down from θ_M to θ_m with time $b1$ by the use of rod1, so the time t2 related to rod2 is updated accordingly. In event release_rod1, which happens just after the release of rod1, the temperature is raised up to temperature θ_M with time a, thus both $t1$ and $t2$ are updated accordingly. The proof of the invariants (in particular that of the safety invariant **inv0_7**) are very simple (some additional "technical" invariants are needed).

The refinement of this discrete system into a continuous one is simple routine. In the sequel, we show how the two discrete variables t_1 and t_2 are refined to time dependent variables t_1_c and t_2_c.

```
variables:  $t_1\_c$
            $t_2\_c$
            $phase$
```

```
inv1_1:  $t_1\_c \in \mathbb{R}^+ \nrightarrow \mathbb{R}$
inv1_2:  $t_2\_c \in \mathbb{R}^+ \nrightarrow \mathbb{R}$
inv1_3:  $now \in \mathrm{dom}(t1\_c) \cap \mathrm{dom}(t2\_c)$
inv1_4:  $t_1 = t_1\_c(now)$
inv1_5:  $t_2 = t_2\_c(now)$
```

Here are the corresponding refined events:

```
cool_rod1
when
  $phase = 0$
  $t_1\_c(now) \geq T$
then
  $phase := 1$
  $t_2\_c := \lambda\, t \cdot t \in now \,..\, now + b1 \mid$
        $t_2\_c(now) + t - now$
  $t_1\_c := \lambda\, t \cdot t \in now \,..\, now + b1 \mid$
        $t_1\_c(now)$
  $\theta := \theta_m$
  $now := now + b1$
end
```

```
release_rod1
when
  $phase = 1$
then
  $phase := 0$
  $t_1\_c := \lambda\, t \cdot t \in now \,..\, now + a \mid$
        $t - now$
  $t_2\_c := \lambda\, t \cdot t \in now \,..\, now + a \mid$
        $t_2\_c(now) + t - now$
  $\theta := \theta_M$
  $now := now + a$
end
```

Further refinements (not shown here) deal with making the variable θ continuous. The global proof effort for the Rodin Platform on this example is 157 proof obligations, all proved automatically.

3.3 Controlling Trains [4]

Our new example comes from the book of A. Platzer [4]. It involves one (or several) trains evolving on a single line. The goal of this system is to provide safe moves of the trains.

Preliminary Study. Each train is regularly made aware by a radio broadcasting (RBC) of a certain point situated at position m, on the line, where it should at the latest stop. In other words, the train shall never pass this point. Given the position z of the train, our main invariant is clearly the following: $m - z \geq 0$. Every ϵ second, the train controller (a piece of software) examines the situation concerning the position z, speed v, and acceleration a of the train. The controller can change the acceleration as follows: it can order a constant positive acceleration A (where A is positive), a constant negative acceleration $-b$ (where b is positive), or no acceleration. Notice that when the train has a negative acceleration $-b$ it should at last stop when $v = 0$ and thus never get a negative speed. If the train is at position z with speed v at time 0, it will circulate with speed $v + at$ at time t and its position will then be $z + vt + \frac{at^2}{2}$. In order to guarantee that the train will not pass the position m, one should be certain that the negative acceleration $-b$ will be sufficient to stop the train before the position m. Since the train should stop, we have $v + at = 0$ (with a = -b), that is $t = \frac{v}{b}$.This gives us the following position at $z+\frac{v^2}{2b}$. This quantity should be smaller than or equal to m, that is $z+\frac{v^2}{2b} \leq m$. Here is thus our final constraints:

$$2b(m-z) \geq v^2 \tag{1}$$

It can also be said that breaking with deceleration $-b$ is able to "absorb" the kinetic energy of the train (this will give us the same result in a shorter way). This is a necessary invariant of the system. We notice that it implies the previous invariant $m-z \geq 0$. At each control time (every other ϵ seconds), the controller must ensure that the train will preserve this invariant in the next control position (in ϵ second). The speed of the train will be $v+a\epsilon$ and the position of the train will be $z+v\epsilon+\frac{a\epsilon^2}{2}$. Therefore substituting these values for v and z in (1) yields the following, $2b(m-z-v\epsilon-\frac{a\epsilon^2}{2}) \geq (v+a\epsilon)^2$, that is:

