HHL Prover: An Improved Interactive Theorem Prover for Hybrid Systems

Shuling Wang, Naijun Zhan, and Liang Zou

State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences

Paris, November 2015
Hybrid systems exhibit combinations of discrete jumps and continuous evolution, many of which are safety-critical.
Verification Approaches

- Reachability analysis and model checking
  - Hybrid automata + temporal logic based specification languages + model checkers;
  - Abstractions or numerical approximations.
Verification Approaches

- **Reachability analysis and model checking**
  - Hybrid automata + temporal logic based specification languages + model checkers;
  - Abstractions or numerical approximations.

- **Deductive verification**
  - A formal modelling language with (de-)compositionality and a specification logic for verifying the corresponding models;
  - The differential invariants are the key for verifying differential equations.
Related Work

- Reachability analysis and model checking
  - \(\frac{d}{dt}\) [Asarin, Bournez, et al.], reachability analysis of hybrid systems with linear continuous dynamics and uncertain bounded input;
  - iSAT-ODE [Eggers, Ramdani, et al.], a numerical SMT solver based on interval arithmetic that conducts bounded model checking;
  - Flow* [Chen, Ábrahám, et al.], computing the over-approximations of the reachable sets of hybrid systems in a bounded time;
  - ...
Related Work

- Reachability analysis and model checking
  - \(d/dt\) [Asarin, Bournez, et al.], reachability analysis of hybrid systems with linear continuous dynamics and uncertain bounded input;
  - iSAT-ODE [Eggers, Ramdani, et al.], a numerical SMT solver based on interval arithmetic that conducts bounded model checking;
  - Flow* [Chen, Ábrahám, et al.], computing the over-approximations of the reachable sets of hybrid systems in a bounded time;
  - ...

- Deductive verification
Contributions of this Work

An interactive theorem prover, called HHL prover, for verifying hybrid systems that are

- modelled by hybrid CSP (HCSP), an extension of CSP [Hoare] with differential equations for describing hybrid systems, and
- specified by hybrid Hoare Logic (HHL), an extension of Hoare logic with history formulas for reasoning about HCSP models.
Contributions of this Work

An interactive theorem prover, called HHL prover, for verifying hybrid systems that are

- modelled by hybrid CSP (HCSP), an extension of CSP [Hoare] with differential equations for describing hybrid systems, and
- specified by hybrid Hoare Logic (HHL), an extension of Hoare logic with history formulas for reasoning about HCSP models.

HHL prover is based on the mechanization of HCSP and HHL in the proof assistant Isabelle/HOL.
This is an improved HHL prover of a previous version [Zou, Wang, Zhan, 2013]:

- HHL verification framework: shallow embedding in Isabelle/HOL, reducing the proof effort to a big extent;
- We re-verify a real-world example: the slow descent guidance control program of a lunar lander, as an illustration.
Outline

1. Preliminaries: HCSP and HHL
2. HHL prover, as an Embedding in Isabelle/HOL
3. Case Study: the Control Program of a Lunar Lander
4. Concluding Remarks
Hybrid CSP (HCSP) [He&Zhou, 1994] is an extension of CSP by introducing timing constructs, continuous evolution and interrupts.
Hybrid CSP (HCSP) [He&Zhou, 1994] is an extension of CSP by introducing timing constructs, continuous evolution and interrupts.

HCSP inherits from CSP the communication-based parallelism:

- Message-passing by \texttt{ch!}e and \texttt{ch?}x.
- A communication is a synchronisation of \texttt{ch!}e and \texttt{ch?}x.
- The parallel composition \( P \parallel Q \) behaves as if \( P \) and \( Q \) run independently except that the communications along the common channels connecting \( P \) and \( Q \) are to be synchronized.
**Continuous evolution** \( \langle \dot{s} = e \& B \rangle \):

- It evolves according to \( \dot{s} = e \) as long as \( B \) holds, and terminates when \( B \) turns false.
- wait \( d \triangleq t := 0; \langle t = 1 \& t \leq d \rangle \).
Communication interruption \( \langle \dot{s} = e \& B \rangle \triangleright i \in i (i o_i \rightarrow Q_i) \):

