Synthesis of Quantized Feedback Control Software For Discrete Time Linear Hybrid Systems

Enrico Tronci
Dipartimento di Informatica – Sapienza University of Rome
Via Salaria 113, 00198 Roma – Italy - tronci@di.uniroma1.it

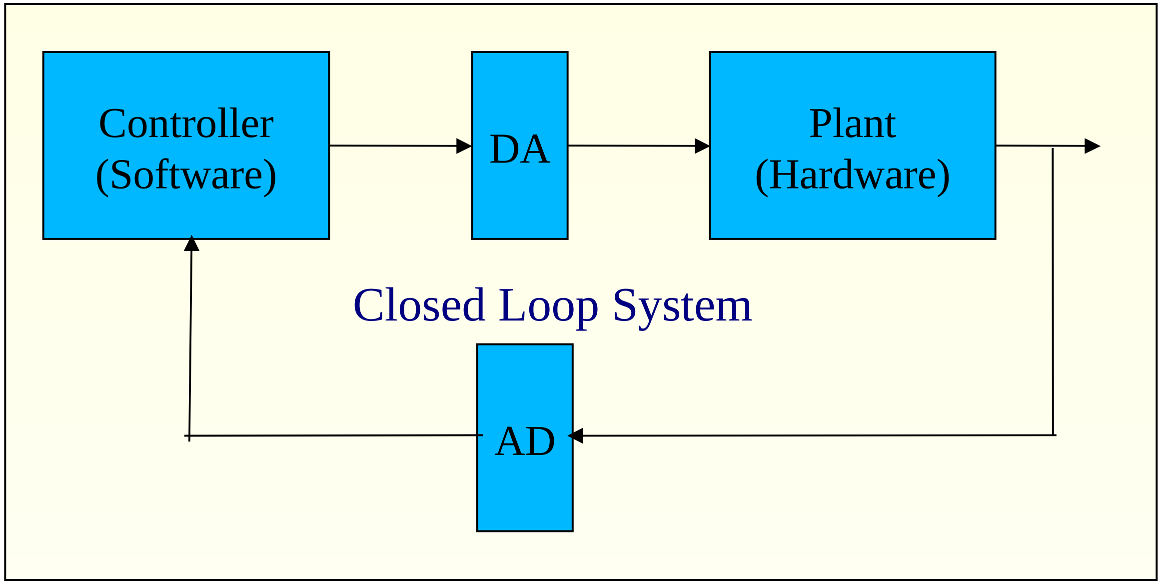
Joint Work with: Federico Mari, Igor Melatti, Ivano Salvo

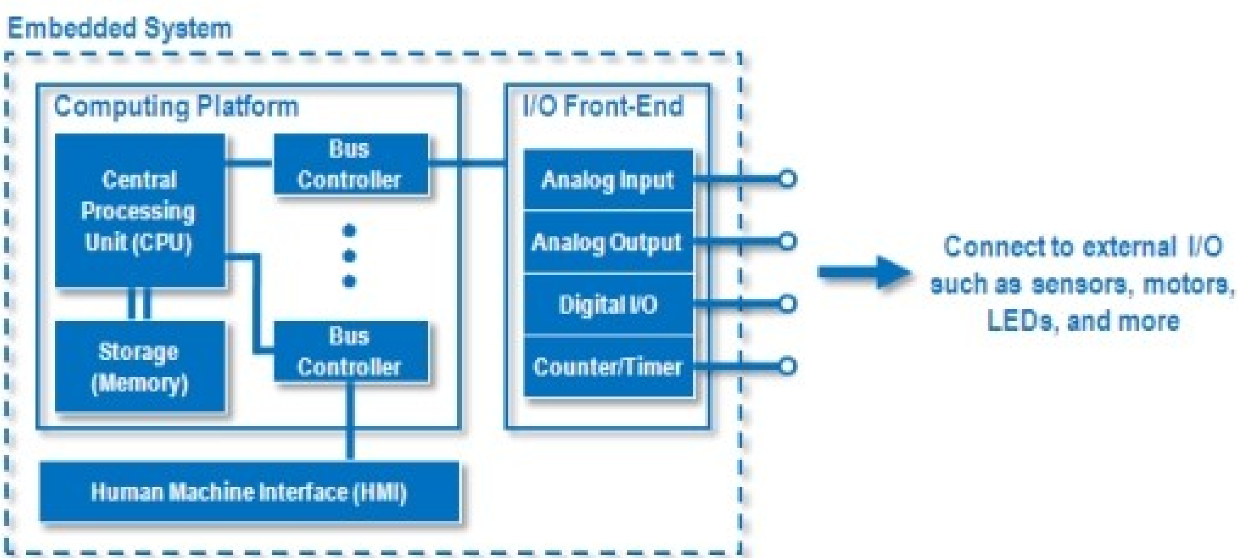
CP Meets CAV, Turunç, Turkey, 25-29 June 2012

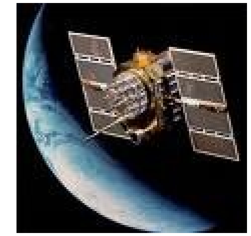
DIPARTIMENTO
DI INFORMATICA

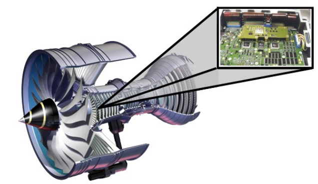
SAPIENZA
UNIVERSITÀ DI ROMA

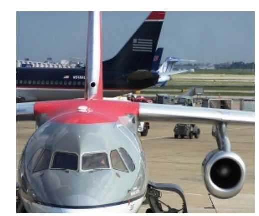
Our Focus: Control Software





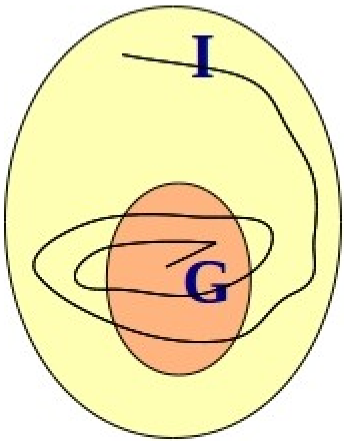




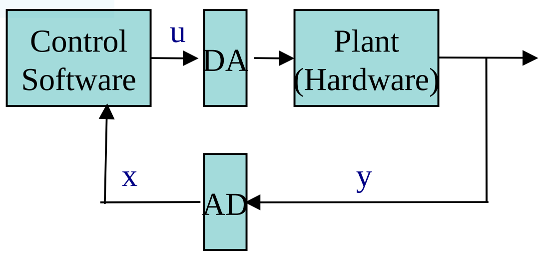


Examples of PLANTS

Synthesis Problem




```
Closed_Loop_System()
every T seconds do      // T = Sampling Time
      Read quantized plant output x from sensors;     // Acquire x = AD(y)
            if (!Controllable_Region(x))   // Fault Detection
               then Exception: start fault Isolation and Recovery (FDIR);
               else // Nominal case: compute command u to send to actuator
                      u = Control_Law(x);
                      Send command u to plant;
             endif; od
```



Control Software = **Controllable_Region** + **Control_Law**

Remark: Control Software WCET <= Sampling Time T

Closed Loop Specifications: I = desired controllable region, G = goal region ($G \subseteq I$):

- **Liveness**: Any computation path starting from I eventually reaches G
- **Safety**: All states reachable from I meet given safety constraints

Synthesis Problem: Find **Control Software** such that:

- $I \subseteq \{x \mid$ **Controllable_Region** $(x)\}$
- Closed Loop Specifications are met.

Current Control Software Design Approach

Plug and Pray

Hardware in the Loop Simulation

Design Iteration

Closed Loop System Specifications

Control Law = Functional Requirements for Control_Law() software
+ Software Desired WCET <= Sampling Time (T)

Control Software Design + Implementation

Evaluate Closed Loop System Implementation
If it satisfies Specs, OK else revise Control Law and/or Control Software

What Could Possibly Go Wrong ?

