Formal Approaches to Model-Based Development of Real-Time/Hybrid Systems

Kunihiko HIRAISHI
School of Information Science
Japan Advanced Institute of Science and Technology

Contents

- Formal Modeling and Verification of
  - Reactive Systems
  - Real-Time / Hybrid Systems
- Techniques and Algorithms for the Verification of Hybrid Systems
  - Discrete Abstraction
  - Symbolic Simulation
  - Polyhedral Libraries
  - Quantifier Elimination
  - MLD systems and MIQP Solvers

Design Validation

- Design validation: ensuring the correctness of the design at the *earliest* stage possible.
- Currently practices methods: *simulation* and *testing*.
  - One is never sure when they have reached their limits or even an estimate of how many bugs may still lurk in the design.
- The approach of *formal verification* is an alternative to these techniques.
  - While simulation and testing explore *some* of the possible behavior of the systems, formal verification conducts *an exhaustive exploration* of all possible behaviors.

Model Checking

- *Model checking* is one of approaches to formal verification. Compared with other approaches, it has the following advantages:
  - It is fully automatic, and its application requires no user supervision or expertise in mathematical disciplines such as logic and theorem proving.
  - When the design fails to satisfy a desired property, the process of model checking always produces *a counterexample* that demonstrates a behavior which fails the property and is useful for fixing the problem.
- Basically, model checking is applied to finite-state systems.

The Process of Model Checking

Modeling: Convert a design into a formalism accepted by a model checking tool.

Specification: State the properties that the design must satisfy. It is common to use *temporal logic* (CTL, LTL, ...).

Verification: Check that the model of the design satisfies the specification. When the answer is *no*, the model checking algorithm usually provides an error trace which will be used debugging.

Reactive Systems

- A reactive system is a system that maintains an ongoing interaction with its environment.
- Reactive systems include
  - concurrent programs
  - embedded and process control programs,
  - operation systems, …
  - These systems must be highly reliable.

Example: Concurrent Program

P = m: **cobegin** $P_0 \parallel P_1$ **coend** m'.

P_0 :: l_0: **while** *True* **do**

NC_0: **wait**($turn = 0$);

CR_0: $turn := 1$; Critical region

end while

l_0'.

P_1 :: l_1: **while** *True* **do**

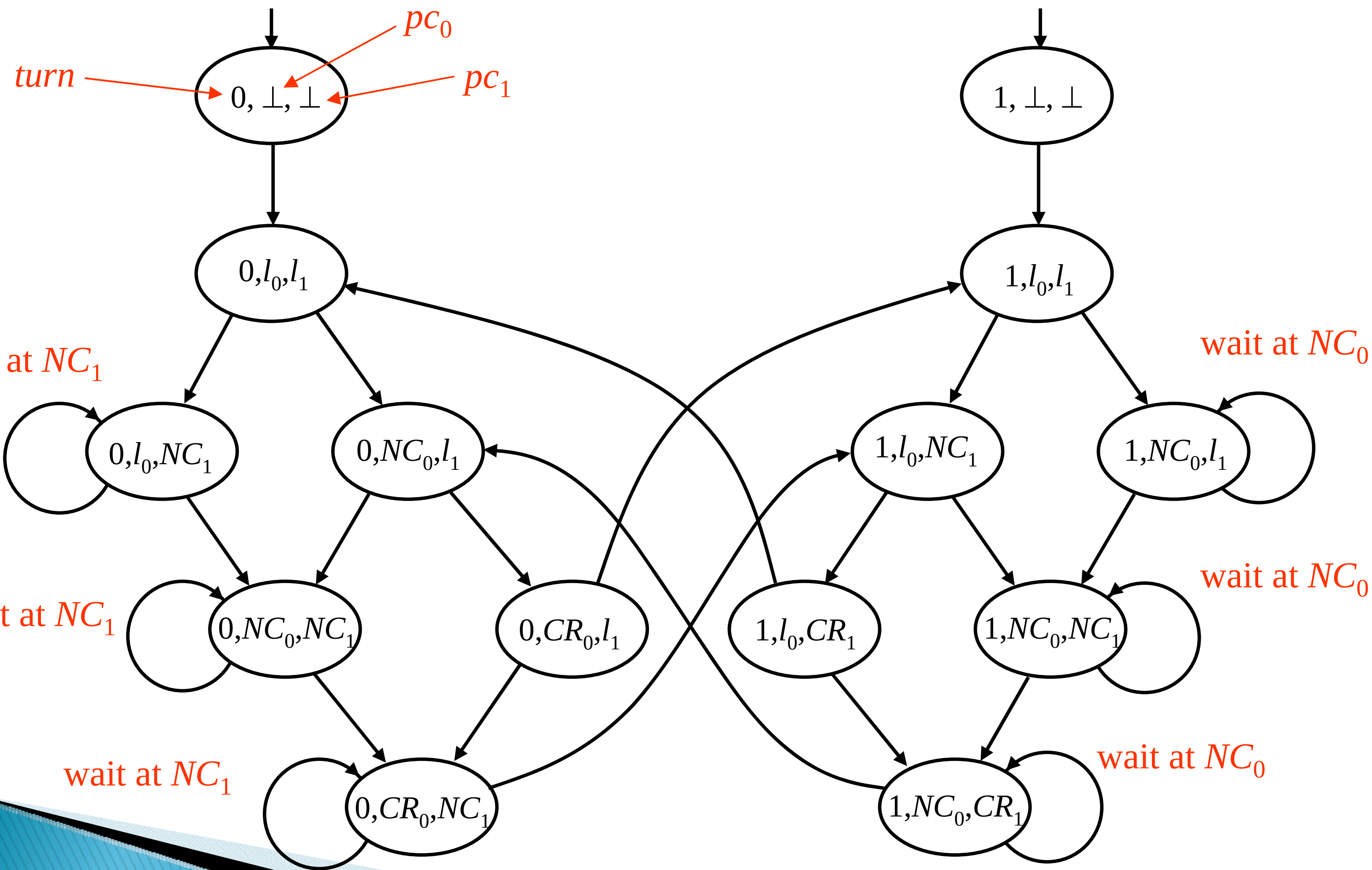
NC_1: **wait**($turn = 1$);

CR_1: $turn := 0$; Critical region

end while

l_1'.

