HHML: A Hierarchical Hybrid Modeling Language for Mode-based Periodic Controllers

Zhiming Hu^1, Zheng Wang^{2,3}, Hongjian Jiang^1, Yuyuan Zhang^2 and Yongxin Zhao^{1*}
^{1} Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
^{2} Beijing Sunwiseinfo Technology Ltd, China
^{3} Beijing Institute of Control Engineering, China

Abstract—**In cyber-physical systems, the controllers are widely designed into mode-based periodic modules, which are used to control physical plants. Such a system can be modeled as a hybrid one, i.e., one of the real-time controller programs and interactive continuous plants that obey dynamical laws. In this work, to facilitate the modeling and analysis of periodic hybrid control systems in the field of aerospace and smart cities, a hierarchical hybrid modeling language (HHML) is proposed, which contains a two-hierarchy structure, i.e., mode-hierarchy and module-hierarchy. The former supports modeling a hybrid system at the abstraction level, while the latter is used to describe the behavior of the modules. The operational semantics is investigated for formal analysis, and the translation rules to hybrid automaton are explored for formal verification. A case study is conducted with the lunar lander to demonstrate the effectiveness of the approach.**

Index Terms—**Hybrid systems, Cyber-physical system, Formal semantics, Model verification**

I. Introduction

A hybrid system [1] is an interactive system of real-time controllers programs and continuous plants that obey dynamical laws. Such systems are pervasively applied in aerospace, smart cities, and automotive industry, etc. In cyber-physical systems [2], the embedded software and its operating environment have the characteristics of high complexity, uncertainty and high real-time requirements. In general, the controllers are designed into mode-based periodic modules to monitor and control the evolution of physical plants. Formal analysis for such mode-based periodic controllers is still an enormous challenge due to very complicated combinations of computation and control, and high safety requirements of system designs.

Recently, there are a number of formal methods developed for hybrid systems, which can be divided into three main categories: automata [3], process algebra [4] and state diagrams [5]. In automata model, each state contains a large number of differential equations, invariants, and transitions with reset operations. However, the automata model cannot achieve good scalability and composability. In cyber-physical systems, however, a practical model may have hundreds of controllers and sensors. Therefore, automata may not be the suitable choice. As a theoretical basis of formal analysis, process algebra is difficult to be accepted by a wide range of practitioners due its complicated symbols and mathematical logic. The lack of good readability causes even simple controllers to require the help and guidance from experts in the formal field. Hence, the method based on process algebra still needs more improvements. State diagram is a commonly modeling method in the industry of which the typical representative is Simulink/Stateflow. In general, it has the characteristics of a high-level programming language. With the support of tools, the modeling, simulation, verification of hybrid systems can be effectively achieved. However, there are few related researches on its formal semantics.

In this paper, we are motivated by the above methods to put forward a hierarchical hybrid modeling language HHML for the mode-based periodic controllers, which contains mode-hierarchy and module-hierarchy to make the model more scalable and composable. Mode-hierarchy supports modeling a hybrid system at the abstraction level and facilitates the graphical representation of the model that is easy to understand. Module-hierarchy is used to describe the behaviors of the modules, which contains the time predicates unique to the period model. The operational semantics is investigated for formal analysis, and the translation rules to hybrid automaton are explored for formal verification. The contributions of this work are the followings:

- **Hierarchical hybrid modeling language**. The language provides hierarchy model to support the hybrid systems. The discrete mode has periodicity and supports nesting to describe the control system. The continuous mode represents the physical world by ordinary differential equations. Furthermore, the operational semantics in the mode-hierarchy and module-hierarchy are explored respectively, which helps developers to understand and ensures the correctness of the model built.
- **End-to-end translation.** Several translation rules of transformation from HHML models to hybrid automata are provided to support property verification in tool Flow*. An illustrative example of a lunar lander is used to demonstrate the feasibility of translation rules.

The rest of the paper is organised as follows. Related work of hybrid system modeling and verification is introduced in Section II. A modeling language HHML is proposed in Section III. In Section IV, the operational semantics is presented. The hybrid automatic translation rules are given in Section V. An example of lunar lander is shown in Section VI. Section VII concludes the paper and introduces the future work.