Control Law correctness wrt **closed loop specifications** via **simulation**
Controllable region and **safety properties** established by **simulation**.
Problem: time consuming, no formal *guarantee* about absence of errors.

Control Software Correctness wrt closed loop specifications typically established using (*hardware in the loop*) **simulation**.
Problem: Same problem as above.

Control Software Performance Control systems are Hard Real Time Systems. Thus it must hold: **Control Software WCET > Sampling Time (T)**. Checked at posteriori (after software design). If it fails redesign of the control software and/or of the control law (to simplify computation) is needed.

Design space exploration difficult, because, although there exist many control laws meeting the given closed loop specs, **only one control law is provided to the software designer**. This limits *a priori* the software design space (e.g., to save on RAM, CPU, Energy)

Model Based Control Software Design

Formal Specifications for Closed Loop System (CLS) =

Plant Model (DTHS in our setting) +

Set of Goal States (G) +

Desired Controllable Region I ($G \subseteq I$) +

AD/DA Characteristic (bits) +

Sampling Time (T) +

Desired Robustness

Quantized Control Software Synthesizer (QKS)

Control Software K such that:

- K is correct by construction (AD/DA OK, no arithmetical overflows, etc),
- K meets robustness requirements
- K meets closed loop system specifications
- WCET of K = Time_IFTE*State_bits*Actuators_Bits

(check if WCET < T even BEFORE computing K)

The World Seen From the Software Side

Controller should work notwithstanding nondeterminism.

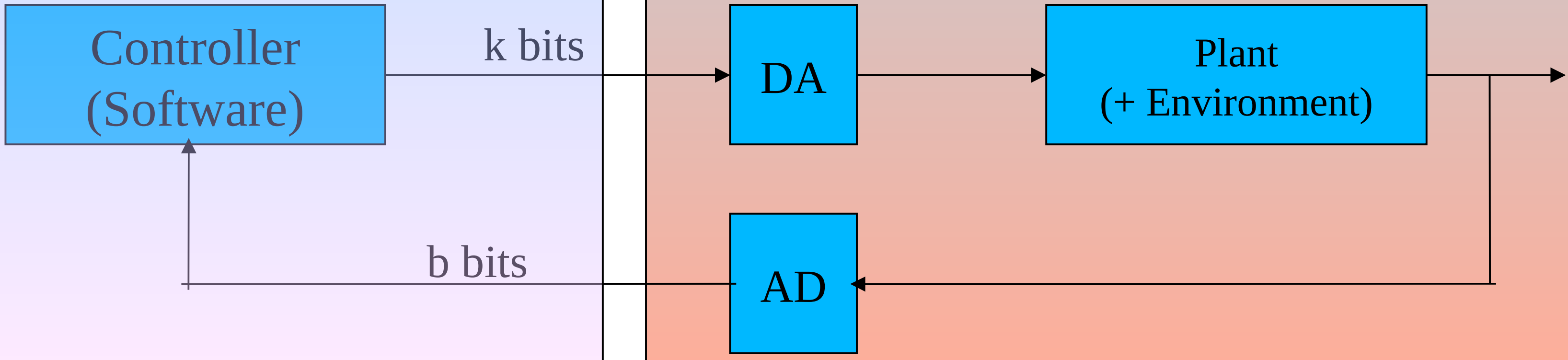
Increasing nondeterminism increases robustness, but decreases our chances of finding a controller.

Decreasing nondeterminism decreases robustness, but increases chances of finding a controller.

Nondeterministic Finite State Machine (FSM).
States: 2^b
Actions = outgoing transitions from each state: 2^k

Nondeterminism stem from quantization.

Modeling quantization as Stochastic Noise not suited here, since we want to certify correctness under *any* admissible scenario.



Outline

Compute FSM M modeling plant H as seen from the Control Software because of quantization. To increase our chances of finding a solution, M should be as deterministic as possible (*Control Abstraction*).

Compute Control Software K for M. Show that using the given AD/DA conversion, K also works on H.

Experimental results on interesting and challenging example: the Buck DC-DC converter.

Discrete Time Linear Hybrid Systems (DTLHS)

Roughly, any hybrid system which dynamics can be described using linear differential equation can be modeled as a DTLHS (using a suitable sampling time T).

$$
\begin{aligned}
N(X,U,Y,X') = (&(i_L' = (1 + Ta_{1,1})i_L + Ta_{1,2}v_O + Tb_{1,1}v_D)\\
\wedge\, &(v_O' = Ta_{2,1}i_L + (1 + Ta_{2,2})v_O + Tb_{2,1}v_D)\\
\wedge\, &(v_u - v_D \leq (1+\rho)V_i) \wedge (v_u - v_D \geq (1-\rho)V_i)\\
\wedge\, &(i_D = i_L - i_u) \wedge (q \rightarrow v_D = 0) \wedge (q \rightarrow i_D \geq 0)\\
\wedge\, &(\bar{q} \rightarrow v_D \leq 0) \wedge (\bar{q} \rightarrow v_D = R_{off} i_D)\\
\wedge\, &(u \rightarrow v_u = 0) \wedge (\bar{u} \rightarrow v_u = R_{off} i_u))
\end{aligned}
$$

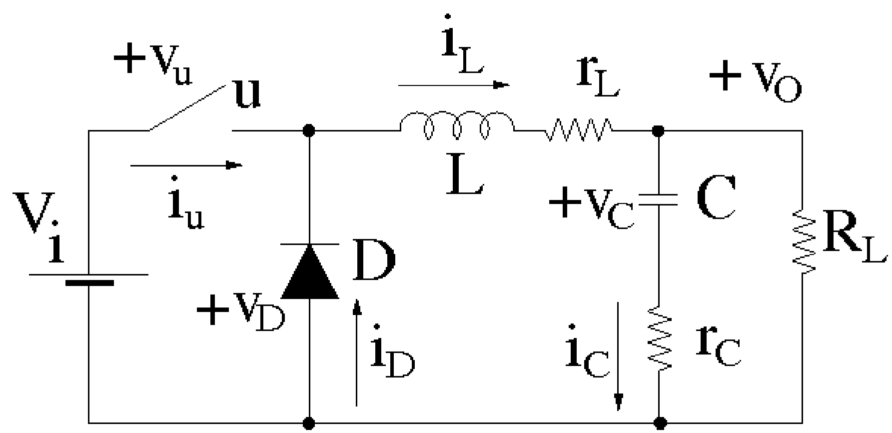
$\rightarrow$ = if-then

$\rho = 0.25$ (25%) (tolerance)

$X = [i_L, v_O]$

$U = [u]$

$Y = [v_D, v_u, i_D, i_u, q]$



Control Problem

GIVEN

a DTLHS H = (X, U, Y, N)
a set I (as AND of linear constraints) of initial states
a set G (as AND of linear constraints) of goal states (typically G is a subset of I)