- \( i o_i \) - a set of communication events (i.e. \( ch!e \) or \( ch?q \));

- It initially proceeds like the continuous evolution \( \dot{s} = e \), and is interrupted on some communication in \( i o_i \), and then proceeds like \( Q_i \).
Communication **interruption** $\langle \dot{s} = e \& B \rangle \triangleright \left[ i \in I (io_i \rightarrow Q_i) \right]$: 

- $io_i$ - a set of communication events (i.e. $ch!e$ or $ch?qx$); 
- It initially proceeds like the continuous evolution $\dot{s} = e$, and is interrupted on some communication in $io_i$, and then proceeds like $Q_i$.

Composite constructs are taken originally from CSP:

- $P; Q$, sequential composition; 
- $B \rightarrow P$, conditional; 
- $P^*$, repetition; 
- $P \sqcup Q$, non-deterministic choice.
HCSP : An Example

A description of a continuously evolving plant with discrete control:

\[
(\langle \dot{x} = f(x, u) \& \text{True} \rangle \triangleright \square (\text{sensor}!x \rightarrow \text{actuator}?u))^* \\
\parallel (\text{wait } d; \text{sensor}?s; \text{actuator}!\text{Comp}(s))^*
\]
Hybrid Hoare Logic (HHL) for deductive verification of HCSP.

- A specification for a process $P$: $\{Pre\} P \{Post; HF\}$
  - $Pre, Post$ - pre-/post-conditions, in first-order logic (FOL);
  - $HF$ - history formula, in the interval-based Duration Calculus (DC).
Hybrid Hoare Logic (HHL) for deductive verification of HCSP.

- A specification for a process $P$: $\{\text{Pre}\} P \{\text{Post}; \text{HF}\}$
  - $\text{Pre}$, $\text{Post}$ - pre-/post-conditions, in first-order logic (FOL);
  - $\text{HF}$ - history formula, in the interval-based Duration Calculus (DC).

- The HCSP constructs are axiomatized in HHL by a set of inference rules.
  - The inference system constitutes the basis for the verification condition generator of HHL prover.
Two ways to embed the whole HHL verification framework in Isabelle/HOL:

- **Shallow embedding**
  - It defines the assertions of HHL (i.e. FOL and DC formulas) by HOL predicates on process states;

- **Deep embedding** [Zou, Wang, and Zhan, 2013]
  - It defines the FOL and DC assertions as new datatypes, and defines the meanings of the datatypes by the deductive rules (of FOL and DC resp.).

In this paper, the first approach is adopted.
HCSP expressions by datatype $\textit{exp}$, and HCSP processes by datatype $\textit{proc}$. 
HHL prover : HCSP Syntax Encoding

HCSP expressions by datatype \texttt{exp}, and HCSP processes by datatype \texttt{proc}.

Each construct of \texttt{proc} corresponds to the one in HCSP syntax:
HCSP expressions by datatype `exp`, and HCSP processes by datatype `proc`.

Each construct of `proc` corresponds to the one in HCSP syntax:

- $B \rightarrow P$ is encoded as $\text{IF } B \text{ } P$;
HCSP expressions by datatype \texttt{exp}, and HCSP processes by datatype \texttt{proc}.

Each construct of \texttt{proc} corresponds to the one in HCSP syntax:

- \( B \rightarrow P \) is encoded as \texttt{IF B P};
- \( \langle \dot{s} = e & B \rangle \) is encoded as \texttt{<s : e&&Inv&B>}, where Inv is the differential invariant of the differential equation \( \dot{s} = e \);
HCSP expressions by datatype `exp`, and HCSP processes by datatype `proc`.

Each construct of `proc` corresponds to the one in HCSP syntax:

- `B → P` is encoded as `IF B P`;
- `<s = e&B>` is encoded as `<s : e&&Inv&B>`, where `Inv` is the differential invariant of the differential equation `s = e`;
- `P*` is encoded as `P*&&Inv`, where `Inv` is the loop invariant.
HHL prover : HCSP Syntax Encoding

HCSP expressions by datatype \textit{exp}, and HCSP processes by datatype \textit{proc}.