*Corresponding author: yxzhao@sei.ecnu.edu.cn
DOI reference number: 10.18293/SEKE2021-149

II. Related work

This section mainly gives some brief introduction to typical hybrid system modeling methods and common verification tools, which provides a reference for this work.

A. Hybrid system modeling methods

Formal methods to model hybrid systems are in progress for many years. The two which have most influenced our language are hybrid action system [6] and Zélus [7].

Hybrid action system was proposed by Mauno et al. It maps continuous-time events to model variables in the form of data flows, so that the overall system behavior can be split into independent differential equations. However, hybrid action system does not support the stepped refinement development methods in action systems, which limits the types of modeling systems. Our discrete mode is hierarchical, so it is suitable for modeling large-scale hybrid systems.

Zélus was a synchronous language with ordinary differential equations proposed by Benveniste et al. It reuses the principles and compilation techniques developed for synchronous languages, extending them to deal with models that mix discrete and continuous-time. However, it uses the type systems to separate discrete and continuous calculations, while we use a clearer discrete mode and continuous mode to distinguish, which makes the interaction between the controller and physical world easier. On the other hand, our language is aimed at periodic controllers, and each discrete mode can have periodicity and use periodicity-related predicates.

B. Hybrid system verification tools

There are many tools that can model and verify specific types of hybrid systems nowadays. Traditional tools include d/dt [8], HyTech, etc., and newer tools contain Flow* [9], SpaceEx [10], etc. They mainly use hybrid automata as the underlying semantic model of hybrid systems. In order to enable the proposed hybrid modeling method to complete the verification of the property in these verification tools, there are many researches on translating from hybrid modeling languages to hybrid automata. For example, in [11], a subset of Simulink language was proposed to translate into hybrid automata. In [12], the conversion rule from ECML to SpaceEx model was proposed. Its essence is the translation of part of ECML to linear hybrid automata.

The HHML model was translated into the above four tools respectively, with requirements being satisfied potentially, to verify the common examples of hybrid system. Taking the running time, running scale and supported operation format into consideration, Flow* is finally chosen.

III. Hybrid description language

This section proposes the hybrid description language HHML to provide the rich control logic, and events that can drive the conversion among different modes. HHML is a two-hierarchy structure containing mode-hierarchy and module-hierarchy. The former one supports to model a hybrid system at the abstraction level, while the latter one describes the detailed behavior of modes.

A. Mode-hierarchy syntax

The following syntax elements are in support of the modeling of the hybrid system architecture at an abstract level:

$$\begin{aligned}
&HModel ::= (Dictionary, Modes)\\
&Dictionary ::= \{var \mid var = (name, attri, type, initval)\}\\
&Modes ::= (dModes, cModes)\\
&dMode ::= (name, period, (dflow \mid dModes), dTrans)\\
&cMode ::= (name, cflow, cTrans)\\
&dTran ::= (dm, priority, dguard, dm')\\
&cTran ::= (cm, priority, cguard, cm')\\
&dguard ::= cond \mid Duration(cond, c) \mid After(cond, c)\\
&cguard ::= When(cond)
\end{aligned}$$

$HModel$ indicates the hybrid model which is composed of $Dictionary$ and $Modes$. $Dictionary$ is the set of four-tuple variables var, where, $name$ is the label of the variable, $attri$ is the attribute which can be continuous, discrete and constant, $type$ is the basis type such as Boolean, int, float, and $intval$ means the initial value.

$Modes$ illustrates the behavior of the hybrid system which is made up of discrete modes $dModes$ and continuous modes $cModes$. A discrete mode $dMode$ is used to describe the control system which consists of label $name$, $period$, discrete control flow $dflow$ or sub-modes $dModes$, and discrete transitions $dTrans$. A continuous mode $cMode$ denotes the changes in the physical world with the differential equations, which consists of $name$, continuous statements $cflow$ and continuous transitions $cTrans$.

