Interactive Verification of Cyber-physical Systems: Interfacing Averest and KeYmaera

Xian Li, Kerstin Bauer, Klaus Schneider
Embedded Systems Group
Department of Computer Science
University of Kaiserslautern, Germany
Email: {xian.li,k_bauer,klaus.schneider}@cs.uni-kl.de

Abstract—**Verification is one of the essential topics in research of cyber-physical systems. Due to the combination of discrete and continuous dynamics, most verification problems are undecidable and need to be dealt with by various kinds abstraction techniques. As systems grow larger and larger, most verification problems are difficult even for purely discrete systems. One way to address this problem is the use of interactive verification. Recently, this approach has also been considered by cyber-physical verification tools like KeYmaera and other classical theorem provers.**

Important requirements for the interactive verification are a precise and readable modeling language as well as the possibility to decompose the system into smaller subsystems. Here, tools like KeYmaera and PVS still need further improvement. On the other hand, these modeling aspects are both addressed within the language Quartz as it provides a complete programming language for cyber-physical systems with standard data types and programming statements as well as a precise compositional semantics that is well-suited for compositional verification.

In this paper, we take the advantages of two different tools, the Averest system and KeYmaera, for the interactive verification of cyber-physical systems. This way, we combine modeling and verification capabilities of Averest and the verification capability of KeYmaera, in order to provide a basis for powerful tool set for the interactive verification of cyber-physical systems.

I. Motivation

CYBER-physical systems are systems that combine discrete and continuous dynamics. The environment of embedded reactive systems often consist of continuous behaviors that are determined by the laws of physics. Formal verification is already hard for discrete systems because of the size of the transition systems, and most verification problems for cyber-physical system are even undecidable, due to the combination of discrete and continuous dynamics.

Inspired by the success of model checking [1] in hardware verification and protocol analysis, there has been increasing research on developing techniques for the automated verification of cyber-physical systems. The main line of research concentrates on model checking of finite abstractions of restricted subclasses of the general model. Most techniques proposed so far in this area either rely on bounded state reachability or on abstraction refinement techniques [2–4]. While the first approach suffers inherently of incompleteness, the latter approach often introduces unrealistic behaviour that may yield spurious errors being reported within the analysis.

Despite the theoretic achievements in research, only a few tools are available to verify non-trivial cyber-physical systems. Tools like e.g. PHAVer [5], HyTech [6], Charon [7, 8] focus on the continuous dynamics. They lack typical program statements and data types.

Furthermore, current tools often require an explicit enumeration of the discrete state space. Although the discrete state space typically consists of only finitely many states, the number of these discrete states can become too large to be handled properly by current computers [9].

HySAT [10] and MathSAT [11, 12] are built based upon a SAT-solver that calls a linear program solver for conjunctions of the linear continuous-part constraints. As this technique requires to encode the whole problem space first, the size of the handleable problems is quite small. BACH [13, 14] provides a convenient GUI to construct rectangular hybrid automata with linear location invariants together with a powerful bounded reachability checker for these systems. Another tool, HybridSAL relation abstracter [15, 16] abstracts the discrete and continuous dynamics of the hybrid system automatically to infinite state discrete transition systems that can be model checked by SAL tools [17].

An alternative approach to verification is based on interactive theorem provers. An approach based on higher-order logic [18] for specification and verification of hybrid control system is described in [19]. A verification framework is presented in [20] to strike the balance between the expressiveness of theorem proving and the efficiency and automation of the state exploration techniques. In order to assist in the deductive verification of hybrid systems, [21] presents a tool implemented as a part of STeP [22]. Deductive methods are used in [23] to deal with the parallel composition of hybrid systems, the operational step semantics and a number of proof-rules within PVS [24] have been formalized as well. KeYmaera [25, 26] is an automated and interactive theorem prover for specification and verification logics for differential dynamic logics for hybrid system. It integrates numerous techniques for automated theorem proving, combining deductive, real algebraic, and computer algebraic prover technologies.

Recently, a new language for modeling, simulation, and verification of cyber-physical systems has been developed in our research group [27]. This language is an extension of the synchronous language Quartz that is derived from the Esterel language. Originating from a programming language for discrete systems, there is a rich set of data types, and many statements for expressing discrete behaviors in a convenient way. In particular, generic statements and module hierarchies allow one to describe large parametric systems in a concise way. Thus, the modeling capabilities of Quartz are in many cases better than in comparable languages. Like Quartz, also the extension to cyber-physical systems has a precise formal semantics that defines unique behaviors for given input traces. For this reason, the language lends itself well for formal verification. In particular, Quartz programs can be translated to equivalent symbolic transition relations, and thus provide a sound basis for formal verification. The determinism of the language is also very important for simulation, since it allows one to reproduce once observed behaviors. Based on the programing language Quartz, the *Averest* toolset has been developed. Besides of transformations, hardware/software synthesis, a symbolic model checker and other tools, the Averest toolset has recently been extended by a technique for interactive verification [28]. However, up to now these interactive verification techniques are restricted to the discrete component of Quartz, models with hybrid components cannot yet be dealt with.

