Inductive Verification of Hybrid Automata with Strongest Postcondition Calculus

June 12, 2013 @ iFM

Daisuke Ishii

Tokyo Institute of Technology

Guillaume Melquiond

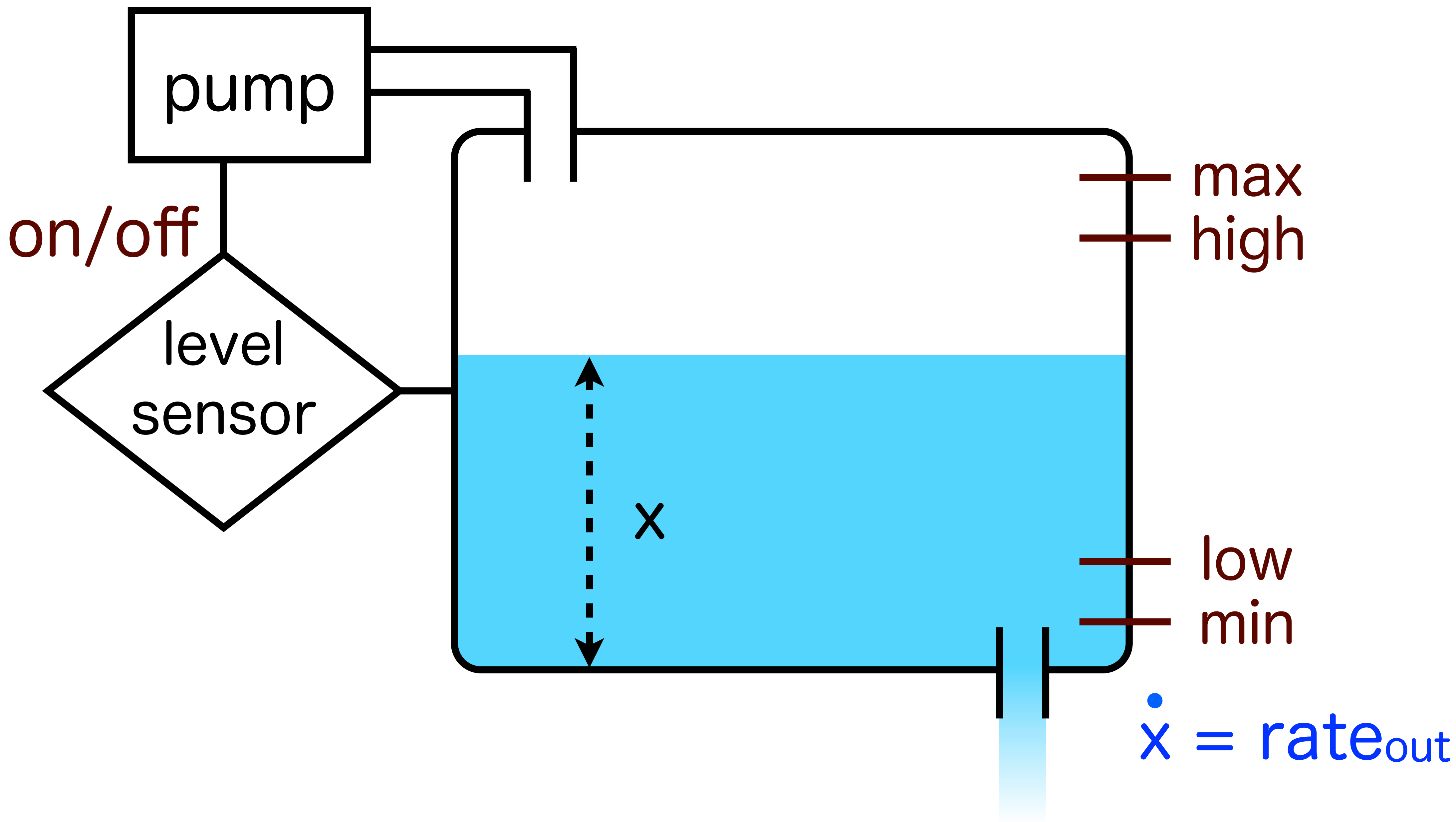
INRIA Saclay / LRI, Université Paris Sud 11