Each construct of \textit{proc} corresponds to the one in HCSP syntax:

- \( B \rightarrow P \) is encoded as \textit{IF B P};
- \( \langle \dot{s} = e \& B \rangle \) is encoded as \textit{<s :e&&Inv&B>}, where \textit{Inv} is the \textit{differential invariant} of the differential equation \( \dot{s} = e \);
- \( P^* \) is encoded as \textit{P*&&Inv}, where \textit{Inv} is the \textit{loop invariant};
  - Both invariants are annotated for the purpose of verification.
HCSP expressions by datatype \texttt{exp}, and HCSP processes by datatype \texttt{proc}.

Each construct of \texttt{proc} corresponds to the one in HCSP syntax:

- \( B \rightarrow P \) is encoded as \texttt{IF B P};
- \( \langle \dot{s} = e \& B \rangle \) is encoded as \texttt{<s :e&&Inv&B>}, where Inv is the differential invariant of the differential equation \( \dot{s} = e \);
- \( P^* \) is encoded as \texttt{P*&&Inv}, where Inv is the loop invariant;
  - Both invariants are annotated for the purpose of verification.
- ...
Semantic functions:

- **state** - functions from variables to values, and
- **flow** - functions from time to states (recording the execution interval)
Assertion Languages : FOL

Semantic functions :

- **state** - functions from variables to values, and
- **flow** - functions from time to states (recording the execution interval)

The FOL formulas `fform` are defined as **predicates on states**,

**type_synonym** `fform = state ⇒ bool`
Semantic functions:

- **state** - functions from variables to values, and
- **flow** - functions from time to states (recording the execution interval)

The FOL formulas are defined as **predicates on states**, where

**type_synonym** \( \text{fform} = \text{state} \Rightarrow \text{bool} \)

The FOL constructs can be derived as special Isabelle functions of type \( \text{fform} \), e.g.,

**definition** \( \text{fEqual} :: \text{"exp} \Rightarrow \text{exp} \Rightarrow \text{fform}" \ ("[=]"") \) **where**

\[
\text{fEqual} (e, f) \equiv \lambda s. \text{evalE} e s = \text{evalE} f s
\]
The DC formulas $dform$ are represented as predicates on flows and time intervals,

```
type_synonym dform = flow ⇒ real ⇒ real ⇒ bool
```
Assertion Languages : DC

The DC formulas $d\text{form}$ are represented as predicates on flows and time intervals,

type_synonym $d\text{form} = \text{flow} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{bool}$

The main operators of DC include :

- $\text{elE } T$ : the length of the interval is $T$,

  definition $\text{elE } :: \text{real} \Rightarrow d\text{form}$ where
  $\text{elE } T \equiv \lambda h \ n \ m. (m-n) = T$
The DC formulas \textit{dform} are represented as \textit{predicates on flows and time intervals},

\textbf{type_synonym} \textit{dform} = \textit{flow} \Rightarrow \textit{real} \Rightarrow \textit{real} \Rightarrow \textit{bool}

The main operators of DC include:

- **\text{elE} T**: the length of the interval is \( T \),

\hspace{1cm} \text{definition} \text{elE} :: \textit{real} \Rightarrow \textit{dform} \text{ where }
\hspace{1cm} \text{elE} \ T \equiv \lambda \ h \ n \ m. (m - n) = T

- **almost \( p \)**: \( p \) holds almost everywhere in the interval,

\hspace{1cm} \text{definition} \text{almost} :: \textit{fform} \Rightarrow \textit{dform} \text{ where }
\hspace{1cm} \text{almost} \ p \equiv \lambda \ h \ n \ m. (m > n) \land (\forall a \geq n. \forall b \leq m. a < b \Rightarrow \\
\hspace{3cm} \exists t. t > a \land t < b \land p(h(t))))
Assertion Languages : DC

The DC formulas $\text{dform}$ are represented as **predicates on flows and time intervals**, 

*type_synonym* $\text{dform} = \text{flow} \rightarrow \text{real} \rightarrow \text{real} \rightarrow \text{bool}$