FIND

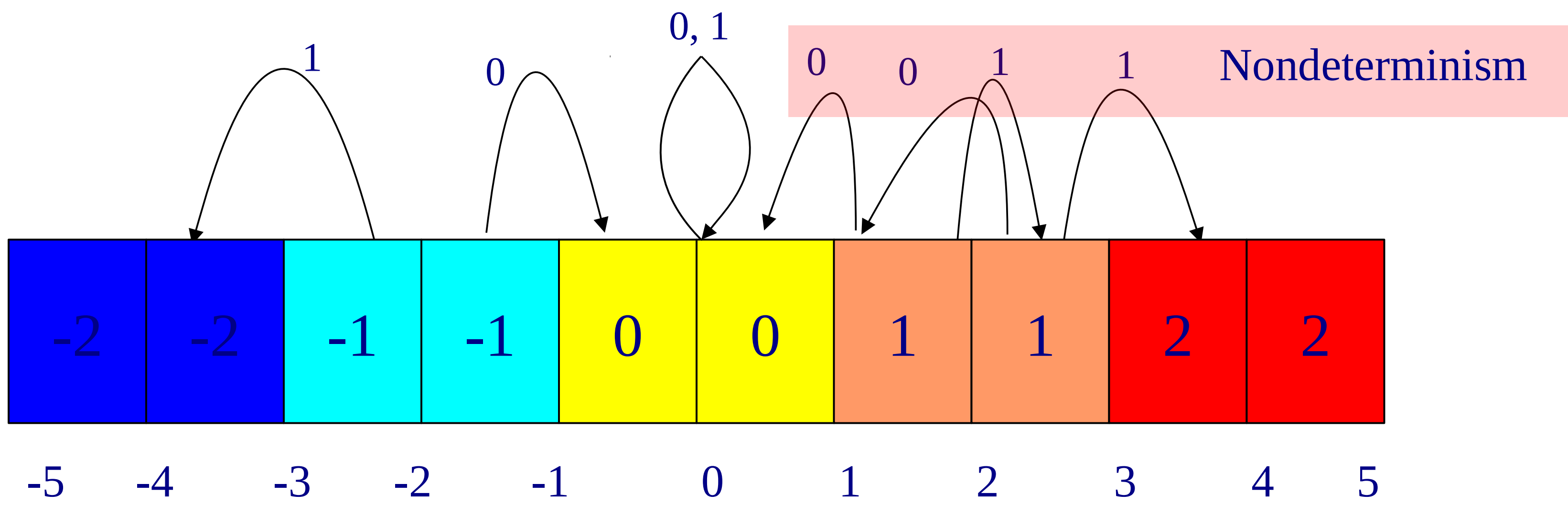
K : AD(X) → AD(U) (AD(U) is just a finite subset of U)

s.t. (by selecting suitable values for control input u)
K drives any state in I to a state in G within a finite number of steps.

Example of Quantized Control Problem

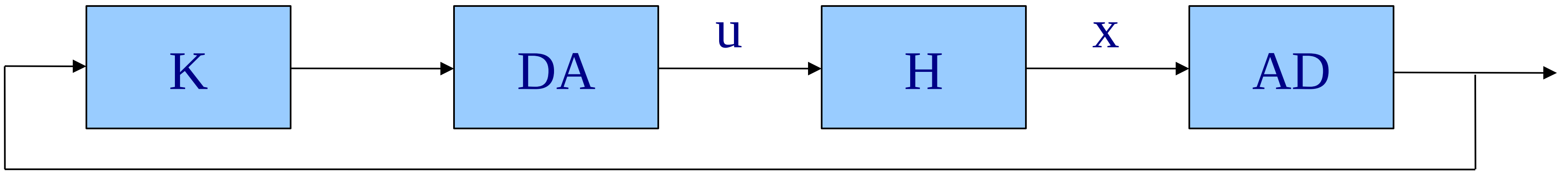
H : Bounds = {-2.5 <= x <= 2.5, u = 0, 1}, N = {!u → x' = 0.5x, u → x' = 1.5x}

AD:



I = {x | -2.5 <= x <= 2.5}, G = {x | -1 <= x <= 1}

K : AD(X) → {0, 1} s.t. K(-1) = K(0) = K(1) = 0 is a controller solving (H, I, G).



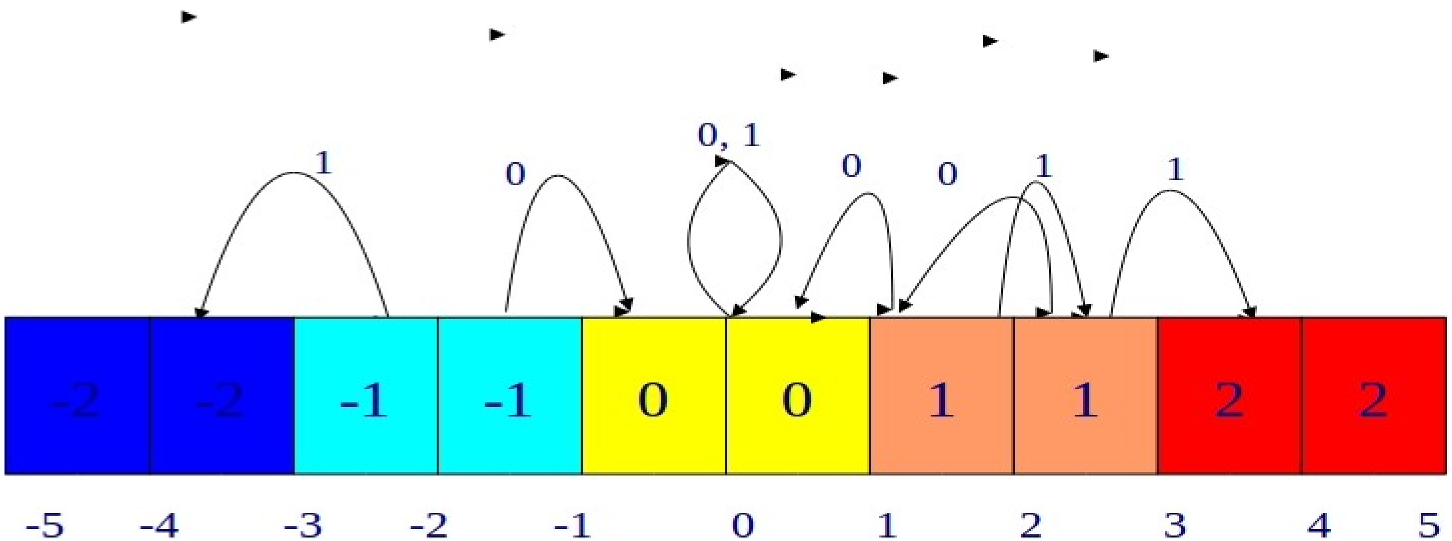
How To Compute Controller

In general this problem is undecidable. We present:
- a sufficient condition
- a necessary condition

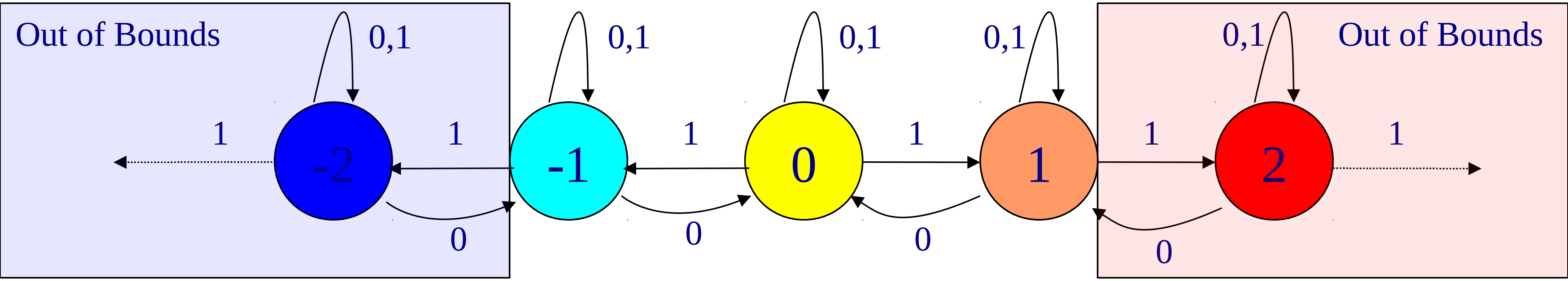
Strategy:
compute the FSM M for plant H as seen from the software
compute controller for M and use it on H (using AD/DA).

