Formal verification of multitask hybrid systems by the OTS/CafeOBJ method

Masaki Nakamura [1] Kazutoshi Sakakibara [1] Yuki Okura [1] Kazuhiro Ogata [2]

[1]Toyama Prefectural University, Toyama, Japan

[2]Japan Advanced Institute of Science and Technology, Ishikawa, Japan

***Abstract*- Hybrid systems combine both continuous and discrete behaviors. Formal descriptions of hybrid systems may help us to verify desired properties of a given system formally with computer supports. In this paper, we propose a way to describe a formal specification of a given multitask hybrid system as an observational transition system in CafeOBJ algebraic specification language and verify it by the proof score method based on equational reasoning implemented in CafeOBJ interpreter.**

Keywords-component; hybrid system; algebraic specification; observational transition system; proof score method

I. Introduction

Formal methods are mathematically based techniques for specification and verification of software and hardware systems. Formal specification languages play important role in formal methods. CafeOBJ is an executable formal specification language, which provides specification execution based on a rewrite theory. The OTS/CafeOBJ method is a formal method in which a system is modeled as an observational transition system (OTS), its specification is described in CafeOBJ, and properties are verified formally by using specification execution function implemented in CafeOBJ, called the proof score method [1, 2]. In our previous work [3], we have proposed a way to describe and verify multitasking real-time systems in the OTS/CafeOBJ method. A real-time system is regarded as a hybrid system where only time is its continuous data. In the literature [4], a way to describe and verify formal specifications of hybrid systems based on CafeOBJ has been proposed, however, only a single-task system is considered. In this study, we propose a way to describe a formal specification of a multitask hybrid system as an observational transition system in CafeOBJ algebraic specification language by extending the existing results, and verify it by the proof score method based on equational reasoning implemented in CafeOBJ interpreter.

This work was supported by JSPS KAKENHI Grant Number JP19K11842.

DOI reference number: 10.18293/SEKE2021-029

II. Preliminaries

A Hybrid automata of a signal control system

In this article, we consider a hybrid automaton of a simple signal control system, represented in Fig.1. The system consists of a signal and a car such that the car is prohibited from being in a specific area, between cs_0 and cs_1, while the signal is red. The specific area is called the critical section. The signal has three modes indicating its color label. Each color of the signal should be kept more than t_0 time units. The car has two modes: going and not-going. In the going mode, the car moves forward according to time advancing. The car stays there in the not-going mode. If the signal label is not green, the car cannot enter the critical section. If the signal label is changed into yellow while the car exists in the critical section, the car should keep the going mode, that is, it should not stop. Thus, if the interval t_0 is more than the time which the car needs to go through the critical section ($cs_1 - cs_0$), the car does not exist in the critical section while the signal is red.

Hybrid automata are models of hybrid systems with discrete and continuous behavior. We give a model of our simple hybrid system according to the literature [5]. Figure 2 represents a hybrid automaton

$$(Loc, Lab, Edg, X, Init, Inv, Flow, Jump)$$

of our simple signal control system with a single car. Locations Loc and edges Edg with labels lab are depicted as circles and arrows between them. X is given as $\{pos, now, l\}$. The initial values are all zero, that is, $Init(pos) = Init(now) = Init(l) = 0$, where pos and now stand for the position of the car and the current time, and l is