In this paper, we therefore propose a new approach for the interactive verification of cyber-physical systems by interfacing the Averest toolset and KeYmaera. While the overall verification task remains within the interactive Averest prover, assertions for the continuous components can be verified with the help of KeYmaera. Thus, the advantages of both systems – the modelling and verification capabilities of Quartz especially w.r.t. the discrete component and the verification capabilities of continuous components within KeYmaera – can be combined in order to result in a powerful tool for the interactive verification of cyber-physical systems. Due to the underlying synchronous language this approach will be very well suited for compositional verification that is a great challenge in the context of cyber-physical systems.

The outline of the paper is as follows. In Section II the synchronous language Quartz and the hybrid language used by KeYmaera are briefly introduced. Then, Section III gives a detailed overview of the interfacing of Averest and KeYmaera for applying interactive verification. The paper will be concluded with the application of the interactive verification to a widely known example in section IV.

II. Preliminaries

In the following, we give a brief overview over the synchronous language Quartz, its hybrid extension for modeling cyber-physical systems and the KeYmaera language.

A. The Synchronous Language Quartz

Quartz is a synchronous language that is derived from the Esterel language. The execution of a Quartz program is defined by so-called *micro and macro steps*, where a macro step consists of finitely micro steps whose maximal number is known at compile time. Macro steps correspond to reaction steps of reactive systems, and micro steps correspond with atomic actions like assignments of the program that implement these reactions. Variables of a synchronous program are *synchronously updated* between macro steps so that the execution of the micro steps within a macro steps is done in the same variable environment of their macro step. This synchronous update is important for avoiding data races, and therefore to ensure determinism.

The language offers many data types like booleans, bitvectors, signed and unsigned integers that may be bounded or unbounded, real numbers, as well as compound data types like arrays and tuples. Modules are declared with an interface that determines inputs and outputs, and a body statement that may use additional local variables. In the following, we list some of the possible statements to describe the examples given in this paper. A complete definition of the language is found in [29] for the discrete case, and in [27] for the hybrid extension.

Provided that S, S_1, and S_2 are statements, ℓ is a location variable, x is a variable, σ is a boolean expression, and α is a type, then the following are statements (parts given in square brackets are optional):

- $x = \tau$ and $\texttt{next}(x) = \tau$ (assignments)
- $\texttt{assume}(\varphi)$, $\texttt{assert}(\varphi)$ (assumptions and assertions)
- $\ell : \texttt{pause}$ (start/end of macro step)
- $S_1; S_2$ (sequences)
- $S_1 \parallel S_2$ (synchronous concurrency)
- $\texttt{if}\ (\sigma)\ S_1\ \texttt{else}\ S_2$ (conditional)
- $\texttt{do}\ S\ \texttt{while}(\sigma)$ (loops)
- $\{\alpha\ S\}$ (local variable)

the pause statement defines a control flow location ℓ – a boolean variable being true iff the control flow is currently at $\ell : \texttt{pause}$. Since all other statements are executed in zero time, the control flow only rests at these positions in the program, and thus the possible (discrete) control flow states are the subsets of these locations.

There are two variants of assignments that both evaluate the right-hand side τ in the current macro step: Immediate assignments $x = \tau$ transfer the value of τ to the left-hand side x directly, while delayed assignments $\texttt{next}(x) = \tau$ assign the value in the next macro step.

If the value of a variable is not determined by assignments of the current of previous macro step, a default value is used according to the declaration of the variable. To this end, declarations of variables consist of a *storage class in addition to their type*. There are two storage classes, namely mem and event that choose the previous value (mem variables) or a default value (event variables) in case no assignment determines the value of a variable.

In addition to the statements known from other imperative languages (conditionals, sequences and loops), Quartz offers synchronous concurrency $S_1 \parallel S_2$ and sophisticated preemption and suspension statements (not shown in the above list), as well as many more statements for the comfortable descriptions of reactive systems. There is also the possibility to call once implemented modules and to store modules in packages to support the re-use in the form known from software libraries.

Our Averest system provides algorithms that translate a synchronous program to a set of guarded actions [29], i.e., pairs (γ, α) consisting of a trigger condition γ and an action α. Actions are thereby assignments $x = \tau$ and next(x) = τ, assumptions assume(φ), or assertions assert(φ). The meaning of a guarded action is obvious: in every macro step, all actions are executed whose guards are true. Thus, it is straightforward to construct a symbolic representation or extended finite state machine (EFSM) of the transition relation in terms of the guarded actions (see [29]).

While time in synchronous languages is given in the abstract form of macro steps, cyber-physical systems require the consideration of physical time. In order to combine these inherently different concepts of time, the computational model of macro steps is endowed by a continuous transition that takes place between the immediate and delayed assignments of the macro step. During the continuous transition, which consumes physical time, variables of the new storage class `hybrid` change their values according to the new *flow assignments* x $\leftarrow$ τ or drv(x) $\leftarrow$ τ (that equate variable x or its derivation on time drv(x) with the expression τ).