Quantization Effects

H : Bounds = {-2.5 <= x <= 2.5, u = 0, 1}, N = {!u → x' = 0.5x, u → x' = 1.5x}



FSM M for H as seen from the control software



Controlling FSM from Quantization

H : Bounds = {-2.5 <= x <= 2.5, u = 0, 1}, N = {!u → x' = 0.5x, u → x' = 1.5x}

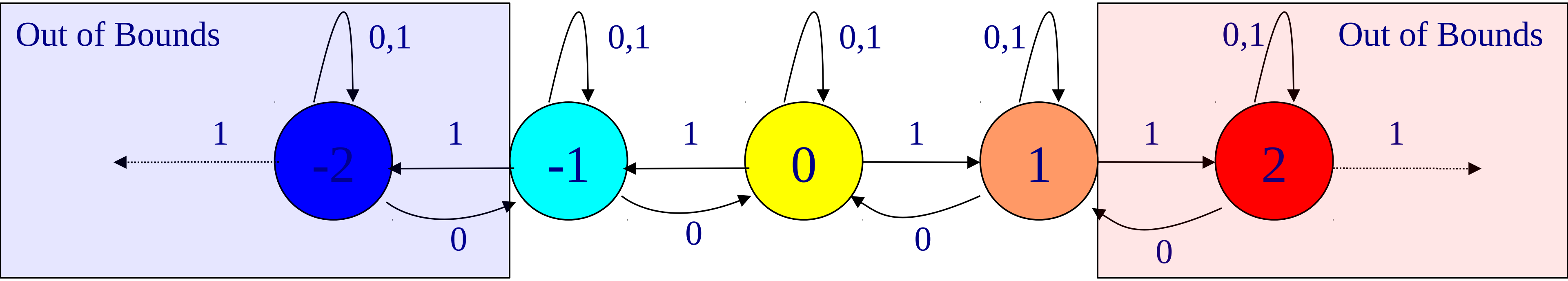
I = {x | -2.5 <= x <= 2.5}, G = {x | -1 <= x <= 1}

Because of *nondeterminism*, from M it looks like **no controller** can guarantee driving H to G!

Indeed, because of *nondeterminism*, even our previously found solution
K : {-1, 0, 1} → {0, 1} s.t. K(-1) = K(0) = K(1) = 0
is not a solution by looking at M!!!!

Our Approach: Replace M with a suitable FSM with less nondeterminism than M.

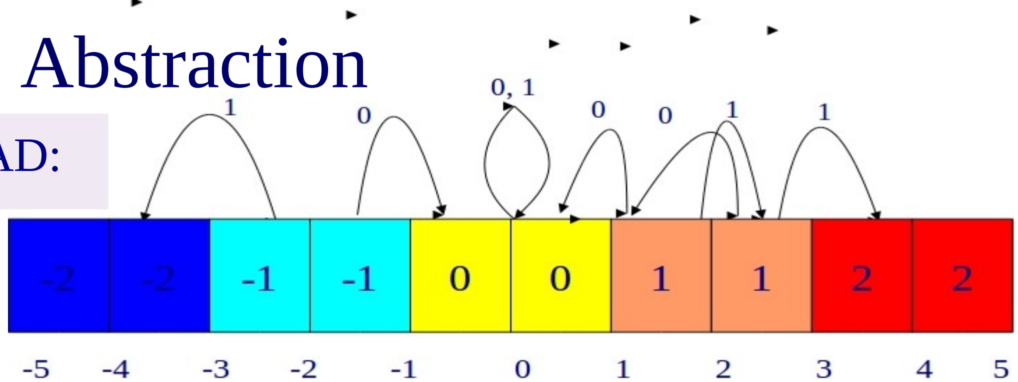
FSM M for H as seen from the control software



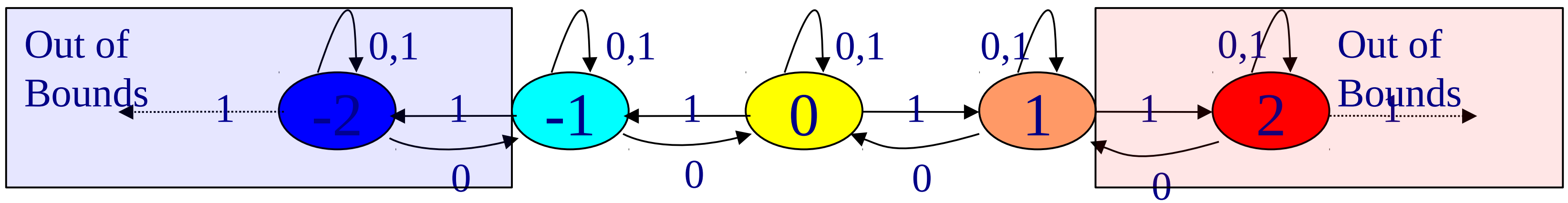
Control Abstraction

H :
Bounds = {-2.5 <= x <= 2.5, u = 0, 1},
N = {!u → x' = 0.5x, u → x' = 1.5x}

AD:

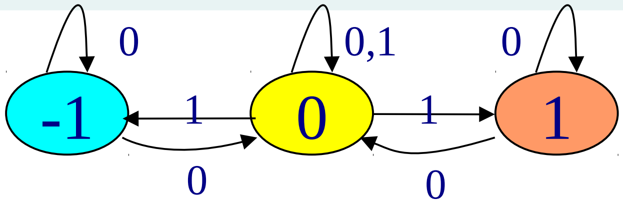


FSM M for H modeling quantization (**MaxCtrAbs**)



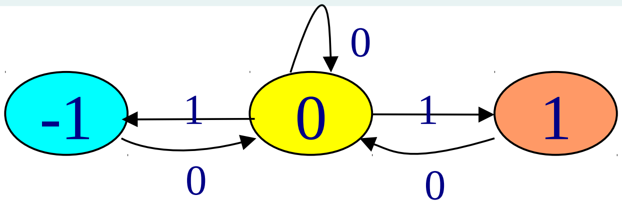
Control Abstraction Step 1 (restrict to **safe transitions**)

M1



Ctr Abs Step 2 (remove a **self-loop** if it eventually disappears) (**MinCtrAbs**)

M2



K(-1) = K(0) = K(1) = 0
is a controller for M2 and thus for H.

Main Theorem

Control Problem (H, I, G) + Quantization AD

Compute AD **MaxCtrAbs** W (FSM)

Check if **from each** quantized initial state s in AD(I) a quantized goal state in AD(G) is reachable in W (**Weak Controller**)
Use [Tronci - ICFEM98].

No: No Solution

Yes: Unknown

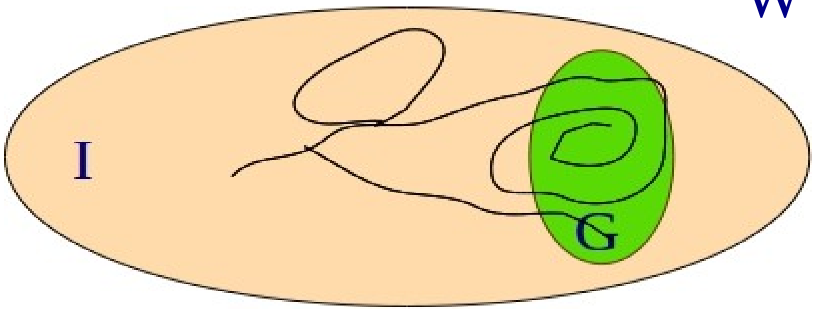
Compute AD **MinCtrAbs** P (FSM)

Check if there exists a restriction (**Strong Controller**) K of P such that in KP any path from a state in AD(I) (quantized initial state) reaches a state in AD(G) (quantized goal state) within a finite number of steps.
Use [Cimatti - AIPS98].

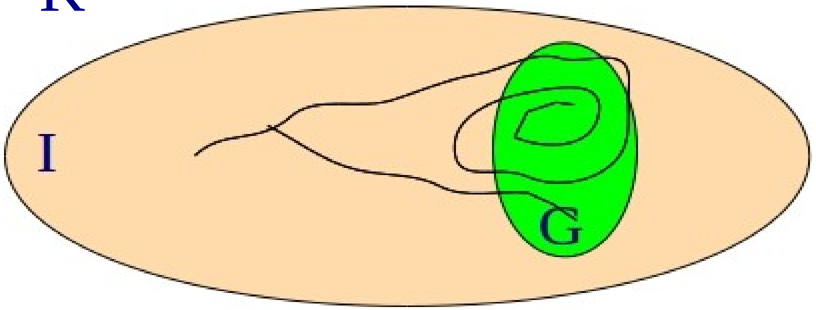
No: Unknown

Yes: Solution for H: λxu. K(AD(x),AD(u))

W



K



Strategy

Compute **MinCtrAbs**.
This yields a **sufficient condition** for existence of a solution to (H, I, G).
Unfortunately computing the *Minimum Control Abstraction* is undecidable (since it entails solving a reachability problem on linear hybrid systems).
We look for a *small enough* (i.e., *deterministic enough*) control abstraction.