$dTran$ is the transfer relationship between discrete modes including the source mode dm, $priority$, guarded conditions $dguard$, and target mode dm'. $Duration$ and $After$ are HHML's special time predicates in $dguard$ to express the property of periodicity based on the basic Boolean expression $cond$ and constant c. $Duration(cond, c)$ is true in a period p if the first c periods within the current period meet $cond$. $After(cond, c)$ is true in a period p if there is another period p' such that $cond$ can be satisfied in period p' and it travels c periods from p' to p.

$cTran$ is similar to $dTrans$, where $cguard$ denotes the continuous guarded condition, cm and cm' denote the source and target continuous mode. $cguard$ uses $When(cond)$ to mean that the system will always wait for $cond$ to be satisfied. The difference between $dTrans$ and $cTrans$ is that $cTrans$ is not controlled by the period. Therefore the transition occurs immediately when $cguard$ is met, and time predicates are missing in $cguard$.

B. Module-hierarchy syntax

Module-hierarchy is divided into discrete flow $dflow$ and continuous flow $cflow$, which denotes the calculation process of the hybrid model, and the following elements are used to specify the behavior of the hybrid system.

$$\begin{aligned}
&dflow ::= declare \mid stmts \mid dflow; dflow\\
&stmts ::= pstmt \mid cstmt\\
&pstmt ::= x := stmt \mid x \leftarrow cv \mid call\ \mathsf{func} \mid skip \mid \quad \perp\\
&cstmt ::= stmt; stmt \mid while\ cond\ do\ stmts \mid\\
&\qquad\quad if\ cond\ then\ stmts\ else\ stmts\\
&cflow ::= eq\ \ until\ cond\\
&eq ::= der\ v = expr \mid eq \parallel eq \mid Idle
\end{aligned}$$

$dflow$ represents the execution task and calculation process of the discrete mode, including local declarations $declare$, control statements $stmts$ and the combinations of $dflow$. The control statements $stmts$ consist of primitive statements $pstmt$ and compound statements $cstmt$. $pstmt$ has the following types: assignment $x := stmt$, sampling of continuous variables $x \leftarrow cv$, function call $call$ func, nil statement $skip$ and divergence $\perp$. $cstmt$ contains three basic control structures, namely sequential composition, iteration and conditional.

$cflow$ describes changing laws of the physical world in continuous modes, which is constructed by $until$ operator inserted between differential equation eq and termination conditions $cond$. eq uses explicit ordinary differential equations $der\ v = expr$ to express the changing laws. Furthermore, eq can be a combination of multiple equations, where $Idle$ is a special case denoting that the continuous variables remain unchanged.

IV. Operational semantics of HHML

This section displays the operational semantics of HHML according to the transition system. They are divided into two semantics based on the mode-hierarchy and the module-hierarchy respectively.

A. Operational semantics of the mode-hierarchy

HHML supports periodic modeling of hybrid systems where the discrete mode will be executed at a specific point in time, and the continuous mode will change with time. It is assumed that there is an operating system that supports a hybrid system's modes to be executed. The semantics of the mode-hierarchy can be described into a five-tuple shown as below:

$$mode\_config ::= (cm, dm, l, per, Tr)$$

where:

- cm represents the continuous mode of the system.
- dm stands for the discrete mode of the system.
- $l \in \{Begin, Execute, End\}$ indicates the stage of the discrete mode which the system is in. $Begin$ means the beginning of the period which is mainly used for data sampling. $Execute$ expresses the preparation to perform periodic tasks, and End determines whether a transition occurs at the end of the period.
- per denotes the counter of period.
- Tr represents the historical value sequence of the variable in the discrete mode, which is used to judge the guarded condition of the transition.

State transition rules of the mode-hierarchy are listed in TABLE I.

(Sample) indicates that in the initial stage of each period of the discrete mode, the system will sample the continuous variables required in the mode and Tr is updated to Tr'. Then system phase will change from $Begin$ to $Execute$.

(Enter_sub) means that if the discrete mode contains sub-modes, the system will immediately enter the sub-modes from the current mode.