The continuous transition of the macro step starts with the variable environment determined by the immediate assignments as initial values. To distinguish between the 'discrete' value and the changing value during the continuous transitions, a new operator cont(x) is introduced: x always refers to the discrete value of a variable, whereas cont(x) refers to the (changing) value during the continuous evolution. For memorized and event variables x and cont(x) always coincide as these variables do not change during continuous evolutions.

The continuous actions may only occur in special statements of the form flow S until(σ) where S is a list of flow assignments and σ is a so-called *release condition* that terminates the continuous phase defined by the flow statement. Figure 1 depicts a program fragment together with the corresponding EFSM. Starting from location ℓ_1, the immediate assignment x = 0.0 is executed so that the continuous transition starts with initial value x = 0.0. The derivation of x is then 1 during the continuous transition, and the continuous transition terminates as soon as the continuous value of x is 1. Then, the control flow will move to ℓ_2. However, it may be the case that another flow-statement runs in parallel, and that its continuous transition terminates before x = 1.0 holds. In this case, the control flow moves to ℓ_2', and it will be restarted from there in the next macro step.

B. KeYmaera: Hybrid Programs

KeYmaera is a verification tool for hybrid systems that combines deductive, real algebraic, and computer algebraic prover technologies [26]. It is an automated and interactive theorem prover for a natural specification and verification logic for hybrid systems. In this section, we give an incomplete overview of the syntax and semantics of KeYmaera programs. Statements not needed in the remainder of the paper will be omitted here, for more detailed information consider [25, 26].

The relevant program statements of KeYmaera are summarized in Table I. During a discrete transition, all right hand sides of the actions $x_i := \tau_i$ are computed in parallel and

Program Statement

ℓ_1: pause
x = 0.0;
ℓ_2, ℓ_2':flow{drv(x)<-1.0} until (cont(x)>=1.0)

Extended Finite State Machine

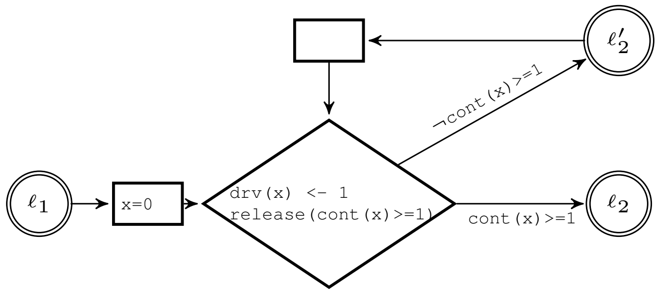


Fig. 1. The Flow Statement of Quartz Programs

TABLE I
SYNTAX OF KEYMAERA (INCOMPLETE)

1	$x_1 := \tau_1, \ldots, x_n := \tau_n$	Discrete jump set
2	$\{x_1' = \tau_1, \ldots, x_n' = \tau_n, H\}$	Continuous evolution
3	α ; β	Sequential composition
4	if(ϕ) then α else β fi	Deterministic choice
5	$< \alpha > \phi$	Existential operator
6	$[\alpha]\phi$	Universal operator
7	$[[\alpha]]\phi$	Universal Path operator

assigned in a second step, which essentially corresponds to the delayed actions of synchronous languages. Continuous transitions are defined by the differential equation systems $x_i' = \tau_i$ of the variables and the evolution domain H, which is defined by a set of location invariants. Continuous evolutions *may* be terminated at any point of time, they *must* be terminated at the latest, when location invariants would be violated. Thus, continuous transitions are always non-deterministic. As stated in line 3, KeYmaera provides sequential composition. The statement if-then-else in line 4 provides a deterministic choice, that depending on the condition ϕ either executes α or β.

Further statements such as loops and non-deterministic choice will not be considered here. Thus, the only non-determinism provided in the presented fragment of KeYmaera lies within the continuous transition, as all other statements in Table I are deterministic.

The formulas from line $5 - 7$ are used for verification. $< \alpha > \phi$ is an existential operator that evaluates to true iff ϕ holds after at least one valid run of the hybrid program α. Analogously, $[\alpha]\phi$ is an universal operator that evaluates to true iff ϕ holds after all valid runs of the hybrid program α. $[[\alpha]]\phi$ is a universal path operator that holds true iff during all runs of the hybrid program α the condition ϕ is satisfied.