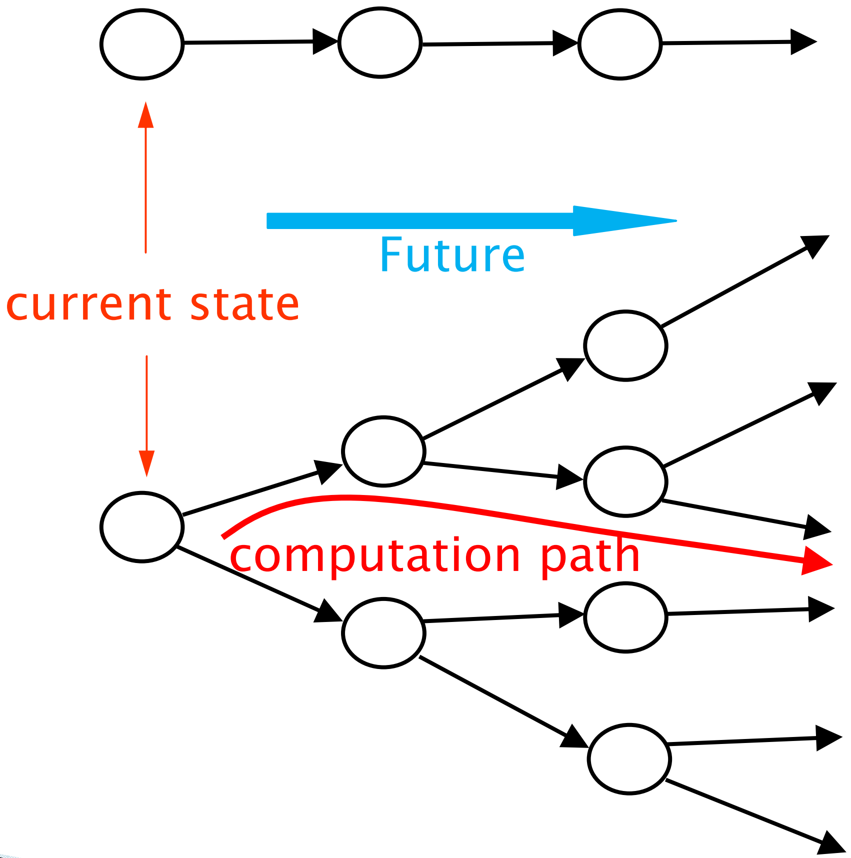
Modeling: Kripke Structure



Specification: Temporal Logic

- *Temporal logic* is a formalism for describing properties on sequences of transitions in discrete state systems.
- Temporal logic was first suggested by Pnueli in 1977 as a tool for the verification of concurrent programs. There exist various versions of temporal logics.
- In this talk, a version of temporal logic called *CTL* (*Computation Tree Logic*) is considered. CTL is *a branching time* temporal logic.

Linear Time and Branching Time



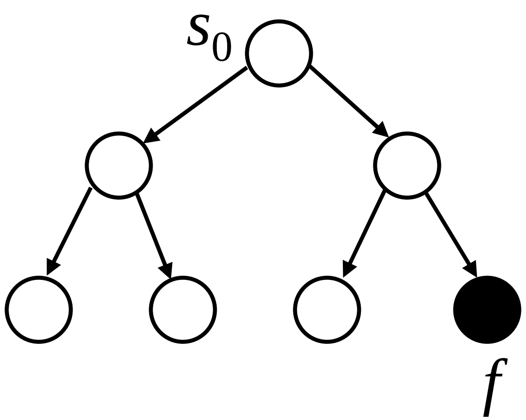
Linear time:
single computation path

Branching time:
multiple computation paths

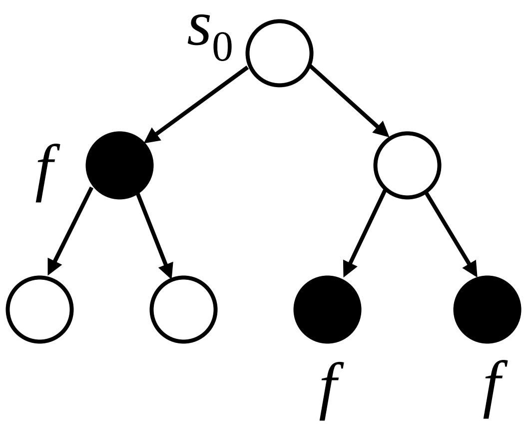
CTL - semantics

- State Formula
  - E g : g holds for some computation paths (Exist).
  - A g : g holds for all computation paths (All).
- Path formula
  - X g: g holds in the *next* state (neXt).
  - F g : g holds at some state on the path (Future).
  - G g : g holds at every state on the path (Globally).
  - g_1 U g_2: g_1 is true *until* g_2 becomes true (Until).

CTL - Semantics

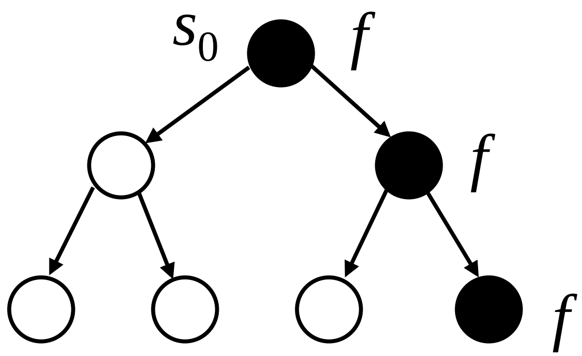


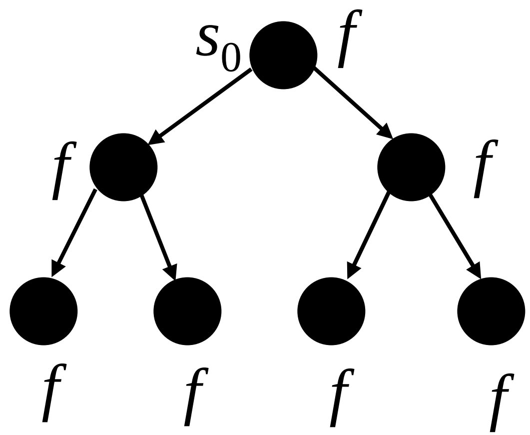
$<M, s_0> \models \mathrm{EF}\ f$



$<M, s_0> \models \mathrm{AF}\ f$

CTL - Semantics





$< M, s_0 > \models \mathrm{EG}\, f$

$< M, s_0 > \models \mathrm{AG}\, f$

Representing Properties by CTL

- Mutual exclusion:

$$\mathsf{AG}\neg(pc_0 = CR_0 \wedge pc_1 = CR_1).$$

- Each process never waits forever:

$$\mathsf{AG}(pc_0 = NC_0 \rightarrow \mathsf{AF}(pc_0 = CR_0)) \wedge$$
$$\mathsf{AG}(pc_1 = NC_1 \rightarrow \mathsf{AF}(pc_1 = CR_1)).$$

SMV: Symbolic Model Verifier

- The SMV system is a tool developed in CMU for checking finite state system against specifications in the temporal logic CTL.

  http://www-2.cs.cmu.edu/~modelcheck/smv.html
- It provides a programming language for describing the transition relation of a finite Kripke structure.
- All computations are performed on ROBDDs.