TABLE I: State Transition Rules at the Mode-Hierarchy

(Sample)	$\dfrac{dflow(dm) \neq empty}{(cm, dm, Begin, per, Tr) \rightarrow (cm, dm, Execute, per, Tr')}$
(Enter_Sub)	$\dfrac{dflow(dm) = empty}{(cm, dm, Begin, per, Tr) \rightarrow (cm, dm', Begin, per, Tr)}$
(Excute)	$\dfrac{execute(dm.dflow, Tr) = Tr'}{(cm, dm, Exceute, per, Tr) \rightarrow (cm, dm, End, per, Tr')}$
(no_dTran)	$\dfrac{\forall tran \in dTrans \cdot \neg(Tr \models tran.dguard)}{(cm, dm, End, per, Tr) \rightarrow (cm, dm, Begin, per+1, Tr)}$
(dTran)	$\dfrac{\exists tran \in dTrans \cdot ((Tr \models tran.dguard) \wedge Hpri(tran))}{(cm, dm, End, per, Tr) \rightarrow (cm, dm', Begin, 1, Tr)}$
(cTran)	$\dfrac{\exists tran \in cTrans \cdot ((Tr \models tran.cguard) \wedge Hpri(tran))}{(cm, dm, l, per, Tr) \rightarrow (cm', dm, l, per, Tr)}$

(Execute) describes that when the discrete mode enters the execution phase, the system processes the variables according to the discrete control flow and then changes from the $Execute$ phase to the End phase. $execute(dm.dflow, Tr) = Tr'$ means the result of the execution.

(no_dTran) indicates that when none of the discrete transition conditions can be met, the discrete mode which the system is in remains unchanged, and the phase of the system is updated from End to $Begin$. Then the period is increased by 1.

(dTran) shows that when at least one discrete transition condition is met, the system will select the highest priority for transition. The current discrete mode dm will transfer to the target discrete mode dm', and the operating period will be reset to 1. $Hpri(tran) = \forall tran' \in dTrans \cdot (tran' \neq tran \wedge Tr \models tran'.dguard \wedge tran'.priority > tran.priority)$ is defined to denote the highest priority for transition.

(cTran) expresses that the continuous mode will only transfer to the target continuous mode cm' with the highest priority $Hrpi(tran)$ when the transfer conditions are met. The transition will occur immediately when the $cguard$ is met, because the $cguard$ is always in a $wait$ state.

B. Operational semantics of the module-hierarchy

A triple $module\_config$ is used to represent the semantics of the module-hierarchy as shown below.

$$module\_config ::= (\sigma, stmts, status)$$

where:

- σ stands for the set of variables.
- $stmts$ represents the statements to be executed.
- $status$ denotes the states of the module layer, and it includes three states: $term$, $wait$ and div. State $term$ expresses that the previous statement runs successfully and you can continue to execute the current statement. State $wait$ indicates that the previous statement is blocked and a guard event is needed to activate the system. State div means that the system has an error and cannot execute subsequent programs.

TABLE II: State Transition Rules of the Module-Hierarchy

(Assign)	$\frac{}{(\sigma, x := stmt, term) \rightarrow (\sigma[stmt/x], \varepsilon, term)}$	**(Sample)**	$\frac{}{(\sigma, x \leftarrow v, term) \rightarrow (\sigma[v/x], \varepsilon, term)}$
(Div)	$\frac{}{(\sigma, \bot, term/wait) \rightarrow (\sigma, \bot, div)}$	**(Loop1)**	$\frac{\sigma \models cond}{(\sigma, while\ cond\ do\ stmts, term) \rightarrow (\sigma, stmts; while\ cond\ do\ stmts, term)}$
(Func)	$\frac{excute(\mathsf{func})(\sigma) = \sigma'}{(\sigma, call\ \mathsf{func}, term) \rightarrow (\sigma', \varepsilon, term)}$	**(Loop2)**	$\frac{\sigma \not\models cond}{(\sigma, while\ cond\ do\ stmts, term) \rightarrow (\sigma, \varepsilon, term)}$
(Skip)	$\frac{}{(\sigma, skip, term) \rightarrow (\sigma, \varepsilon, term)}$	**(Cond1)**	$\frac{\sigma \models cond}{(\sigma, if\ cond\ then\ stmt_1\ else\ stmt_2, term) \rightarrow (\sigma, stmt_1, term)}$
(Seq)	$\frac{(\sigma, stmt_1) \rightarrow (\sigma', stmt_1')}{(\sigma, stmt_1; stmt_2, term) \rightarrow (\sigma', stmt_1'; stmt_2, term)}$	**(Cond2)**	$\frac{\sigma \not\models cond}{(\sigma, if\ cond\ then\ stmt_1\ else\ stmt_2, term) \rightarrow (\sigma, stmt_2, term)}$
(EQ1)	$\frac{\sigma \models cond}{(\sigma, eq\ until\ cond, wait) \rightarrow (\sigma, \varepsilon, term)}$	**(EQ2)**	$\frac{\sigma \not\models cond}{(\sigma[v(t)], eq\ until\ cond, term/wait) \rightarrow (\sigma[v(t+\delta)], eq\ until\ cond, wait)}$