III. INTERFACING AVEREST AND KEYMAERA

Due to the combination of discrete and continuous dynamics, most verification problems are undecidable for cyber-physical systems. Even the simple reachability problem is undecidable for most families of cyber-physical systems and the few

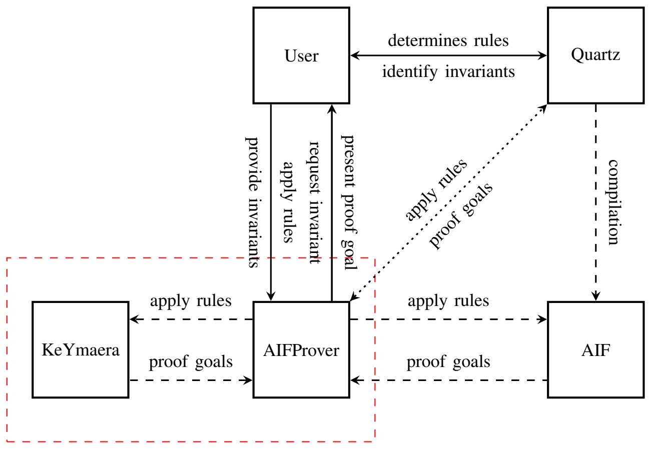


Fig. 2. Idea of our Approach

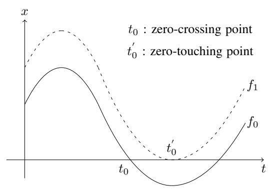


Fig. 3. Zero-Crossing and Zero-Touching Function

```
1,1':flow{
     drv(x_1) ← τ_1; ...; drv(x_n) ← τ_n;
}until(φ ≤ 0)
```

Fig. 4. Continuous Transition within Flow Statement

decidability results depend on strong restrictions of either the discrete or continuous component.

An interactive verification approach is pursued by the tool KeYmaera that is especially suitable for verifying parametric hybrid systems [26, 30]. Unfortunately, like most tools for cyber-physical systems, KeYmaera focuses on the continuous component, e.g. only real-valued variables can be modelled. Furthermore, the tool still lacks good capabilities for modelling the parallel composition of systems.

Recently, our research group proposed a new verification approach for discrete Quartz programs (see Figure 2), which is based on the Averest system. Assuming that the system is given as a Quartz program, the user determines rules based on the program structure which are then verified on the intermediate code format AIF, that essentially is a set of guarded actions. By rule applications, the guarded actions are decomposed into smaller AIF files. The proof goals can also be decomposed flexibly, so that the compositional reasoning could be used for verification by the AIFProver, meaning that already proved goals/assertions of some program fragments can be used as assumptions for the remaining program fragments. At the moment, this approach only supports discrete Quartz programs.

In this section, we propose a way to integrate the continuous component of Quartz programs into that framework by interfacing the tool KeYmaera with AIFProver as depicted in Figure 2. As already mentioned in the preliminaries, Quartz and the underlying language of KeYmaera differ in major points (especially the discrete semantics) which makes it difficult to translate complete Quartz programs to KeYmaera. Thus, we will interface Averest and KeYmaera such that proof rules that depend on continuous transitions will be proved by KeYmaera while the overall verification remains within the Averest.

A. General Idea

Recall that continuous transitions of Quartz programs and KeYmaera programs are based on quite different semantics.

In Quartz programs, continuous evolutions *must* terminate at exactly the first point of time, where an active release condition evaluates to true. The continuous assertions given in the form of constraints (which essentially correspond to location invariants) do not influence the control flow of the Quartz program and may be used only for verification purposes. Thus, continuous transitions in Quartz programs are completely deterministic. Contrary to that, continuous transitions of KeYmaera programs are non-deterministic. They *must* be terminated before active location invariants would be violated, they *may* be terminated at any point of time before that. There is no equivalent to our release condition.

In the following, we explain the general idea of how to adapt continuous transitions in Quartz to KeYmaera and still provide the required semantics. Assume for simplicity to have a continuous transition (compare Figure 4) with only one active release condition in the form of $\phi(t) \leq 0$ together with the proof goal σ that shall hold true at the end of the continuous transition, i.e. when the transition terminates according to its release condition $\phi \leq 0$. Assume furthermore, that the release function ϕ satisfies the condition, that it will have a zero-crossing in finite time and that the first zero-crossing point is a 'real' crossing point instead of only a 'zero-touching' point as depicted by the straight line in Figure 3. Then it holds, that for $0 \leq t \leq t_0$ the invariant $\phi \geq 0$ holds, while the same invariant will be violated for $t > t_0$. Thus, adding this invariant to the KeYmaera program as the evolution domain of the continuous transition enforces the transition to terminate at t_0 at the latest while not changing the transition otherwise.

Now, in order to enforce KeYmaera only to consider *one* path, where the continuous transition terminates at time t_0, the proof goal σ at the end of the KeYmaera program must be changed to $\phi \leq 0 \to \sigma$. According to our assumption w.r.t. the release function ϕ we know that at least one path exists where the continuous transition terminates at t_0. Thus, $\phi \leq 0 \to \sigma$ evaluates to true iff σ holds on thepath we are interested in.

