Karlsruhe Institute of Technology

A Theory of Refinement of Cyber-Physical-Systems into Implementations

Bachelor's Thesis of

Daniel H. Draper

at the Department of Informatics
Institute for Theorethical Informatics

Reviewer:	Prof. Dr. Bernhard Beckert
Second reviewer:	
Advisor:	Dr. rer. nat. Mattias Ulbrich

15. April 2015 – 15. August 2015

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

. . . . . .

(Daniel H. Draper)

Abstract

English abstract.

Zusammenfassung

Deutsche Zusammenfassung

Contents

List of Figures

List of Tables

1. Introduction

This bachelorthesis will try to formalize the following process: Replacing the abstract notion of the control program in a verified (by KeYmaera) Cyber-Physical-System (*CPS*) with an actual verified (by KeY) implementation through a form of Formal Refinement and being able to verify that the entire CPS still satisfys the required safety constraints, using both KeYmaera and KeY.

CPS are generally modelled as either Hybrid Automata or - Programs.(See ref. [**platzer2010b**]). Mostly, this means, that an abstract version of the discrete control program is modelled, as a non-deterministic assignment of one or multiple control value (See app. A.1). To replace this non-deterministic assignment with an actual implementation a certain "glue" or "coupling" has to be found to translate discrete and real continous values into each other. Glue in this case refers to a relation that encompasses both the discrete and real values. In certain cases (See sect. 2.4), glue will be a concrete function, but not in general.

If we try to express the goal of this thesis in logic, we get equation 1.1. Here, ψ is the safety condition that acts both as a specification for our later implementation and the goal the program result is verified against. Glue is the aforementioned relation to be able to translate from the real into the discrete world and vice versa.

$$\begin{aligned} \textbf{If } (&\models [controlValue := *, ?\psi(controlValue) \dots]\alpha \text{ verified} \\ &\wedge \textit{"glue"(discreteVariables, continuousVariables)} \text{ verified} \\ \textbf{Then } &[controlValue := JavaProgram \dots]\alpha \text{ also verified} \end{aligned} \tag{1.1}$$

To explain this process we will take a look at the following:

I: Example-based Refinement on CPS Watertank (Example taken from KeYmaera).

II: Introduction of formalized process of using Refinement to gain a concrete implementation from a hybrid model.

III: Application of formalized process on example: CPS Gear-Backlash (See ref. [**bla**]).

1.1. Introducing Cyber-Physical-Systems

In this thesis we take a close look at **CPS**. These are systems in which a physical aspect or value is controlled by a computer (program). For example, an aircraft control system in which the computer exerts a form of speed control on the airplane would be a CPS.

In our case we take a closer look at the closely related notion of *Hybrid Systems*, in which discrete values (in the control program) and continuous values (in the physical world)

coexist. The difficulty in analyzing these kinds of systems stems from the “hybridness” of the systems: There is always some form of translation necessary to go from the program (discrete values) to the physical (continous reals) world.

two basic modelling approaches exist for hybrid systems: Hybrid Automata that are based on Non-Determnistic Finite Automata.

2. Example-based Refinement on CPS Watertank

To get a better understanding of the tasks involved in refining a hybrid model into a implementation with all necessary intermediate verification steps, we took a look at one of the out-of-the-box examples provided in the KeYmaera tutorial [**keYmaera**], a Watertank (See Fig. 2.1).

The "control" part of this hybrid system is the valve, it can either drain the watertank with a rate of -2 or it can fill the tank further with a rate of 1. The goal of the entire system then is, to keep the water level y of the tank between 1 and 12. Normally this is proven by KeYmaera without actually implementing a control program. As our goal is to find a concrete implementation, we first took a look at the hybrid model of the watertank. It is provided both in the form of a hybrid automata (See Fig. 2.2) and a hybrid program (See Fig. 2.3).