The state transition rules of the Module-Hierarchy are given in Table II.

(Assign) and **(Sample)** respectively denote the assignment of discrete variables and the sampling of continuous variables. If the current state is $term$, the operation can be performed, and the variables in the variable set σ are modified. **(Func)** means that calling the function $func$ which makes the current variable set σ become σ' and sets the currently executed statement to empty. **(Skip)** means doing nothing. **(Div)** indicates that the system lies in a divergent state when $\bot$ holds.

(Seq) describes the systems executing the sequential statements. **(Loop1)** and **(Loop2)** respectively represent that the loop statement holds or not. **(Loop1)** denotes that the statements are in the loop and the entire loop will be executed as sequential statements, while **(Loop2)** denotes the termination of loop.

(Cond1) and **(Cond2)** are two transition rules to denote the if statement, where the former holds when the condition is true and the latter holds with false condition.

(EQ1) denotes the condition is meet and the $until$ statement is executed. While the condition is unsatisfied, **(EQ2)** denotes the system must wait for δ periods till the condition holds.

V. Translations of HHML

This section introduces the modeling essence of Flow* and proposes some translation rules to translate the model established by HHML into a hybrid automaton. According to the generated automaton, formal verification of safety and reachability has been successfully implemented by the tool Flow*.

A. Hybrid automata

Flow* [9] works on systems that can be modeled by hybrid automata. Hybrid automata can be expressed as:

$$(loc, var, inv, flow, trans, guards, resets, init)$$

where:

- loc is a finite set of continuous states, also called modes.
- var consists of several real-valued variables.
- inv means the invariant of each mode.
- $flow$ represents the continuous dynamics defined by ordinary differential equations of each mode.
- $trans$ is the set of possible transition between modes.
- $guards$ is the set of transition conditions between modes.
- $resets$ assigns a reset map to a jump. After a jump occurs, the values of the continuous variables will be updated according to the reset mapping.
- $init$ denotes the initial of the automaton.

B. Translation rules

To simplify the expression of hybrid automata, we use the $jumps$ set to denote the union set of $guards$, $resets$ and $trans$. $jump$ represents an execution of system resets.

$$jumps ::= \{jump \mid jump = \{l_{begin}, l_{end}, guard, reset\}\}$$

Therefore, hybrid automata can be expressed as a six-tuple .

$$(loc, var, inv, flow, jumps, init)$$

Now, the translation rules containing the variables, discrete modes, continuous modes and some flows in module-hierarchy are shown in turn.

*1) **Variables:*** Variables v are translated by the below rule where using ”$-$” to denote the unchanged elements.

$$Tr(v) = (-, var \cup v, -, -, -, init \cup v.inival)$$

The variables in HHML are divided into continuous, discrete and constant. The types include integer, floating-point and Boolean. The variables in Flow* are unified as floating-point continuous state variables. Therefore, the Boolean variables are transformed to 1/0 and other variables are converted into floating-point. Then these variables can be converted into state variables in the hybrid automaton directly. To reduce the number of translated variables, we will change the constants to values. Assigning initial values $inival$ to variables in HHML is corresponding to the initial variables $init$ in hybrid automata.