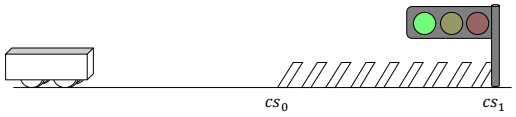

Figure 1. A signal control system

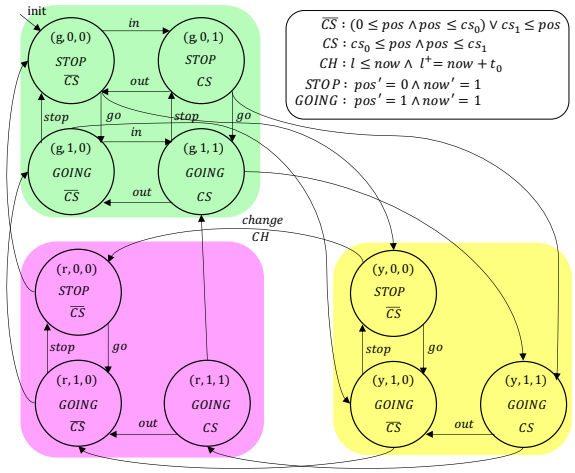

Figure 2. A hybrid automaton of a signal control system

used for the signal interval control. $Loc = \{g, y, r\} \times \{0, 1\} \times \{0, 1\}$ is the set of locations where $(label, going, cs) \in Loc$ represents the location where the color of the signal is *label*, the car moves when *going* and the car exists in the critical section when cs. The top, the middle and the bottom labels in a location correspond to locations, flow conditions and invariants of the location respectively. For example, $Flow(l) = STOP$ and $Inv(l) = \overline{CS}$ for $l = (g, 0, 0)$.

The flow conditions of locations are given by $STOP : pos' = 0 \wedge now' = 1$ or $GOING : pos' = 1 \wedge now' = 1$, where x' stands for the time derivative of x. Since $now' = 1$ holds in each location, the value of *now* always increases according to time advancing. Thus, *now* keeps the elapsed time from the initial state. The value of *pos* is unchanged when $STOP$ holds, and increases when $GOING$ holds. The invariants $CS : cs_0 \le pos \le cs_1$ and $\overline{CS} : (0 \le pos \le cs_0) \vee cs_1 \le pos$ mean that the car exists and does not exist in the critical section respectively.

The edges e between different color's locations are labeled by *change* and the jump condition $Jump(e)$ is given as $CH : l \le now \wedge l^+ = now + t_0$, which means that the edge can be executed when $l \le now$ and then the value of l is updated by $now + t_0$. We omit some of *change* and CH in Figure 2. We also omit unchanged variables in the jump condition, that is, $x^+ = x$ for each variable x.

The state transition system is obtained by a hybrid automaton, where a state $(l, v) \in S$ is a pair of a location and values of variables. From the above conditions, there is no path $s_0 \to s_1 \to \cdots \to s_n$ such that s_0 is an initial state, and $s_n = ((r, x, y), v)$ and $cs_0 < v(pos) < cs_1$, that is, the car does not exist in the critical section when the signal is red. We call it the safety property.

Hybrid automata of plural cars can be obtained by composing the copies of the above hybrid automaton. Consider the hybrid automaton with two cars. A location is a pair (l_1, l_2) where $l_i = (label_i, going_i, cs_i)$. There is an edge between (l_1, l_2) and (l'_1, l'_2) if there is an edge e_i between l_i and l'_i and $l_j = l'_j$ $(i, j \in \{1, 2\}, i \ne j)$. When the number of cars increases, the state space is exponentially increases, e.g., the number of locations becomes 10^n for n cars.

In the following sections, we describe OTS/CafeOBJ specifications of the above hybrid automaton with a single car and with plural cars, and give a formal proof of the safety property.

B *The OTS/CafeOBJ method*

We introduce some notion and notation of the OTS/-CafeOBJ method including the proof score method [1].

A CafeOBJ specification consists of modules. A CafeOBJ module $\mathtt{SP} = (\Sigma_{\mathtt{SP}}, E_{\mathtt{SP}})$ consists of its signature and axioms. A signature $\Sigma_{\mathtt{SP}} = (S_{\mathtt{SP}}, \le_{\mathtt{SP}}, F_{\mathtt{SP}})$ consists of a set of sorts, an ordering on the sorts and an $S_{\mathtt{SP}}$-sorted set of operations. A $\Sigma_{\mathtt{SP}}$-algebra A is an algebra which has a carrier set A_s for each $s \in S_{\mathtt{SP}}$ and an operation $A_f : A_{s_0} \times \cdots \times A_{s_n} \to A_s$ for each $f \in (F_{\mathtt{SP}})_{s_0 \ldots s_n s}$. An axiom $l = r$ if $c \in E$ is a conditional equation whose both sides l and r of the equation are terms of a same sort and condition is a term of boolean sort constructed from the operations in $\Sigma_{\mathtt{SP}}$ and variables. A SP-algebra is a $\Sigma_{\mathtt{SP}}$-algebra which satisfies all equations in $E_{\mathtt{SP}}$. When SP has tight denotation, it denotes the initial SP-algebra. When SP has loose denotation, it denotes all SP-algebras.