2.1. Finding the concrete Control Value Assisgnment

In order to be able to apply Eq. 1.1 to this concrete example, the first challenge we faced was finding a spot in which a (or multiple) concrete control value is actually assigned. We call this assignment(s) the *hook* as the actual implementation will "hook" into our hybrid model at this exact point. Taking a look at the Hybrid Automata describing the Watertank (See Fig. 2.2), it became obvious, that the hybrid model of the watertank required changing to be able to find our actual "hook". In the original model, a control value is never explicitly assigned, rather does the valve change its state non-deterministically, making a deterministical control program implementation impossible. Therefore, we tried to remodel the model to better serve our purpose, featuring a clear ticked hook that is called upon at deterministic times. (See Fig. 2.4).

We tried to keep the model mostly equivalent to the original, preserving the 2 clock ticks of time needed for the valve to change its state from drain to fill and vice-versa. This then proved difficult, when trying to think of possible implementations, as the program hook could be called again before the valve would have actually changed its state. To simplify this problem, we decided to only model cases in which the actual program tick (so the time between two hook calls) is greater than 2 clock ticks (however long they may be). This means, that the valve or control program doesn't have to have a (possibly infinite) list of the last control values assigned, which could be the case if the program was called more frequently than the valve is able to change states.

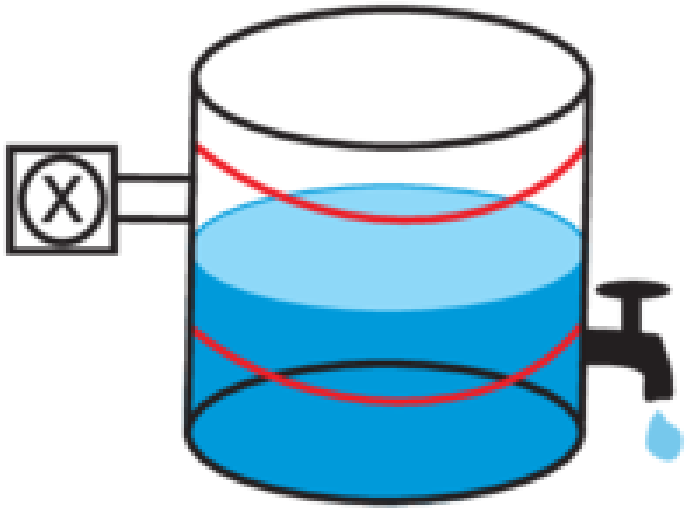

Figure 2.1.: Picture of possible Watertank configuration: Watertank that can get drained at all times and has a valve controlled by a control program [**keymaeraGuide**].