Since there is no discrete modes in hybrid automata, the names of discrete modes are added as variables into var. We use flag 1/0 to distinguish whether the system lies in the discrete mode. Finally, time term t is added to record the period in automata whose initial value is set to 0.

*2) **Discrete modes:*** The discrete mode supports mode nesting in HHML which may contain several sub-modes. So before translating, the discrete mode has to be flattened, i.e., all discrete modes after simplification do not contain submodes, and keep the semantic consistency during the translation. Flattened discrete modes can be described as below.

$$dmodes' = \{dm \mid dm.dflow \neq empty \wedge dm \in dmodes\}$$

Since loc is a set of states in the hybrid automata, the discrete modes need to be translated into each state. A certain state $l \in loc$ is used in the translation rules and other states are the same. The following rule is about when the discrete mode transfers.

$$\begin{aligned}
&Tr(dm) = (-,-,-, flow \cup t' = 1, jumps \cup jps, -)\\
&\text{where } jps = \{jp \mid jp = (l, l, (t \geq dm.period; dm.name\\
&== 1; dguard), (dm.name = 0; dm'.name = 1; dm'.dflow;\\
&t = 0)) \wedge (dm, -, dguard, dm') \in dm.dTrans\}
\end{aligned}$$

In order to translate $dm.period$, $t' = 1$ is added to each $flow$ to represent the periodic process. Condition $t \geq dm.period$ is attached to $guard$, and t will be set to 0 in $resets$ to indicate the end of the period. $dguard$ is converted into $guards$ to translate the conditions of discrete transfer process. Once the transition occurs, the discrete control flow of the target mode $df.dflow$ is executed, and the current mode is modified to dm'. There is no transition between continuous modes, hence the target state is still the source state l.

The following rule is about when the discrete mode does not transfer.

$$\begin{aligned}
&Tr(dm) = (-,-,-, flow \cup t' = 1, jumps \cup jps, -)\\
&\text{where } jps = \{jp \mid jp = (l, l, t \geq dm.period;\\
&\qquad dm.name == 1, dm.dflow; t = 0)\}
\end{aligned}$$

The rule indicates that when the discrete mode does not transfer, it will execute $dm.dflow$ and then enter the next period at the end of the period.

3) ***Continuous modes:*** Continuous modes are translated by the rule as follows.

$$\begin{aligned}
&Tr(cm) = (loc \cup cm.name, -, inv \cup cm.cond,\\
&\qquad flow \cup cm.eq, jumps \cup cm.cguards, -)
\end{aligned}$$

As mentioned above, the continuous mode is a triple in HHML. The corresponding translation is performed between the hybrid automata and the continuous mode. The continuous modes' $name$, differential equation eq and termination condition $cond$ will be translated to loc, $flow$ and inv in hybrid automata respectively. The transfer between continuous modes is equivalent to the $jumps$ behavior in the hybrid automaton, while $resets$ in $jumps$ does nothing during the transfer.

4) ***Some flows in module-hierarchy:*** Since Flow* only supports part of the discrete control flow, conditional statement along with the time predicates $Duration$ and $After$ is translated to enable expressive models to be verified.

First, the translation rule of conditional statement $if\ cond\ then\ stmt_1\ else\ stmt_2$ in $dflow$ can be expressed as:

$$\begin{aligned}
&Tr(dflow.cd) = (-,-,-,-, jumps \cup jps, -)\\
&\text{where } jps = (l, l, cond, stmt_1) \cup (l, l, \neg cond, stmt_2)
\end{aligned}$$

$dflow$ that contains the conditional statement will be split into two, and conditional statements will be replaced with $stmt_1$ and $stmt_2$ and set to the $reset$ respectively. The corresponding $cond$ and $\neg cond$ are added to the $guards$.