The following is a loose module specifying an arbitrary set with a binary predicate.

```
mod* PID {  [ Pid ]
  op _=_ : Pid Pid -> Bool {comm} }
```

A loose module (`mod*`) denotes all models satisfying axioms. Module `PID` denotes an arbitrary set with a binary relation. The following is a tight module specifying labels of a traffic signal.

```
mod! LABEL{  [Label]
  ops gr re ye : -> Label
  pred _=_ : Label Label {comm}
  op next : Label -> Label
  eq (L:Label = L) = true . eq (gr = re) = false .
  eq (gr = ye) = false .   eq (re = ye) = false .
  eq next(re) = gr .       eq next(gr) = ye .
  eq next(ye) = re .  }
```

A tight module (`mod!`) denotes the initial model. In the initial mode, any elements of a carrier set is represented by a term constructed from its signature, and no two elements of a carrier set are equivalent unless the corresponding terms can be shown to be equal using its axioms. Module `LABEL` has three constant operators `gr`, `ye` and `re` of `Label`, a binary predicate `_=_`, and a unary operation `next` on `Label`. The first four equations define the equality predicate, which

takes two labels and returns true if they are same, otherwise false. The unary operation `next` returns a next label, defined by `re` ⇒ `gr` ⇒ `ye` ⇒ `re`.

An OTS/CafeOBJ specification consists of data modules and a system module. Modules `LOC` and `PID` are examples of data modules. A system module is given as a behavioral specification of CafeOBJ. A behavioral specification has a special sort, called a hidden sort, and special operations, called behavioral operations, whose arguments include the hidden sort. A behavioral operation whose returned sort is not hidden is called an observation. A behavioral operation whose returned sort is hidden is called a transition. Two elements of the hidden sort are observationally equivalent if their observed values are equivalent for each observation. An OTS/CafeOBJ specification is a restricted behavioral specification, where observational equivalence is preserved by transitions.

The following is an OTS/CafeOBJ specification of a signal control system:

```
mod* SIGNAL{  pr(LABEL)  *[Sys]*
  op init : -> Sys  bop color : Sys -> Label
  bop change : Sys -> Sys
  eq color(init) = gr .
  eq color(change(S:Sys)) = next(color(S)) . }
```

Module `SIGNAL` imports module `LABEL` with the protecting mode, where a model of the importing module includes a model of the imported module as it is. Hidden sort `Sys` is declared, which denotes the state space of a system to be specified. Constant `init` is declared as an initial state. Observation `color` observes a color, where term `color(`s`)` represents the current color in state s of the signal control system. Transition `change` changes a color, where term `change(`s`)` represents the state obtained after changing.

The first equation `color(init) = gr` specifies the initial state through observation such that the initial color is green. The second equation specifies the behavior of transition `change` through observation. Term `change(S)` represents the state obtained by applying `change` to `S`. The color of `change(S)` is defined as the next color of `S`. For example, `color(change(change(init)))` is equivalent to `re`.

III. OTS/CafeOBJ specifications of hybrid systems

In this section, we introduce a way to describe OTS/-CafeOBJ specifications of hybrid systems. We model signal control systems with a single car and with plural cars as hybrid automata, and describe them as OTS/CafeOBJ specifications.

A An OTS/CafeOBJ specification of a signal control system with a single car

In our OTS model of the signal control system, there are three observations for discrete locations and three observations for continuous variables. Observations *color*, *going* and *cs* observe the value of elements of a location. Observations *now*, *pos* and *l* observes the value of those variables. There are two kinds of transitions in OTS models: discrete and continuous transitions. Discrete transitions *go*, *stop*, *in*, *out* and *change* correspond to edges. A continuous transition $tick_t$ advances the system's time by $t \in \mathbb{Q}^*$.

Table 1 shows the correspondence between the hybrid automaton in Section 1 and our OTS/CafeOBJ specification.

We give a system module `SIGNAL` which imports the built-in module `RAT` of rational numbers and `LABEL` given in the previous section. The following is a part of module `SIGNAL` specifying the initial state.

```
  eq now(init) = 0 .          eq pos(init) = 0 .
  eq going(init) = false .    eq cs(init) = false .
  eq color(init) = gr .       eq l(init) = 0 .
```

The initial values of *now*, *pos* and *l* are defined as `0`. The initial values of *going* and *cs* are false. The initial color is green. Module `SIGNAL` includes declaration `ops cs0 cs1 t0 : -> Rat` of constants cs_0, cs_1 and t_0.