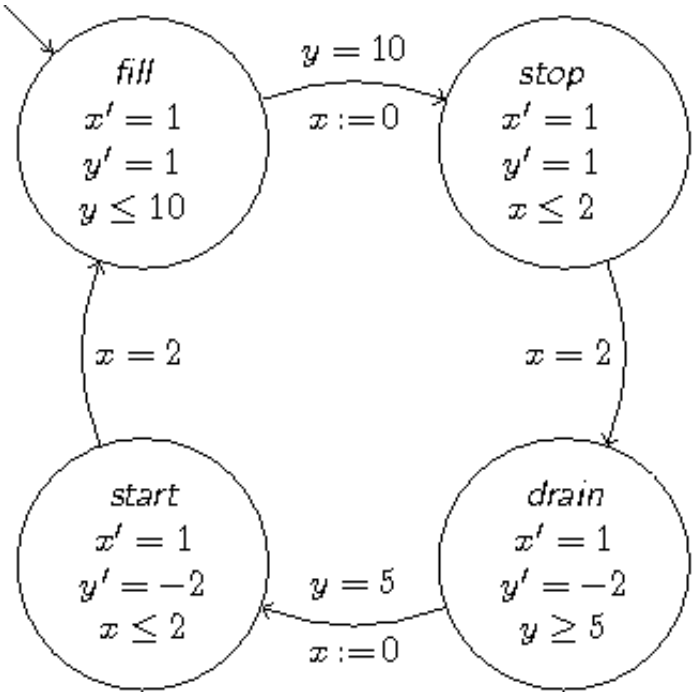

Figure 2.2.: The Watertank CPS expressed as a hybrid automata [**keymaeraGuide**].

```
/* This example does NOT use a good description style */
\programVariables {
    /* state variable declarations */
    R y, x, st;
}

\problem {
    /* initialization */
    \[  x:=0; y:=1; st:=0 \] (
     st = 0  /* initial state characterization */
   ->
     \[  /* system dynamics */
       ( /* repeat the discrete/continuous transitions */
           (?(st=0);
                 (?(y=10); x:=0; st:=1)
              ++ {x'=1,y'=1, y<=10}
           )
         ++ (?(st=1);
                 (?(x=2); st:=2)
              ++ {x'=1,y'=1, x<=2}
           )
         ++ (?(st=2);
                 (?(y=5); x:=0; st:=3)
              ++ {x'=1,y'=-2, y>=5}
            )
         ++ (?(st=3);
                 (?(x=2); st:=0)
              ++ {x'=1,y'=-2, x<=2}
           )
       )*@invariant(1<=y & y<=12
           & (st=3 -> (y>=5-2*x)) & (st=1 -> (y<=10+x)))
     \] (1<=y & y<=12)    /* safety / postcondition */
  )
 }
```

Figure 2.3.: The Watertank CPS expressed as a hybrid program [**keymaeraGuide**].

```
\functions {

    R tick;
    \external R Floor(R);
}

\programVariables {

R y; R x; R old; R new; R valve;
}

/*
invariant:
y >=1 & y <=12
*/

\problem {

 tick > 2  -> \[ x := tick; y := 1; valve := 1;

((?(x = tick ); new := *; x:= 0);
/* (? (y + 2* valve + (tick -2) * new >= 1 & y + 2* valve + (tick -2) * new <= 12 & y + 2* valve >= 1 & y + 2* valve  <= 12); */
(?(y + 2 * valve + tick * new >= 1 & y + 2 * valve + tick * new <= 12 & (new = 1 | new = -2));

((?(new != valve); {x' = 1, y' = valve & x <= 2}); if (x=2) then valve := new; {x' = 1, y' = valve & x <= tick} fi)

++ (?(!(new != valve)); {x' = 1, y' = valve & x <= tick})

))*@invariant(y >=1 & y <=12 & (valve = -2 | valve = 1) & (x = tick -> (y + 2* valve >= 1 & y + 2* valve  <= 12)))
\]
(y >= 1 & y <= 12)
}
```

Figure 2.4.: The Watertank CPS after remodelling to account for hook.

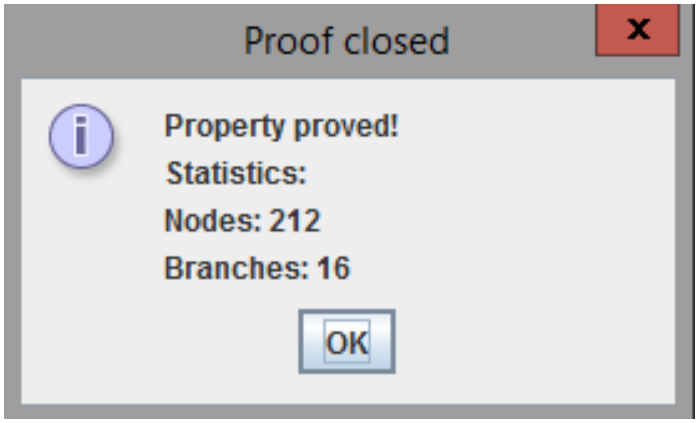

Figure 2.5.: The result of automatic verification by KeYmaera of the Watertank hybrid program that includes our hook (See fig 2.4).