Next, rules of time predicates $Duration$ and $After$ necessary for modeling periodic hybrid systems are introduced. They only focus on changing the $guards$ and $resets$ in the $jumps$ behavior.

$$\begin{aligned}
&Tr(dflow.Duration(cond, c)) = (-, var \cup cnt, -, -,\\
&jumps \cup jps, init \cup cnt = 0)\\
&\text{where } jps = (-,-, cond, cnt = cnt + 1) \cup (-,-,\\
&\qquad \neg cond, cnt = 0) \cup (-,-, cnt \geq c, -)
\end{aligned}$$

$$\begin{aligned}
&Tr(dflow.After(cond, c)) = (-, var \cup cnt, -, -,\\
&jumps \cup jps, init \cup cnt = 0)\\
&\text{where } jps = (-,-, cond, cnt = cnt + 1) \cup (-,-,\\
&\qquad cnt > 0, cnt = cnt + 1) \cup (-,-, cnt \geq c, -)
\end{aligned}$$

An additional count variable cnt is introduced here. For $Duration$, when a period of the discrete mode ends, if $cond$ is true, cnt will increase by 1, otherwise it will be reset to 0. When $cnt \geq c$, the expression of $Duration(cond, c)$ is true. Similarly, for $After(cond, c)$, when $cond$ is true or $cnt > 0$, cnt will increase by 1 at the end of the period, and when $cnt \geq c$, the expression is true.

VI. Case study

In this section, the process of lunar lander's slow descent is modeled into the hybrid system by HHML. Then, the model is translated into a hybrid automaton and Flow* carries out the reachability analysis towards it.

A. Model

The model analyzed in this case study is taken from the descent guidance control program of a lunar lander in [13]. In brief, it is a sampled data control system composed of physical devices and control programs. The thrust exerted on the lander is constantly adjusted by the system to ensure that the lander remains stable during the slow descent phase. So that it enters the free fall phase smoothly and completes the landing. For the specific meaning and value of each parameter, please refer to [13]. This paper only models from the parameter level.

The hybrid system is divided into the current stage of the guidance program and the lander dynamics. The guidance program (i.e. discrete mode) modeled by HHML is shown in top half of Figure 1. The slow descent phase of the guidance program is executed periodically in a sampling period. At each sampling point, various sensors will sample the current state of the lander. The sampled values will be calculated in the guidance program, and the control command will be output, which will then affect the dynamics of the lander. When more than 10 seconds have passed during the slow descent phase (approximately 80 periods), and the height of the lander is less than 6 meters, the system will switch from the slow descent phase to the free fall phase and send out the signal.

Furthermore, the lander dynamic (i.e. continuous mode) modeled by HHML is shown in bottom half of Figure 1 which is considered only in the vertical direction. $dynamic\_1$ and $dynamic\_2$ indicate the change of the lander under different thrusts. After receiving the signal to change to free fall, the dynamics will change to $dynamic\_3$ to indicate free fall.

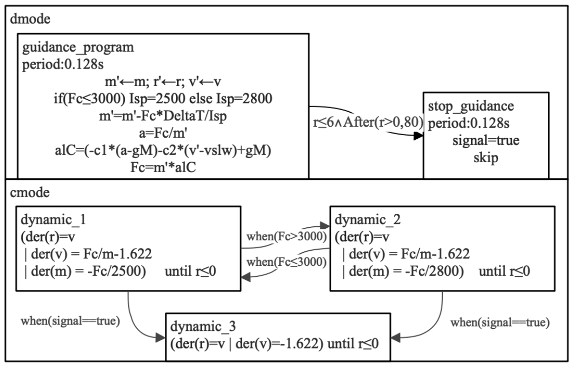


Fig. 1: Lunar Lander Modeled in HHML

B. Translation and verification

The translated model shown as Fig.2 will be verified with regard to three properties [13] in the following.

First, the speed fluctuation of model is supposed to satisfy $|v - vlsw| < 0.05$. Then, the lander will eventually reach the surface of the moon. Finally, the speed of the lander should not exceed $vMax$ when arriving at the destination.

