ISCAS-SKLCS-14-05 **February, 2014**

中国科学院软件研究所
计算机科学国家重点实验室
技术报告

Formal Verification of a Descent Guidance Control Program of a Lunar Lander

by

Hengjun Zhao, Mengfei Yang, Naijun Zhan, Bin Gu, Liang Zou, Yao Chen

State Key Laboratory of Computer Science
Institute of Software
Chinese Academy of Sciences
Beijing 100190. China

Formal Verification of a Descent Guidance Control Program of a Lunar Lander

Hengjun Zhao[1], Mengfei Yang[2], Naijun Zhan[1], Bin Gu[3], Liang Zou[1], and Yao Chen[3]

[1] State Key Lab. of Computer Science, Institute of Software, CAS
[2] Chinese Academy of Space Technology
[3] Beijing Institute of Control Engineering

Abstract. In this paper, we report our recent experience in applying formal methods to the verification of a descent guidance control program of a lunar lander. The descent process of the lander is a specific hybrid system (HS) with the following three distinguished features: 1) the control program has complex data types, branching conditions and numerical computations; 2) the physical process is modelled by ODEs with general elementary functions and is frequently affected by control inputs; 3) the system suffers from various uncertainties. The verification of such a HS is beyond the capacity of many existing verification tools. Our solutions include: firstly, we build a Simulink/Stateflow model of the descent guidance control program and analyze its behaviour by simulation; then we verify its behaviour as expected in a bounded time interval by combining two bounded model checkers iSAT-ODE and Flow*; thirdly, with the tool Sim2HCSP we automatically translate the Simulink/Stateflow graphical model to a formal model given by HCSP, a formal modelling language of HSs, and then perform unbounded safety verification of the control program using HHL Prover, a theorem prover for HSs, by extending our previous work on invariant generation for polynomial HSs to HSs with general elementary functions. The descent process consists of 6 phases, of which we only focus on the slow descent phase.

Keywords: Lunar lander, formal verification, hybrid systems, reachable set, invariant

1 Introduction

Recently, the China National Space Administration (CNSA) just launched a lunar lander. The official mission objective is to achieve China's first soft-landing and roving exploration on the moon, as well as to demonstrate and develop key technologies for future missions. The scientific objectives include lunar surface topography and geology survey, lunar surface material composition and resource survey, Sun-Earth-Moon space environment detection, and lunar-based astronomical observation.

The lander entered a 100 kilometers (km)-high circular lunar orbit 6 days later after being launched. The orbit was obtained after 361 seconds of variable thrust engine braking from its single main engine. Later, the spacecraft adopted a 15km $\times$ 100km elliptic orbit. The landing took place another one week later. At periapsis, its variable thrusters were again fired in order to begin the powered descent phase.

The powered descent is the most important and risky process of the lunar lander mission. During a period of around 700 seconds (s), the lander descends from a height of 15km to the lunar surface, and at the same time decelerates from a velocity of 1.7km/s (relative to the lunar surface) to 0. The variable thrust engine can provides retro-thrust varying almost continuously between 1500 and 7500 newtons (N). The process from the ignition of the retro-thrust engine to its shutdown can be divided into 6 phases as illustrated by Figure 1.

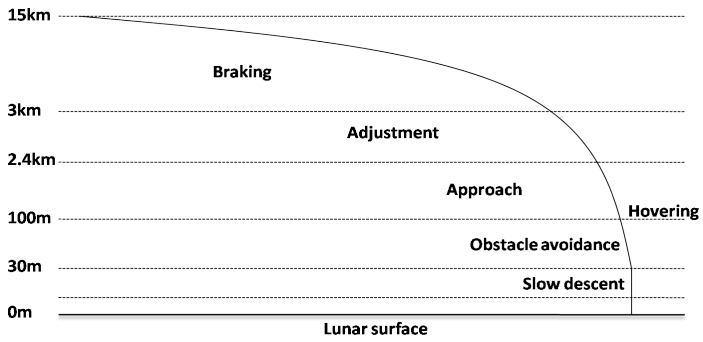

Fig. 1. The powered descent process of the lunar lander.

- **Braking** (15km to 3km): use maximum thrust for braking; at the end of this phase the velocity is slowed down to around 60m/s.
- **Adjustment** (3km to 2.4km): quickly adjust the attitude of the lander to make preparations for the following descent phases.
- **Approach** (2.4km to 100m): also called the rough obstacle avoidance phase; use sensors to recognize rocks and craters with diameters greater than 1m, and roughly select a safe landing area.
- **Hovering** (100m): keep hovering for a period of at most 30s; at the same time move horizontally to avoid obstacles and determine a final landing spot, by recognizing rocks and craters with diameters greater than 0.2m.
- **Obstacle avoidance** (100m to 30m): descend to a height of 30m right above the landing site selected in the hovering phase.
- **Slow descent** (30m to shutdown): descend slowly to the landing site; upon receiving an engine shutdown signal at several meters above the landing point, shut down the thrust engine.

The slow descent phase is followed by a free fall of the lander to the lunar surface. One of the reasons to shut down the thruster before touchdown is to reduce the amount of stirred up dust that can damage onboard instruments. The shock of landing is cushioned by four legs at the bottom of the lander.

Powered descent is the most challenging task of the lunar lander mission because it is fully autonomous. Due to communication delay, it is impossible for stations on earth to track the rapidly moving spacecraft and remote control commands from earth cannot take effect immediately. The spacecraft must rely on its own guidance, navigation and control (GNC) system to, in real time, acquire its current state, calculate control commands, and use the commands to adjust its attitude and engine thrust. Therefore the reliable functionality of GNC system is the key to the success of the lunar lander's soft-landing. In the rest of this paper we will focus on one of the 6 phases, namely the slow descent phase, to demonstrate how formal verification techniques can be applied to enhance the dependability of aerospace control programs.