The main operators of DC include:

- $\text{elE } T$: the length of the interval is $T$,
  
  ```
  \text{definition elE :: real } \Rightarrow \text{dform where}
  \text{elE } T \equiv \lambda h \ n \ m. (m-n) = T
  ```

- **almost** $p$: $p$ holds almost everywhere in the interval,
  
  ```
  \text{definition almost :: fform } \Rightarrow \text{dform where}
  \text{almost } p \equiv \lambda h \ n \ m. (m>n) \land (\forall a \geq n. \forall b \leq m. a < b \Rightarrow
  \exists t. t > a \land t < b \land p(h(t))))
  ```

- $H[^]\ M$: the interval can be separated into two sub-intervals such that $H$ and $M$ hold on them resp,
  
  ```
  \text{definition chop :: dform } \Rightarrow \text{dform } \Rightarrow \text{dform ("[^]\") where}
  H[^]\ M \equiv \lambda h \ n \ m. (\exists nm. (nm \geq n \land nm \leq m \land H h n nm \land M h nm m))
  ```
Assertion Languages : FOL and DC in Deep Emb.

In deep embedding, both FOL and DC formulas are constructed in the syntax level step by step from the bottom-most expressions.

The datatype $\text{fform}$ encodes the FOL formulas:

```
\textbf{datatype} fform = [False] | exp [=] exp | exp [>] exp
| [\neg] fform | fform [\lor] fform | [\lor] string fform
```

The datatype $\text{dform}$ encodes the DC formulas:

```
\textbf{datatype} dform = [[True]] | dexp[=[ ]dexp | dexp[[< ]dexp
| almost fform | dform[^]dform [[\neg]dform | dform[[\lor]dform
```

As new datatypes, the deductive systems of FOL and DC are defined as axioms for the reasoning of FOL and DC formulas.
Specification and Inference Rules

ValidS \ p \ c \ q \ H, represents a valid specification \{p\} c \{q; H\}:

ValidS \ p \ c \ q \ H \equiv \ \forall \ now \ h \ now' \ h'. \ semB \ c \ now \ h \ now' \ h' \rightarrow h(now) \models p \\
\rightarrow (h'(now') \models q \land h', [now, now'] \models H)

- \semB \ c \ now \ h \ now' \ h', representing the big-step semantics, c starts execution from the initial flow h and time now, and terminates with flow h' and time now';
- the precondition p holds under h and now, implies that the postcondition q and the history formula H hold under h' and now'.

Mingshuai Chen Institute of Software, CAS
In Isabelle/HOL, we have proved a set of lemmas stating that all the inference rules of HHL are valid.
In Isabelle/HOL, we have proved a set of lemmas stating that all the inference rules of HHL are valid.

**lemma** ContinuousRule : \( \forall s. \)

\[
((p \rightarrow \text{Inv}) \\
[\land] (\text{exeFlow} <v :E \& \text{Inv} \& b> \text{Inv} \rightarrow \text{Inv}) \\
[\land] (\text{Inv} [\land] \text{cl}([\rightarrow]b) [\rightarrow] q)) s \\
\Rightarrow \ \forall h \ \text{now} \ \text{now}'. ((\text{elE} 0 [\forall]) \text{almost} (\text{Inv} [\&] b)) [\rightarrow] H) h \ \text{now} \ \text{now'} \\
\Rightarrow \{p\} <v :E \& \text{Inv} \& b> \{q \ ; \ H\}
\]

- Inv is indeed a **sufficiently strong invariant** (lines 1-3).
In Isabelle/HOL, we have proved a set of lemmas stating that all the inference rules of HHL are valid.