SMV: Input Language

```
MODULE main
VAR
   turn : boolean;
   p1 : process proc1(turn);
   p2 : process proc2(turn);
ASSIGN
   init(turn) := { 0, 1 };
SPEC
   AG !(p1.state = CR & p2.state = CR)
SPEC
   AG (p1.state = NC -> AF p1.state = CR)
```

SMV: Input Language

```
MODULE proc1(turn)
VAR
   state : { L0, NC,CR };
ASSIGN
   init(state) := L0;
   next(state) :=
         case
                   state = L0 :  NC;
                   state = NC & !turn : CR;
                   state = CR : L0;
                   1 : state;
         esac;
   next(turn) :=
         case
                   state = CR : 1;
                   1 : turn;
         esac;
```

SMV: Verification Result

```
% smv concurrent.smv
-- specification AG (!(p1.state = CR & p2.state = CR)) is true
-- specification AG (p1.state = NC -> AF p1.state = CR) is false
-- as demonstrated by the following execution sequence
state 1.1:
turn = 0
p1.state = L0
p2.state = L0
[stuttering]
state 1.2:
[executing process p1]
-- loop starts here --
state 1.3:
p1.state = NC
[stuttering]
state 1.4:
[stuttering]

resources used:
user time: 0.01 s, system time: 0 s
BDD nodes allocated: 667
Bytes allocated: 1245184
BDD nodes representing transition relation: 61 + 6
```

Theorem Proving

- Theorem proving is an alternative way for the formal verification.

	Theorem Proving	Model Checking
State Space	Infinite	Finite
Verification Procedure	Limited Automatic	Fully Automatic
Counter Example	No Automatic	Automatic
Obtaining Insight of the Systems	Tell how the system is correct	Tell how the system is incorrect

Isabelle/HOL

- Isabelle is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus.
- Isabelle/HOL is the specialization of Isabelle for HOL, which abbreviates Higher-Order Logic.
  http://www.cl.cam.ac.uk/research/hvg/Isabelle/index.html

Isabelle/HOL: Theory

A theory is a named collection of types, functions, and theorems, much like a module in a programming language.

```
theory List
imports Datatype
begin

datatype 'a list = Nil ("[]")
                 | Cons 'a "'a list" (infixr "#" 65)

primrec app :: "'a list => 'a list => 'a list" (infixr "@" 65)
where
"[] @ ys = ys" |
"(x # xs) @ ys = x # (xs @ ys)"

primrec rev :: "'a list => 'a list" where
"rev [] = []" |
"rev (x # xs) = (rev xs) @ (x # [])"
```

Isabelle/HOL: Proof Script

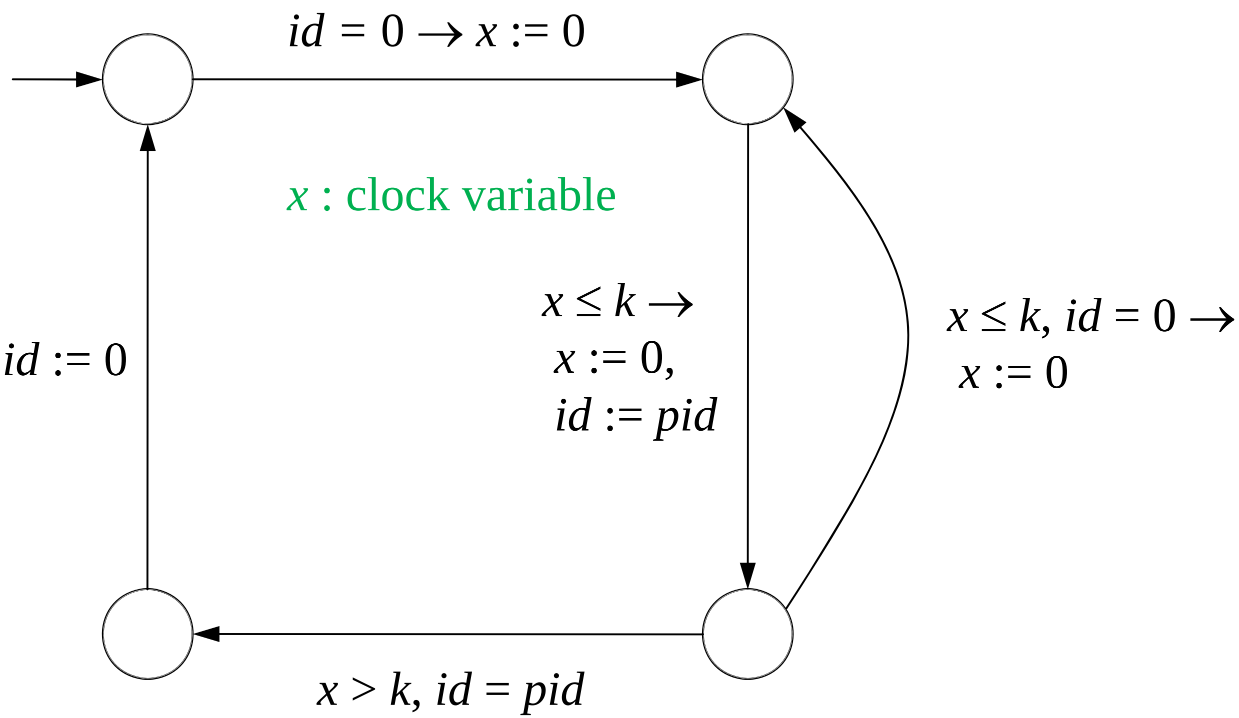
```
lemma app_Nil2 [simp]: "xs @ [] = xs"      ← Subgoals
apply(induct_tac xs)
apply(auto)
Done

lemma app_assoc [simp]: "(xs @ ys) @ zs = xs @ (ys @ zs)"
apply(induct_tac xs)
apply(auto)
done

lemma rev_app [simp]: "rev(xs @ ys) = (rev ys) @ (rev xs)"
apply(induct_tac xs)
apply(auto)
done

theorem rev_rev [simp]: "rev(rev xs) = xs"      ← Goal
apply(induct_tac xs)
apply(auto)
done
end
```

System support for automatic generation and proof of subgoals.

Real-Time Systems

- Real-time systems maintain a continuous interaction with their environment and are often subject to timing constraints, i.e., operational deadlines from event to system response.