The motivation of this work is to prove the correctness of the descent guidance control program controlling the soft-landing with formal methods. The general framework of our approach is as follows:

- we first build a Simulink/Stateflow model and analyze the behaviour of the control program by simulation;

– then we verify the the system's behaviour as expected in a bounded time interval by combining two bounded model checker iSAT-ODE and Flow*;
– thirdly, we automatically translate the Simulink/Stateflow graphical model to a formal model given by HCSP [5, 8] with Sim2HCSP [11, 10], and then perform unbounded safety verification of the control program using HHL Prover [9].

Coping with hybrid systems (HSs) with a large set of initial states, different data types, complex logical structures and numerical computations, general nonlinear ODEs, and frequent interactions between the continuous and discrete parts is beyond the capacity of many existing verifiation tools for HSs, either based on reachable set computation or deduction. To overcome the difficulties, we first prove the behaviour of the program as expected in a bounded time interval by combining iSAT-ODE and Flow*, where iSAT-ODE can deal with different data types and logical structures very well by using SMT techniques, while Flow* can cope with HSs with large initial sets and various uncertainties using Taylor model representation of flowpipes. As both iSAT-ODE and Flow* are based on interval arithmetic, which suffers from the inflation of numeric errors, we further resort to theorem proving to do unbounded verification by extending our previous work [7] on invariant synthesis for polynomial HSs to HSs with general elementary functions (rational, exponential, trigonometric, logarithmic functions etc).

In this paper, we mainly focus on applying the above framework to the slow descent phase, which is a very typical and representative phase.

1.1 Related work

Applying formal methods to real-world industrial case studies has attracted increasingly attentions in recent years, and many successful stories are available, e.g. line 14 of Paris subway and shuttle at Roissy Airport [1], etc. In addition, formal methods have been applied to verify and design spacecrafts, e.g. in [6] Johnson et al proved satellite rendezvous and conjunction avoidance, while in [4] Katoen et al reported their experience in developing the control program for an European satellite under development using formal methods.

2 Description of the Verification Problem

Overview of the Slow Descent Phase. The slow descent phase begins at an altitude (relative to lunar surface) of approximately 30m and terminates when the engine shutdown signal is received. The task of this phase is to ensure that the lander descends slowly and smoothly to the lunar surface, by nulling the horizontal velocity, maintaining a prescribed uniform vertical velocity, and keeping the lander at an upright position. The descent trajectory is nearly vertical w.r.t. the lunar surface (see Figure 2).

The operational principle of the GNC system for the slow descent phase (and any other phases) can be illustrated by Figure 3. The closed loop system is composed of the spacecraft dynamics and the guidance program for the present phase. The guidance program is executed periodically with a fixed sampling period. At each sampling point, the current state of the spacecraft is measured by IMU (inertial measurement unit) or various sensors. Processed measurements are then input into the guidance program, which outputs control commands, e.g. the magnitude and direction of thrust, to be imposed on the spacecraft dynamics in the following sampling cycle.

We next give a mathematical description of the spacecraft dynamics as well as the guidance program of the slow descent phase. For the purpose of showing the technical feasibility and effectiveness of formal methods in the verification of aerospace guidance programs, we neglect the

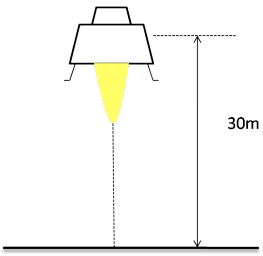

Fig. 2. The slow descent phase.

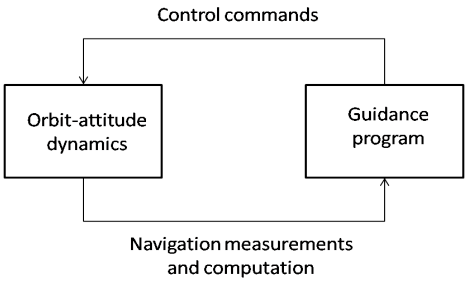

Fig. 3. A simplified configuration of GNC.

attitude control as well as the orbit control in the horizontal plane, resulting in a one-dimensional (the vertical direction) orbit dynamics.

Dynamics. Let the upward direction be the positive direction of the one-dimensional axis. Then the spacecraft dynamics is given by

$$\begin{cases} \dot{r} &= v \\ \dot{v} &= \frac{F_c}{m} - gM(r) \\ \dot{m} &= -\frac{F_c}{Isp(F_c)} \end{cases} , \tag{1}$$

where

- r, v and m denote the altitude (relative to lunar surface), vertical velocity and mass of the lunar lander, respectively;
- F_c denotes the thrust imposed on the lander, which is a constant between two sampling points;
- $gM(r)$ is the magnitude of the gravitational acceleration on the moon at height r:

$$gM(r) = \frac{4.9028 \times 10^{12} \mathrm{m}^3\mathrm{s}^{-2}}{(r + rM)^2} ,$$

  where rM denotes the radius of the moon. Since r ($0 \leq r \leq 30$m) is rather small compared to rM (the mean radius is 1737.10km), its effect on $gM(r)$ can be neglected. Therefore in the rest of this paper, $gM(r)$ will be taken as a constant, i.e. the surface gravity $1.622\mathrm{m/s}^2$, denoted by gM.
- $Isp(F_c)$ is the *specific impulse*[1] of the main engine of the lander. It takes the constant value 2500 if $F_c \in [1500, 3000]$, and the value 2800 if $F_c \in (3000, 5000]$. Here it is assumed that the commanded thrust given by the guidance control program is always kept in the range $[1500, 5000]$. Hence the dynamics (1) can be simplified to

$$\begin{cases} \dot{r} &= v \\ \dot{v} &= \frac{F_c}{m} - 1.622 \\ \dot{m} &= -\frac{F_c}{2500} \end{cases} , \quad \text{for } F_c \in [1500, 3000] , \tag{2}$$

$$\begin{cases} \dot{r} &= v \\ \dot{v} &= \frac{F_c}{m} - 1.622 \\ \dot{m} &= -\frac{F_c}{2800} \end{cases} , \quad \text{for } F_c \in (3000, 5000] . \tag{3}$$

[1] Specific impulse is a physical quantity describing the efficiency of rocket engines. It equals the thrust produced per unit mass of propellant burned per second.