**Lemma** ContinuousRule : ∀ s.

\[(p \rightarrow Inv) \land (\text{exeFlow} <v:E&Inv&b> Inv \rightarrow Inv) \land (\text{Inv} \land cl([b] : b) \rightarrow q)) \land s \Rightarrow \forall h \text{ now now}'. ((\text{elE} 0[[\forall]]) \text{ almost} (\text{Inv} [\&] b)) [[\rightarrow]] H) \text{ h now now'} \Rightarrow \{p\} <v:E&Inv&b> \{q ; H\}

- Inv is indeed a sufficiently strong invariant (lines 1-3).
- H is implied by the strongest history formula (line 4).
Eventually, the **proof of the continuous evolution** is reduced to an equivalent **differential invariant generation problem** (a set of constraints w.r.t. $\text{Inv}$).
Eventually, the proof of the continuous evolution is reduced to an equivalent differential invariant generation problem (a set of constraints w.r.t. Inv).

HHL prover will call an external invariant generator, an oracle inv_oracle_SOS in Isabelle/HOL, to solve the invariant generation problem.
All the inference rules of HHL together constitute a verification condition generator of HHL prover for proving HCSP specifications.
Proof Process

All the inference rules of HHL together constitute a verification condition generator of HHL prover for proving HCSP specifications.

Proof Process

- first, by applying the verification condition generator, an HCSP specification is transformed step by step to a set of logical formulas (FOL and DC formulas);
- then, by applying the proof tactics and rules of HOL, the validity of these logical formulas, which is equivalent to the correctness of the original HCSP specification, is proved.
Case Study: a Lunar Lander

Control commands

Orbit-attitude dynamics

Guidance program

Navigation measurements and computation

30m
The lunar lander's **dynamics** is mathematically represented by

\[
\begin{align*}
\dot{r} &= \mathbf{v} \\
\dot{\mathbf{v}} &= \frac{F_c}{m} - gM \\
\dot{m} &= -\frac{F_c}{I_{sp_i}}
\end{align*}
\]
Sample time: 0.128s.

In every period, the guidance program
- reads $r$ and $v$ via the sensor,
- updates $m$, and calculates $F_c$.

The new thrust $F_c$ will then be used for the next sampling cycle.
Sample time: 0.128s.

In every period, the guidance program
- reads $r$ and $v$ via the sensor,
- updates $m$, and calculates $F_c$.

The new thrust $F_c$ will then be used for the next sampling cycle.

The safety property to be proved is:

$|v - v_{slw}| \leq \varepsilon$, where $\varepsilon = 0.05\text{m/s}$ is the tolerance of fluctuation of $v$ around the target $v_{slw} = -2\text{m/s}$. 
First, the **HCSP model** for the control program is constructed:

```definition
LL :: proc where
   LL ≡ PC_Init ; PD_Init ; t := (Con Real 0) ;
      (PC_Diff_f ; t := (Con Real 0) ; PD_Rep)*
```

By applying HHL prover, we have proved the following lemma:

**lemma goal**: `{fTrue} LL {safeProp ; (elE 0 [[|]] almost safeProp)}`

which indicates that, starting from any state, the control program satisfies the safety property almost everywhere during the whole execution.
Modelling and Verification in HHL Prover

First, the **HCSP model** for the control program is constructed:

```plaintext
definition LL :: proc where
   LL ≡ PC_Init ; PD_Init ; t := (Con Real 0) ;
   (PC_Difff ; t := (Con Real 0) ; PD_Rep)*
```

By applying HHL prover, we have proved the following **lemma**:

```plaintext
lemma goal : {fTrue} LL {safeProp ; (elE 0 [[[]]] almost safeProp)}
```

which indicates that, starting from any state, the control program satisfies the **safety property** almost everywhere during the whole execution.
Comparison: Shallow vs. Deep

- Both the proofs for the case study are composed of a sequence of rule applications of Isabelle/HOL.

---

1. a certified integration of third-party automated theorem provers and SMT solvers including Alt-Ergo, Z3, CVC3, and etc.
Both the proofs for the case study are composed of a sequence of rule applications of Isabelle/HOL.

The proof length in shallow embedding (about 200 lines) is about one half of the one in deep embedding.
Both the proofs for the case study are composed of a sequence of rule applications of Isabelle/HOL.

The proof length in shallow embedding (about 200 lines) is about one half of the one in deep embedding.

Shallow embedding
- The rules applied mainly comprise of two kinds: the inference rules of HHL, and the rules for unfolding the HOL predicates defining FOL and DC formulas.
- Most proofs for deciding validity of formulas can be found by the built-in tool sledgehammer\(^1\) of Isabelle/HOL automatically.