Shin Nakajima

National Institute of Informatics, Tokyo

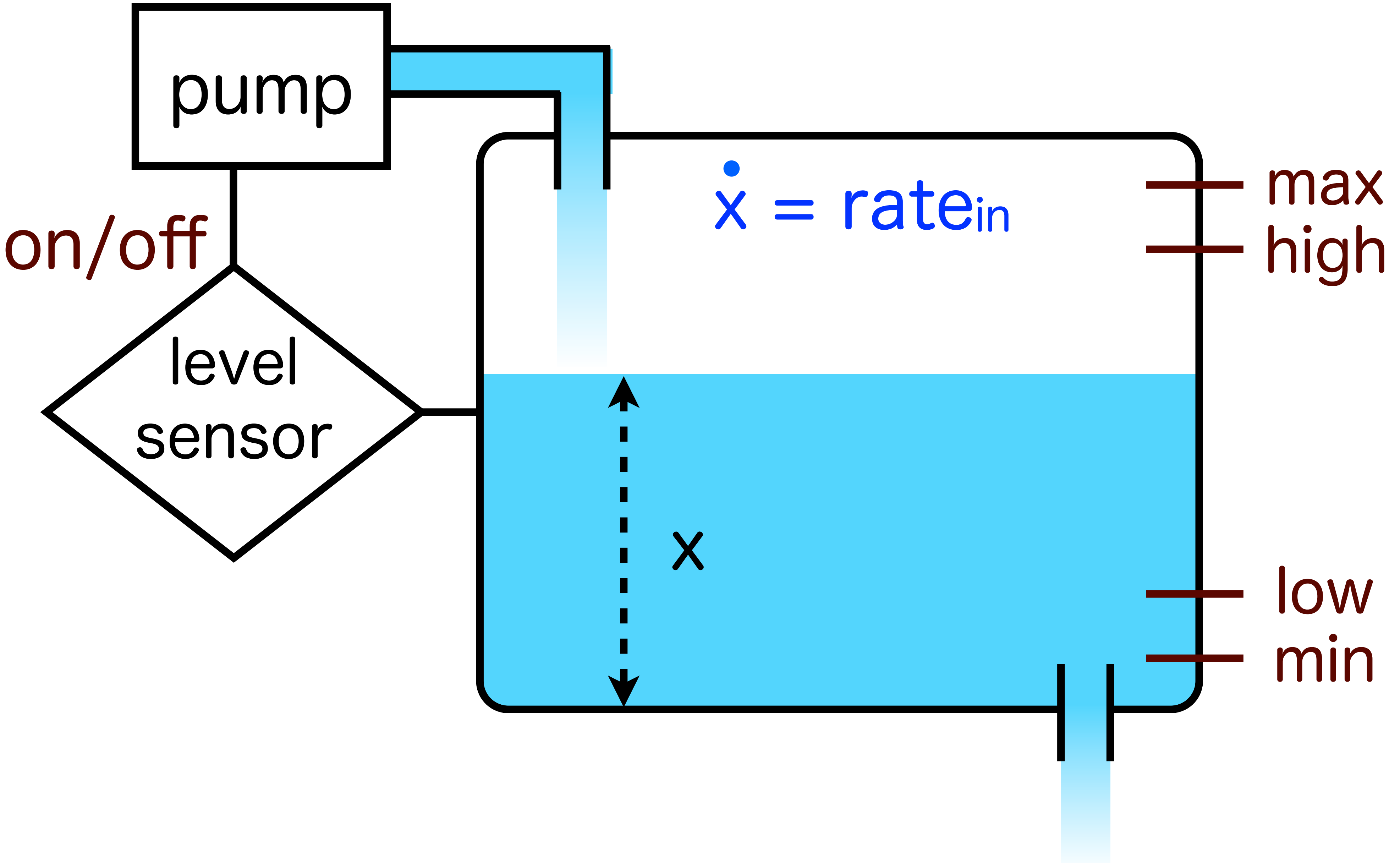
Hybrid Systems

- Systems whose states can make both continuous and discrete changes
- Example: Water-level monitor



Hybrid Systems

- Systems whose states can make both continuous and discrete changes
- Example: Water-level monitor