Note that the term $\frac{F_c}{m}$ in (2) and (3) makes the dynamics non-polynomial.

Guidance Program. The control flow of the guidance program for the slow descent phase is demonstrated by the left part of Figure 4.

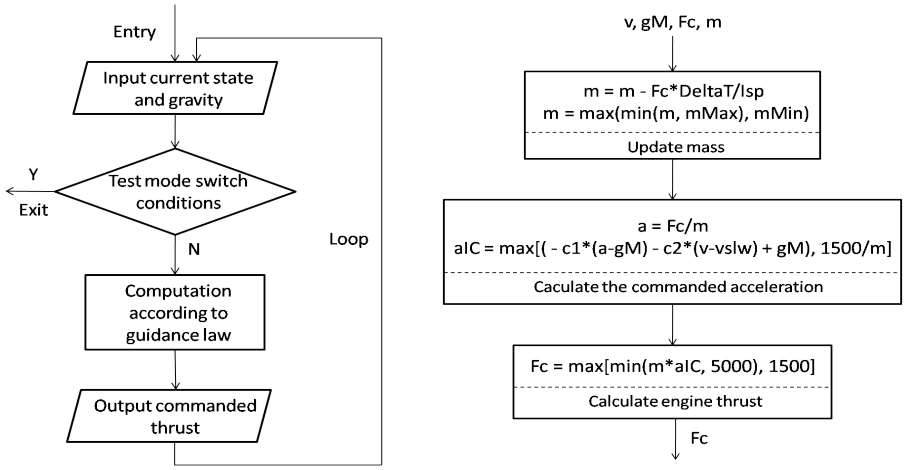

Fig. 4. The guidance program for the slow descent phase.

The guidance program is executed once for every 0.128s, which is the sampling period of the slow descent phase. The program first reads data produced by navigation computation, and then decides whether to stay in the slow descent phase or switch to other phases by testing the following conditions:

(SW1) shutdown signal 1, which should normally be sent out by sensors at the height of 6m, is received and the lander has stayed in slow descent phase for more than 10s;
(SW2) shutdown signal 2, which should normally be sent out by sensors at the height of 3m, is received and the lander has stayed in slow descent phase for more than 10s;
(SW3) no shutdown signal is received and the lander has stayed in slow descent phase for more than 20s.

If any of the above conditions is satisfied, then the GNC system switches from slow descent phase to no-control phase and a shutdown command is sent out to the thrust engine; otherwise the program will stay in the slow descent phase and do the guidance computation as shown in the right part of Figure 4, where

- v and gM are the vertical velocity and gravitational acceleration from navigation measurements or computation; note that we have assumed gM to be a constant;
- F_c and m are the computed thrust and mass estimation at last sampling point; they can be read from memory;
- $DeltaT = 0.128$s is the sampling period;
- Isp is the specific impulse which can take two different values, i.e. 2500 or 2800, depending on the current value of F_c;
- $mMin = 1100$kg and $mMax =$3000kg are two constants used as the lower and upper bounds of mass estimation;
- $c_1 = 0.01$ and $c_2 = 0.6$ are two control coefficients in the guidance law;

– $vslw = -2$m/s is the target descent velocity.
– the output F_c will be used to adjust engine thrust for the following sampling cycle; it can be deduced from the program that the commanded thrust F_c always lies in the range $[1500, 5000]$.

The Verification Tasks For the closed loop system in Section 2, after discussing with our industrial partner, they are mainly concerned the following safety-critical properties: whether the guidance law is effective, stable, and of high precision; whether the thrust engine will finally be shut down; whether the speed of the lander is low enough when contacting the lunar surface to avoid crash; and so on. Therefore the following verification problems are proposed.

I. With the initial condition $m = 1250$kg, $F_c = 2027.5$N, $r = 30$m, $v = -2$m/s:
   1. during the slow descent phase and before touchdown[2], $|v - vslw| \leq \varepsilon$, where $\varepsilon = 0.05$ is the specified bound for fluctuation of v around the target $vslw = -2$;
   2. the system will finally exit the slow descent phase;
   3. $|v| < vMax$ at the time of touchdown, where $vMax = 5$ is the specified upper bound of $|v|$ for guaranteeing safe landing;

II. Analysis or verification of the same properties in I, by taking into account various uncertainties due to sensor errors, disturbance of dynamics, floating-point calculation errors, etc.

3 Simulation

First of all, we build a Simulink/Stateflow model consisting of the physical process and the guidance program. Then based on the model we analyze the system's behaviour by simulation.

According to Section 2, the physical dynamics is specified by (2) and (3) which is modelled by the following Simulink diagram. In Figure 5, several blocks contain parameters that are not

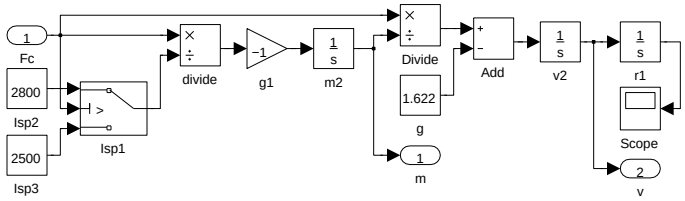

Fig. 5. The dynamics of the slow descent phase.

displayed.

– The threshold of Isp1 is 3000, therefore Isp1 outputs 2800 when F_c is greater than 3000 and 2500 otherwise;
– the initial values of m, v, and r are specified as initial values of block m2, v2, and r1 respectively (where $m = 1250$kg, $r = 30$m, $v = -2$m/s).

