Formal Specification and Verification of an Autonomous Vehicle Control System by the OTS/CafeOBJ method

Yifan Wang Masaki Nakamura Kazutoshi Sakakibara Yuki Okura

Toyama Prefectural University, Toyama, Japan

Abstract- **The autonomous vehicle control system is a typical kind of hybrid system that combines both continuous and discrete behavior. Formal specification and verification techniques help us to verify desired properties of given systems. In this study, we propose a way to describe a formal specification of an autonomous vehicle control system in CafeOBJ algebraic specification language. The control system is a hybrid system with continuous variables of time, velocity, and position controlled by discrete pedal actions including acceleration, braking, and no-operation. We also verify the safety property of the autonomous vehicle control system by a theorem proving technique called the proof score method** [*]

Keywords-Autonomous vehicle; Hybrid system; Formal verification; Observational transition system; Proof score method

I. Introduction

A hybrid system is a dynamic system that includes both continuous and discrete dynamic behavior. The autonomous vehicle has both continuous behaviors (velocity and position) and discrete behaviors (pedal actions), therefore, in order to design and verify an autonomous vehicle control system we model it as a hybrid system. There are many techniques used to test and simulate autonomous vehicles, such as CarMaker [†] and SUMO [‡], but only testing and simulation just provide some kinds of limited or predetermined paths, it is far from enough for safety. We need formal specification and verification techniques to make sure safety of the autonomous vehicle control system.

Formal verification is an approach to verify that a given specification satisfies some desired properties formally. One approach of formal verification is model checking. In our previous work [1], we proposed a way to describe and verify an autonomous vehicle group control system by Maude model checker and showed Maude is useful to design the system with the safety property. Although model checking is fully automated, the state space is limited where we choose a time sampling strategy instead of dense time.

The other approach of formal verification is theorem proving where mathematical proofs are made by the interaction of humans and computers, it is semi-automated but applicable to infinite state space and it supports continuous variables. CafeOBJ supports specification execution based on a rewrite theory for theorem proving. 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 based on the specification execution, called the proof score method [2].

There have been several case studies of the OTS/-CafeOBJ method which deal with systems with only discrete behaviors ([2] and so on). In [3], the OTS/CafeOBJ method is applied to distributed real-time systems, which are kinds of hybrid systems with only one continuous variable of time. In [4], the OTS/CafeOBJ method is applied to multitask hybrid systems, which includes only a simple hybrid system for explanation's sake. In [5], the safety property of an autonomous vehicle intersection traffic control system by the OTS/CafeOBJ method is shown, where the system includes only one continuous variable of real-time.

In this study, we propose a way to describe a formal specification of an autonomous vehicle control system in CafeOBJ algebraic specification language. The control system is a hybrid system with continuous variables of time, velocity, and position controlled by discrete pedal actions including acceleration, braking, and no-operation. We also verify the safety property of the autonomous vehicle control system by a theorem proving technique called the proof score method.

II. Hybrid automaton

Hybrid automata are models of hybrid systems which contain the discrete and continuous behavior and we give a model of an autonomous vehicle control system as a hybrid system according to the literature [6]. In this article, we consider a hybrid automaton of a single autonomous vehicle control system, represented in Figure 1. The system con-

[*]DOI reference number: 10.18293/SEKE2023-170.
This work was supported by JSPS KAKENHI Number JP19K11842.

[†]https://ipg-automotive.com/cn/products-solutions/software/carmaker/
[‡]https://www.eclipse.org/sumo/

sists of a single autonomous vehicle with three locations: acceleration, nothing, and brake. The pedal states of acceleration, brake, and no-operation (nothing) can be selected freely.

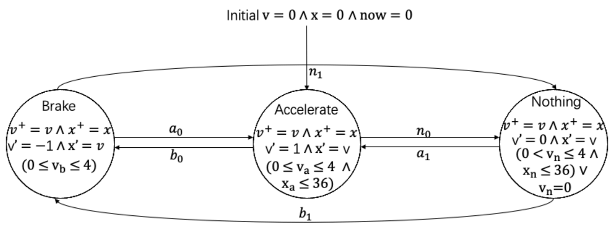

Figure 1. A hybrid automaton of an autonomous vehicle control system

We assume that there will be an obstacle at a certain position and the vehicle must jump into the brake mode and the pedal state is fixed to the braking to prevent to crash when the vehicle is close to the obstacle which is represented in Figure 2.