2.2. Finding the correct Program Safety Condition

The next step we attempted was finding the correct postcondition for the program hook (ψ in Eq. 1.1), which would then serve as the abstraction of the java control program we would later implement. This means, that the program would be built accordingly, so that it could be verified against this postcondition. The original postcondition we devised:

$$\begin{aligned} \psi \equiv ?(&y + 2 * valve + tick - 2 * new >= 1 \ \& \\ &2 * valve * (tick - 2) * new <= 12 \ \& \\ &y + 2 * valve >= 1 \ \& \ y + 2 * valve <= 12) \end{aligned} \tag{2.1}$$

When trying to verify the entire hybrid program (see comment in line 5 of the problem in Fig. 2.4), this postcondition did not work. This means, that even in such a simple CPS as this watertank control system, finding the hook postcondition was non-trivial and only with the help of KeYmaera's counterexamples did we manage to find the correct postcondition (See Eq. 2.2).

$$\begin{aligned} \psi \equiv (?(&y + 2 * valve + tick * new >= 1 \ \& \\ &y + 2 * valve + tick * new <= 12 \ \& \\ &(new = 1 | new = -2)) \end{aligned} \tag{2.2}$$

KeYmaera verified our new hybrid program fully automatically (See Fig. 2.5).

2.3. The (simple) Java Control Program

After completing verification of our hybrid model with KeYmaera, we proceeded to implement a (simple) control program for the Watertank. Finding a suitable implementation proved difficult, as a parallel implementation (Both the control system and the differential

equations working in parallel, modifying the water level) seemed more intuitive. This would defeat the purpose of actually finding a suitable hook and was more based on our understanding of the original hybrid model and not our version including the hook.

Using our understanding of a singular hook of the control system into the Watertank as well as the hook postcondition from the previous section as our specification for the actual control method, that would return the control value to the Watertank's valve, we then managed to implement a suitable discrete control method (See Lst. 2.3).

```
1  public int getControlValue (int y, int old) {
2        //Waterlevel in two time units
3        int inTwo = y + 2 * old;
4        //If we are raising level, keep raising if possible without
             hitting max_level before next tick
5        if (old == 10) {
6           if (inTwo + tick * 1 <= 116) {
7              return 10;
8           }
9           else {
10                return -20;
11          }
12       }
13       //ELSE if we are currently lowering level, keep lowering if we can
              lower further without hitting min_level b4 next tick
14       else {
15          if (old == -20) {
16             if (inTwo - tick * 2 >= 12) {
17                return -20;
18             }
19             else {
20                   return 10;
21
22             }
23
24          }
25       }
26       //Only returned if old != 10 && old != -20, unreachable.
27       return 0;
28 }
29 }
```

The control method consists only of simple if-else-statements and just assigns the best-case (keep raising level if we were raising, keep lowering if we are lowering as long as postcondition is still valid) value to the valve. We translated the hook postcondition (See Eq. 2.1) into, what we thought to be, a suitable JDL postcondition statement for verification with KeY (See List. 2.3). In the next section we will explain why this postcondition would not work for our complete proof. KeY verified our program fully automatically, aside from the fact, that the $tick > 2$ precondition added too much computational complexity, so we let KeY prove the property with the more concrete $tick == 30$; $tick == 31$; . . . assignments. See Fig. 2.6, 2.7.

For testing purposes, we also implemented a complete simulator of the Watertank CPS (See App. **??**).

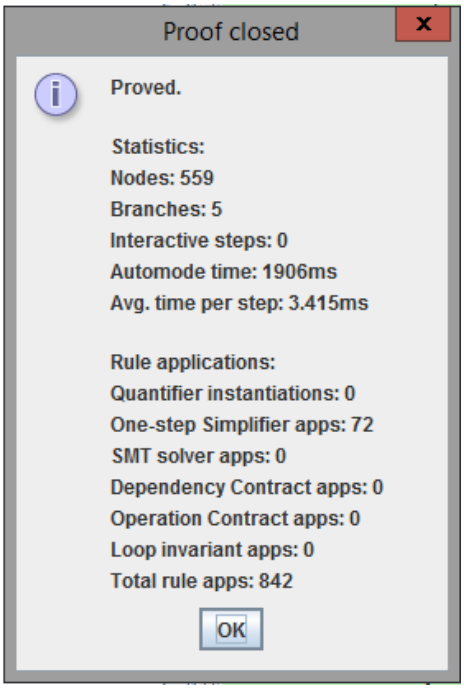

Figure 2.6.: Automatic proof result with tick== 30.