As specified in Figure 4, The guidance program includes three parts: updating mass m, calculating acceleration *aIC*, and thrust F_c.

The Simulink diagram for the guidance program is in Figure 6, in which all sample times of all blocks are fixed as 0.128s, that is the period of the guidance program. As we can see in

[2] Note that if no shutdown signal is received, there exists possibility that the lander stays in the slow descent phase after landing.

Figure 6, it senses the mass and speed of the lander and calculate the thrust every 0.128s, and the relation between mass m, speed v, and thrust F_c is as:

$$F_c \;=\; (1.622 - 0.01 * (F_c/m - 1.622) - 0.6 * (v - 2)) * m \tag{4}$$

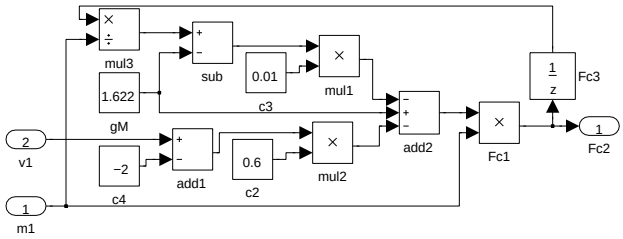

Fig. 6. The guidance program of the slow descent phase.

The simulation result is shown in Figure 7, where the speed of the lander is between -2 and -1.9999 and the lander switches from slow descent phase to no-control phase at the time 12.032s assuming that shutdown signal 1 is sent out at 6m and is successfully received.

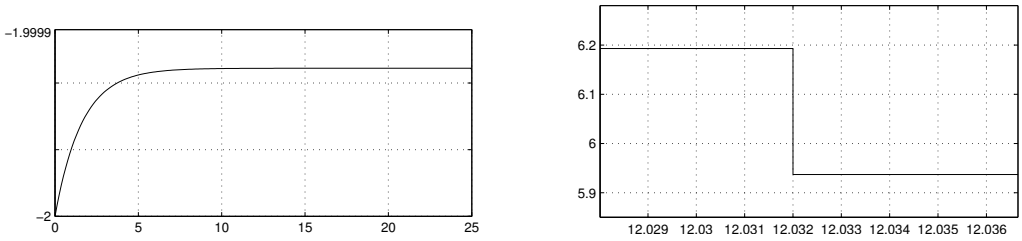

Fig. 7. The simulation result.

4 Verification

In this section, we study the proposed verification problems I.1-I.3 and II about the descent guidance program. Firstly, we verify properties I.1-I.3 using the bounded model checker iSAT-ODE [3]; then we build a more realistic model by considering all kinds of uncertainties in the system, of which properties similar to I.2 and I.3 are verified by the flowpipe computation tool Flow* [2]; afterwards, we perform unbounded verification of property I.1 using HHL Prover [9].

4.1 Verification by Bounded Model Checking

Bounded model checking (BMC) is a verification technique that, for a given state transition system and an initial set of states, answers whether an unsafe state can be reached by unwinding the transition system to a depth of k. For embedded software in sampled control systems such as space vehicles, usually a sub-procedure is only invoked during certain phases and executed at each sampling point, and thus can be suitably analyzed by BMC tools. In this paper, we will apply the iSAT-ODE tool [3] to verify properties I.1-I.3. Essentially, iSAT-ODE can be seen as an SMT solver that handles nonlinear arithmetic and nonlinear differential equations. It can be used as bounded model checker for nonlinear hybrid systems.

Modelling in iSAT. Thanks to the support of Boolean, integer and real data types, as well as such functions as max, min, abs etc in iSAT-ODE, modelling of the closed loop system in Figure 3 for the slow descent phase is straightforward. We first define two Boolean variables mode_slow and mode_free to represent the slow descent phase and the free fall phase respectively. We further require that at any time one and only one of mode_slow and mode_free is TRUE. Each sampling cycle induces two kinds of state transitions, i.e. the continuous and discrete transition, which are distinguished by a Boolean variable jump. For example, the following texts

```
mode_slow and !jump -> (d.r / d.time = v);
mode_slow and !jump -> (d.v / d.time = Fc/m - gM);
mode_slow and !jump -> (d.m / d.time = -Fc/Isp1);
mode_slow and !jump -> (d.Fc / d.time = 0);
```

can be used to define a continuous transition under dynamics (2), where !jump denotes the negation of jump and Isp1 is the constant 2500. Similarly, an update of Fc by a discrete computation has the following form

```
mode_slow and jump -> Fc' = (the updating assignment of Fc);
```

where Fc′ denotes the value of Fc after transition. The duration of each transition is represented by a real variable delta_time, which equals the sampling period in the continuous case and 0 in the discrete case.

The critical part is to model the conditions of switching from the slow descent phase to the free fall phase, i.e. (SW1)-(SW3). Based on whether the shutdown signal 1 or 2 is received, we build three different models using different switching conditions. For example, if the shutdown signal 1 is sent out at exactly a height of 6m and is successfully received, then (SW1) will be used in the model and it is encoded as

```
mode_slow and jump -> (mode_free' <-> r <= 6 and time > 10);
```

where time denotes the time elapsed in the slow descent phase. We will verify I.1-I.3 for all three models with different switching conditions.

The impact dynamics of landing may be quite complex, but for simplicity, in our model we assume the lander's velocity becomes 0 immediately upon touchdown and stays at 0 afterwards.

Verification in iSAT. Bounded model checking in iSAT-ODE can be done by specifying a target set (formula) whose reachability is to be checked, as well as the minimal and maximal unwinding depth of the state transition system for constructing BMC formulas. For the model of the slow descent phase, since each sampling cycle corresponds to two transition steps[3], if the system's reachable set in n sampling cycles is going to be checked against a target formula, then the minimal and maximal depth should be specified as 0 and $2n$ (or $2n-1$) respectively.