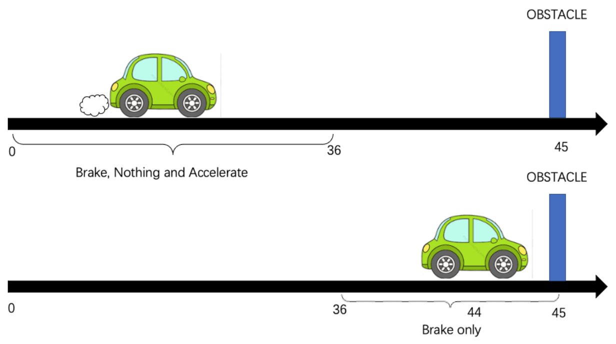

Figure 2. A target model

The location `Brake` stands for the state where the brake pedal is pushed ($v' = -1$). The location `Nothing` stands for the state where neither the brake nor the accelerator pedals are pushed ($v' = 0$). The location `Accelerate` stands for the state where the accelerator pedal is pushed ($v' = 1$).

The initial condition $v = 0 \wedge x = 0 \wedge now = 0$ for the location `Accelerate` means that when the system starts, the state of the vehicle is acceleration, the velocity and position of the vehicle are 0 respectively. We assume the vehicle does not goes back and have a max velocity, that is, we give the limitation of the velocity of vehicle between the ($0 \leq v \leq 4$) at three locations `Brake`, `Nothing` and `Accelerate`.

And there is an obstacle at position 45 and to avoid collision, the position should be less than or equal to 36 ($x \leq 36$) at locations `Nothing` and `Accelerate`. If the position is over 36, the vehicle has to jump into location *Brake* only and vehicle has to reduce the velocity ($v' = -1$) and stop before position 44 with just the brake pedal is pushed, that is, the vehicle does not exist the position over 44 so that prevents the crash with the obstacle. We call it the safety property.

The operational semantics of hybrid automata is given as a state transition system where a state is represented by a tuple of a location and values of variables and there are discrete jump transitions between locations and continuous flow transitions for time elapsing (See [1] for more details). The safety property of our system is given as follows formally: for any reachable state, the position of the vehicle is less than forty-five, that is the obstacle position.

III. OTS/CafeOBJ specification

In this section, we introduce a way to describe the OTS/CafeOBJ specifications of a hybrid system. We model an autonomous vehicle control system with a single vehicle as hybrid automata and describe them as OTS/CafeOBJ specifications. Then we give the simulation of our system and certify that the vehicle will stop before the obstacle.

A CafeOBJ specification consists of modules, in which sorts, operators, and equations are declared in module `LABEL`, module `RAT`, and module `VEHICLE`. A module is declared with `module` or just `mod`. The name of a module is written after `mod`. Module elements are declared between {}. CafeOBJ modules can be classified into tight modules and loose modules. Tight and loose modules begin with `mod!` and `mod*` respectively. A loose module `mod*` denotes all models satisfying axioms. A tight module `mod!` denotes the initial model.

A Data modules

An OTS/CafeOBJ specification consists of data modules and a system module. We first give a data module `LABEL` for the vehicle.

```
mod! LABEL{  [Label]
 ops accel nothing brake : -> Label
 pred _=_ : Label Label {comm}
 var L : Label . eq (L = L) = true .
 eq (accel = nothing) = false .
 eq (accel = brake) = false .
 eq (brake = nothing) = false .
}
```

The name of the module is `LABEL`. The module declaration with `mod!` denotes the tight denotation, where the module denotes only the initial model. In the initial mode, any elements of a carrier set are 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.

We set a tight data module `LABEL` specifying three constant operators `accel`, `nothing`, and `brake` of `Label`, a binary predicate `_=_` , and a variable is declared with the keyword var. And we define the equality predicate, which takes two labels and returns true if they are the same, otherwise false.