$$2b(m-z) \geq v^2 + (a\epsilon^2 + 2v\epsilon)(a+b) \tag{2}$$

For the controller to decide that the acceleration a could be A for the next ϵ seconds, we must have: $2b(m-z) \geq v^2 + (A\epsilon^2 + 2v\epsilon)(A+b)$. If this is not the case (i.e. if the previous predicate is false), the controller must decide that the acceleration a must be $-b$. This decision is indeed safe since then $a+b=0$ in (2) and we already have the following invariant, $2b(m-z) \geq v^2$. At the end of the process, when the speed v is equal to 0 and when the train cannot proceed further, we have: $m-z < \frac{A\epsilon^2(A+b)}{2b}$. The final specification of this train controlling process has now become very clear. It can be stated informally as follows: move the train by accelerating or decelerating it until it reaches the speed 0 and get to a position z such that $m-z < \frac{A\epsilon^2(A+b)}{2b}$ holds. Notice that it is quite possible for the train to reach a position where its speed is 0 but with $m-z \geq \frac{A\epsilon^2(A+b)}{2b}$: it means that the train stops and restart immediately because it can move further.

Event-B Development. The previous preliminary elementary calculations dictate the way things can be implemented with Event-B and the Rodin Platform. We first define the four constants A, b, ϵ, and m: they are all positive real numbers. The dynamic state of the system introduces variables z, v, and a (position, speed, and acceleration of the train). We also introduce a technical variable *phase* that can be 1 or 2. In phase 1, the controller will decide what to do, whereas in phase 2, the train will makes some progress or stop. The main invariants are the following (**inv1_1** to **inv1_5**).The first decision event decide_1, decelerates the train as below:

```
inv1_1: $z \in \mathbb{R}^+$
inv1_2: $v \in \mathbb{R}^+$
inv1_3: $a \in \{0, A, -b\}$
inv1_4: $phase \in \{1, 2\}$
inv1_5: $2b(m-z) \geq v^2$
```

```
decide_1
  when
    $phase = 1$
    $2b(m-z) < v^2 + (A\epsilon^2 + 2v\epsilon)(A+b)$
  then
    $phase := 2$
    $a := -b$
  end
```

The second decision event, decide_2, accelerates the train. But at this point we introduce another constraint, namely that there is an upper constant speed limit v_M. We have thus two more decision events:

```
decide_2
  when
    $phase = 1$
    $2b(m - z) \geq$
        $v^2 + (A\epsilon^2 + 2v\epsilon)(A + b)$
    $v + \epsilon A \leq v_M$
  then
    $phase := 2$
    $a := A$
  end
```

```
decide_3
  when
    $phase = 1$
    $2b(m - z) \geq$
        $v^2 + (A\epsilon^2 + 2v\epsilon)(A + b)$
    $v + \epsilon A > v_M$
  then
    $phase := 2$
    $a := 0$
  end
```

The first driving event, drive_1, stops the train before the end of ϵ seconds since otherwise the speed would become negative. The second driving event, drive_2, continues the progression of the train.

```
drive_1
  when
    $phase = 2$
    $v + a\epsilon \leq 0$
  then
    $phase := 1$
    $v := 0$
    $z := z + \frac{v^2}{2b}$
  end
```

```
drive_2
  when
    $phase = 2$
    $v + a\epsilon > 0$
  then
    $phase := 1$
    $v := v + a\epsilon$
    $z := z + v\epsilon + \frac{a\epsilon^2}{2}$
  end
```

We can now refine this model by adding a second train. Nothing changes for the first train that still gets its limit point to be m. The second train with position z_2, speed v_2, and acceleration a_2 will now get its limit being z (the position of the first train) rather than m. This refinement is very easily done with Event-B and the Rodin Platform. After this second refinement, an interesting animation can be performed with the AnimB animator of the Rodin Platform: one can see the two trains accelerating and decelerating in an appropriate way. The global proof effort for the Rodin Platform on this example is 103 proof obligations, all proved automatically except two of them that are proved interactively (easy). It would be simple to refine the time independent variables z and v to time dependent variables as we have done in previous examples.