B. Transformation

Figure 5 shows the transformation of the continuous transition of a macro step to an equivalent KeYmaera program according to the ideas in the previous subsection. The left-hand side of the figure depicts the continuous transition of the macro step together with the given assumptions and proof goals (assertions). The corresponding KeYmaera program (for ease of notation without variable initializations) on the right-hand-side is described in detail below. The transformation of the continuous transition is divided into four parts:

- *Initialization*
  Already obtained assumptions by the interactive verification together with known initial values of the variables are gathered in the assumption ψ_{assume}. Furthermore, a new location label $s := -1$ is introduced that is used for bookkeeping whether the continuous transition has been terminated because of the release-conditions or prematurely.
- *Continuous transition*
  The differential equations of the hybrid variables can be translated one-to-one. $\Psi := \bigvee_i \sigma_i$ is the disjunction of all active release conditions and states the termination criterion for the continuous transition. As already sketched previously, a set of location invariants $\hat{\Psi}$ needs to be added, that is determined by the active release conditions Ψ:
  For ease of notation assume that each single release condition σ is a conjunction of basic expressions of the form $f(x_0, .., x_n) \leq 0$ or $f(x_0, .., x_n) = 0$. If disjunctions occur, these can be dealt with in the same way as several release conditions in parallel. Define now for the first case
  $$\hat{f}_i := f(x_0, .., x_n) \geq 0$$
  and for the latter case
  $$\hat{f}_i := f(x_0, .., x_n) \begin{cases} \leq 0 : & f(x_0, .., x_n) < 0 \textit{ initially} \\ \geq 0 : & f(x_0, .., x_n) > 0 \textit{ initially} \end{cases}$$
  Then, the release condition σ is replaced by $\hat{\sigma} := \bigvee \hat{f}_i$, as the continuous transition must be terminated only, when all of the basic expressions evaluate to true.
  Several release conditions in parallel correspond to a disjunction of these conditions, as it is only necessary that at least one release triggers. Thus, the location invariants determined by each single release condition must hold true in parallel, i.e. $\hat{\Psi} := \bigwedge_i \hat{\sigma}_i$
- *Termination of the continuous transition*

  Lines $7 - 10$ define the control flow of the program for the next macro step. Depending on the fulfillment of the active release conditions Ψ, new values of the location variables are determined.
  By definition, the continuous transition terminates due to the release conditions iff $\Psi := \bigvee_i \sigma_i$ evaluates to true. Thus, the arc from line $7 - 9$ will be executed iff the continuous transition terminates because of the active release conditions. For ease of notation, this information is stored in the location label s, that is either set to 1 or 0, regarding to the termination condition.
- *Proof goal assertion*
  The proof goal ψ_{assert} of the Quartz program must evaluate to true iff the considered path terminated the continuous transition according to the active release conditions. Thus, it must be proved that $s = 1$ holds which is enforced by the KeYmaera proof goal $s = 1 \rightarrow \psi_{assert}$

C. Correctness

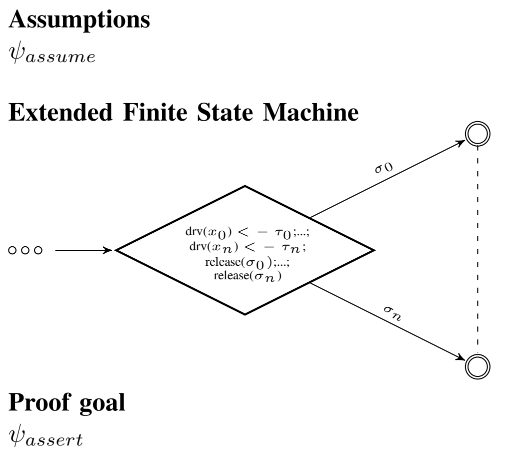
In this section, we present how to use the transformation given in the previous subsection for the interactive verification. The main difficulty lies within the different semantics of KeYmaera and Quartz, namely the difference between determinism (Quartz) and non-determinism (KeYmaera). These difficulties have been dealt with by the introduction of the location invariants $\hat{\Psi}$ together with the additional location label s. The following theorem now states the correctness of transfering the proof results within KeYmaera back to the Quartz language.

Theorem. *Consider a continuous transition in Quartz as given in Figure 5 together with assumptions and proof goals (assertions). Assume furthermore, that the release functions of the continuous transition provide a real zero-crossing (see Figure 3) and that the continuous transition will be terminated in finite time t_0. Then, the following holds:*

1) *The continuous evolution of the variables given by the differential equations are analogous to the ones in Quartz.*
2) *The continuous transition of the KeYmaera program must be terminated at the time $0 \leq t \leq t_0$ and after executing the KeYmaera program, $s = 1$ holds iff $t = t_0$, otherwise $s = 0$.*
3) *If the goal $s = 1 \rightarrow \psi_{assert}$ is proven in KeYmaera, then ψ_{assert} holds after the continuous transition in Quartz.*

Proof.