Transition *change* is specified as follows:

```
   eq c-change(S) = l(S) <= now(S) .
  ceq change(S) = S if not c-change(S) .
  ceq color(change(S)) = next(color(S))
      if c-change(S) .
  ceq l(change(S)) = now(S) + t0 if c-change(S) .
   eq now(change(S)) = now(S) .  ...
```

The effective condition `c-change(S)` is defined in the first equation such that $l \leq now$. The second (conditional) equation specifies that the state is unchanged if *change* is not effective. The third and fourth equations specify the updated values of *color* and *l* such that *color* becomes the next color

TABLE 1. THE CORRESPONDENCE BETWEEN OUR HYBRID AUTOMATON AND OTS/-CAFEOBJ SPECIFICATION

Hybrid Automaton	OTS/CafeOBJ specification
Loc	`color, going, cs`
Edg	`go, stop, in, out, change`
X	`now, pos, l`
Init	`init`
Inv	`c-tick`
Flow	observed values after `tick`
Jump	effective conditions and observed values after transitions

*We assume a temporal domain is dense, that is, there is a point between any pair of points. We use rational numbers for the clock values.

and l is set to t_0 time later. The remaining equations specify other variables are unchanged.

Transition *in* is specified as follows:

```
  eq c-in(S) = (cs0 = pos(S) and color(S) = gr) .
 ceq cs(in(S)) = true if c-in(S) .
```

Transition *in* is effective when the car exists at cs_0 and the signal is green. When *in* is effective, `cs(in(S))` becomes true, that is, location $(\_, \_, 0)$ is changed into location $(\_, \_, 1)$. The other discrete transitions are defined similarly.

Next, we specify the continuous transition. Time advancing $tick_r$ is described as follows:

```
 ceq now(tick(X,S)) = now(S) + X if c-tick(X,S) .
 ceq pos(tick(X,S)) =
     (if going(S) then pos(S) + X else pos(S) fi)
     if c-tick(X,S) .
```

Term `tick(`r`,`s`)` is the result state of applying $tick_r$ to state s. Since $now' = 1$, the value of *now* increases by $1 \times r$. If we have more complex differential equations in *Flow*, equations become larger.

For a given $r \in$ `Q`, when $tick_r$ is done, the current time *now* increases by r. The position *pos* increases by r if *going* is true, that is, the car moves forward. The effective condition `c-tick` is given by invariants of the hybrid automaton.

```
eq c-tick(X,S) = 0 <= X and X <= cs1 - cs0
 and (cs(S) and going(S) implies
      cs0 <= pos(S) + X and pos(S) + X <= cs1)
 and ...
 and (cs1 < pos(S) + X implies not cs(S)) .
```

Invariants in hybrid automata should always hold, which means that time cannot advance if the invariants do not hold. Thus, the effective condition `c-tick` is given as the conjunction of all invariants.

B *An OTS/CafeOBJ specification of a signal control system with plural cars*

Consider the case that more than one cars appear in our signal control system. The system is an example of multitask hybrid systems. In OTS models of multitask systems, observations and transitions related to processes are parameterized, for example, pos_p and go_p are an observation and a transition for a process p respectively. In OTS/CafeOBJ specifications, multitask systems are described with the import of loose module `PID`. For example, pos_p is given by `pos : Pid Sys -> Rat`.

We give a system module `MS` which imports data modules `RAT`, `LABEL` and `PID` for a signal control system with plural cars. Observations and initial state of module `MS` are specified as follows:

```
eq now(init) = 0 .    eq going(P,init) = false .
eq pos(P,init) = 0 .  eq cs(P,init) = false .
eq color(init) = gr . eq l(init) = 0 .
```

where observations `pos`, `going`, `cs` related to cars are parameterized. Equation `pos(P,init) = 0` means that the initial positions are zero for all cars `P`, for example. The definition of discrete transitions are modified as follows:

```
  eq c-in(P,S) =
     (cs0 = pos(P,S) and color(S) = gr) .
 ceq in(P,S) = S if not c-in(P,S) .
  eq cs(P',in(P,S)) = P' = P or cs(P',S)  . ...
```

In the third equation above, term `cs(`p'`,in(`p`,`s`))` means that the value of $cs_{p'}$ at the result state of applying in_p to state s for processes p and p'. Thus, when $p = p'$, it is true otherwise it is unchanged.