3.4 Aircraft Collision Avoidance [4] [5]

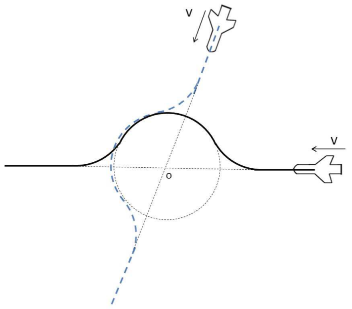
This example is taken from the book of A. Platzer [4]. It has also been developed in an independent paper by Platzer and Clarke [5]. The problem is to study an *horizontal* collision avoidance maneuver to be performed by two aircrafts flying at the same altitude. This maneuver must be performed when the two aircrafts have the possibility to "almost" collide, i.e. when the distance between them could become smaller than or equal to a predefined constant distance p. The maneuver is said to be "horizontal" as both aircrafts continue to fly at the same altitude before, during, and after executing the maneuver.

Simplified Case: Preliminary Study

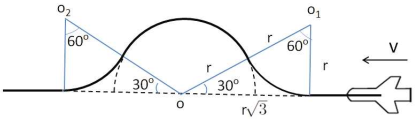
Platzer and Clarke studied a simple case with the following constraints:

1. Both aircrafts have the same linear speed v which has to be maintained during the maneuver.
2. Both aircrafts would *exactly collide* at some point o should they continue to fly without performing the maneuver.
3. Both aircrafts are situated at the same distance of the colliding point o when they decide to maneuver

The maneuver consists for both aircrafts to reach a certain circle centered in o and with a radius r (to be made precise later). Once they have reached this circle, both aircrafts follow it in the same direction until they both leave it at the same time in order to eventually return to their original direction. All this can be illustrated in the following figure:



Entering and leaving the circle as well as following the circle is always done at the same original linear speed v. In order to ensure this, both aircrafts should enter and leave the circle by using portions of external circles (called the entering circles) that are tangent to the main one and with the same radius. As a consequence, both aircrafts start following the entering circles when they are both at a distance $r\sqrt{3}$ of the circle center o. Likewise, they enter the main circle when the angle with their original trajectory is exactly $\frac{\pi}{6}$. This can be illustrated in the following figure:

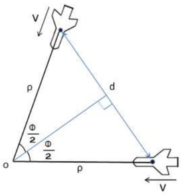


During the maneuver the distance between the two aircrafts must always be smaller than the predefined distance p. If the angle between the two trajectories is ϕ and the common distance of both aircrafts to the circle center o is ρ then the distance d of both aircrafts is the following:

$$d = 2\rho \sin \frac{\phi}{2}$$

This is illustrated in the right figure.
The main invariant of our system is thus the following:

$$2\rho \sin \frac{\phi}{2} \geq p$$



Notice that the angle ϕ between the two trajectories will not change during the maneuver. As a consequence, the only quantity that counts in order to compute the distance and thus check whether the safety condition holds is the common distance ρ of both aircrafts to the center o. The smallest distance between the aircrafts is reached when they are both flying on the main circle (this will be formally proved below). We must have then $2r \sin \frac{\phi}{2} \geq p$. This gives us a *lower value* for r: $r \geq \frac{p}{2\sin\frac{\phi}{2}}$. If both aircrafts decides to maneuver when they are at a distance ρ_i from the point o, this distance must be greater than the distance where they start turning (i.e. $r\sqrt{3}$). This gives us an *upper value* for r. We must have $r\sqrt{3} \leq \rho_i$. We have thus the following constraint for r (that is, r can be chosen non-deterministically between these two values): $\frac{p}{2\sin\frac{\phi}{2}} \leq r \leq \frac{\rho_i}{\sqrt{3}}$.