We first try to verify problem I.2 by setting the target formula to !mode_slow, that is, we test whether the current phase is NOT the slow descent phase; if it is, then the result unsatisfiable will be returned. In our model mode_slow is initially set to TRUE. We check for each $k \geq 0$ to find the first k that produces the satisfiable answer, which means phase switching happens at k. However, according to our experience in using iSAT-ODE, at the unwinding depth where the target formula becomes satisfiable, iSAT-ODE will run for a long time until memory is exhausted

[3] In our model, the k-th transition is a continuous transition if k is an odd number, and a discrete transition if k is an even number.

without giving an answer. In practice, when this phenomenon is observed, we negate the target formula and check against its negation at the same depth. It can be expected that iSAT-ODE will give the unsatisfiable answer with the negated target, i.e. mode_slow, which means the system has exited from the slow descent phase. In this way, we have shown that:

- if signal 1 is received, phase switching happens at $k = 188$, that is, the end of the 94th sampling cycle, or equivalently at the time 12.032s (consistent with Figure 7);
- if signal 1 is not received and signal 2 is received, phase switching happens at $k = 212$;
- if no signal is received, phase switching happens at $k = 314$.

We then try to verify I.1 by setting the target formula to the negation of I.1:

$$\mathsf{r} > 0 \ \mathsf{and} \ !(\mathsf{v} >= -2.05 \ \mathsf{and} \ \mathsf{v} <= -1.95) \ .$$

Since we are only considering the lander's velocity in the slow descent phase, this target is checked for depth $0 \leq k \leq 187$, $0 \leq k \leq 211$, $0 \leq k \leq 313$ respectively for three different models. In this way we have successfully verified I.1 (consistent with Figure 7).

We finally try to verify I.3. To this end, we will first try to get an estimation of the ranges of v and r of the lander at $k = 188, 212, 314$ for the three different models. The ranges of v has already been obtained by I.1. To get the ranges of r, we have to set the target formula appropriately by guessing a possible range of r first. The basic idea is that if iSAT-ODE returns unsatisfiable then the negation of the target formula gives a guaranteed bound of r. It's a process of trial and error. Bipartition of intervals can be applied. Finally, we get

- if signal 1 is received, then $r \in [5.9, 6.0]$ (consistent with Figure 7) when phase switching happens;
- if signal 1 is not received and signal 2 is received, then $r \in [2.8, 2.9]$ when phase switching happens;
- if no signal is received, then $r = 0, v = 0$ when phase switching happens.

Since the slow descent phase is followed by free fall, using the range estimations of v and r and the dynamics of free fall, it can be shown that in all three cases $|v| < 5$ upon touchdown.

The cost of the above verification, on the platform with Intel Q9400 2.66GHz CPU running a Debian virtual machine with 3GB memory allocated, can be summarized in the Table 1.

Table 1. Time and memory cost of the verification in iSAT-ODE.

	Model with (SW1)	Model with (SW2)	Model with (SW3)
I.1	2min46sec, 477MB	3min46sec, 594BM	14min3sec, 1.8GB
I.2	1min22sec, 290MB	2min1sec, 350MB	2min7sec, 62MB
I.3	24sec, 304MB	31sec, 378MB	50sec, 602MB

4.2 Verification with Uncertainties

We have shown how the properties I.1–I.3 in Section 2 can be verified by iSAT-ODE, by assuming the initial state to be a single point, and the continuous dynamics, sampling time points, navigation and guidance computations etc are all exact. However, in practice such ideal models do not exist because disturbances and noises are unavoidable in the physical world. Therefore it is meaningful to analyze the performance of the lander's GNC system by taking into account various uncertainties that may exist therein. The tool for such analysis used here is Flow* [2].

Flow* is a Taylor model-based tool for computing over-approximations of the reachable sets of continuous dynamical systems and hybrid systems. The prominent features of Flow* includes the handling of non-polynomial nonlinear ODEs, ODEs with uncertainties, reset functions with uncertainties, and so on, which all facilitate our modelling here.

Modelling as Hybrid Automata. Basically, in Flow* a hybrid system is modelled as a hybrid automaton. For the slow descent phase shown by Figure 3, Figure 4 and dynamics (2) and (3), we make the following assumptions:

- the shutdown signal 1 is sent out at the height of 6m and is successfully received by guidance program;
- throughout the computation of the guidance program, the value of m lies in the range $[mMin, mMax]$, and the value of

$$F_c' \widehat{=} -c_1 \cdot (F_c - m \cdot gM) - c_2 \cdot (v - vslw) \cdot m + m \cdot gM \tag{5}$$

  lies in $[1500, 5000]$, which means the computation of thrust in the guidance program can be simplified to $F_c := F_c'$.

Then we can construct in Flow* a hybrid automaton (HA) as illustrated by Figure 8.

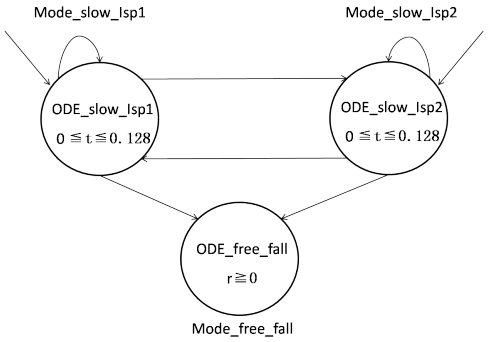

Fig. 8. The HA model of the slow descent phase.

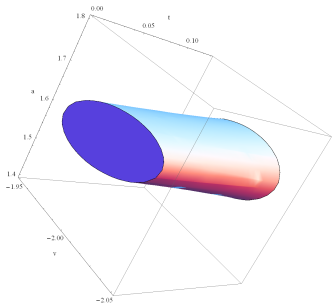

Fig. 9. The generated invariant for HHL Prover.