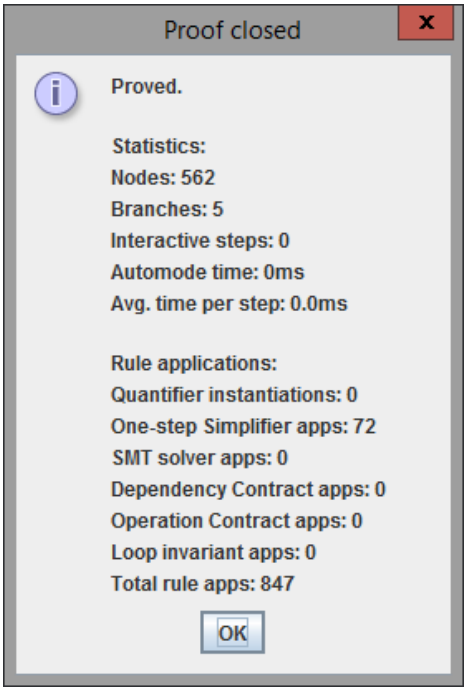

Figure 2.7.: Automatic proof result with tick== 31.

```
1  /*@ public normal_behavior //
2    @ requires tick == 30 && y >= 10 && y <= 120 && y + 2 * old <= 120
         && y + 2 * old >= 10 && (old == 10 || old == -20); //
3    @ ensures \result * (tick / 10) + y + 2 * old  >= 12 & \result * (
         tick / 10) + y + 2 * old <= 116 && (\result == 10 || \result ==
          -20); //
4    @*/
```

2.4. Finding the glue between Java and the Hybrid Model of the system

The most important and also difficult part of our process, was finding the glue between the real parts of the hybrid system and our java control program. As real values (used in the hybrid model) and discrete values (used in our java implementation) are fundamentally different, a certain transformation has to occur. In the Watertank example, this means, that the waterlevel as well as the last valve-value (be it fill or drain), which are passed to the control method, have to be converted into one direction by the glue.

Also, the result of the computation in the method (so the new valve value) has to be converted in the other direction. In general the glue is not necessarily a bijective function, as some values will not exist in both worlds, making it a relation. For the Watertank, fortunately, this problem doe not exist, so finding a function translating the waterlevel and valve values into each other proved relatively easy.

As we went along, we figured out how a general glue proof should look (See Fig. 2.3). Here x are the discrete values in our control program, y are the real world values, ψ is the postcondition of the control program expressed using discrete values and ϕ is the actual safety condition we want to show for the entire hybrid system.

$$\forall x \in (DiscreteWorld)\ \forall y \in (RealWorld) : (x, y) \in glue \implies (\psi(x) \implies \phi(y)) \quad (2.3)$$

Following this basic guideline, we were able to find a suitable glue relation (or in our case a bijective function) for this cps (See Eq. 2.4).