The effective condition of $tick_r$ should check all invariants for all processes. First, we parameterize `c-tick(P,X,S)` by processes `P` as follows:

```
eq c-tick(P,X,S) =
 (cs(P,S) and going(P,S) implies
  cs0 <= pos(P,S) + X and pos(P,S) + X <= cs1)
 and ...
 and (cs1 < pos(P,S) + X implies not cs(P,S)) .
```

We give a way to describe multitask hybrid systems in OTS/CafeOBJ specifications which denote hybrid automata without fixed numbers of processes. We introduce a specification `PSET` of a set of processes:

```
mod* PSET{  [Pid < PSet]
 op _ _ : PSet PSet -> PSet {assoc comm idem}
 op nil : -> PSet  pred _in_ : Pid PSet
 vars P Q : Pid  var PS : PSet
 eq (P in (P PS)) = true .
 eq (P in nil) = false .  eq (P in Q) = (P = Q) .
 eq (P in (Q PS)) = (P = Q) or (P in PS) . }
```

Sort `PSet` is declared as a super sort of `Pid`, and the sequence of two elements of `PSet` is also an element of `PSet`. Thus, a sequence of `Pid` is a term of `PSet`, for example, `p1 p2 p3` is a term of `PSet` when `p1`, `p2` and `p3` are terms of `Pid`. Operation `_in_` denotes the membership predicate on `PSet`.

We also introduce an observation *ps* which is a set of active processes. A process becomes active when it moves. After a process becomes active, it is active until it stops. The initial value of *ps* is empty. When a car p starts to move, *ps* is updated by $p\,ps$. The following is a part of description related to *ps*:

```
  eq ps(init) = nil .  eq ps(go(P,S)) = P ps(S) .
```

The effective condition `c-tick` is defined for *ps*. The effective condition of $tick_r$ is defined on *ps* inductively:

```
  eq c-tick(nil,X,S) = true .
  eq c-tick(P PS,X,S) = c-tick(P,X,S) and
                        c-tick(PS,X,S) .
  eq c-tick(X,S) =  0 <= X and X <= cs1 - cs0 and
                    c-tick(ps(S),X,S) .
```

For example, `c-tick(p q r,`x`,`s`)` = `c-tick(p,`x`,`s`)` and `c-tick(q,`x`,`s`)` and `c-tick(r,`x`,`s`)`.

IV. Verification of multitask hybrid systems

In this section we give a formal proof of the safety property. We declare the relationship between constants cs_0, cs_1 and t_0 such that $0 < cs_0 < cs_1$ and $cs_1 - cs_0 \leq t_0$. Then, the specification denotes all models satisfying the above condition including the test model in the previous section. First, we give a state predicate $inv_1(p, s)$ such that the car p does not exist in the critical section when the signal is red at s.

```
eq inv1(P,S) = not (color(S) = re and
    cs0 < pos(P,S) and pos(P,S) < cs1) .
```

If we prove `inv1(`p`,`s`)` for all processes p and all reachable state s from the initial state, the safety property holds. We denote the set of all reachable states by RS. We prove $\forall s \in RS.\forall p \in A_{\mathtt{Pid}}.inv_1(p, s)$ by the induction on structure of reachable terms.

A A proof passage for the induction on the reachable state space

As the induction basis, we apply the reduction command to `inv1(p,init)` to prove the initial state to satisfy inv_1, where `p` is a fresh constant as an arbitrary element. CafeOBJ interpreter returns `true`, which implies the induction basis holds, that is, inv_1 holds at the initial state.