For Figure 8, we give the following explanations:

- the three modes represents the slow descent phase with specific impulse 2500, with specific impulse 2800, and the free fall phase, respectively; the mode domains are shown in the picture; the continuous dynamics are (2), (3) and the standard dynamics of free fall, respectively;
- all the discrete jumps take place at $t = 0.128$, where t is a local variable representing the elapsed time in the current sampling cycle; t is reset to 0 for every jump;
- the jumps between Mode_slow_lsp1 and Mode_slow_lsp2 depend on the comparison of F_c' (defined in (5)) to 3000; the value of F_c is updated to F_c' for every such jump;
- the jumps from Mode_slow_lsp1 or Mode_slow_lsp2 to Mode_free_fall depend on the the test of the switching condition (SW1), which is equivalent to $r \leq 6 \wedge T > 10$ by our assumption, where T denotes the total elapsed time since entering the slow descent phase.

Introducing Uncertainties. We next modify the model in Figure 8 by introducing into it various kinds of uncertainties according to different origins:

- The initial states are chosen to be intervals, e.g. $v \in [-2.5, -1.5]$, $r \in [29.5, 30.5]$, $m \in [1245, 1255]$ $F_c \in [2020, 2035]$,[4] and so on.
- Add interval uncertainties to dynamics (2) and (3), as well as dynamics of the free fall mode. The causes of such uncertainties could be: the direction of F_c may deviate from the vertical direction; the specific impulse may not be exactly 2500; the engine may not be able to keep a constant thrust in one cycle; the acceleration of gravity changes with height; and so on. For example, we have
$$\dot{m} = -\frac{F_c}{2500} + [-0.1, 0.1]$$
if the specific impulse has roughly a ± 300 uncertainty around 2500.
- In the guidance program, the values of m and F_c are stored in memory so they are not changed between two sampling points. Due to uncertainties of ODEs, the stored values may not be identical with the actual physical quantities. Therefore we introduce two new variables m_p and F_p, whose time derivatives are zero, to distinguish between program variables and continuous state variables.
- The measurement of time in the computer system may not be accurate, and thus the length of one sampling cycle may vary in the range of, say $[0.127, 0.129]$. This should be reflected in the domain and transition guard of the hybrid automaton.
- The measured height may suffer from sensor errors, say ± 0.1, and thus the shutdown signal may be sent out at a height of 6 ± 0.1m. Therefore we revise the mode switching conditions accordingly by taking into consideration such imprecision.
- The measured velocity may also suffer from sensor errors, say ± 0.1. Since the value of m is greater than 1000, by (5), this may cause a fluctuation of nearly 100 of the commanded thrust. Therefore we revise (5) by
$$F_p' \widehat{=} -c_1 \cdot (F_p - m \cdot gM) - c_2 \cdot (v - vslw) \cdot m + m \cdot gM + [-100, 100] \ . \quad (6)$$
In the computation of F_p, there may also exist floating point errors, which we claim can be absorbed by the large interval $[-100, 100]$.

Computation Results. We compute the reachable set of the above described model with a time bound of 25s and a jump depth of 200. The computation costs 19 minutes and 769MB memory on the platform with Intel Q9400 2.66GHz CPU and 4GB RAM running Ubuntu Linux. From the computed reachable set we draw the pictures depicting the changes of v, r, F_p, m_p with respect to time T. The meanings of Figure 10 can be explained as follows.

- The ranges of T in all pictures are within $[0, 18]$. It can be inferred[5] that neither the time bound 25 or the jump depth 200 is reached during the flowpipe computation, and thus the result covers all the reachable states of the hybrid automaton in Figure 8.
- The top left picture shows the v-T relation. Since the initial range of v is $[-2.5, -1.5]$. The verification problem I.1 does not make sense. However, we can still conclude that the guidance program has a good asymptotic property, that is, the value of v converges to a stable

[4] Thus we know that the initial mode should be the slow descent phase with specific impulse 2500.

[5] Rigorous proof can be obtained by looking into the details of computed flowpipes.

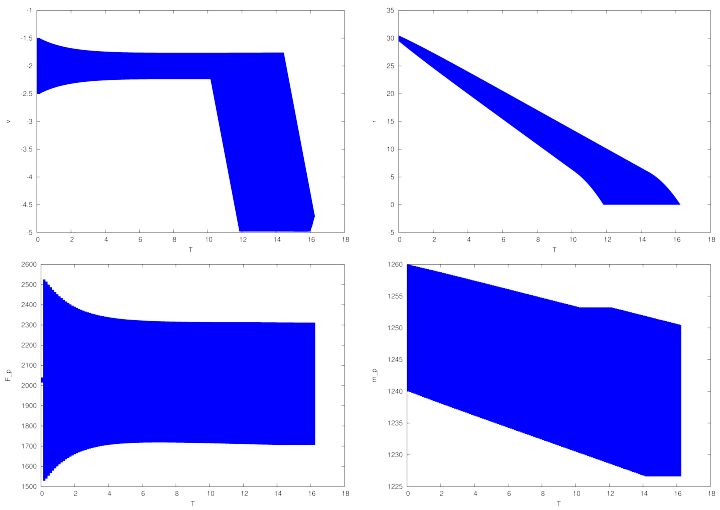

Fig. 10. Reachable sets given by Flow*.

interval, approximately $[-2.25, -1.75]$ after some time. We can also infer (see footnote 5) from the sharp decrease in v that starting from any initial state the system will finally switches to the free fall phase. Thus by incorporating other phase switching conditions, i.e. (SW2) and (SW3) into the model, property I.2 can be verified. Besides, it can be seen from the picture that v is always above the level -5 at any time; actually property I.3 can be formally verified with the support of safety property checking in Flow*.