Notice that if ρ_i is too small, the maneuver is impossible: it is too late. In fact, we must have the following relationship between the three constants ρ_i, p, and ϕ: $\rho_i \geq \frac{p\sqrt{3}}{2\sin\frac{\phi}{2}}$

Simplified Case: Event-B Development. In this development, we model the behavior of one aircraft only. We can do so since in this simplified framework the behavior of the second aircraft can be deduced from that of the first one by a simple rotation with constant angle ϕ. Here are first a number of constant definitions: ρ_i, p and ϕ. They are all real numbers constrained as follows:

axm0_1: $2\rho_i \sin\frac{\phi}{2} \geq p\sqrt{3}$

The state of the first model is defined by means of the following variables: *phase*, ρ, θ, and r. Variables ρ and θ are the polar coordinates of the first aircraft. But as mentioned above, only the ρ polar cordinate is useful in order to compute the distance d between both aircrafts. As a consequence, we can discard the θ polar coordinate. The following invariants must hold between these variables. Mind the invariant **inv0_4** stating the main safety property, i.e. the distance between aircrafts is always greater than or equal to the constant p. These variables are initialized as follows and the initial agreement makes a non-deterministic choice for r:

```
inv0_1:  $phase \in \{0,1,2,3,4,5\}$
inv0_2:  $\rho \in \mathbb{R}^+$
inv0_3:  $r \in \mathbb{R}^+$
inv0_4:  $2\rho \sin\frac{\phi}{2} \geq p$
```

```
INIT
  begin
    $\rho := \rho_i$
    $phase := 0$
    $r :\in \mathbb{R}^+$
  end
```

```
agree
  any  $c$  where
    $phase = 0$
    $p \leq 2c \sin\frac{\phi}{2}$
    $c\sqrt{3} \leq \rho_i$
  then
    $phase := 1$
    $r := c$
  end
```

The next phases are described in the following events:

```
start
when
  $phase = 1$
then
  $phase := 2$
  $\rho := r\sqrt{3}$
end
```

```
enter
when
  $phase = 2$
then
  $phase := 3$
  $\rho := r$
end
```

```
cycle
when
  $phase = 3$
then
  $phase := 4$
  $\rho := r$
end
```

```
leave
when
  $phase = 4$
then
  $phase := 5$
  $\rho := r\sqrt{3}$
end
```

We are now going to refine this initial model by introducing the continuous time function for ρ, that is ρ_c. We also introduce the variable *now*. Our main invariant is **inv1_4** is to be compared to **inv0_4** where the variable ρ corresponded to a discrete transition.

```
inv1_1:  $\rho\_c \in \mathbb{R}^+ \nrightarrow \mathbb{R}$
inv1_2:  $now \in \mathrm{dom}(\rho\_c)$
inv1_3:  $\rho = \rho\_c(now)$
inv1_4:  $\forall t \cdot t \in \mathrm{dom}(\rho\_c) \Rightarrow$
           $2\rho\_c(t) \sin\frac{\phi}{2} \geq p$
```

```
INIT
  begin
    $\rho\_c := \{0 \mapsto \rho_i\}$
    $phase := 0$
    $r :\in \mathbb{R}^+$
    $now := 0$
  end
```

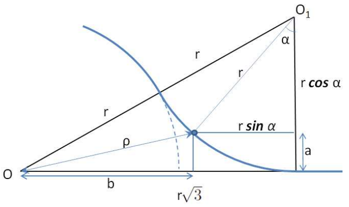
```
agree
  any  $c$  where
    $phase = 0$
    $p \leq 2c \sin\frac{\phi}{2}$
    $c\sqrt{3} \leq \rho_i$
  then
    $phase := 1$
    $r := c$
  end
```

The event INIT is refined in a simple way and the event agree is not modified. In the event start, the time function ρ_c is decreased from ρ_i in a linear fashion according to the constant speed v of the aircraft. The variable *now* is incremented with a time corresponding to the linear movement of the aircraft from the initial position at ρ_i to the position at $r\sqrt{3}$ where it starts turning.

```
start
  when
    $phase = 1$
  then
    $phase := 2$
    $\rho\_c := \lambda t \cdot t \in now \mathinner{..} now + \frac{(\rho_i - r\sqrt{3})}{v} \mid \rho_i - v(t - now)$
    $now := now + \frac{(\rho_i - r\sqrt{3})}{v}$
  end
```

