HyRev: A tool for the automatic generation of real-time routines for enabling fail-safe control in a class of safety-critical embedded systems using backwards reachability analysis

Hallstein A. Hansen, Buskerud University College, Norway

2013-09-23

Motivation

- Buskerud University College does research on the use of remote laboratories in engineering education
- Some student experiments can fail - expensively
- We wish to protect equipment by adding a safety system that overrides the student experiment when (if) it fails
- The goal of this work: To give a method and tool for determining this 'when'

Safety-critical and real-time systems

- The failure of a *safety-critical* system can lead to
  - Human death or severe injuries
  - Loss or serious damage to property
  - Environmental harm
- A *hard real-time* system is a safety-critical system which can fail if some timing constraint is not met
  - Typically by missing a deadline

Fail-safe systems

- A *fail-safe* system responds to a failure by placing itself in a state where it can cause no, or minimal, harm
- Failing safe is easy for some types of systems
  - Typically by cutting power
- Failing safe is difficult for other types of systems
  - Aircraft

Control and safety

- A way of designing fail-safe systems is through separate control and safety systems
  - The *control* system provides the functionality required of the system
  - The *safety* system brings the system to a safe state
- The control system is typically complex, since there is a demand for system functionality
- The safety system is typically simple, since this generally improves its dependability
- The two systems should be physically and logically separate to isolate failures of the control system

Hazards

- A *hazard* is a potential accident as a result of system failure
- Identifying hazards and analyzing their consequences and *conditions* is a crucial step of developing safety-critical systems
  - An unmanned aerial vehicle will crash if it hits the ground with at least a certain speed

Our approach

- Given
  - The conditions of a hazard
  - The description of a safety system
  - The description of an out-of-control system, where 'anything' is possible
- We want to automatically generate a *safety check*
  - Evaluates a conservative condition for when the safety system must act to avoid an accident
- Note that we neither need to model the (complex) control system, nor (arbitrary) control system errors

Safety check

- The conservative condition is given as a subset of the possible values of the system state variables
  - Position, velocity, acceleration, temperature, battery level, etc.
- If the condition holds, then the safety system must act within some *deadline*, in order to fail safe

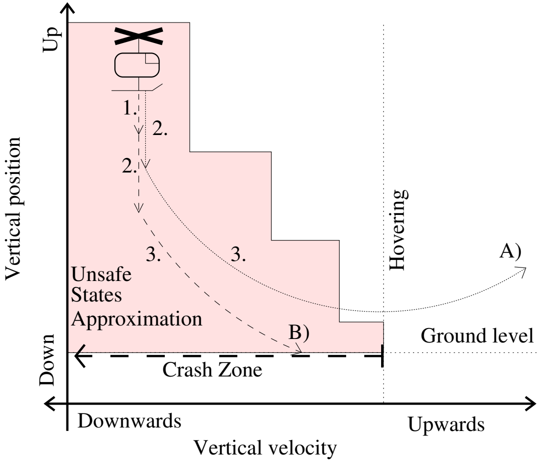
Timing considerations

- The safety check operates in parallel with the control system
- If the safety check activates the safety system too late, then the safety system may miss its deadline
- Thus, we can assume the safety check in general is a *hard real-time* system

Timing considerations

- The system is assumed to be out-of-control during the time interval
  - From meeting the conservative condition for transitioning to the safety system
  - To operation of the safety system
- We can assume the system may exhibit *any* physically allowed behavior
- We call this behavior the *free* behavior, and the time interval the *free* time

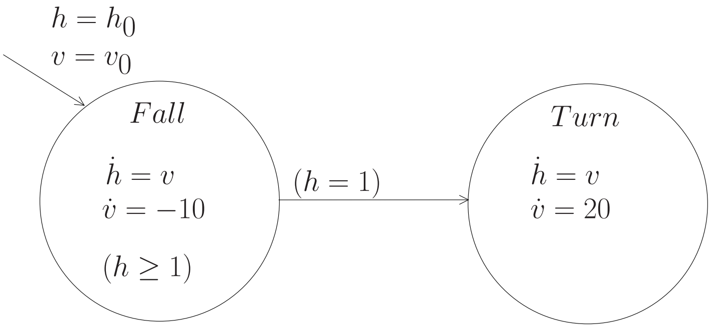
Example


Up
Down
Vertical position
1.
2.
2.
3.
3.
Unsafe
States
Approximation
Hovering
A)
B)
Ground level
Crash Zone
Downwards
Upwards
Vertical velocity