- The top right picture shows the r-T relation.
- The bottom left picture shows the F_p-T relation. Although the initial range of F_p is narrow (defined to be $[2015, 2040]$), by (6), the variation of v in $[-2.5, -1.5]$ would cause fluctuation of F_p by several hundreds. Nevertheless, the picture shows that the range of F_p also stabilizes after some time. Besides, F_p always lies in $[1500, 2600]$, which justifies our assumption for getting (5).
- The bottom right picture shows the m_p-T relation. The initial value of m_p is defined to be $[1240, 1260]$. It can be seen that m_p always lies in $[1225, 1260]$, which also justifies our assumption for getting (5).

4.3 Verification by Theorem Proving

In order to perform unbounded verification of the correctness of the control program by theorem proving, we first translate automatically by the tool Sim2HCSP [11] the Simulink/Stateflow model in Section 3 into a formal model given by HCSP [5, 8], which is a formal modelling language for HSs by extending CSP. Then, based on the formal model, using decomposition rules of HHL Prover, several constrains related to differential invariant remain unresolved; finally, using all the constraints we synthesize a differential invariant which can ensure the safety property.

Transformation to HCSP. Basically the transformed HCSP process is as follows[6],

```
definition P :: proc where
```

[6] The detail of this case study can be downloaded at `http://lcs.ios.ac.cn/~zoul/casestudies/hcs.rar`.

```
"P == PC_Init;PD_Init;t:=0;(PC_Diff;t:=0;PD_Rep)*"
```

in which

- PC_Init and PD_Init are initial procedures for dynamics and guidance program, respectively;
- PC_Diff models dynamics of the lander within a period 0.128s, which is either formula (2) guarded by $F_c \leq 3000$ or formula (3) guarded by $F_c > 3000$;
- PD_Rep calculate thrust F_c for the next period according to (4).

Hence, process P is initialised at the beginning by PC_Init and PD_Init, and behaves as a repetition of dynamics PC_Diff and computation PD_Rep afterwards. Variable t is used to ensure that PD_Diff terminates in 0.128s.

Verification in HHL Prover. In the slow descent phase, we are interested in whether the speed of the lander v satisfies $|v - vslw| \leq \varepsilon$. Hence, the proof goal is

```
lemma goal : "{True} P {safeProp; (l=0 | (high safeProp))}"
```

where safeProp is $|v - vslw| \leq \varepsilon$.

After using rules in HHL Prover, the following three lemmas remain.

```
lemma constraint1: "(t<=0.128) & Inv |- safeProp"
lemma constraint2: "(v=-2) & (m=1250) & (Fc=2027.5)
  & (t=0) |- Inv"
lemma constraint3: "(t= 0.128) & Inv
  |- substF([(t,0)], substF([(Fc, m*
     (1.622 - 0.01*(Fc/m-1.622) - 0.6*(v+2)))],Inv))"
```

Inv is the differential invariant for differential equations. Based on these three lemmas, we only need to discover an differential invariant to ensure the safety.

Invariant Generation. The three lemmas left unresolved in the above proof process are all about the undetermined invariant Inv. In a more readable way, these constraints on Inv are:

(C1) $0 \leq t \leq 0.218 \wedge \text{Inv} \longrightarrow |v - vslw| \leq \varepsilon$;
(C2) $v = -2 \wedge m = 1250 \wedge F_c = 2027.5 \wedge t = 0 \longrightarrow \text{Inv}$;
(C3) $t = 0.128 \wedge \text{Inv} \longrightarrow \text{Inv}(0 \leftarrow t; F_c' \leftarrow F_c)$, with

$$F_c' = -c_1 \cdot (F_c - m \cdot gM) - c_2 \cdot (v - vslw) \cdot m + m \cdot gM \ ; \tag{7}$$

(C4) Inv is the invariant of the two constrained dynamical systems

$$\langle ODE1; 0 \leq t \leq 0.128 \wedge F_c \leq 3000 \rangle \quad \text{and} \quad \langle ODE2; 0 \leq t \leq 0.128 \wedge F_c > 3000 \rangle \ ,$$

with $ODE1$ and $ODE2$ defined in (2) and (3) respectively.

As we have pointed out, the dynamics (2) and (3) are non-polynomial, which makes the invariant generation for them an intractable problem. However, if we replace the non-polynomial terms $\frac{F_c}{m}$ in $ODE1$ and $ODE2$ by a new variable a, then by simple computation of derivatives we can get two transformed polynomial dynamics:

$$ODE1' \mathrel{\widehat{=}} \begin{cases} \dot{r} = v \\ \dot{v} = a - 1.622 \\ \dot{a} = \frac{a^2}{2500} \end{cases} \quad \text{and} \quad ODE2' \mathrel{\widehat{=}} \begin{cases} \dot{r} = v \\ \dot{v} = a - 1.622 \\ \dot{a} = \frac{a^2}{2800} \end{cases} \ . \tag{8}$$

Furthermore, it is not difficult to see that the update of F_c as in (7) can be accordingly transformed to the update of a:

$$a' = -c_1 \cdot (a - gM) - c_2 \cdot (v - vslw) + gM \ . \tag{9}$$

As a result, if we assume Inv to be a formula over variables v, a, t, then Inv should satisfy (C1) and

(C2') $v = -2 \wedge a = 1.622 \wedge t = 0 \longrightarrow$ Inv;
(C3') $t = 0.128 \wedge$ Inv $\longrightarrow$ Inv$(0 \leftarrow t; a' \leftarrow a)$, with a' defined in (9);
(C4') Inv is the invariant of the two constrained dynamical systems

$$\langle ODE1'; 0 \leq t \leq 0.128 \rangle \quad \text{and} \quad \langle ODE2'; 0 \leq t \leq 0.128 \rangle^{7} \ ,$$

with $ODE1'$ and $ODE2'$ defined in (8) respectively.