$$\begin{aligned} &\forall y \in \mathbb{R}.\forall y_j \in \mathbb{Z}.\forall valve \in \mathbb{R}.\forall tick \in \mathbb{R}.\forall tick_j \in \mathbb{Z}.\forall new \in \mathbb{Z}.\forall result \in \mathbb{R} : \\ &((y_j = \text{floor}(10 * y) \land old_j = \text{floor}(10 * valve) \land tick_j = \text{floor}(10 * tick) \land result = 10 * new \land \\ &\quad y >= 1 \land y <= 12 \land (valve = 1 \lor valve = -2) \land tick > 2 \implies \\ &\quad ((result * tick_j/10 + y_j + 2 * old_j <= 116 \land result * tick_j/10 + y_j + 2 * old_j >= 12 \land \\ &\quad\quad (result = 10 \lor result = -20)) \implies \\ &\quad\quad y + 2 * valve + tick * new >= 1 \land y + 2 * valve + tick * new <= 12 \land \\ &\quad\quad\quad (new = 1 \lor new = -2)) \end{aligned} \quad (2.4)$$

As can be seen, the $\psi(x)$ that we found for the glue is not equal to the original trivial postcondition for our java program (See List. 2.3). This can be explained by the floor functions that has to be used for translation from the $\mathbb{R}$ into $\mathbb{Z}$ world. Due to the usage of the floor function, we only can approximate the original value in the discrete world, and therefore these approximation erorrs have to be accounted for in ψ. Again the fact that we only found this error in our original postcondition devised for the java program, proves the non-triviality of the process even in seemingly simple examples as this watertank.

2.5. Verification based on KeYmaera

Verification of the glue proved difficult since KeYmaera had problems with the floor function. Therefore we found an abstraction of the function using two inequalities to approximate it (See Eq. 2.5).

$$f(x) = y \implies x - 1 < y \land y <= x \tag{2.5}$$

Unfortunately, this still lead to problems when using the automatic proof-mechanism, so we had to add a full new rule fdef to the definition file to be able to use the automatic proover (See List. 2.5).

```
1  \rules {
2     fdef {
3        \schemaVar \term R x;
4        \schemaVar \skolemTerm R c;
5           \find(f(x))
6        \sameUpdateLevel
7        \varcond ( \new(c, \dependingOn(x)) )
8        \replacewith(c)
9        \add(x-1 < c & c <= x & (x >= 0 -> c >= 0) & (x < 0 -> c < 0)
            ==> )
10    };
11 }
```

Applying the rule manually to the floor function and changing the first-order-strategy (so the time at which quantifier elimination using mathematica is tried) from lazy (extending every possible leave with case differentiation) to eager (quantifier elimination as soon as possible), the proof runs automatically aside from the application of our own defined rule (See Fig. 2.8).

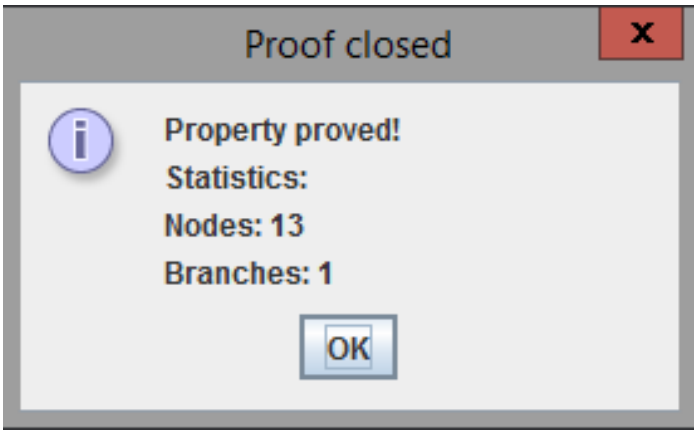

Figure 2.8.: Successful proof of our glue with KeYmaera.

3. Introduction of formalized process of using Refinement to gain a concrete implementation from a hybrid model

What the Watertank example shows is the non-triviality of refining the hybrid model into an implementation and of the verification of all necessary parts. Overall it is obvious, that a formalized approach to the general problem presented in chapter 1 is necessary. In this chapter we present a possible formalized approach to the problem, that we deemed feasible.

To aid readability we will now give an overview of the process without explanation, then detailing each step in the following sections.