Verification of Hybrid Systems

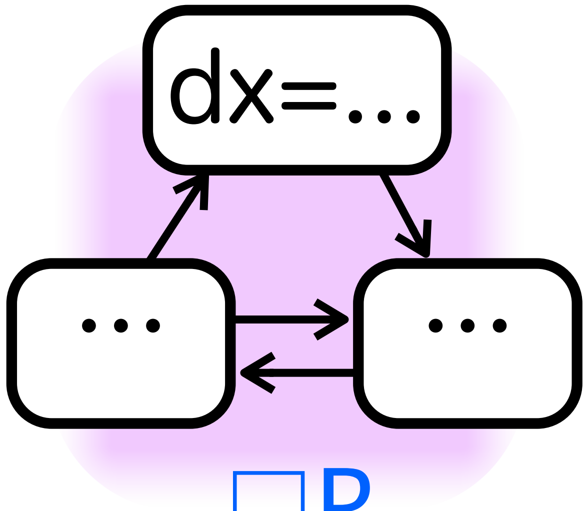
- **Model-checking approach**
  - Based on **hybrid automata**
  - **Many practical automated tools**: e.g. HyTech [Henzinger+ 96] and PHAVer [Frehse 02]
  - **Tractable problems are limited**: small linear models, without uncertain parameters

Verification of Hybrid Systems

- **Model-checking approach**
  - Based on **hybrid automata**
  - **Many practical automated tools**: e.g. HyTech [Henzinger+ 96] and PHAVer [Frehse 02]
  - **Tractable problems are limited**: small linear models, without uncertain parameters
- **Logical analytic approach**
  - Based on annotated **imperative programs**
  - Theoretically studied but **few practical tools**
  - KeYmaera [Platzer+ 08] has been successful
  - **Applicable to larger class of systems**
    - systems with symbolic parameters, nonlinear systems
  - **Verification process is not fully automatic**

Talk Overview

- We propose an **algorithmic logical analytic method** built on top of symbolic arithmetic solvers
  - **Clear and correct scheme**: conjunction of simple transformations
  - **Applicable to a large class of hybrid automata**
    - with nonlinear flows and uncertain parameters