Note that the constraints (C1) and (C2')-(C4') are all polynomial. Then the invariant Inv can be discovered by using the well known SOS (sum-of-squares) relaxation method in the study of polynomial hybrid systems. Simply, we first define a template $p(v, a, t, \mathbf{u}) \leq 0$ for Inv, where $\mathbf{u}$ is the vector of undetermined coefficients in p; then encode (C1) and (C2')-(C4') into SOS constraints, which can be solved efficiently using numerical solvers; finally from the assignments of $\mathbf{u}$ given by numerical computation, we can get an invariant instantiation. Here we define $p(v, a, t, \mathbf{u})$ to be a polynomial of degree 6 with all coefficients undetermined. Then we model the SOS encodings in the Matlab-based environment YALMIP, which calls the numerical solver SDPT-3 for solving the underlying SDP (semi-definite programming) problems. On the platform with Intel Q9400 2.66GHz CPU and 4GB RAM running Windows XP, with a time cost of 2s and a memory cost of 5MB, an invariant $p(v, a, t) \leq 0$ is generated as depicted by Figure 9.

Due to floating point errors, the invariant generated by numerical solvers is not guaranteed to be a true invariant. Therefore we perform post-verification using symbolic computation to show that the constraints (C1) and (C2')-(C4') can indeed be satisfied by the synthesized $p(v, a, t)$. The post-verification is done with the Maple-based tool RAGlib[8]. All constraints pass the verification with an overall time cost of 10 minutes and memory cost of 70MB on the same platform mentioned above. Thus we have successfully completed the proof of property I.1 by theorem proving. The detail of invariant generation will be reported in another paper.

5 Conclusion and Discussion

We studied a short piece of program used for the guidance and control in the terminal slow descent process of a lunar lander. In collaboration with experts from the aerospace industry, a closed loop system linking the program and the lander's dynamics was mathematically described, and safety-critical properties about the system were proposed. These properties are all successfully verified by using or extending several existing formal verification techniques that can handle continuous-discrete interactions, general nonlinear differential equations and uncertainties. The tools applied in this paper includes iSAT-ODE, Flow* and HCSP/HHL-based theorem prover. Our experience with using these tools can be summarized as follows.

– **Modelling.** The input model of HHL Prover is originally developed in Simulink/Stateflow, and then automatically transformed into HCSP. Both Simulink/Stateflow and HCSP are very expressive. iSAT-ODE also has a rich modelling language. Only continuous state variables are supported in Flow*.

[7] We have abstracted the domain constraint on F_c to be TRUE.
[8] `http://www-polsys.lip6.fr/~safey/RAGLib/`

– **Verified Properties.** The current version of HHL Prover cannot prove liveness properties. iSAT-ODE and Flow* can both be used as bounded model checker, so safety and liveness do not make much difference for them. In this paper, more properties are proved by iSAT-ODE and Flow*.
– **Expenses and Scalability.** iSAT-ODE is very expensive in memory cost, especially when using intervals as initial sets. The Taylor model-based Flow* can effectively control the wrapping effect in flowpipe computation, and thus can handle large intervals and exhibits high scalability in the verification of uncertain models. Theoretically HHL Prover scales well and can deal with very large complex systems since it does not rely on brute-force computations; however, the call to external constraint solvers for invariant generation may also be costly.
– **Human Interaction.** Flow* is fully automated. It computes flowpipes of hybrid systems and draw pictures of them, from which the system's behaviour within a considered time range can be quite clear. For verification in iSAT-ODE, as we have no prior knowledge of the system's properties, sometimes trial and guess are necessary. HHL Prover is an interactive theorem prover so human interaction is essential for using it.

The studied program in this paper is very typical for sampled-data control systems. The preliminary result shows good prospect of closed loop verification of embedded control software. In the next step, we will do more case studies on spacecrafts. We also plan to extend HHL Prover to deal with proof of liveness properties, to develop more effective invariant generation methods for nonlinear ODEs, and to investigate error control in numerical computation of ODEs.

References

1. Abrial, J.R.: Formal methods in industry: achievements, problems, future. In: ICSE 2006. pp. 761–768 (2006)
2. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: CAV 2013. pp. 258–263 (2013)
3. Eggers, A., Ramdani, N., Nedialkov, N., Fränzle, M.: Improving the SAT modulo ODE approach to hybrid systems analysis by combining different enclosure methods. Software & Systems Modeling pp. 1–28 (2012)
4. Esteve, M.A., Katoen, J.P., Nguyen, V.Y., Postma, B., Yushtein, Y.: Formal correctness, safety, dependability, and performance analysis of a satellite. In: ICSE 2012. pp. 1022–1031 (2012)
5. He, J.: A classical mind: Essays in honour of C. A. R. Hoare. chap. From CSP to hybrid systems, pp. 171–189. Prentice Hall International (UK) Ltd. (1994)
6. Johnson, T.T., Green, J., Mitra, S., Dudley, R., Erwin, R.S.: Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems. In: FM 2012. pp. 252–266 (2012)
7. Liu, J., Zhan, N., Zhao, H.: Computing semi-algebraic invariants for polynomial dynamical systems. In: EMSOFT 2011. pp. 97–106 (2011)
8. Zhou, C., Wang, J., Ravn, A.P.: A formal description of hybrid systems. In: Hybrid Systems, LNCS 1066. pp. 511–530 (1996)
9. Zou, L., Lv, J., Wang, S., Zhan, N., Tang, T., Yuan, L., Liu, Y.: Verifying Chinese train control system under a combined scenario by theorem proving. In: VSTTE 2013. LNCS, vol. 8164, pp. 262–280 (2013)
10. Zou, L., Zhan, N., Wang, S., Fränzle, M.: Formal verification of simulink/stateflow diagrams. Tech. Rep. ISCAS-SKLCS-13-07, State Key Lab. of Computer Science, Institute of Software, CAS (2013)
11. Zou, L., Zhan, N., Wang, S., Fränzle, M., Qin, S.: Verifying simulink diagrams via a hybrid hoare logic prover. In: EMSOFT 2013. pp. 1–10