The aircraft travels on the external circle centered in o_1 as shown in this figure:

$$\begin{aligned}\rho^2 &= a^2 + b^2\\ &= r^2(1-\cos\alpha)^2 + r^2(\sqrt{3}-\sin\alpha)^2\\ &= r^2(5-4\cos(\tfrac{\pi}{3}-\alpha))\\ \rho &= r\sqrt{5-4\cos(\tfrac{\pi}{3}-\alpha)}\end{aligned}$$



The part of this circle that is used corresponds to an angle of $\frac{\pi}{3}$. Since the aircraft still flies at the same linear speed v, the time it takes to turn in this circle is $\frac{\frac{\pi}{3}r}{v}$: this is therefore the quantity used to increment the variable *now* in the event enter. The computation of the new function ρ_c is a little more complicated. It is explained above (see the figure). We can notice that the quantity $\sqrt{5-4\cos(\frac{\pi}{3}-\alpha)}$ is well defined since $5-4\cos(\frac{\pi}{3}-\alpha)$ is always positive. Also, this quantity is greater than or equal to 1 (when α varies from 0 to $\frac{\pi}{3}$), so ρ is greater than or equal to r. Now the angle α can be related to the time $t - now$, which is the time elapsed on the circle to progress from the angle 0 to the angle α at linear speed v, that is: $\alpha = \frac{v(t-now)}{r}$

```
enter
  when
    $phase = 2$
  then
    $phase := 3$
    $\rho\_c := \lambda t \cdot t \in now \,..\, now + \frac{\pi r}{3v} \mid r\sqrt{5-4\cos(\frac{\pi}{3}-\frac{v(t-now)}{r})}$
    $now := now + \frac{\pi r}{3v}$
  end
```

The last two phases correspond to cycling on the main circle and then leaving the circle: they are not shown here. The global proof effort for the Rodin Platform on this example is 84 proof obligations, all proved automatically.

The General Case. The general case is not very different from the simplified one. In what follows, we give a short account on this generalization. We still suppose that both aircrafts are flying *at the same linear speed* v. They are converging to a point o as in the simplified case, but this time they are not necessarily colliding at this point, but their distance might become smaller than or equal to the predefined distance p. The first thing to do is to determine the distance between both aircrafts and the way this distance evolves. The distance δ can be calculated as follows:

$$\delta^2 = \rho_2^2\sin^2\phi + (\rho_2\cos\phi - \rho_1)^2 = \rho_1^2 + \rho_2^2 - 2\rho_1\rho_2\cos\phi$$

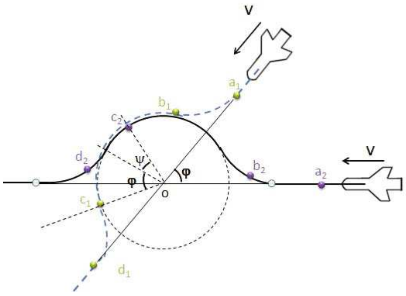
As ρ_1 and ρ_2 are moving from their initial values ρ_{i1} and ρ_{i2} at the same speed v, we have thus: $\rho_1 = \rho_{i1} - vt$ and $\rho_2 = \rho_{i2} - vt$ that is:

$$\delta^2 = (\rho_{i1} - vt)^2 + (\rho_{i2} - vt)^2 - 2(\rho_{i1} - vt)(\rho_{i2} - vt)\cos\phi$$