Deep embedding
- The rules applied comprise of two kinds: the inference rules of HHL, and the deductive rules of FOL and DC.
- The proof needs to be conducted by the user completely, to apply the deductive rules of both logic manually.

---

1. a certified integration of third-party automated theorem provers and SMT solvers including Alt-Ergo, Z3, CVC3, and etc..
In both embeddings, the proof in HHL prover cannot be fully automatic:

- The intermediate assertions in the SequentialRule, CommunicationRule, etc, need to be instantiated in the proof process by the user manually.
- The constraints related to unknown differential and loop invariants need to be gathered manually so that they are solved by the external invariant generator as a whole.
- In shallow embedding, due to the limitation of SMT solvers, the HOL verification conditions containing quantifiers usually cannot be proved automatically.
- In deep embedding, the FOL and DC verification conditions are proved by applying their deductive rules manually.
Remarks on Automation

In both embeddings, the proof in HHL prover cannot be fully automatic:

- The intermediate assertions in the SequentialRule, CommunicationRule, etc, need to be instantiated in the proof process by the user manually;

- The constraints related to unknown differential and loop invariants need to be gathered manually so that they are solved by the external invariant generator as a whole;

- In shallow embedding, due to the limitation of SMT solvers, the HOL verification conditions containing quantifiers usually cannot be proved automatically;

- In deep embedding, the FOL and DC verification conditions are proved by applying their deductive rules manually.
Remarks on Automation

In both embeddings, the proof in HHL prover cannot be fully automatic:

- The intermediate assertions in the SequentialRule, CommunicationRule, etc, need to be instantiated in the proof process by the user manually;
- The constraints related to unknown differential and loop invariants need to be gathered manually so that they are solved by the external invariant generator as a whole;
Remarks on Automation

In both embeddings, the proof in HHL prover cannot be fully automatic:

- The intermediate assertions in the SequentialRule, CommunicationRule, etc, need to be instantiated in the proof process by the user manually;
- The constraints related to unknown differential and loop invariants need to be gathered manually so that they are solved by the external invariant generator as a whole;
- In shallow embedding, due to the limitation of SMT solvers, the HOL verification conditions containing quantifiers usually cannot be proved automatically;
In both embeddings, the proof in HHL prover cannot be fully automatic:

- The intermediate assertions in the SequentialRule, CommunicationRule, etc, need to be instantiated in the proof process by the user manually;
- The constraints related to unknown differential and loop invariants need to be gathered manually so that they are solved by the external invariant generator as a whole;
- In shallow embedding, due to the limitation of SMT solvers, the HOL verification conditions containing quantifiers usually cannot be proved automatically;
- In deep embedding, the FOL and DC verification conditions are proved by applying their deductive rules manually.
In both embeddings, the proof in HHL prover cannot be fully automatic:

- The intermediate assertions in the SequentialRule, CommunicationRule, etc, need to be instantiated in the proof process by the user manually;

- The constraints related to unknown differential and loop invariants need to be gathered manually so that they are solved by the external invariant generator as a whole;

- In shallow embedding, due to the limitation of SMT solvers, the HOL verification conditions containing quantifiers usually cannot be proved automatically;

- In deep embedding, the FOL and DC verification conditions are proved by applying their deductive rules manually.

HHL prover is capable of modelling and verifying more complex hybrid systems, because of the expressiveness of both HCSP and HHL.
Concluding Remarks

- We present an improved HHL prover as a shallow embedding of Isabelle/HOL for verifying hybrid systems.
We present an improved HHL prover as a shallow embedding of Isabelle/HOL for verifying hybrid systems.

We apply the HHL prover on safety verification of a real-life example, i.e. the slow descent control program of a lunar lander.
We present an improved HHL prover as a shallow embedding of Isabelle/HOL for verifying hybrid systems.

We apply the HHL prover on safety verification of a real-life example, i.e. the slow descent control program of a lunar lander.

The proof results show that the shallow embedding has better performances in the proof size and automation than deep embedding.