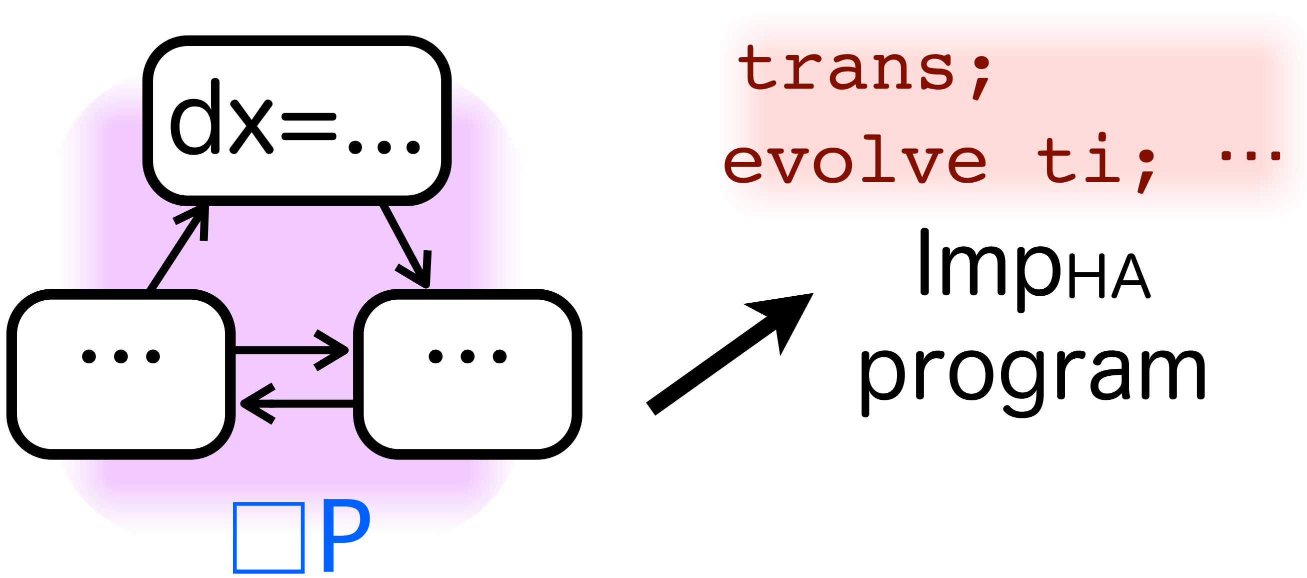


□P

safety verification of hybrid automaton

Talk Overview

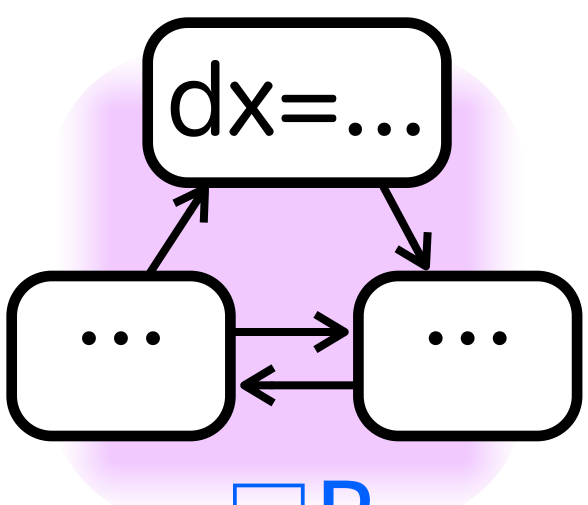
- We propose an **algorithmic logical analytic method** built on top of symbolic arithmetic solvers
  - **Clear and correct scheme**: conjunction of simple transformations
  - **Applicable to a large class of hybrid automata**
    - ✳ with nonlinear flows and uncertain parameters



safety verification of hybrid automaton

Talk Overview

- We propose an **algorithmic logical analytic method** built on top of symbolic arithmetic solvers
  - **Clear and correct scheme**: conjunction of simple transformations
  - **Applicable to a large class of hybrid automata**
    - ✳with nonlinear flows and uncertain parameters