Formalism: Hybrid automaton

- The state variables over which both conditions are evaluated are in general continuous
- The description of a system includes in general both discrete and continuous behavior
- Can be expressed using *hybrid automata*
  - Linear hybrid automata: The continuous behavior is given by constant differential equations, $\dot{x} = c$
  - Affine hybrid automata: The continuous behavior is given by linear differential equations, for example $\dot{x} = ax + b$

Example



Reachability

- Usually the formal analysis of a hybrid automaton model is performed using a reachability search:
    - From a set of *initial* states
    - The state space of the model is explored
    - And either one of a set of *final* states is reached, 'Yes', or the state space is exhausted, 'No', or the search does not terminate
- If a linear *approximation* of the (general) model is used, then only a 'No' answer can be trusted

Reachability

- In our setting, where we consider a safety system model, the usual analysis is not applicable
- We have no initial states to begin with
- Indeed, our purpose is to produce a set of initial states for the safety systems
- However, we do know the *final* states, these are the hazard conditions

Reversal

- By *reversing* the safety system model we get that:
    - The hazard conditions are the initial states
    - The reachable state space includes all the system trajectories satisfying the hazard conditions
- Only trajectories in the reachable state space satisfy the hazard conditions
- Thus, the condition for the safety system to act can be given as an over-approximation of the reachable state space

Reachability solver

- It is possible to perform a backwards reachability search to generate this over-approximation
- However, most existing reachability solvers perform *forward* reachability searches
  - We generate the *reverse* of a linear hybrid automaton
  - Then perform the reachability search, or rather the state space exhaustion, using an existing tool

Safety check

- The safety check routine which determines if the safety control should act or not has a hard real-time requirement
- In its most general form the routine can be written as
  - `if currentState in overApprox then act()`
- Thus the form and size of the over-approximation influences the worst case execution time of the safety check

Reverse linear hybrid automaton

- A reverse H_r of a linear hybrid automaton H is an automaton with:
  - The same variables, locations and invariants, and edges as H
  - Any constant differential equation $\dot{x} = c$ of the activities of H is replaced by $\dot{x} = -c$
  - The guards and assignments are modified as well

Free behavior

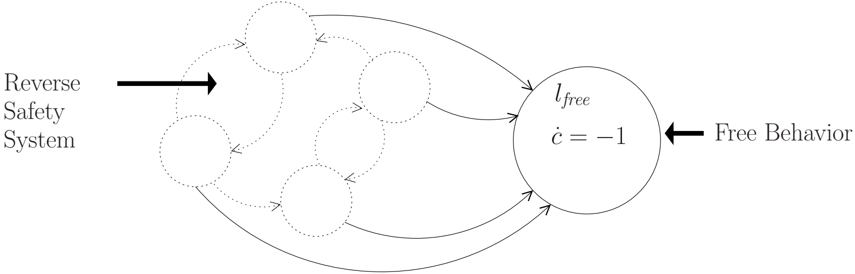
- We model the free, out-of-control, behavior in the following manner:
- A special clock variable, c
- A special location called l_{free} is defined, with
  - Invariant $c \leq t_{free}$, with t_{free} the free time
  - $\dot{c} = 1$
  - $\dot{x} \in [\epsilon_x^-, \epsilon_x^+]$, for all variables $x \neq c$
- $[\epsilon_x^-, \epsilon_x^+]$ represents the free behavior of x

Linear emergency control automaton

- Consider a linear hybrid automaton H, the safety system, extended with free behavior in the following manner:
  - l_{free} is added to the set of locations of H
  - c is added to the set of variables but only modified by the activity of l_{free}
  - The system can leave l_{free} at any time, but never return
- We call the reverse of such an automaton a *linear emergency control automaton*

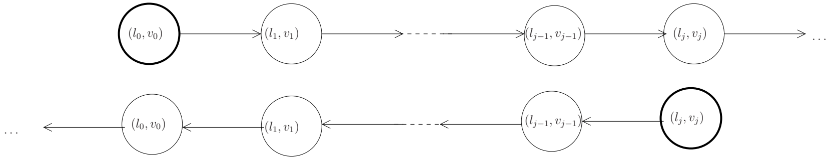
Linear emergency control automaton

- The initial states of the linear emergency control automaton should be the hazard conditions