Compute **MaxCtrAbs**.
This yields a **necessary condition** for existence of a solution to (H, I, G).
Computing the *Maximum Control Abstraction* is decidable and easier than Computing MinCtrAbs. Thus, in the following we focus on computing MinCtrAbs.

Main Algorithm (1): Computing Control Abstractions

Input: A quantization AD, a DTLHS H = (X, U, Y, N), a control problem (H, I, G).
Output: OBDDs for: **N** (transition relation of MinCtrAbs), **I** (quantization of I), **G** (quantization of G)

```
minCtrAbs(AD, H, I, G)  {
X = [x_1, ... x_n];   X' = [x_1', ... x_n'];   U = [u_1, ... u_r];  N(X, U, Y, X') = 0;   I(X) = 0;  G(X) = 0;
forall (s in AD(STATE)) do {                 // MILP (… X …) is feasible = EXISTS X ( … )
        if (s is the quantization of an initial state)  I = I ∪ {s};        // MILP1 add initial state
        if (s is the quantization of a goal state)  G = G ∪ {s};         // add goal state
        forall (u in AD(CTR)) do {
            if (action u from a state X with quantization s may lead to an unsafe state) // MILP5 skip unsafe
                {continue;}
            if (action u from a state x with quantization s may lead to a selfloop)  //  MILP2  add selfloop
                {N = N ∪ {(s, u, s)}};        // Use OBDDs here as well as in I and G
            forall (i = 1, ...  n)  do {
                    m_i = min value for u-successor of x_i ;          // MILP3 min reach x
                    M_i = max value for u-successor of x_i;        // max reach x
            Over_Img(s, u) = ∏_{i=1}^n [AD(m_i), AD(M_i)] ;        // Overapprox of 1-step reachable x
            forall (s'  in  Over_Img(s, u)) do  {
                    if  (s != s'  and s' is a (quantized) u-successor of s)    // MILP4
                        N = N ∪ {(s, u, s');  }          // add transition  (s, u, s')
            } }     //  end exploration
return (N, I, G);  }
```

Main Algorithm (2): MILPS

Discrete state s is the quantization of an initial state =
EXISTS X [I(X) $\wedge$ AD(X) == s] = MILP(I(X) $\wedge$ AD(X) == s) is feasible = MILP1

Discrete state s is the quantization of a goal state =
EXISTS X [G(X) $\wedge$ AD(X) == s] = MILP(I(X) $\wedge$ AD(X) == s) is feasible

Action u from a state X with quantization s may lead to an unsafe state =
MILP(N(X, U, Y, X') $\wedge$ AD(X) = s $\wedge$ AD(U) = u $\wedge$ X' is not in STATE) is feasible = MILP5

Action u from a state X with quantization s may lead to a selfloop = SelfLoop(s, u) = MILP2

m_i = min value for u-successor of x_i =
$m_i = x_i'^*$, where $X'^* = [x_1'^*, \ldots, x_n'^*]$ is a solution to the MILP
(min, x_i', N(X, U, Y, X') $\wedge$ AD(X) = s $\wedge$ AD(U) = u) = MILP3

M_i = max value for u-successor of x_i =
$M_i = x_i'^*$, where $X'^* = [x_1'^*, \ldots, x_n'^*]$ is a solution to the MILP
(max, x_i', N(X, U, Y, X') $\wedge$ AD(X) = s $\wedge$ AD(U) = u)

Discrete state s' is a (quantized) u-successor of s =
MILP (N(X, U, Y, X') $\wedge$ AD(X) = s $\wedge$ AD(U) = u $\wedge$ AD(X') = s') is feasible = MILP4

Main Algorithm (3): Checking Self Loops

SelfLoop(s, u) {

For each real valued state component x_i, **do** // check gradient of x_i

let w_i be the min elongation of x_i, that is the solution to

MILP(min, $x_i' - x_i$, N(X, U, Y, X') $\wedge$ AD(X) = s $\wedge$ AD(U) = u);

let W_i be the max elongation of x_i, that is the solution to

MILP(max, $x_i' - x_i$, N(X, U, Y, X') $\wedge$ AD(X) = s $\wedge$ AD(U) = u);

If for some i [(w_i != 0) $\wedge$ (W_i != 0) $\wedge$ (w_i and W_i have the same sign)]

then return 0 // any *long enough* sequence of u actions will drive

// state component x_i outside of AD^{-1}(s)

else return 1 // unable to show that self loop can be eliminated

}

Computing a Controller for MinCtrAbs

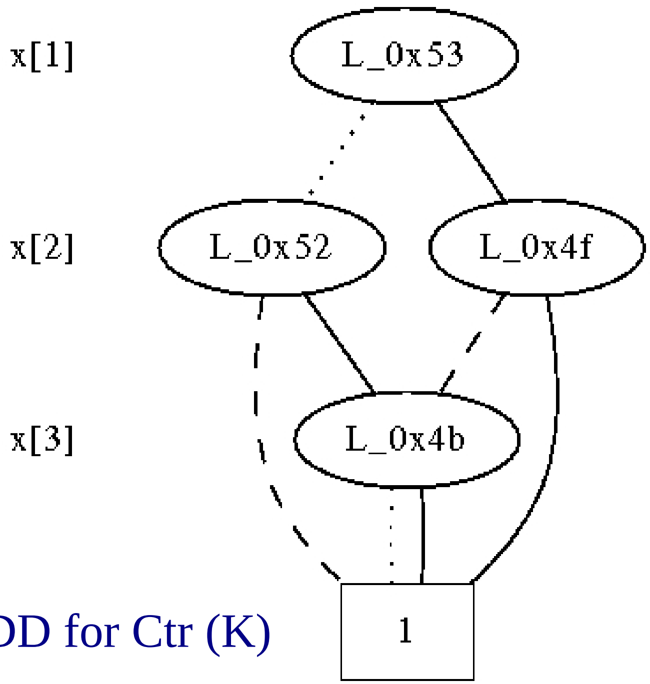
From OBDDs (N, I, G) we compute symbolically an OBDD K(x, u) for the Strong Controller using the algorithm in [Cimatti – AIPS98].

Using the algorithm in [Tronci – ICFEM98] From K we generate a C implementation F(x) for K(x, u) s.t. K(x, F(x)) holds for any state x in the controllable region. That is, F satisfies the predicate:

$$\forall x\ [\exists u\ K(x, u) \rightarrow K(x, F(x))\ ]$$

A Glimpse on Control Software Generation

Ctr Software generated from OBDD for K.