In the induction step, we prove that each transition preserves the state predicate. We first assume an arbitrary state `s` satisfies `inv1(p,s)` for all processes `p`. Then, under the assumption we prove `inv1(p,s')` for the state `s'` obtained by any transition. The following is a template module for the induction step.

```
eq istep1(P:Pid) = inv1(P,s) implies inv1(P,s') .
```

The proposition `istep1(P)` means that the proposition `inv1(P,s')` holds under the induction hypothesis `inv1(P,s)`. The following is a part of the induction steps with respect to $tick_{t_1}$, where `s'` is obtained by applying $tick_{t_1}$ to `s`.

```
eq s' = tick(t1,s) .  red istep1(p) .
```

If the above reduction returns `true`, it guarantees that implication $inv_1(p, s) \Rightarrow inv_1(tick_{t_1}(p, s))$ holds. Unfortunately, it returns neither `true` nor `false` for the above proof passage. We need to give more information for proofs. A typical proof strategy is a case splitting by the effective condition as follows:

```
eq c-tick(t1,s) = false .  eq s' = tick(t1,s) .
red istep1(p) .
```

```
eq c-tick(t1,s) = true .  eq s' = tick(t1,s) .
red istep1(p) .
```

If the above two proof passages both return `true`, the original proof passage is satisfied, since $(c - tick(t_1, s) \Rightarrow istep1(p)) \wedge (\neg c - tick(t_1, s) \Rightarrow istep1(p)) \Rightarrow istep1(p)$. If it does not so, we proceed case-splitting more.

B Lemma introduction

Repeating the process of case splitting, we may face `false` as a returned value of the reduction command. The following is such an example.

```
eq cs(p,s) = true .  ...
eq cs0 <= pos(p,s) = false . eq s' = tick(t1,s) .
red istep1(p) .
```

Since cs_p is true, the position pos_p is greater than or equal to cs_0. The equation `cs0 <= pos(p,s) = false` contradicts to it. Such a proof passage represents unreachable states.

For such a case, we introduce another appropriate safety property, called a lemma. We add the following lemma to the module `INV` and `ISTEP`:

```
eq inv3(P,S) = cs(P,S) implies
   cs0 <= pos(P,S) and pos(P,S) <= cs1 .
eq istep3(P) = inv3(P,s) implies inv3(P,s') .
```

The lemma `inv3(P,S)` denotes that the position of car p is between cs_0 and cs_1 whenever cs_p is true. By replacing the reduction command by adding the lemma, we obtain `true` for the above proof passage.

```
red inv3(p,s) implies istep1(p) .
```

Proceeding the case splitting and introducing lemma, we obtain `true` for all the remaining proof passages for inv_1.

C Proving lemma

We proved invariant inv_1 for all states reachable from the initial state by the induction scheme under the assumption of some lemmata. To complete the proof, we need to show (1) the lemmata hold for the initial state, and (2) the lemmata hold for result state of applying every transition to states satisfying inv_1 and the lemmata. In the other words, we make a proof score of the conjunction of $inv_1 \wedge \cdots \wedge inv_n$, where the induction base is represented by $inv_1(p, init) \wedge \cdots \wedge inv_n(p, init)$ and the induction step is represented by $inv_1(p, s') \wedge \cdots \wedge inv_n(p, s')$ under the induction hypothesis $inv_1(p, s) \wedge \cdots \wedge inv_n(p, s)$. Note that in the previous section, we prove $inv_3(p, s) \Rightarrow (inv_1(p, s) \Rightarrow inv_1(p, s'))$. The formula is equivalent to $(inv_1(p, s) \wedge inv_3(p, s)) \Rightarrow inv_1(p, s')$. If we prove $(inv_1(p, s) \wedge inv_3(p, s)) \Rightarrow inv_3(p, s')$ for lemma inv_3, we obtain $(inv_1(p, s) \wedge inv_3(p, s)) \Rightarrow inv_1(p, s') \wedge inv_3(p, s')$.

To complete a proof of inv_1, we make seven lemmas and 136 proof passages[†], all of which return `true`. The following is the declaration of the lemmata `inv2`, `inv3`, `inv4`, `inv5`, `inv6` and `inv7`.

```
eq inv1(P,S) = not (color(S) = re and
  cs0 < pos(P,S) and pos(P,S) < cs1) .
eq inv2(P,S) = not (cs(P,S) and
  pos(P,S) < cs1 and color(S) = re) .
eq inv3(P,S) = cs(P,S) implies
  cs0 <= pos(P,S) and pos(P,S) <= cs1 .
eq inv4(P,S) = cs(P,S) and not color(S) = gr
  and l(S) <= now(S) implies cs1 <= pos(P,S) .
eq inv5(P,S) = cs(P,S) and not color(S) = gr
  implies cs1 - pos(P,S) <= l(S) - now(S) .
eq inv6(P,S) = cs(P,S) or cs0 = pos(P,S) or
  going(P,S) implies P in ps(S) .
eq inv7(P,S) = cs0 < pos(P,S) and pos(P,S) < cs1
  implies cs(P,S) .
```