Timed Automaton



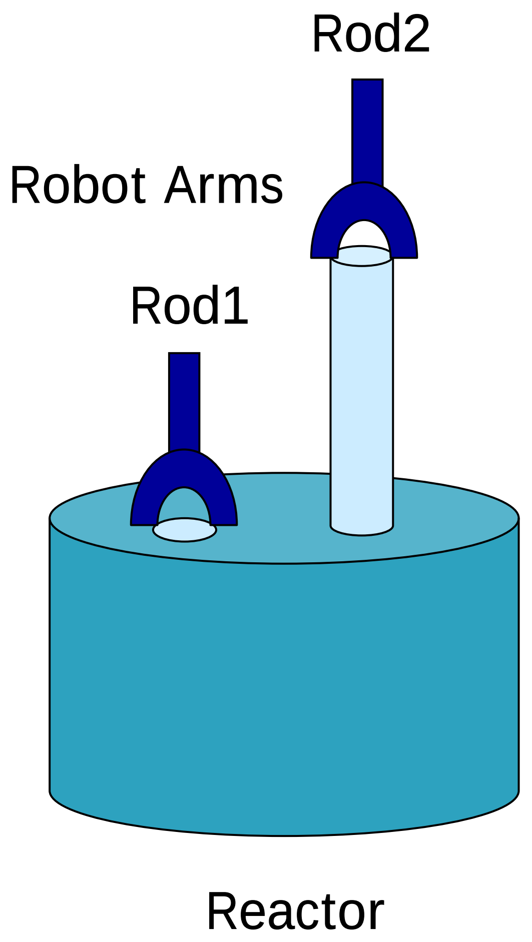
Fisher’s Mutual Exclusion Protocol

Tool: UPPAAL http://www.uppaal.com/

Hybrid Systems

- Hybrid systems combine both digital and analog components.
- Hybrid systems have been used as mathematical models for many important applications, such as
  - automated highway systems,
  - air-traffic management systems,
  - embedded automotive controllers,
  - manufacturing systems,
  - chemical processes,
  - robotics,
  - real-time communication network,
  - real-time circuits, ...

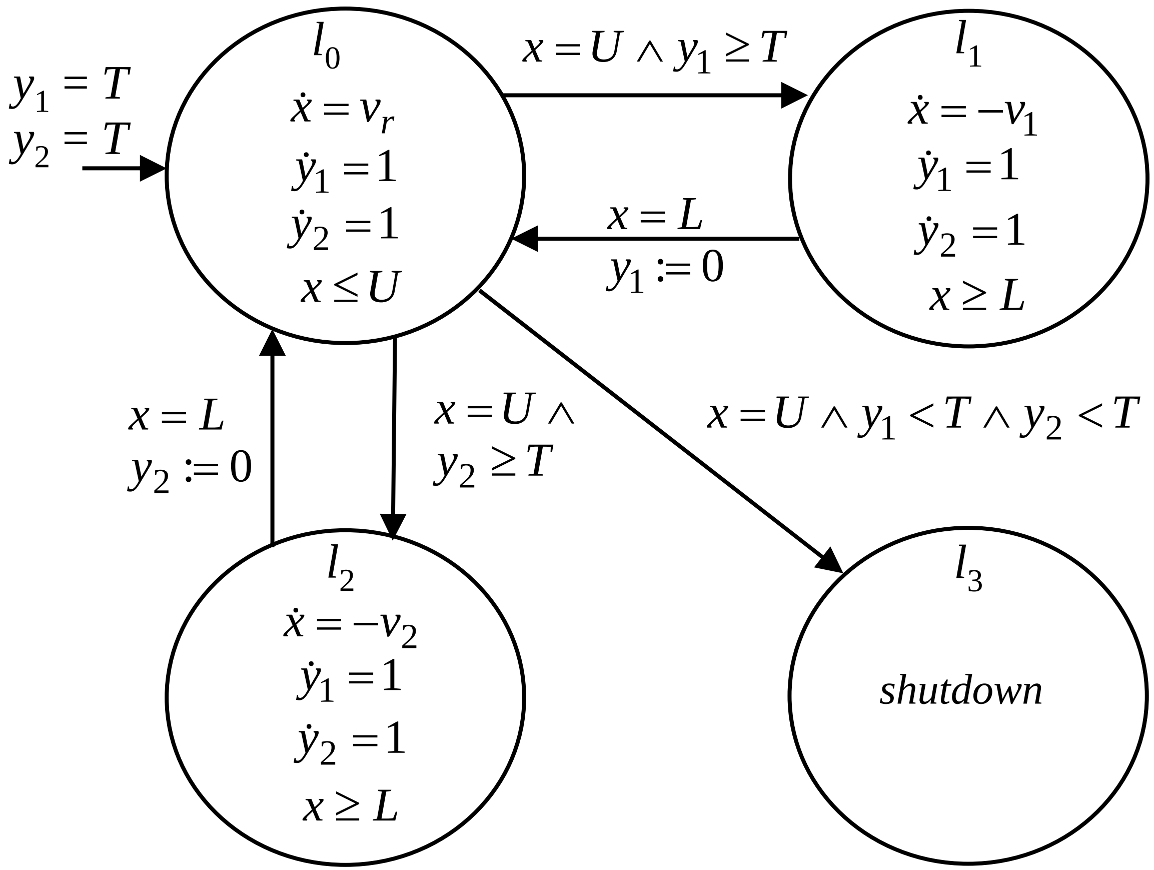
Reactor Tank



Reactor Tank

- The system controls the coolant temperature in a reactor tank by moving two independent control rods.
- The goal is to maintain the coolant between the temperatures L and U.
- When the temperature reaches its maximum value U, the tank must be refrigerated with one of the rods.
- A rod can be moved again only if T time units have elapsed since the end of its previous movement.
- If the temperature of the coolant cannot decrease because there is no available rod, a complete shutdown is required.