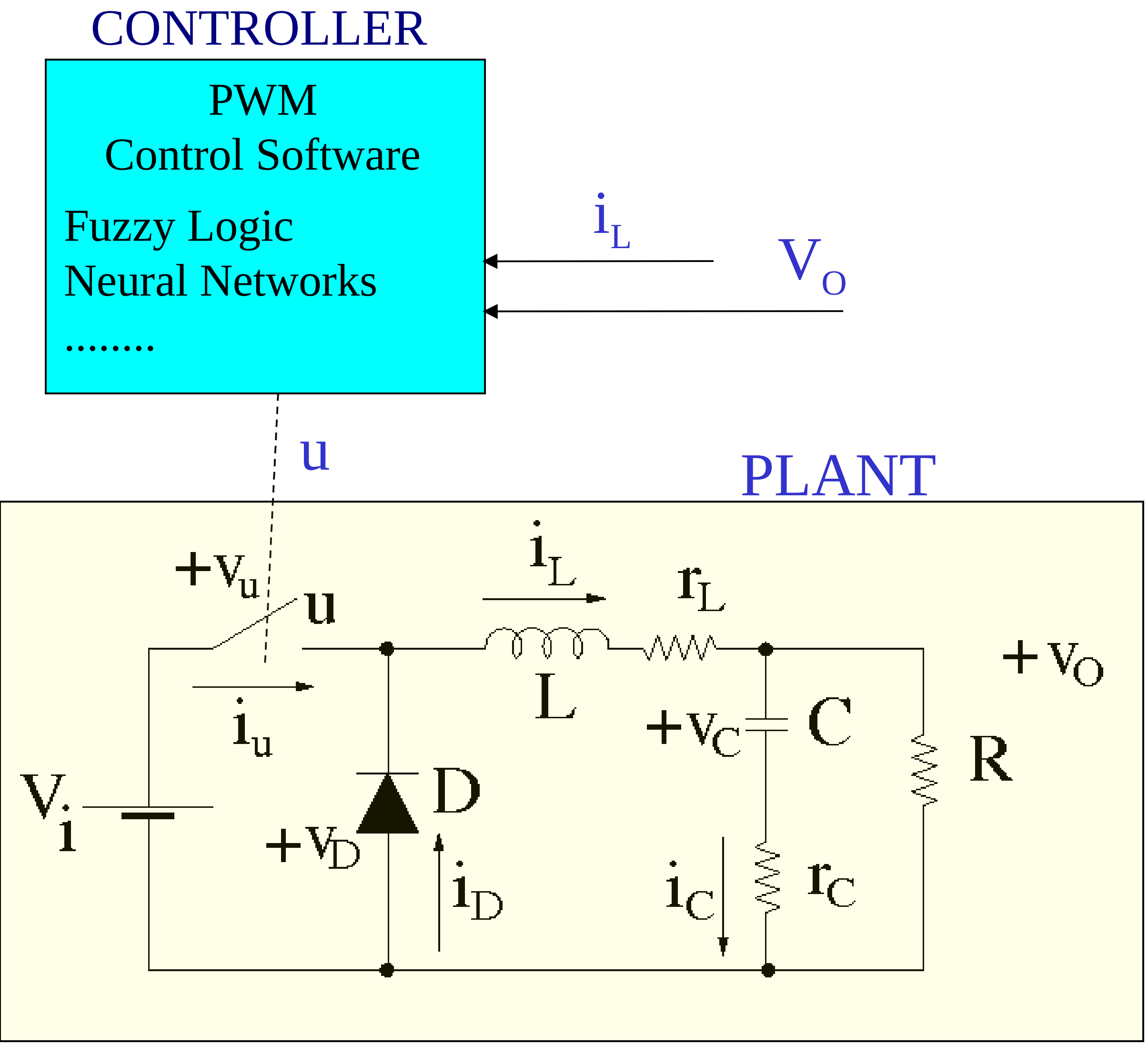
OBDD for Ctr (K)

```
char obdd_in_C(char *x)  {  char return_bit = 1;

L_0x53:   if (x[1] == 1) goto L_0x4f;
          else {return_bit = !return_bit;  goto L_0x52;}

L_0x52:   if (x[2] == 1) goto L_0x4b;  else goto L_1;

L_0x4f:   if (x[2] == 1) goto L_1; else goto L_0x4b;

L_0x4b:   if (x[3] == 1) goto L_1;
           else {return_bit = !return_bit;  goto L_1;}

L_1:  return return_bit;
}
```

Software Implementation of Ctr K

WCET = A * <size of longest path in K OBDD> * <number of U bits>,   where:
A = Time to compute an if-then-else and a goto. Thus: WCET <= A*X_BITS*U_BITS

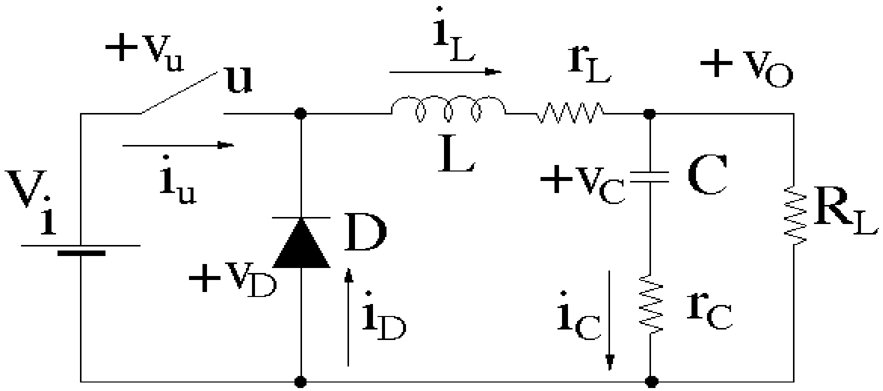
Example: Buck Converter



Applications:

- Consumer Electronics
- Airplanes.
- Satellites.
- Switching power suppliers (off-chip).
- On-chip power suppliers for multicore processors (energy saving).

Experimental Setting



Parameters: $T = 10^{-6}$s,
$L = 2*10^{-4}$H, $r_L = 0.1$ Ohm,
$C = 5*10^{-5}$F, $r_C = 0.1$ Ohm,
$R = 5 \pm 25\%$ Ohm, $Vi = 15 \pm 25\%$ V,
Vref = 5 V, p = 0.01V (converter precision)

Safety Bounds: $|i_L| <= 4$, $-1 <= v_O <= 7$, $|i_u| <= 10^3$, $|i_D| <= 10^3$, $|v_u| <= 10^7$, $|v_D| <= 10^7$.

$I = \{(i_L, v_O) \mid |i_L| <= 2, 0 <= v_O <= 6.5\}$, $G = \{(i_L, v_O) \mid |i_L| <= 2, |v_O - V_{ref}| <= p\}$

Experimental Results: Ctr Abs + Ctr Software

Table shows CPU Time (s) needed to compute a near-optimal control law and its C implementation (K). All computations run within 200MB RAM.
Main Alg returns UNK for b=8, SOL for all other cases. Thus we know, on a formal ground, that for b=10 our synthesized controller works correctly on the desired set of initial states.

Arcs: Arcs in MinCtrAbs (our *close to minimum* Control Abstraction)
MaxLoops: Loops in MaxCtrAbs.
LoopFrac: Fraction of self loos in MaxCtrAbs that is also in MinCtrAbs.

Experiments on an Intel 3.0 Ghz Dual Quad Core Linux Pcwith 4GB of RAM

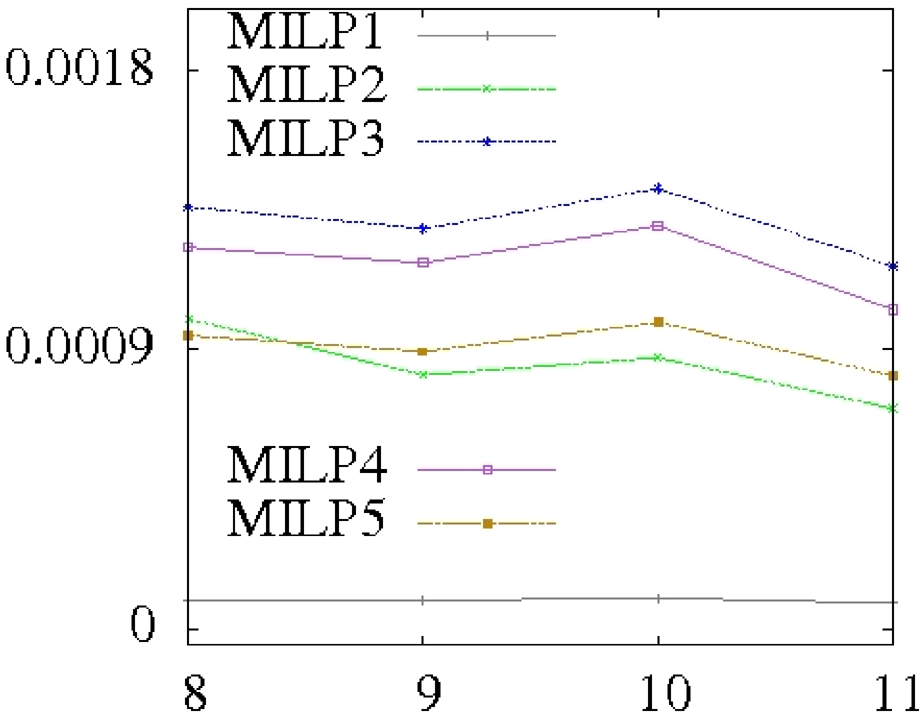
	Control Abstraction				Controller Synthesis		Total CPU
b	CPU (s)	Arcs	MaxLoops	LoopFrac	CPU	OBDD	CPU
8	2.50e+03	1.35e+06	2.54e+04	0.00323	0.00e+00	1.07e+02	2.50e+03
9	1.13e+04	7.72e+06	1.87e+04	0.00440	1.00e+02	1.24e+03	1.14e+04
10	6.94e+04	5.14e+07	2.09e+04	0.00781	7.00e+02	2.75e+03	7.01e+04
11	4.08e+05	4.24e+08	2.29e+04	0.01417	5.00e+03	7.00e+03	4.13e+05