□P

safety verification of hybrid automaton

```
trans;
evolve ti; …
```

Imp_{HA} program

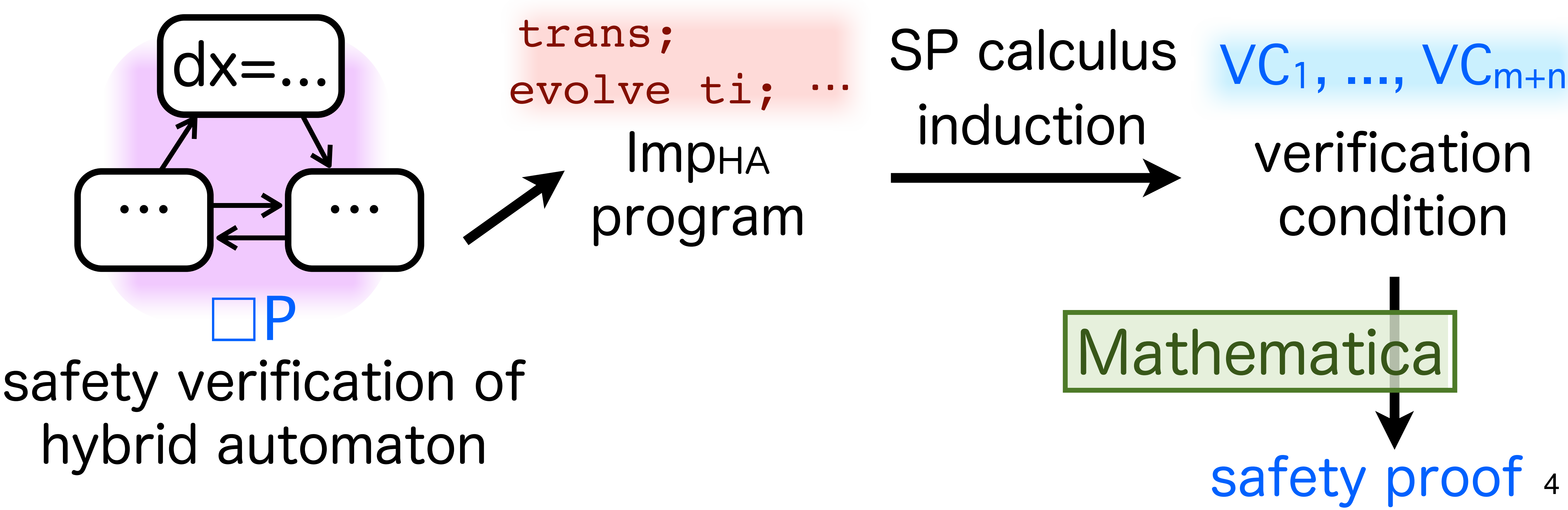
SP calculus
induction

$VC_1, \ldots, VC_{m+n}$

verification condition

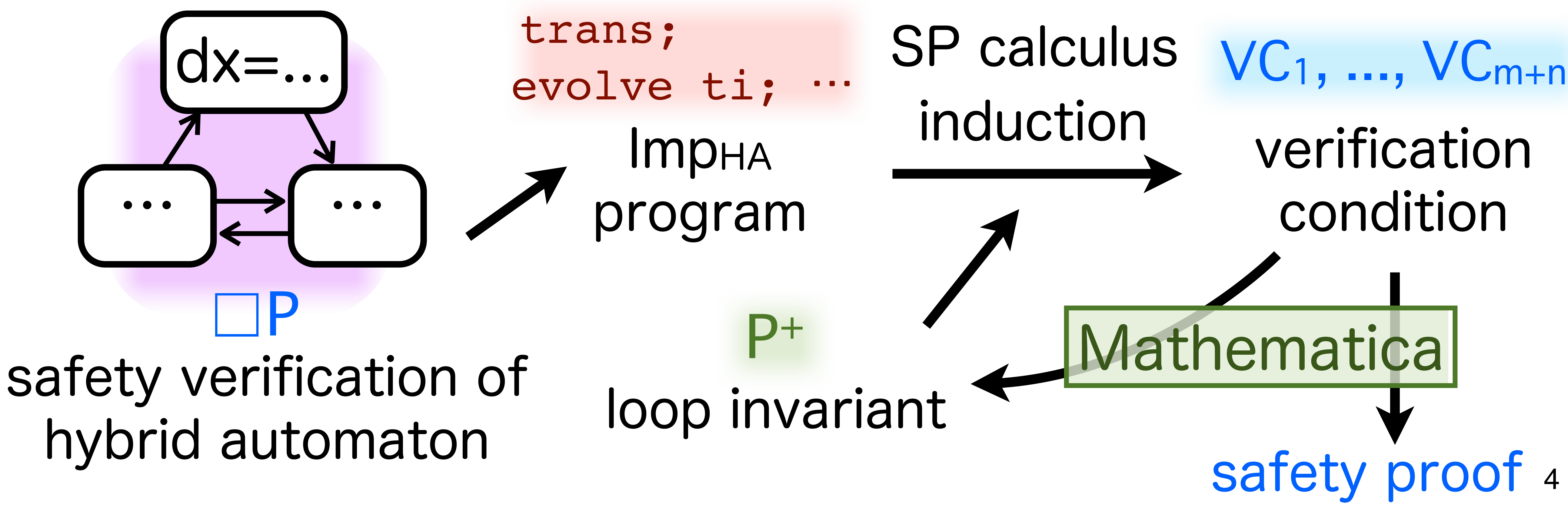
Talk Overview

- We propose an **algorithmic logical analytic method** built on top of symbolic arithmetic solvers
  - **Clear and correct scheme**: conjunction of simple transformations
  - **Applicable to a large class of hybrid automata**
    - ✳ with nonlinear flows and uncertain parameters