1) Obvious.
2) The continuous evolutions of the hybrid variables are equivalent within Quartz and KeYmaera. By assumption t_0 is the first point of time where an active release condition evaluates to true. Thus, the location invariants as defined in the previous section are satisfied for all times $0 \leq t \leq t_0$. Furthermore, according to the assumption that the release conditions satisfy the condition as depicted in Figure 3, at least one of the location invariants will be directly violated for $t > t_0$.
   Thus, the duration of the continuous transition of the KeYmaera program can be any time t with $0 \leq t \leq t_0$. The arc given in lines $7-9$ will be executed iff at least one release condition evaluates to true, i.e. if the continuous transition of KeYmaera terminates at time $t = t_0$. This yields that after executing the program $s = 1$ holds iff $t = t_0$, otherwise $s = 0$.



```
\problem {
    /* Initialization */
    \ [ $\psi_{assume}$
    (      $s = -1$
$\rightarrow$ \ $\langle$
    /* Continuous transition */
    {   $x_0^{'} = \tau_0, ..., x_n^{'} = \tau_n$,    $\hat{\Psi}$   } ;
    /* Control flow assignment*/
    if( $\Psi$ )    then {
        if($\sigma_0$) then $l_0 := 1$ fi ; ... ;
        if($\sigma_n$) then $l_n := 1$ else $\ell_n = 0$ fi ;
        $s := 1$
    } else {   $s := 0$   }    fi
\ $\rangle$
    /* Proof goal assertion*/
    ) ( ($s = 1$)$\rightarrow$ $\psi_{assert}$ )
}
```

Fig. 5. Continuous transition in Quartz together with the corresponding KeYmaera program

3) Since by assumption the continuous transition in Quartz terminates at finite time t_0, according to 2), there exists exactly one path satisfying $s = 1$ after executing the KeYmaera program, which corresponds to the continuous transition in Quartz. Thus, if the condition $s = 1 \rightarrow \psi_{assert}$ can be proven by KeYmaera, then ψ_{assert} holds after the continuous transition within Quartz.

■

The first assumption, that the release functions have the form as given in Figure 3 is not a severe restriction, as this is the realistic behaviour in most cases and can be checked in advance. The second assumption, that the continuous transition terminates in finite time can directly be checked by KeYmaera, by simply exchanging the proof goal to the existence of a path satisfying $s = 1$ at the end (compare line 5 in Table I).

If one is interested in proving the continuous assertions of Quartz programs, this can be achieved by replacing the proof goal with the universal path operator as depicted in line 7 in Table I.

IV. Interactive Verification

In this section, we apply the framework as depicted in Figure 2 to an adaptation of the well-known bouncing ball example. While the bouncing ball itself is a standard example with relatively simple continuous dynamics, the parallel composition of a number of balls is difficult to model for most tools as without a suitable compositional semantics the model suffers severely from state space explosion.

Figures 6 and 7 depict the Quartz code of N parallel balls, where N is an arbitrary parameter and the balls start from arbitrary heights. In the main module the observer `n_sum` counts the overall number of bounces of *all* balls, whereas the module `Ball` models an individual ball.

The correctness of the implementation of this observer is proved in two steps: In the first step, the correctness of the counters of each single ball is proven. As the module `Ball` works independently of the rest of the program and is stable under parallel composition, this goal can be achieved by only considering that submodule, disregarding of the number of parallel balls (for more information on the stability of Quartz programs under parallel composition compare [27]). In the second step, the only proof goal is the correctness of the overall observer, i.e. that for all times `n_sum` is defined as the sum of the single observers.

A. Verification of a Single Counter

In Quartz programs, it is possible to have several variable values for the same physical point of time, as the semantics of the language does not only consist of physical but also of logical time. Therefore, the event of the ball hitting the floor cannot simply be described by $h \leq 0$. A good alternative is given by 'the ball reaches the floor during a continuous transition, the starting point may be in the air or directly after a bounce'. The latter part is defined by $\sigma :=$ `h=0 and v>0 or h>0`, whereas reaching the floor during the continuous transition is defined by `cont(h)<=0 and cont(v)<0`. To prove the correctness of the event description, it suffices to show, that during any continuous transition satisfying σ, this condition holds as an invariant for all points except the termination point of the continuous transition. Thus, the event of hitting the floor can only be triggered at the end of but not during the continuous transition. In terms of Quartz, the first property is expressed by σ`-> constrainSM(cont(`σ`))`. The corresponding KeYmaera proof for the only flow statement is depicted in Figure 8.

Define now `NEXTSTEP(`ϕ`)` as a macro for the macro step following one, where a given condition ϕ holds. Then, in a second step the global invariant `NEXTSTEP(h>0 and cont(h)<=0)->next(n)= n+1` is proven. Here, KeYmaera must prove that the release condition `cont(h)<=0 and cont(v)<=0` implies the condition `h>0 and cont(h)<=0`, which is obviously the case and omitted here. The rest of the proof rules will be done by the AIFProver.

```
module Ball(real ? init_h, ? init_v ,int n,) {
   hybrid real h, v;
   h = init_h ; v = init_v ;
   loop{
      flow {
         drv(h) <- cont(v);
         drv(v) <- -9.81;
      } until(cont(h) <= 0 and cont(v) <= 0);
       next(v) = -v/2.0;
       next(n) = n + 1 ;
       flow {} until(true);
   }
}
```