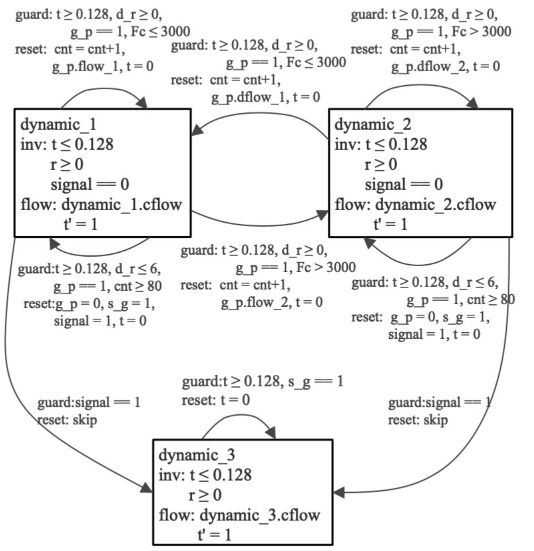
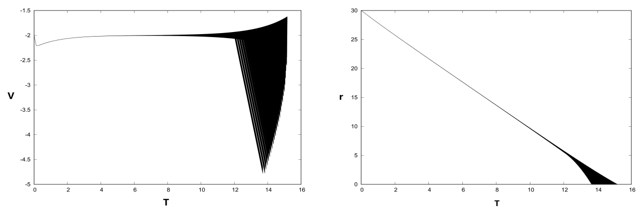


Fig. 2: Translated Mode in Hybrid Automata

The computation costs 9 minutes on the platform with 2.4 GHz Intel Core i5 CPU and 16GB RAM running macOS. The reachable sets of the translated model are given in Figure 3.

VII. Conclusion and future work

This paper has introduced a hierarchical hybrid modeling language (HHML) for periodic controllers. The language uses periodic and hierarchical discrete modes to formalize the control system, and continuous modes to model the physical environment. In addition, translation rules help translate the



(a) T_v_plot (b) T_r_plot

Fig. 3: Reachable Sets Given by Flow*

model into hybrid automata, and implement verification of the properties on the verification tool Flow*. The verification results show that the lander can finally reach the lunar surface smoothly and safely.

In the future, we plan to apply HHML to more cases in smart cities, and support more verification tools.

Acknowledgements

This work is supported by National Key Research and Development Program (2019YFB2102600), Lab of High Confidence Embedded Software Engineering Technology, Beijing Insititue of Control Engineering, No: HCESET-2019-1.

References

[1] L. P. Carloni, R. Passerone, and A. Pinto, *Languages and tools for hybrid systems design*. now Publishers Inc, 2006, vol. 1.

[2] K. Ghorbal, J.-B. Jeannin, E. Zawadzki, A. Platzer, G. J. Gordon, and P. Capell, "Hybrid theorem proving of aerospace systems: Applications and challenges," *Journal of Aerospace Information Systems*, vol. 11, no. 10, pp. 702–713, 2014.

[3] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid systems*. Springer, 1992, pp. 209–229.

[4] F. Wang, Z. Cao, L. Tan, and Z. Li, "Formal modeling and performance evaluation for hybrid systems: a probabilistic hybrid process algebra-based approach," *arXiv preprint arXiv:2012.12716*, 2020.

[5] J. B. Dabney and T. L. Harman, *Mastering simulink*. Pearson, 2004.

[6] M. Rönkkö, A. P. Ravn, and K. Sere, "Hybrid action systems," *Theoretical Computer Science*, vol. 290, no. 1, pp. 937–973, 2003.

[7] T. Bourke and M. Pouzet, "Zélus: A synchronous language with odes," in *Proceedings of the 16th international conference on Hybrid systems: computation and control*, 2013, pp. 113–118.

[8] E. Asarin, T. Dang, and O. Maler, "The d/dt tool for verification of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2002, pp. 365–370.

[9] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 258–263.

[10] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 379–395.

[11] A. Agrawal, G. Simon, and G. Karsai, "Semantic translation of simulink/stateflow models to hybrid automata using graph transformations," *Electronic Notes in Theoretical Computer Science*, vol. 109, pp. 43–56, 2004.

[12] S. Yoon and J. Yoo, "Formal verification of ecml hybrid models with spaceex," *Information and Software Technology*, vol. 92, pp. 121–144, 2017.

[13] H. Zhao, M. Yang, N. Zhan, B. Gu, L. Zou, and Y. Chen, "Formal verification of a descent guidance control program of a lunar lander," in *International Symposium on Formal Methods*. Springer, 2014, pp. 733–748.