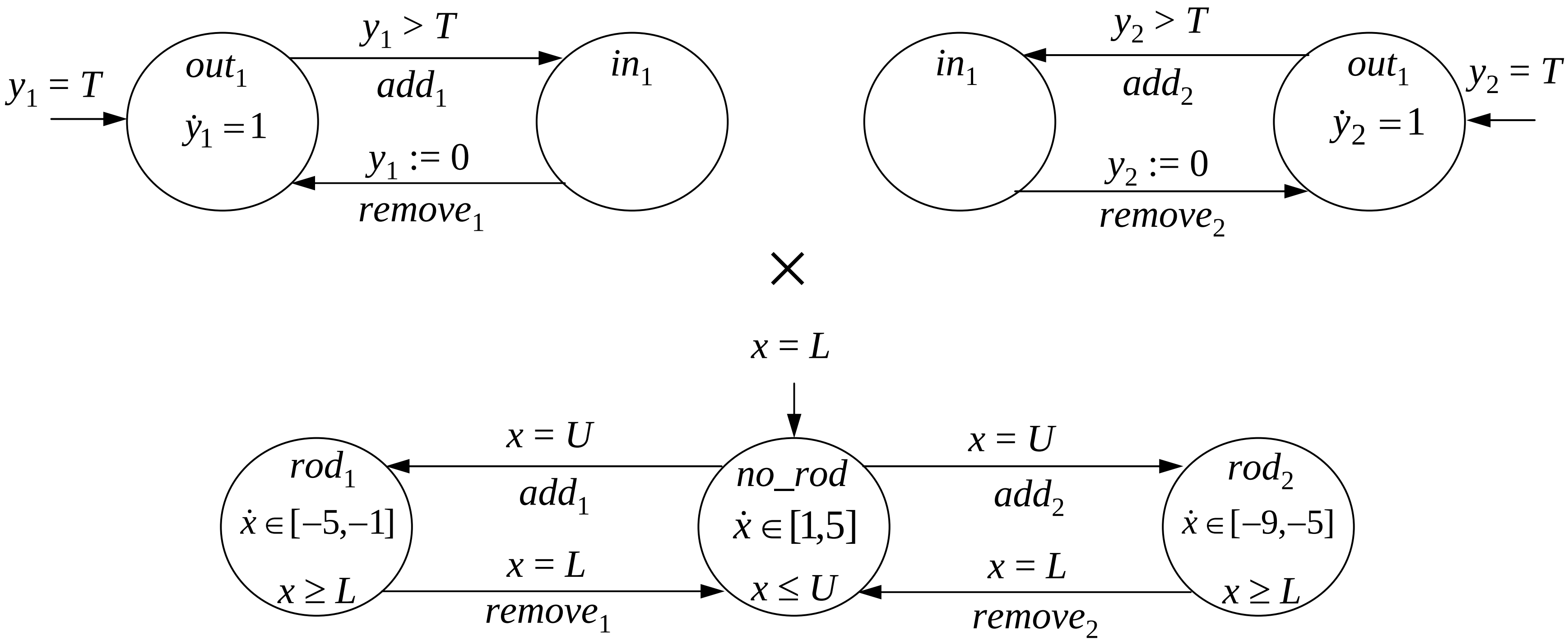
Hybrid Automaton



Hytech: HYbrid TECHnology tool

- HyTech is an automatic tool for the analysis of embedded systems. http://embedded.eecs.berkeley.edu/research/hytech/
- HyTech computes the condition under which *a linear hybrid system* satisfies a temporal requirement. If the verification fails, then HyTech generates a diagnostic error trace.

Reactor Tank: Product Form



Hytech: Input Language

```
var
  y1,          -- timer for rod 1
  y2           -- timer for rod 2
    : clock;
  x            -- clock-translated variable from temperature
    : analog;
  T,           -- minimal time delay before reusing a cooling rod
  L,           -- minimal acceptable temp
  U            -- maximal acceptable temp
    : parameter;
```

variables

Hytech: Input Language

```
automaton rod_1
synclabs: add_1, remove_1;
initially out_1 & y1 = T;

loc out_1: while y1>=0 wait {}
    when y1 >= T sync add_1 goto in_1;

loc in_1: while y1>=0 wait {}
    when True sync remove_1 do {y1' = 0} goto out_1;

end -- rod_1
```

Rod 1

Hytech: Input Language

```
automaton rod_2
synclabs: add_2, remove_2;
initially out_2 & y2 = T;

loc out_2: while y2>=0 wait {}
    when y2 >= T sync add_2 goto in_2;

loc in_2: while y2>=0 wait {}
    when True sync remove_2 do {y2' = 0} goto out_2;

end -- rod_2
```

Rod 2

Hytech: Input Language

```
automaton temp
synclabs: add_1, remove_1, add_2, remove_2;
initially no_rod & x = L;

loc no_rod: while x <= U wait {dx in [1,5]}
   when x=U sync add_1 goto rod_1;
   when x=U sync add_2 goto rod_2;

loc rod_1: while x >= L wait {dx in [-5,-1]}
   when x=L sync remove_1 goto no_rod;

loc rod_2: while x >= L wait {dx in [-9,-5]}
   when x=L sync remove_2 goto no_rod;

end -- temp
```

Tank

Hytech: Input Language

```
var
   init_reg, final_reg, b_reached : region;

init_reg :=   loc[rod_1] = out_1 & y1  = T
           & loc[rod_2] = out_2 & y2 = T
           & loc[temp]  = no_rod & x = L;

final_reg :=   loc[temp] = no_rod & x=U
            & loc[rod_1] = out_1 & y1  <= T
            & loc[rod_2] = out_2 & y2  <= T;

b_reached := reach backward from final_reg endreach;

prints "Control rod NOT available under the following conditions";
print omit all locations hide non_parameters in b_reached &
   init_reg endhide;
```