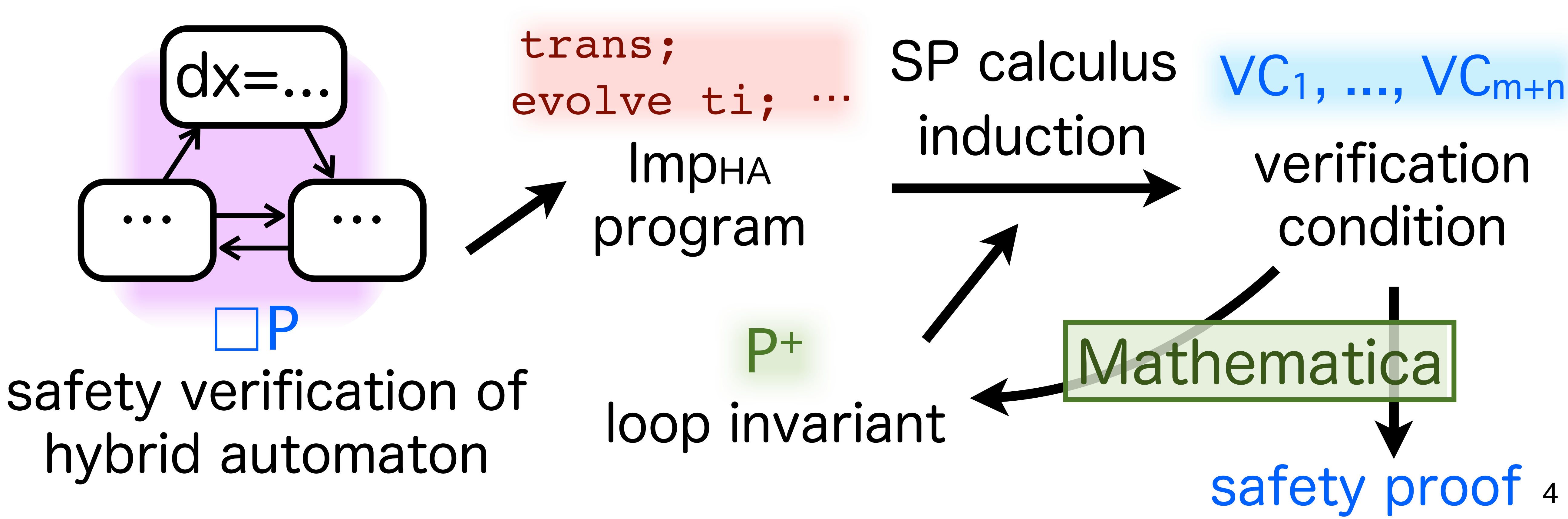
Talk Overview

- We propose an **algorithmic logical analytic method** built on top of symbolic arithmetic solvers
  - **Clear and correct scheme**: conjunction of simple transformations
  - **Applicable to a large class of hybrid automata**
    - ✳ with nonlinear flows and uncertain parameters



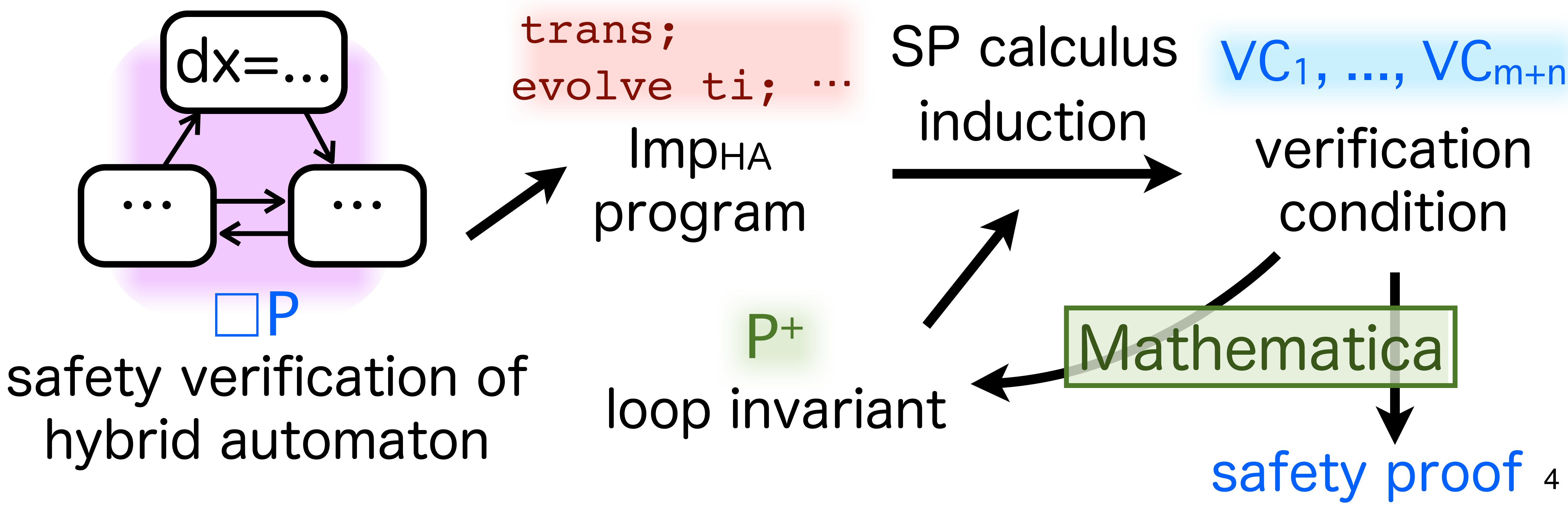
Talk Overview

- We propose an **algorithmic logical analytic method** built on top of symbolic arithmetic solvers
  - **Clear and correct scheme**: conjunction of simple transformations



Talk Overview

- We propose an **algorithmic logical analytic method** built on top of symbolic arithmetic solvers
  - **Clear and correct scheme**: conjunction of simple transformations
  - **Applicable to a large class of hybrid automata**
    - ✳ with nonlinear flows and uncertain parameters