WCET(b=10) = IF_THEN_ELSE_TIME*STATE_BITS*CTR_BITS = $0.5*10^{-7}*20*1 = 10^{-6}$

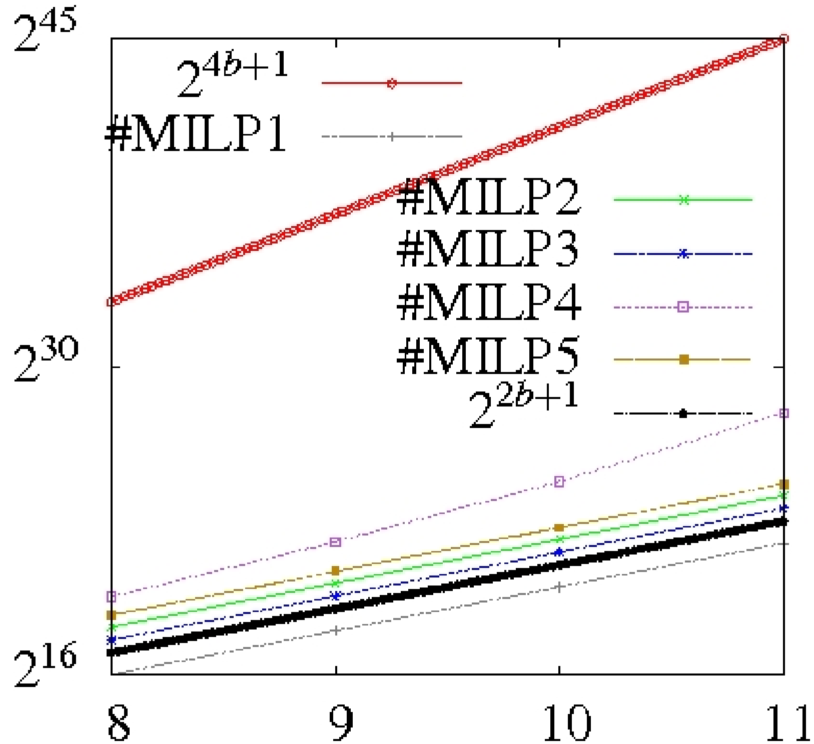
Experimental Results: MILP in Main Algorithm

Avg Execution Time (s) for MILP problems in Main Alg.
This is quite small since all MILPs have about the same size (plant model).

Number of calls to MILP problems in Main Alg. MILP4 most called one.
This is closer to STATE_BITS*CTR_BITS than to STATE_BITS^2*CTR_BITS. This shows effectiveness of Over_Img in main Alg.