V. Related work

There are several tools for analyzing and/or verifying hybrid systems: MATLAB & Simulink[‡] HSolver[§], HyTech[¶], KeYmaera[‖], PHAVer[**] and so on. See the literature [5] for more details. One of the most relevant tools to our study is Maude a language and tool supporting specification description and verification based on rewriting logic [6]. Both Maude and CafeOBJ are algebraic specification languages and support user-defined abstract data type specifications, which is an advantage over the other tools for hybrid systems. Real-time Maude [7] is an extension of Maude which supports formal specification and analysis of real-time and hybrid systems. HI-Maude [8] is another extension of Maude which deals with a wider range of hybrid systems, called interacting hybrid systems. System modules in Maude are based on rewriting logic, where systems transitions are described by rewrite rules. Verification in Maude is based on exhaustive searching for reachable spaces obtained by the rewrite rules. In Maude, systems with discrete and continuous variables can be described, however, only discrete time domains obtained by time sampling strategies can be verified by search and model checking.

One of our advantages against these model checking approaches is that proof scores guarantee that verified properties hold for an arbitrary number of multiple processes. To make state spaces finite, model-checking approaches should restrict the size of the system to finite. Although model checking is fully-automated, the proof score method is semi-automated and needs a human interaction to complete proofs. In the literature [9], automated support of making proof scores has been proposed. A case splitting phase may be automated, however, lemma discovery is heuristic and not easy to be automated.

VI. Conclusion

We described an observational transition system of a simple signal control system with plural cars as an example of multitask hybrid systems, and verified some safety property by the proof score method. One of our future work is to apply the proposed method to practical applications of multitask hybrid systems, such as real-time operating systems, automotive control systems, intelligent transport systems, and so on.

[†]Besides them, we need to add some lemmata which can be proved without induction on reachable states, e.g. `eq lemma1(P,X,S) = (P in ps(S)) and c-tick(X,S) implies c-tick(P,X,S)`. We proved it by the induction on structure of terms of sort `PSet`.

[‡]https://jp.mathworks.com/products/simulink.html

[§]http://hsolver.sourceforge.net/

[¶]https://ptolemy.berkeley.edu/projects/embedded/research/hytech/

[‖]http://symbolaris.com/info/KeYmaera.html

[**]http://www-verimag.imag.fr/%7Efrehse/phaver_web/index.html

References

[1] K. Ogata, and K. Futatsugi, Proof scores in the OTS/-CafeOBJ method, FMOODS 2003, LNCS 2884, pp.170-184. Springer, 2003.

[2] K. Ogata and K. Futatsugi, Modeling and verification of real-time systems based on equations, Science of computer programming, 66(2), pp.162-180, Elsevier, 2007.

[3] M. Nakamura, S. Higashi, K. Sakakibara and K. Ogata, Formal verification of Fischer's real-time mutual exclusion protocol by the OTS/CafeOBJ method, SICE 2020, pp.1210-1215, 2020.

[4] K. Ogata, D. Yamagishi, T. Seino and K. Futatsugi, Modeling and verification of hybrid systems based on equations, DIPES 2004, pp.43-52, 2004.

[5] L. Doyen, G. Frehse, G. J. Pappas, and A. Platzer, Verification of hybrid systems, Handbook of model checking, pp.1047-1110, Springer, 2018.

[6] P. C. Ölveczky and J. Meseguer, Semantics and pragmatics of Real-Time Maude, Higher-order and symbolic computation 20, pp.161-196, Springer, 2007.

[7] D. Lepri, E. Ábrahám and P. C. Ölveczky, Timed CTL model checking in real-time Maude, WRLA 2012, LNCS 7571, pp.182-200, Springer, 2012

[8] M. Fadlisyah and P. C. Ölveczky, The HI-Maude tool, CALCO 2013, LNCS 8089, pp.322-327, Springer, 2013.

[9] D. Gaina, D. Lucanu, K. Ogata, K. Futatsugi, On automation of OTS/CafeOBJ method. SAS 2014, LNCS 8373, pp.578-602, Springer, 2014.