Fig. 6. Hybrid Quartz Module One Ball

```
import BounceBall.*;
macro N = ?;
module NBalls([N]real ? InitH){
    hybrid [N]real h, v;
    [N]int n; int n_sum;
    for(i=0..N-1) {h[i] = InitH(i); v[i] = 0.0; }
    /* n parallel balls */
       {for(i=0..N-1) do || Ball(h[i],v[i],n[i]);}
    ||
       loop{
          n_sum = sum(i =0..N-1 ) n[i];
          pause;
       }
}
```

Fig. 7. Hybrid Quartz Module N Balls

```
\programVariables {
   R $h$, $v$ ;
   R $s$, $t$ ;
   R $l_1$ ,$l_2$ ;
}
\problem {
   $(\sigma) \wedge s = -1$
$\rightarrow$ \ [
   $\{h' = v, v' = -9.8, t' = 1, h \geq 0 \vee v \geq 0\}$;
   if($h = 0 \wedge v \leq 0$)
   then $l_1 := 1$; $l_2 := 0$; $s := 1$
   else $s := 0$
   fi
   \ ] (( $s = 0$ ) $\rightarrow$ (($\sigma$) )
}
```

Fig. 8. KeYmaera Program for only Flow Statement

B. Verification of the Overall Counter

The verification of the overall counter is very simple, as the single counters work correctly. As the counters are discrete variables, it suffices to show that it holds globally `n_sum = n[1] + ... + n[N]`, which is easily done by the AIFProver.

V. Conclusion

Quartz is a powerful synchronous language for the modelling of cyber-physical systems with non-trivial discrete dynamics. This language provides possibilities for the interactive verification of purely discrete programs based on the program structure. In this paper, we presented how to combine the modelling and discrete verification capabilities of Averest with the verification capabilities of KeYmaera. To that end, we interfaced KeYmaera and Averest and showed the capabilities of this tool combination by interactively verifying a non-trivial example.

References

[1] O. Grumberg and H. Veith, Eds., *25 Years of Model Checking – History, Achievements, Perspectives*, ser. LNCS, vol. 5000. Springer, 2008.

[2] A. Fehnker, E. Clarke, S. Kumar Jha, and B. Krogh, "Refining abstractions of hybrid systems using counterexample fragments," in *Hybrid Systems: Computation and Control (HSCC)*, ser. LNCS, M. Morari and L. Thiele, Eds., vol. 3414. Zurich, Switzerland: Springer, 2005, pp. 242–257.

[3] T. Dzetkulic and S. Ratschan, "Incremental computation of succinct abstractions for hybrid systems," in *Formal Modeling and Analysis of Timed Systems (FORMATS)*, ser. LNCS, U. Fahrenberg and S. Tripakis, Eds., vol. 6919. Aalborg, Denmark: Springer, 2011, pp. 271–285.

[4] K. Bauer, R. Gentilini, and K. Schneider, "A uniform approach to three-valued semantics for mu-calculus on abstractions of hybrid automata," in *Haifa Verification Conference (HVC)*, ser. LNCS, H. Chockler and A. Hu, Eds., vol. 5394. Haifa, Israel: Springer, 2009, pp. 38–52.

[5] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," in *Hybrid Systems: Computation and Control (HSCC)*, ser. LNCS, M. Morari and L. Thiele, Eds., vol. 3414. Zurich, Switzerland: Springer, 2005, pp. 258–273.

[6] T. Henzinger, P.-H. Ho, and H. Wong-Toi, "HYTECH: a model checker for hybrid systems," *Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1-2, pp. 110–122, December 1997.

[7] F. Kratz, O. Sokolsky, G. Pappas, and I. Lee, "R-Charon, a modeling language for reconfigurable hybrid systems," in *Hybrid Systems: Computation and Control (HSCC)*, ser. LNCS, J. Hespanha and A. Tiwari, Eds., vol. 3927. Santa Barbara, California, USA: Springer, 2006, pp. 392–406.

[8] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee, "Modular specification of hybrid systems in Charon," in *Hybrid Systems: Computation and Control (HSCC)*, ser. LNCS, N. Lynch and B. Krogh, Eds., vol. 1790. Pittsburgh, Pennsylvania, USA: Springer, 2000, pp. 6–19.

[9] X. Briand and B. Jeannet, "Combining control and data abstraction in the verification of hybrid systems," in *Formal Methods and Models for Codesign (MEMOCODE)*, R. Bloem and P. Schaumont, Eds. Cambridge, Massachusetts, USA: IEEE Computer Society, 2009, pp. 141–150.

[10] M. Fränzle and C. Herde, "HySAT: An efficient proof engine for bounded model checking of hybrid systems," *Formal Methods in System Design (FMSD)*, vol. 30, pp. 179–198, 2007.

[11] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani, "Verifying industrial hybrid systems with MathSAT," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 119, pp. 17–32, 2005.

[12] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani, "The MathSAT 4SMT solver," in *Computer Aided Verification (CAV)*, ser. LNCS, A. Gupta and S. Malik, Eds., vol. 5123. Princeton, New Jersey, USA: Springer, 2008, pp. 299–303.