Specification

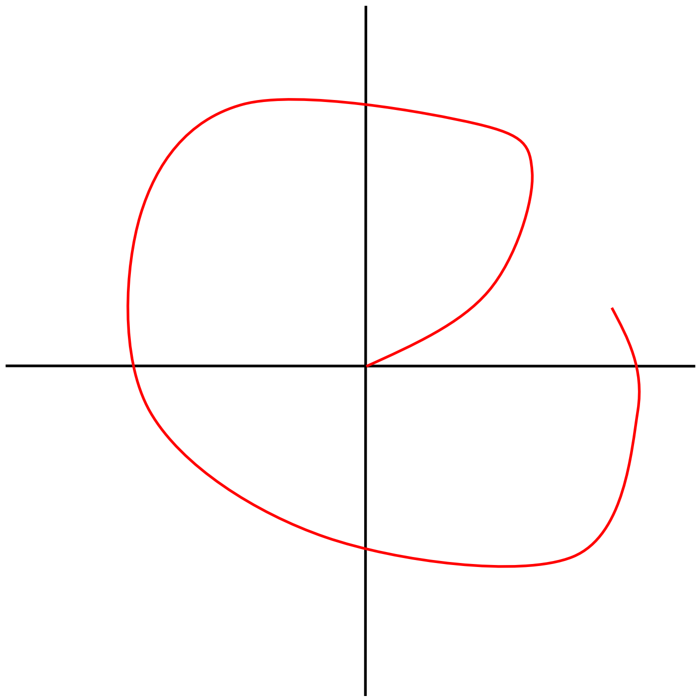
Hytech: Verification Result

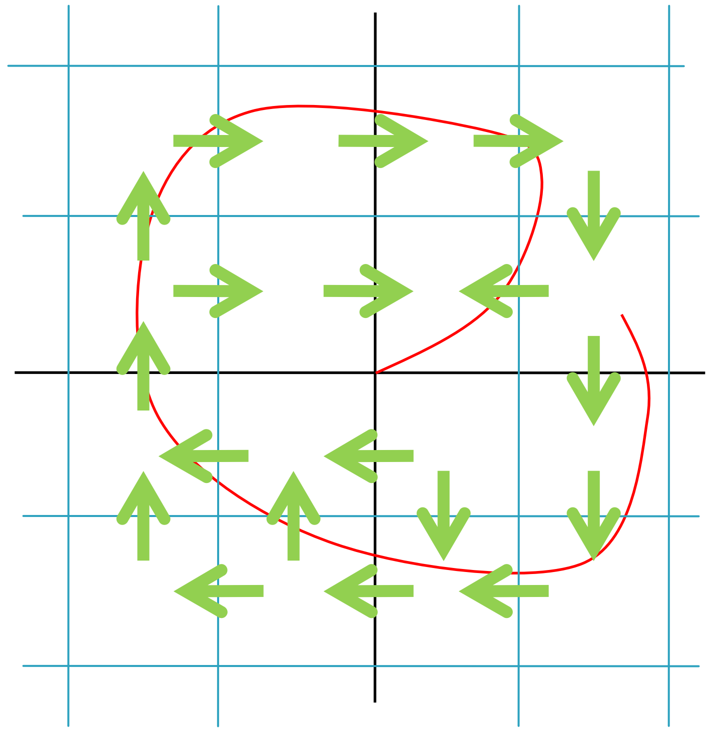
```
================================================
HyTech: symbolic model checker for embedded systems
Version 1.04 10/15/96
For more info:
   email: hytech@eecs.berkeley.edu
   http://www.eecs.berkeley.edu/~tah/HyTech
Warning: Input has changed from version 1.00(a). Use -i for more info
================================================

Number of iterations required for reachability: 8
Control rod NOT available under the following conditions
    23U <= 45T + 23L   & L <= U

================================================
Max memory used =     0 pages =        0 bytes =   0.00 MB
Time spent      =      0.08u +      0.05s =       0.13 sec total
================================================
```

Contents

- Formal Modeling and Verification of
  - Reactive Systems
  - Real-Time / Hybrid Systems
- Techniques and Algorithms for the Verification of Hybrid Systems
  - Discrete Abstraction
  - Symbolic Simulation
  - Polyhedral Libraries
  - Quantifier Elimination
  - MLD systems and MIQP Solvers

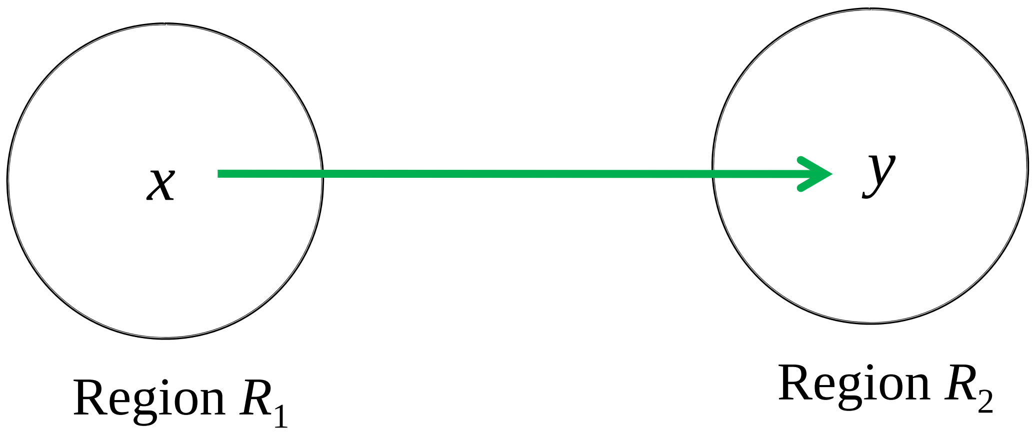
Discrete Abstraction





Partition of State Space

Discrete Abstraction



	Bisimulation	Predicate Abstraction
Approximation	$\forall x \in R_1 \exists y \in R_2 . x \Rightarrow y$	$\exists x \in R_1 \exists y \in R_2 . x \Rightarrow y$
Verification	All CTL formula	Safety property
Partition	To be computed	Given

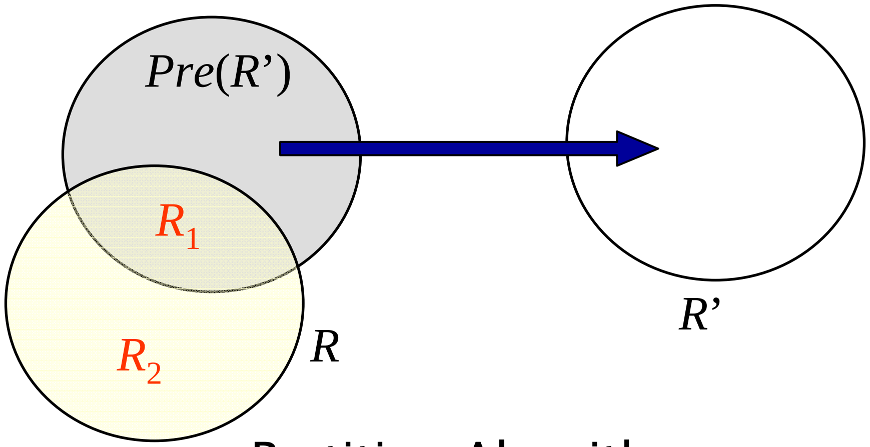
Computing Bisimulation

$$x(t+1) = 0.8\begin{bmatrix} \cos\alpha(t) & -\sin\alpha(t) \\ \sin\alpha(t) & \cos\alpha(t) \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