Thus, the derivative $\frac{d\delta^2}{dt}$ of δ^2 relative to t is the following:

$$\frac{d\delta^2}{dt} = 2v(\cos\phi - 1)(\rho_{i1} + \rho_{i2} - 2vt)$$

When t is smaller than $\frac{\rho_{i1}+\rho_{i2}}{2v}$, the derivative is negative. A minimum is thus reached when t is equal to $\frac{\rho_{i1}+\rho_{i2}}{2v}$, leading to the following minimum δ_m (in this case, we have $\rho_{i1} - vt = \frac{\rho_{i1}-\rho_{i2}}{2}$ and $\rho_{i2} - vt = \frac{\rho_{i2}-\rho_{i1}}{2}$): $\delta_m = (\rho_{i2} - \rho_{i1})\cos\frac{\phi}{2}$ (we suppose $\rho_{i2} > \rho_{i1}$) In summary, both aircrafts "almost" collide when the following holds: $\delta_m \leq p$. In order to avoid this "almost" collision, we are using the same maneuver as in the simplified case, namely to have both aircrafts following a trajectory using a circle centered in the point o as indicated in the following figure:



This time however, we have to take account of both aircrafts in order to ensure that their distance δ remains greater than or equal to p. We suppose $\rho_{i2} > \rho_{i1}$. First of all, when both aircrafts decide on the maneuver, their distance δ_i must be greater than p, that is: $\delta_i^2 = \rho_{i1}^2 + \rho_{i2}^2 - 2\rho_{i1}\rho_{i2}\cos\phi \geq p^2$.

We have to determine the radius r of the circle. The constraints are the following. The first aircraft can reach the turn: $r\sqrt{3} \leq \rho_{i1}$. The distance between both aircrafts is greater than or equal to p when they are both flying on the circle: $2r\sin\frac{\phi+\psi}{2} \geq p$ where ψ is the angular distance due to the difference of the initial positions of the aircrafts. More precisely, we have: $\psi = \frac{\rho_{i2}-\rho_{i1}}{r}$.

4 Conclusion

In this paper, we presented a way of studying hybrid systems in Event-B [7] and the Rodin Platform [8]. Our approach follows that of Action System [1] [2] [3]. It is illustrated by means of many examples taken from the literature. All of them have been developed and fully proved with the Rodin Platform. We have not studied the possible definition of the continuous parts by means of differential equations as is usually done in the hybrid system literature: this will be studied in subsequent papers. As the Rodin Platform does not support (mathematical) real numbers yet, our examples, implemented on Rodin, "cheated a bit". So far, for most of the examples, the "cheating" consisted in giving in the Rodin developments some specific integer values to the constants that are normally assigned to some real values. As a consequence, various calculations ended in integer numbers: we have done so in the saw example (section 3.1), the nuclear plant cooling example (section 3.2), and the train example (section 3.3). In the aircraft collision avoidance example (section 3.4), this simplification could not be done as we were dealing with trigonometric functions and the square root function. So, in this case, we gave some explicit properties of these functions. For instance, $\sqrt{3}$ is left as such. We have done the same for some trigonometric values, and so on. We also sometimes added some real number "axioms" such as $\forall x \cdot x \neq 0 \Rightarrow x * (y/x) = y$ that is not true for integer numbers since "/" is the integer division.

References

1. Back, R.J., Kurki-Suonio, R.: Distributed Cooperation with Action Systems. ACM Transaction on Programming Languages and Systems 10(4), 513–554 (1988)
2. Back, R.-J., Petre, L., Porres, I.: Generalizing Action Systems to Hybrid Systems. In: Joseph, M. (ed.) FTRTFT 2000. LNCS, vol. 1926, p. 202. Springer, Heidelberg (2000)
3. Back, R.J., Cerschi Seceleanu, C., Westerholm, J.: Symbolic Simulation of Hybrid Systems. In: APSEC 2002 (2002)
4. Platzer, A.: Logical Analysis of Hybrid Systems. Springer (2010)
5. Platzer, A., Clarke, E.M.: Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 547–562. Springer, Heidelberg (2009)
6. Alur, R., et al.: The Algorithmic Analysis of Hybrid Systems. Theoretical Computer Science 138, 3–34 (1995)
7. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
8. http://www.event-b.org
9. http://research.sei.ecnu.edu.cn/reports/A_Hybrid_V7.pdf