B System modules : Signature

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 and 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 an autonomous vehicle system:

```
mod* VEHICLE{
 pr(LABEL + RAT)  *[ Sys ]*
 op init : -> Sys          bop loc1 : Sys -> Label
 bops a b n : Sys -> Sys
 bops x1 v1 now : Sys -> Rat
 bop tick : Rat Sys -> Sys
 op c-tick : Rat Sys -> Bool
 var S : Sys                vars V A X T Y Z : Rat
```

The loose module VEHICLE imports module LABEL and RAT 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 loc1 observes a location where the term loc1(s) represents the current location in state s of the vehicle control system. Transition a, b, and n change a system, where term a(s) represents the state s obtained after changing. Observation x1 and v1 observe the current position and speed. The clock observer now observes the current time. The term tick (X,S) represents the state s obtained after advancing time by X. The operation c-tick specifies that a sequence of Rat and Sys is a term of Bool. Variables used in equations can be declared beforehand. A variable S is declared with the keyword var as a system. Plural variables of the same sort such as V, A, X, T, Y, and Z can be declared with the keyword vars as a rational number.

C System modules : Axioms

The initial state of the current time (now), velocity (v1), and position (x1) are defined as zero respectively, the initial state of the location (loc1) is defined as acceleration (accel) by the following equations.

```
eq now(init) = 0 .  eq v1(init) = 0 .
eq x1(init) = 0 .  eq loc1(init) = accel .
```

The transitions of a, b, n, and tick are specified as O($\tau(S)$) = S when O = x1, v1, now and τ = a, b, n, which means the values of now, v1 and x1 will not change with the pedal actions. The following equations specify the updated values of location after the pedal actions.

```
eq loc1(a(S))= accel. eq loc1(b(S))= brake.
eq loc1(n(S))= nothing.
```

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

```
ceq now(tick(T,S)) = now(S) + T if c-tick(T,S) .
ceq loc1(tick(T,S)) = loc1(S) if c-tick(T,S) .
```

The first equation specifies the updated value of now is set to T time later, that is, $now^{+} = now + T$ if the effective condition is satisfied. The second equation specifies the variable *loc* is unchanged if it satisfies the c-tick.

In the OTS/CafeOBJ method, we describe an equation like x(tick(T,S)) = rhs (right-hand side) if c-tick(T,S). x is the observation for a continuous variable. The term tick(T,S) stands for the result state by applying tick(T,_) to the state S, that is, the state after time advancing by T from S. The left-hand side x(tick(T,S)) of the equation stands for the value of x for the state tick(T,S). By this equation, the value of observation (tick(T,S)) is defined as the right-hand side when c-tick(T,S) is true. The value of x after T is obtained from the flow condition of the hybrid automaton.

Let v1(t) be the value of v1 at time t. From the flow condition $v_1' = a$ where a is a constant (a = -1, 0, 1), the value v1(T) of v1 at the state after T is calculated as follows: $v_1(T) = v_1(0) + \int_0^T adt = v_1(0) + [at]_0^T = v_1(0) + a * T$ where the value of v1(0) is the value of v1 at S. Thus, the right-hand side of the equation whose left-hand side is v1(tick(T,S)) can be written as v1(S) + a1(loc1(S)) * T, where a1(accel) = 1, a1(brake) = -1 and a1(nothing) = 0. To describe the equation, we introduce an operation nextv(V,A,T) which calculates the value of v1 after T from the state whose value of v1 is V and acceleration value is A.

```
eq nextv(V,A,T) = V + A * T .
ceq v1(tick(T,S)) = nextv(v1(S), a1(loc1(S)), T)
                                      if c-tick(T,S) .
```