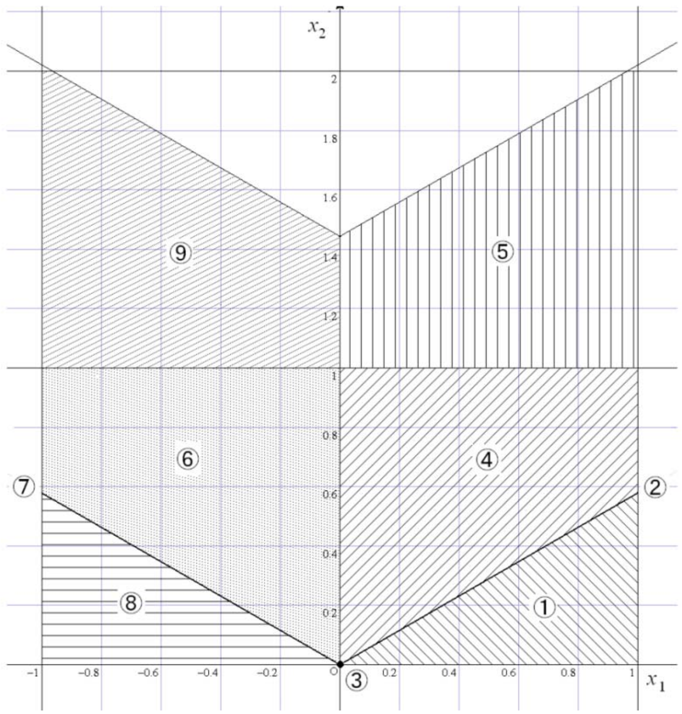
$$\alpha(t) = \begin{cases} \pi/3 & \text{if } \begin{bmatrix}1 & 0\end{bmatrix} x(t) \ge 0 \\ -\pi/3 & \text{if } \begin{bmatrix}1 & 0\end{bmatrix} x(t) < 0 \end{cases}$$

$$u(t) \in \begin{bmatrix}-1 & 1\end{bmatrix}$$

A Piecewise Linear System

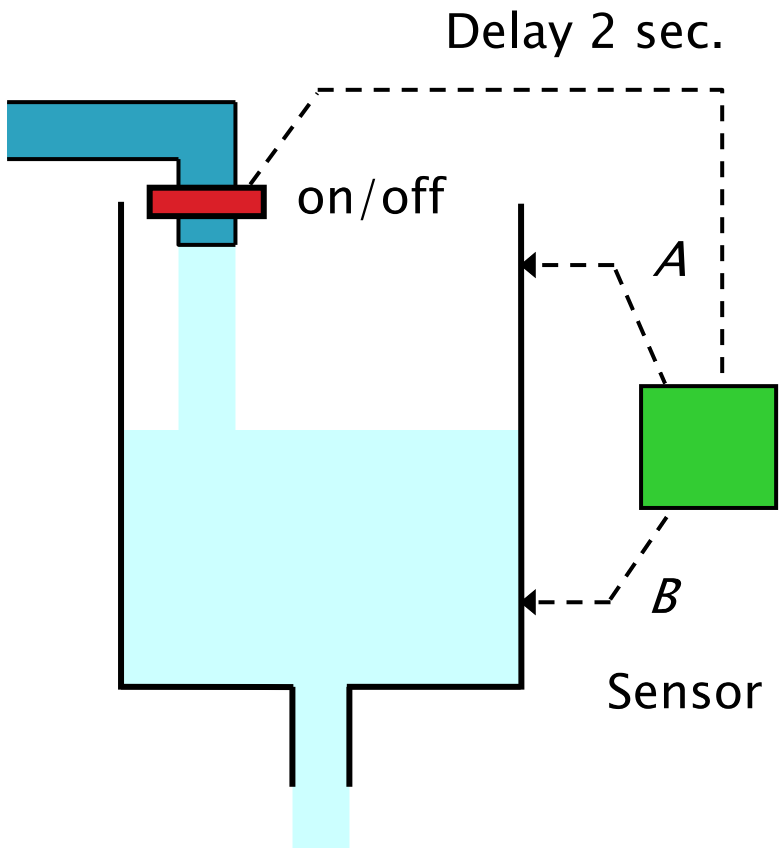


Partition Algorithm



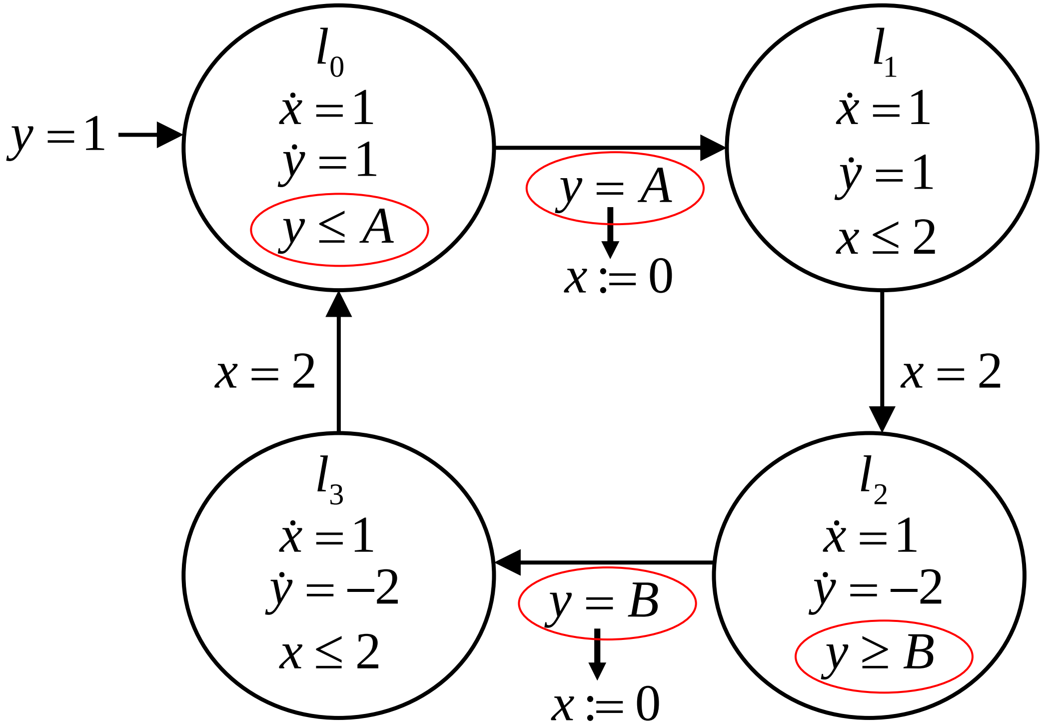
Partition by Bisimulation

Parameter Design: Example



Water Level Monitor

Parameter Design: Formulation

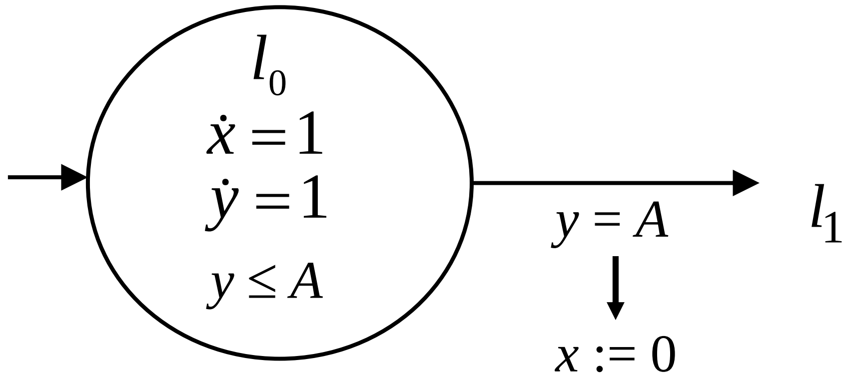


Find the values of A and B so that the water level y always satisfies $1 \le y \le 12$.