Talk Outline

1. **Hybrid Automata**

2. **Imp$_{HA}$ and Strongest Postcondition Calculus**

3. **Inductive Verification**

4. **Experimental Results**

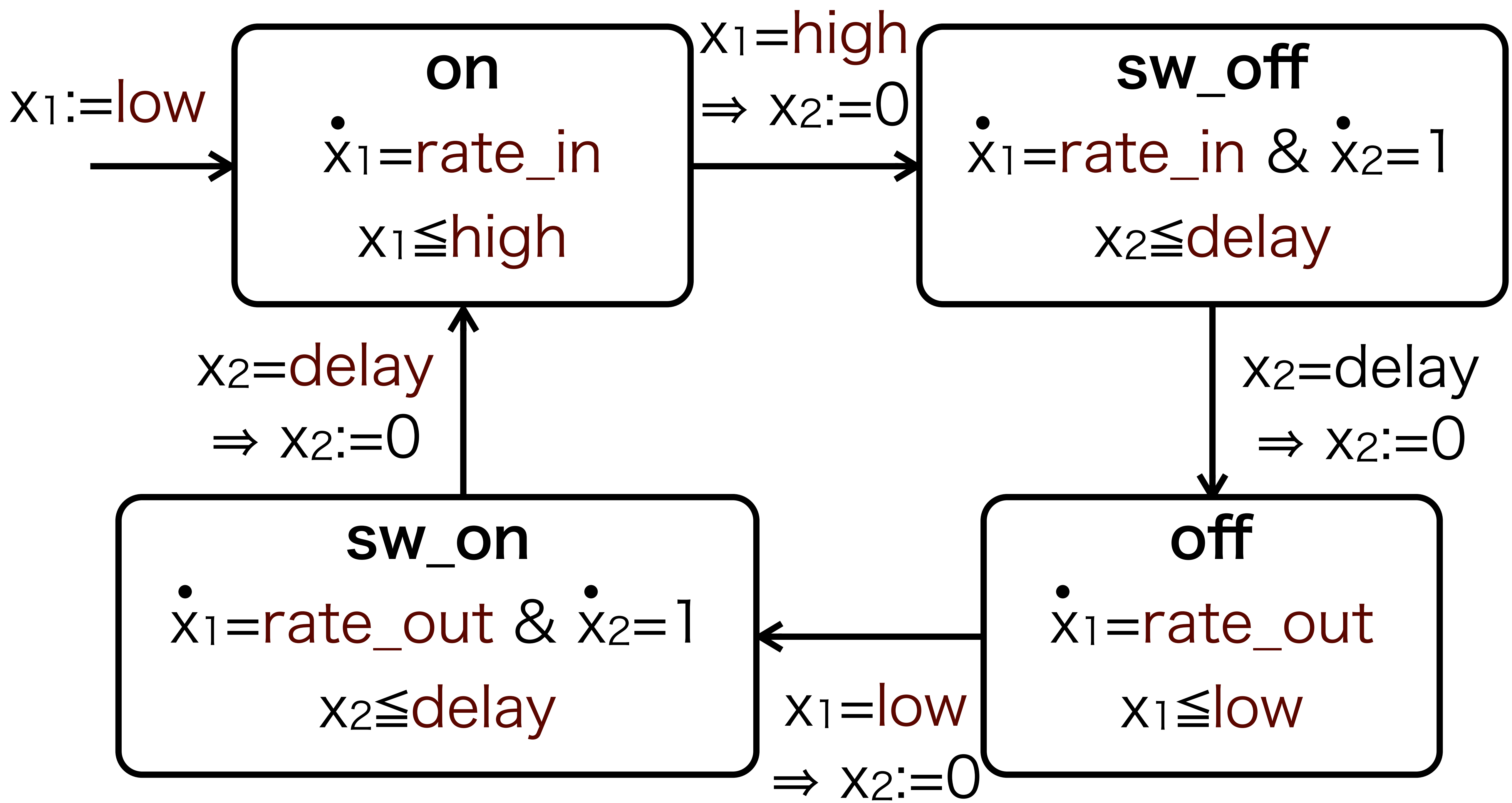
Hybrid Automata (HA)

- **Mathematical model of hybrid systems**
  - Discrete aspect is described by an automaton
  - Continuous dynamics is described by a differential equation indexed by locations of the automaton