Similarly, x1(t) is the value of x1 at time t. From the flow condition $x_1' = v$, the value x1(T) of x1 at the state after T is calculated as follows: $x_1(T) = x_1(0) + \int_0^T v(t)dt = x_1(0)+[v_0+at]_0^T = x_1(0)+v_0*T+\frac{1}{2}*a*T^2$, where the value of x1(0) is the value of x1 at S. Thus, the right-hand side of the equation whose left-hand side is x1(tick(T,S)) can be written as $x_0 + v_0 * T + \frac{1}{2} * a_1(loc1(S)) * T^2$. To describe the equation, we introduce an operation `nextx(X,V,A,T)` which calculates the value of `x1` after `T` from the state whose value of `x1` is `X` and acceleration value is `A`.

```
eq nextx(X,V,A,T) = V * T + 1/2 * A * T * T + X .
ceq x1(tick(T,S)) =
nextx(x1(S), v1(S), a1(loc1(S)), T) if c-tick(T,S).
```

The above conditional equations have `c-tick(T,S)` as their conditions. The effective condition `c-tick(T,S)` is given by invariants of the hybrid automaton which `eq c-tick(T,S) =` inv_b `and` inv_a `and` inv_n.

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. For example, the inv_b equals `(loc1(S) = brake implies (0 <= nextv(v1(S), a1(loc1(S)), T) and nextv(v1(S), a1(loc1(S)), T) <= 4)))` means that $0 \leq nextv \leq 4$ holds after $tick_x$ whenever the location is the braking state. The effective condition of $tick_x$ should check invariant $loc1(S)$ by the updated value of *nextv* and *nextx*. In other words, if future values violate invariants, time cannot advance.

```
ceq tick(T,S) = S if not c-tick(T,S) .
```

IV. Verification of hybrid system

A single reduction can prove a simple equation. More complex properties are proved by combining several reductions, which is called a proof score. First, we give a state predicate `inv1(S)` such that the vehicle does not exist over the position of 45 at the state `S`.

```
 eq inv1(S) = (x1(S) < 45) .
```

If we prove `inv1(S)` for all reachable states from the initial state and get the result is true, the safety property holds. As the induction basis, we apply the reduction command to `inv1(init)` to prove the initial state to satisfy `inv1`.

```
open INV .
red inv1(init) .
close
```

CafeOBJ interpreter returns true, which implies the induction basis holds, that is, `inv1` holds at the initial state.

To complete the proof, we need to prove not only the safety property `inv1` but also a lemma `inv2`. We make two lemmas and 12 proof passages, all of which return true. Because of the page limitation, we omit the induction part of `inv1`, an introduction of a lemma `inv2` and its proof. The codes of them can be found in our GitHub page [§].

[§]https://github.com/evan-jaaapan/SEKE2023.git

V. Conclusion

We described an observational transition system of an autonomous vehicle control system as an example of a hybrid system, and verified the safety property by the proof score method.

One of our future works is to apply the proposed method to practical applications of multitask hybrid systems with multiple autonomous vehicles. There are several related work on formal verification for hybrid system, e.g. SpaceEx[¶] and KeYmaera X[‖]. Another one of our future work is to compare them with our approach and combine them to obtain more efficient formal verification.

References

[1] Wang Y, Nakamura M, Sakakibara K. Modeling an Autonomous Vehicle Group Control System as a Hybrid Automaton and its Specification and Verification in Rewriting Logic. In2021 36th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC) 2021 Jun 27 (pp. 1-4). IEEE.

[2] Ogata K, Futatsugi K. Proof scores in the OTS/CafeOBJ method. InFMOODS 2003 Nov 19 (Vol. 3, pp. 170-184).

[3] Ogata K, Futatsugi K. Modeling and verification of real-time systems based on equations. Science of computer programming. 2007 Apr 30;66(2):162-80.

[4] Nakamura M, Sakakibara K, Okura Y, Ogata K. Formal verification of multitask hybrid systems by the OTS/CafeOBJ method. International Journal of Software Engineering and Knowledge Engineering. 2021 Dec;31(11n12):1541-59.

[5] Igarashi T, Nakamura M, Sakakibara K. Formal Verification of the Lim-Jeong-Park-Lee Autonomous Vehicle Control Protocol using the OTS/CafeOBJ Method.(2022): 574-579.

[6] Doyen L, Frehse G, Pappas GJ, Platzer A. Verification of hybrid systems. Handbook of Model Checking. 2018:1047-110.

[¶]http://spaceex.imag.fr/

[‖]https://keymaerax.org/publications.html