1. Modelling CPS as Hybrid System that includes concrete hook.
2. Finding the necessary safety condition of the control value for verification with KeYmaera.
3. Implementing control program according to safety condition as its specification and Verification by KeY.
4. Finding the correct "glue" between hybrid model and control program and its verification by KeYmaera.
5. Result validation: Was Eq. 1.1 proven?

3.1. Modelling CPS as Hybrid System that includes concrete Hook

Most CPS we took a look at (See [**keymaera**] Tutorial, [**platzer2010b**] …) as examples, did not have a concrete spot in which a control program could "hook" in easily. This means, that the first step in our refinement process has to be finding a suitable hook for the control program, referring to one or more non-deterministic assignments of a/multiple control values. Mostly for already existing hybrid systems, as was the case in Sec. 2.1, a complete changing of the model is necessary. When creating a new model, one should try to find a suitable spot for a non-deterministic control value assignment, and if possible already model a form of tick, so as to make an implementation actually feasible.

To prevent issues in later verification of the model, a hybrid program representation of the model should be chosen (cf. [**platzer2010b**]).

3.2. Finding the necessary safety condition of the control value for verification with KeYmaera

After finding a suitable hook spot in our program we have the analyze the hook for a necessary safety condition. If we think back to the watertank example (See ch. **??**), this was non-trivial as the first postcondition we came up with logically proved to be incorrect. When thinking of possible conditions we often found we overlooked the tick when considering different postconditions. Obviously, more than one valid postcondition exists, as the constraints imposed on the implementation can get arbirtrarily strong. One should try and find the most general postcondition ψ for which KeYmaera still verifies the model, as this makes it easier to code the actual implementation with less constraints.

3.3. Implementing control program according to safety condition as its specification and Verification by KeY.

Implementation of the program can theorethically be done in any preferred language, for usage with KeY Java would obviously be the easiest. The implementation is free from any constraints asides from the postcondition that was devised previously, aside from the constraints provided by KeY itself (No threading etc.). Since the postcondition is in the hybrid model and therefore written in the form of real variables and values, one has to already think of a form of the glue that will be formally devised as the next step, as to find a suitable translation of the postcondition expressed with real values into JDL so that KeY can understand.

The implementation with its postcondition expressed in JDL then has to be verified by KeY before continuing with the glue.

3.4. Finding the correct glue between hybrid model and control program and verifying it with KeYmaera.

As both our model and implementation are now verified, the actual glue between the implementation and the model now has to be devised. In the previous Chapter we came up with a general definition of what the glue relation provides:

$$\forall x \in (DiscreteWorld)\ \forall y \in (RealWorld) : (x, y) \in glue \implies (\psi(x) \implies \phi(y)) \quad (2.3)$$

This means, that for the discrete-world postcondition ψ that is actually proven by KeY and the real-world postcondition ϕ that we came up with in step 3, and the glue that we devise, we reach the goal we set out for (See Eq. 1.1). Coming up with the glue is the hardest part of the entire process, and for us took multiple iterations before a correct and verified glue was found. We used a form of logical deduction to come up with the correct quantifiers for the glue proof (so the $\forall$ in Eq. 2.3), which can be found in App. **??**.

3.5. Validating glue results against implementation

As we realized in our Watertank example (See Sec. 2.4), with verification of our glue some changes might have to be made to the implementation and its postcondition. To complete the process of refinement we have to change the implementation and jdl condition, verify it again with KeY and make sure all our decisions made were actually valid and correct when taking a look back at the entire picture according to Eq. 1.1).

4. Evaluation

…

4.1. First Section

…

4.2. Second Section

…

4.3. Third Section

…

5. Conclusion

…

A. Appendix

A.1. Images

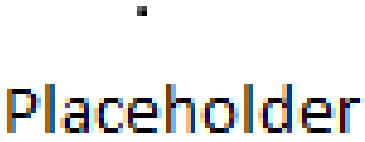

Figure A.1.: Watertank Hybrid Program and - Automata with Non-Deterministic Control Program Abstraction marked.