Symbolic Simulation

- To compute all possibilities, the region of state values at each time step is computed as *a set of inequalities*.
- Solving the inequalities by mathematical programming methods, we can obtain an optimal values for the parameters.

Implementation by CLP




```
l0(X, Y, Time, A, B):-
        X1 = X + D, Y1 = Y + D, D >= 0,
        Y1 = A,
        l1(0, Y1, Time + D, A, B).
```

CLP: Constraint Logic Programming

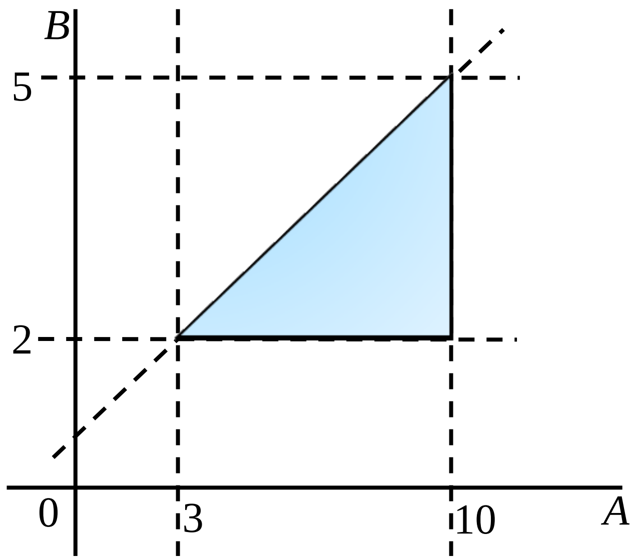
Implementation by CLP

Feasible solution.

```
| ?- l0(X, 1, 0, TT, [A, B]), project([A, B], Z).
A = 10  - _142
B = 12  -2 * _161 - _142
Z = [1 * B >= 5, (-0.5) * B + 0.5 * A >= -1, (-1) * A >= -10]
_142 >= 0
_161 >= 0
-7 =  - _169 -2 * _161 - _142
_169 >= 0

*** yes ***
```

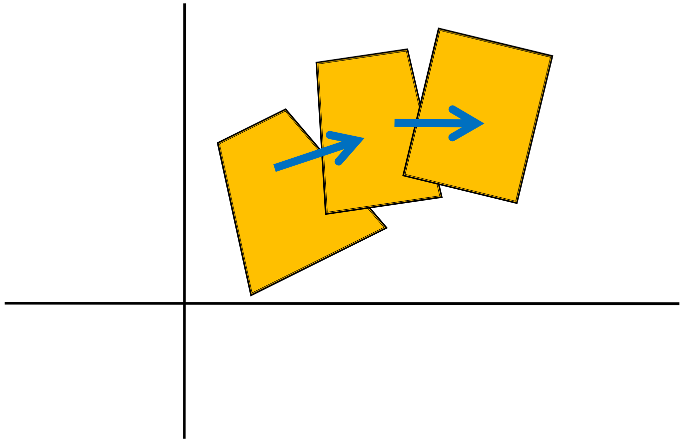
Minimizing the number of switches.

```
| ?- max(TT, l0(X, 1, 0, TT, [A, B])).
A = 10
B = 5

*** yes ***
```



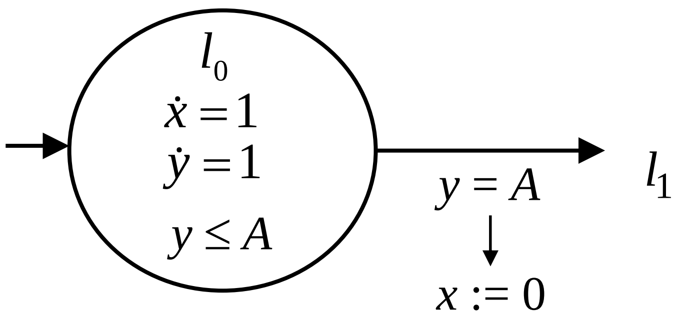
Polyhedral Library

- Manipulations of convex polyhedra are the basis of solving problems on linear hybrid systems.
- There are several libraries for the computation of convex polyhedra.
  - Polylib: http://www.ee.byu.edu/faculty/wilde/polyhedra.html
  - Parma Polyhedra Library, Polka, … , etc.



Quantifier Elimination

- A quantifier elimination (QE) algorithm transforms formulas with quantifiers into equivalent formulas without quantifiers.
- There are several QE algorithms implemented on symbolic computation tools such as Maple, Mathematica, and REDUCE.



$$NextState(x', y', t', A, B) =$$
$$\exists x \exists y \exists x'' \exists y'' \exists d \exists t.$$
$$CurrentState(x, y, t, A, B) \wedge$$
$$x'' = x + d \wedge y'' = y + d \wedge d \geq 0 \wedge y'' = A \wedge$$
$$x' = 0 \wedge y' = y'' \wedge t' = t + d.$$

MLD Systems and MIQP Solvers

- The Mixed Logical Dynamical (MLD) framework is a powerful tool for modeling discrete-time linear hybrid systems.

$$x(k+1) = Ax(k) + B_1u(k) + B_2\delta(k) + B_3z(k)$$
$$y(k) = Cx(k) + D_1u(k) + D_2\delta(k) + D_3z(k)$$
$$E_2\delta(k) + E_3z(k) \le E_4x(k) + E_1u(k) + E_5$$

k is the discrete time-instant, $x(k)$ denotes the states, $u(k)$ the inputs and $y(k)$ the outputs, with both real and binary components. δ and z represent binary and auxiliary continuous variables.

- Optimal control problem for MLD systems can be solved by MIQP (Mixed Integer Quadratic Programming) solvers, such as CPLEX and NUOPT.

Concluding Remarks

- Easy to formalize, hard to solve.
  - Combination of online and offline computations.
  - Guaranteed approximation techniques.