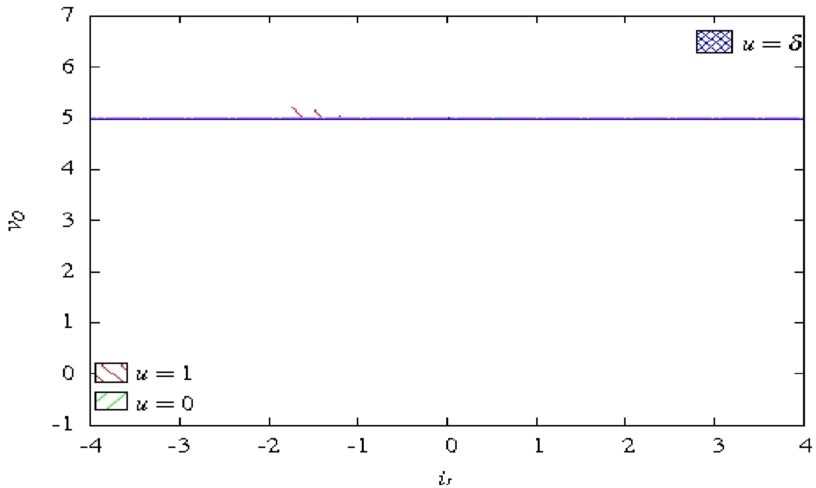
State bits



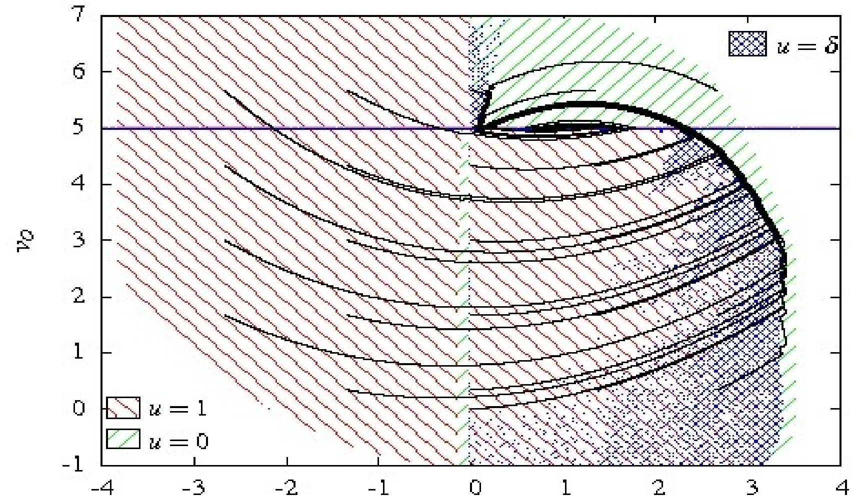
State bits

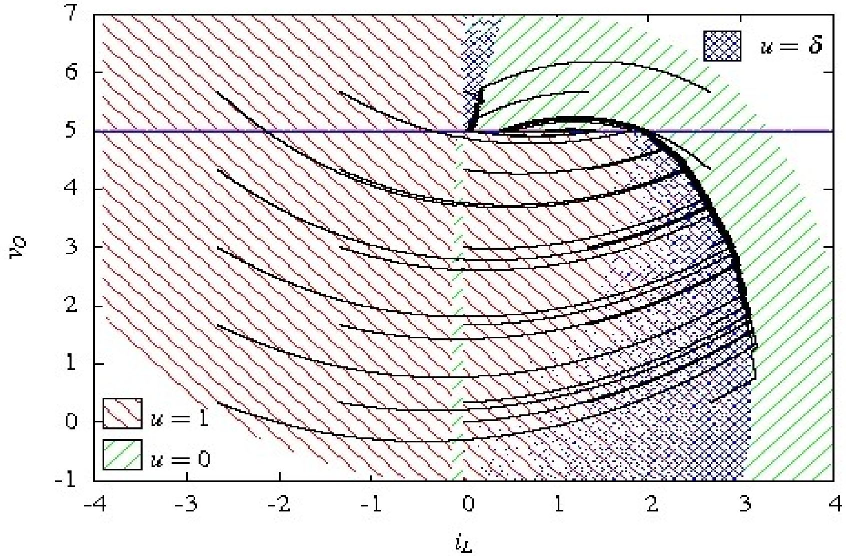
Controllable Regions

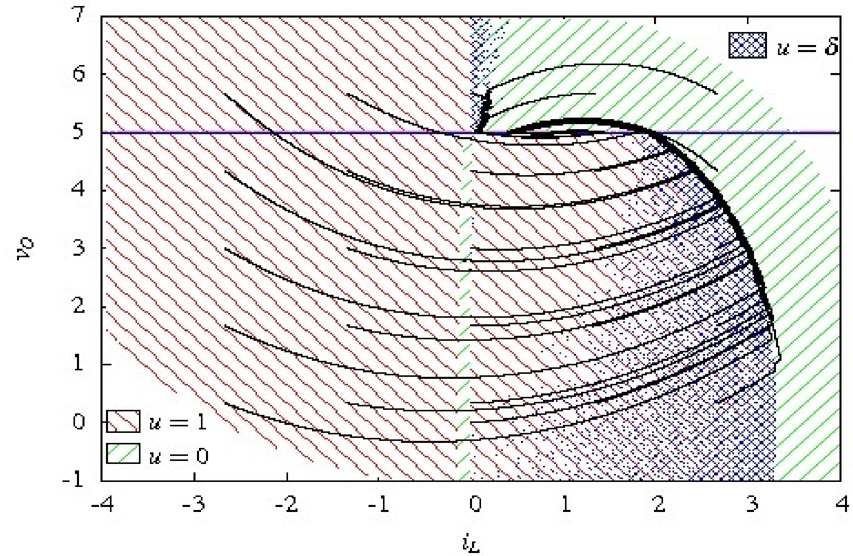
8 bits



9 bits





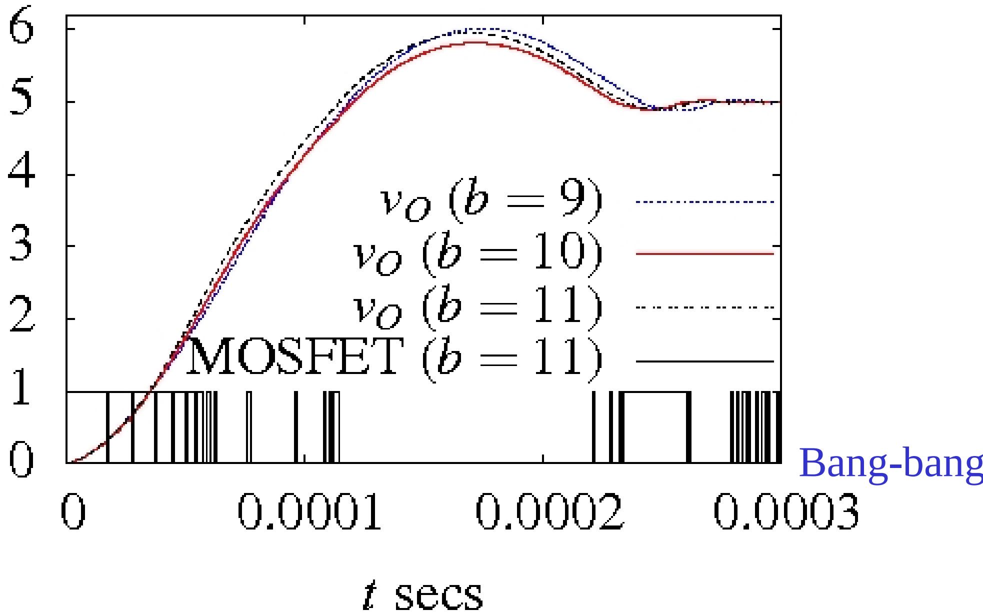


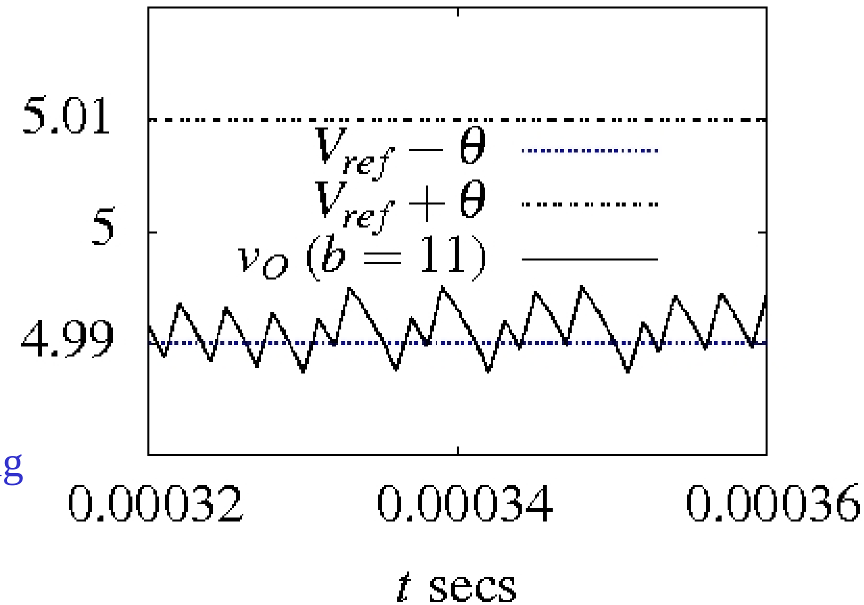
10 bits

Don't cares offer optimization opportunities

11 bits

Control Software Performances
(Hardware in the Loop Simulation)





Transient – 11 bit quantization
State of the art: ~ 2 ms
Automatic Synthesis: ~ 0.3 ms

Ripple – 11 bit quantization
State of the art: ~ 50 mV
Automatic Synthesis: ~ 4 mV

Dark Side
Synthesized Control Software size: about 7K Locs,
Manually designed control software size (fuzzy logic controller): about 300 Locs

Conclusions ...

Automatic Synthesis of **Quantized** Feedback Control Software for DTLHS is possible. Synthesized software properties:

- **Correct-by-construction** (e.g., quantization taken into account, no arithmetical overflows),

- Known **Controllable Region**,

- **Robust** to variations in plant dynamics

- Guaranteed **WCET**

... and Future Work

Fully Symbolic Approaches (e.g, based on quantifier elimination)

Methods to reduce size of synthesized control software (e.g., exploiting don't cares)

Methods to decrease WCET of synthesized control software (e.g., multithread)

Thanks

Main Algorithm(3): Computing Control Abstractions -

Input: A quantization AD, a DTLHS H = (X, U, Y, N), a control problem (H, I, G).

Output: OBDDs for: **N** (transition relation of MinCtrAbs), **I** (quantization of I), **G** (quantization of G)

```
minCtrAbs(AD, H, I, G)  {
  X = [x_1, ... x_n];    X' = [x_1', ... x_n'];   U = [u_1, ... u_r];  N(X, U, Y, X') = 0;   I(X) = 0;  G(X) = 0;
  forall (s in AD(STATE)) do {       // MILP (… X …) is feasible = EXISTS X ( … )
          if (MILP (I(X) ∧ AD(X) == s) is feasible)  I = I ∪ {s};         // MILP1 add initial state
          if (MILP (G(X) ∧ AD(X) = s) is feasible)  G = G ∪ {s};     // add goal state
          forall (u in AD(CTR)) do {
                if (MILP(N(X, U, Y, X') ∧ AD(X) = s ∧ AD(U) = u ∧
                              X' is not in STATE) is feasible) {continue;}     // MILP5 skip unsafe
                if (SelfLoop(s, u)) {N = N ∪ {(s, u, s);   }    //  MILP2  add selfloop
                forall (i = 1, ...  n)  do {
                        m_i = x_i'*, where  X'*  = [x_1'*, ... , x_n'*] is a solution to the MILP
                              (min, x_i', N(X, U, Y, X') ∧ AD(X) = s ∧ AD(U) = u);  // MILP3 min reach x
                        M_i = x_i'*, where  X'*  = [x_1'*, ... , x_n'*] is a solution to the MILP
                              (max, x_i', N(X, U, Y, X') ∧ AD(X) = s ∧ AD(U) = u);  }   // max reach x
                Over_Img(s, u) = ∏_{i=1}^n [AD(m_i), AD(M_i)] ;  // Overapprox of 1-step reachable x
                forall (s'  in  Over_Img(s, u)) do  {
                      if  (s != s'  and (MILP (N(X, U, Y, X') ∧ AD(X) = s ∧
                                            AD(U) = u ∧ AD(X') = s') is feasible )    // MILP4
                              N = N ∪ {(s, u, s');  } } }     // add transition   -  end exploration
  return (N, I, G);  }
```