[13] L. Bu, Y. Li, L. Wang, and X. Li, "BACH: Bounded reachability checker for linear hybrid automata," in *Formal Methods in Computer-Aided Design (FMCAD)*. Portland, Oregon, USA: IEEE Computer Society, 2008, pp. 1–4.

[14] L. Bu, Y. Li, L. Wang, X. Chen, and X. Li, "BACH 2: Bounded ReachAbility CHecker for compositional linear hybrid systems," in *Design, Automation and Test in Europe (DATE)*. Dresden, Germany: EDA Consortium, 2010, pp. 1512–1517.

[15] M. Sabry, A. Sridhar, D. Atienza, Y. Temiz, Y. Leblebici, S. Szczukiewicz, N. Borhani, J. Thome, T. Brunschwiler, and B. Michel, "Towards thermally-aware design of 3D MPSoCs with inter-tier cooling," in *Design, Automation and Test in Europe (DATE)*. Grenoble, France: IEEE Computer Society, 2011, pp. 1466–1471.

[16] A. Tiwari, "HybridSAL relational abstracter," in *Computer Aided Verification (CAV)*, ser. LNCS, P. Madhusudan and S. Seshia, Eds., vol. 7358. Berkeley, California, USA: Springer, 2012, pp. 725–731.

[17] L. de Moura, S. Owre, H. Rueß, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari, "SAL 2," in *Computer Aided Verification (CAV)*, ser. LNCS, R. Alur and D. Peled, Eds., vol. 3114. Boston, Massachusetts, USA: Springer, 2004, pp. 496–500.

[18] M. Gordon, "HOL: A machine oriented formulation of higher order logic," Computer Laboratory, University of Cambridge, Tech. Rep. 68, May 1985.

[19] N. Völker, "Towards a HOL framework for the deductive analysis of hybrid control systems," in *Automation of Mixed Processes: Hybrid Dynamic Systems (ADPM)*, S. Engell, S. Kowalewski, and J. Zaytoon, Eds. Shaker, 2000, pp. 243–250.

[20] T. Mhamdi and S. Tahar, "Providing automated verification in HOL using MDGs," in *Automated Technology for Verification and Analysis (ATVA)*, ser. LNCS, F. Wang, Ed., vol. 3299. Taipei, Taiwan: Springer, 2004, pp. 278–293.

[21] Z. Manna and H. Sipma, "Deductive verification of hybrid systems using STeP," in *Hybrid Systems: Computation and Control (HSCC)*, ser. LNCS, T. Henzinger and S. Sastry, Eds., vol. 1386. Berkeley, California, USA: Springer, 1998, pp. 305–318.

[22] N. Bjørner, A. Browne, E. Chang, M. Colon, A. Kapur, Z. Manna, H. Sipma, and T. Uribe, "STeP: deductive-algorithmic verification of reactive and real-time systems," in *Computer Aided Verification (CAV)*, ser. LNCS, R. Alur and T. Henzinger, Eds., vol. 1102. New Brunswick, New Jersey, USA: Springer, 1996, pp. 415–418.

[23] E. Abraham-Mumm, U. Hannemann, and M. Steffen, "Verification of hybrid systems: formalization and proof rules in PVS," in *International Conference on Engineering of Complex Computer Systems*. Skovde, Sweden: IEEE Computer Society, 2001, pp. 48–57.

[24] S. Owre, J. Rushby, and N. Shankar, "PVS: A prototype verification system," in *Conference on Automated Deduction (CADE)*, ser. LNCS, D. Kapur, Ed., vol. 607. Saratoga Springs, New York, USA: Springer, 1992, pp. 748–752.

[25] A. Platzer and J.-D. Quesel, "KeYmaera: A hybrid theorem prover for hybrid systems (system description)," in *International Joint Conference on Automated Reasoning (IJCAR)*, ser. LNCS, A. Armando, P. Baumgartner, and G. Dowek, Eds., vol. 5195. Sydney, New South Wales, Australia: Springer, 2008, pp. 171–178.

[26] A. Platzer, *Logical Analysis of Hybrid Systems – Proving Theorems for Complex Dynamics*. Springer, 2010.

[27] K. Bauer, "A new modelling language for cyber-physical systems," Ph.D. dissertation, Department of Computer Science, University of Kaiserslautern, Germany, Kaiserslautern, Germany, January 2012.

[28] M. Gesell and K. Schneider, "Interactive verification of synchronous systems," in *Formal Methods and Models for Codesign (MEMOCODE)*, S. Shukla, L. Carloni, D. Kroening, and J. Brandt, Eds. Arlington, Virginia, USA: ACM, 2012, pp. 75–84.

[29] K. Schneider, "The synchronous programming language Quartz," Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, Internal Report 375, December 2009.

[30] A. Platzer and J. Quesel, "European train control system: A case study in formal verification," in *International Conference on Formal Engineering Methods (ICFEM)*, ser. LNCS, K. Breitman and A. Cavalcanti, Eds., vol. 5885. Rio de Janeiro, Brazil: Springer, 2009, pp. 246–265.