Trajectories

- We needed to prove that a reverse automaton R includes all the (reversed) behavior of the orginial automaton H
- If not, then trajectories that eventually lead to an accident may not be included in the condition for switching to the safety system
- Trajectories are infinite in general, but we only need consider trajectories that reach states where the hazard condition holds, and only until that state



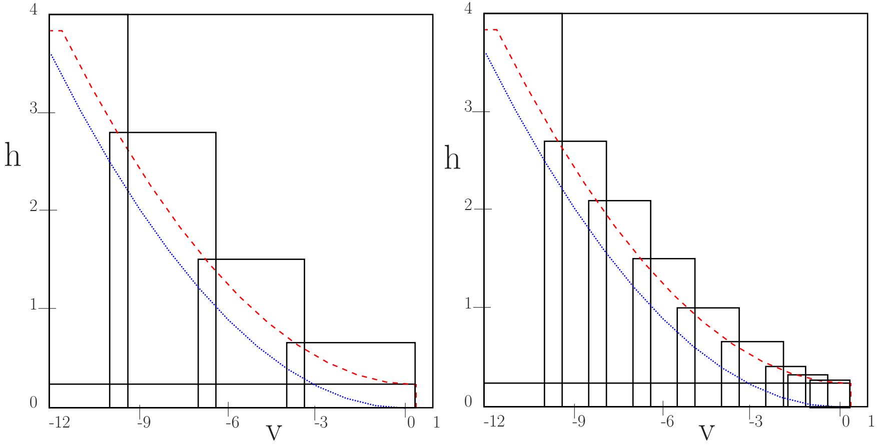
Reach set

- By running an external reachability solver on a linear emergency control automaton then, if the search terminates, we generate the state space reachable from the hazard condition
- The safety check needs to check, within a deadline, if the current system state is contained in this set, which is made up of location,polytype pairs
- Our approach:
  - We show that the location is irrelevant
  - Checking containment in a hyper-rectangle has a trivial worst case execution time (`if low` $\leq$ `x` $\leq$ `high` for each dimension)
  - We can thus over-approximate each polytope with a hyper-rectangle

Reach set

- We can achieve a WCET bound on checking containment for a single hyper-rectangle
- But the reach set can be of arbitrary size
- By *merging* hyper-rectangles together we can restrict the size of the final reach set to be included in the safety check
- The cost is of course the precision of the over-approximation

Merging



HyRev

- The theory has been implemented in the tool-chain HyRev
- `http://folk.uio.no/hallstah/hyrev/`
- Written in Python
- Generates pseudocode
- The implementation is limited to safety systems modeled as single location *affine* hybrid automata

HyRev: Configuration

```
[[name]]
rise

[[vars]]
h 0.0 4.0
v −12.0 12.0

[[init]]
h 0.0 0.0
v −12.0 0.0

[[flow]]
h' = v
v' = 20.0

[[cuts]]
h 1
v 16

[[free behavior]]
h' −12 12
v' −10 20

[[free time]]
0.02

[[safety test]]
5
```

HyRev: Operation

- The tool HyRev performs the following operations on its input
  1. Hybridization, affine to linear automaton
  2. Automaton reversal
  3. Addition of free behavior
  4. Computes reach set (SpaceEx tool)
  5. Approximates and merges reach set
  6. Outputs pseudocode for safety check

HyRev: Final output

```
boolean potentially_dangerous(box[] state_space, point state)

   for box in state_space
      contains = TRUE
      for i in dimension
         if pos[i] not in box[i]
            contains = FALSE
      if contains
         return TRUE
   return FALSE


switch(pos)

   space = [
      box( interval(0.0,0.24), interval(-12.0,0.4)),
      box( interval(0.0,4.0), interval(-12.0,-8.6)),
      box( interval(0.0,2.6025), interval(-9.2,-5.6)),
      box( interval(0.0,1.365), interval(-6.2,-2.6)),
      box( interval(0.0,0.5775), interval(-3.2,0.4))
   ]



   if potentially_dangerous(space,pos)
      do_switch()
   else
      do_nothing()
```

HyRev: Performance

Hybridization	SpaceEx	Merging	Total run time
32	1.0	0.9	2.2
40	1.6	3.3	5.3
48	2.9	9.9	13.4
56	5.6	24.3	31.0
64	6.5	42.8	50.8

Future work

- More complex models:
  - Hybridization of non-linear dynamics
  - Reversal of affine automata
- Better case studies
- Analysis of precision
- More precise over-approximation
- More efficient merging algorithm

Thank you!

Any questions?