Hybrid Automata (HA)

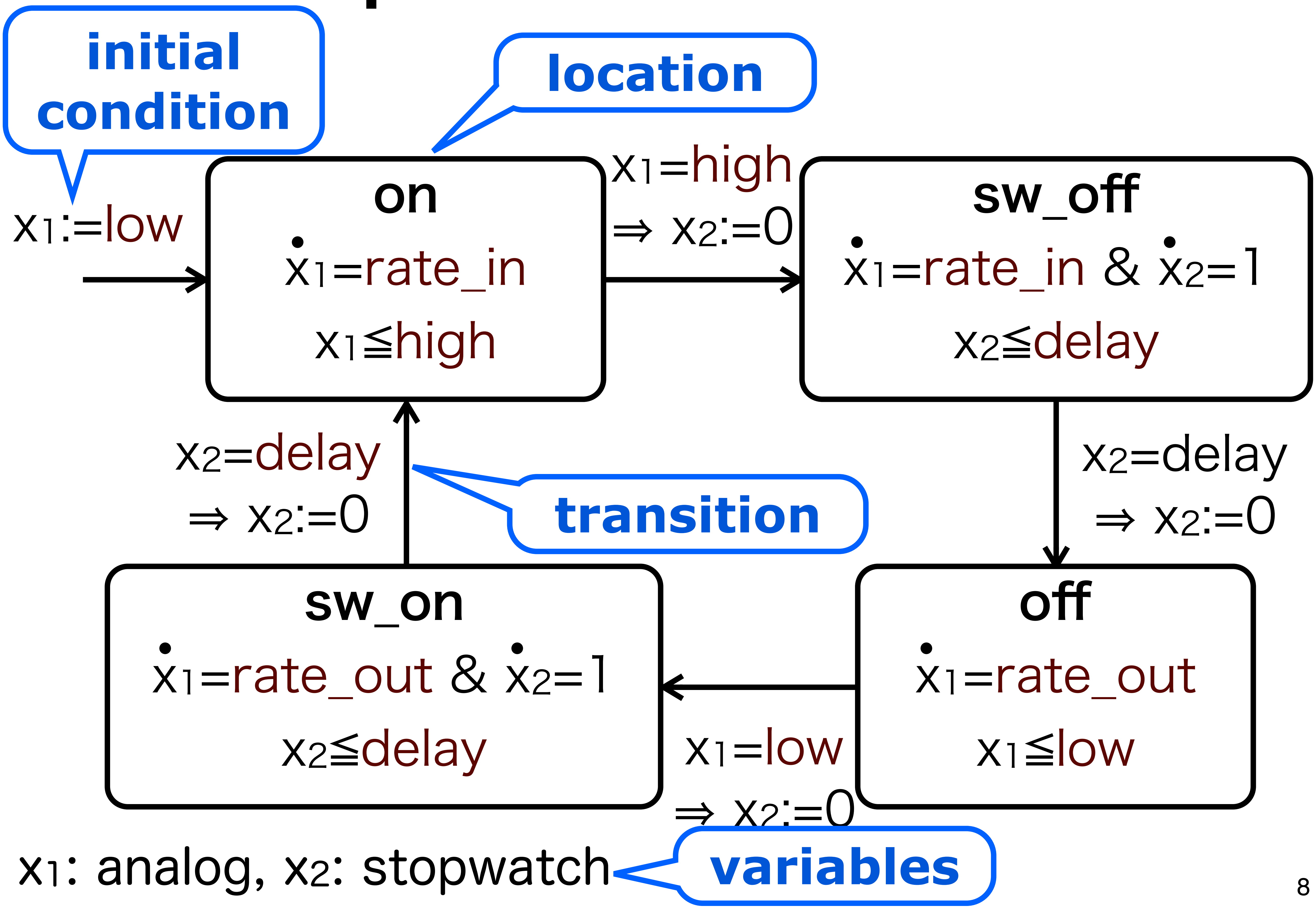
- A **hybrid automaton** is a septet $HA = \langle Loc, Var, Init, Grd, Rst, Flow, Inv \rangle$ that consists of:
  - Finite set $Loc = \{L_1, \ldots, L_p\}$ of **locations**
  - Finite set $Var = \{x_1, \ldots, x_q\}$ of real-valued **variables**
  - **Initial condition** $Init$ in $L \times \mathbf{R}^{Var}$
  - Family $\mathbf{Grd} = \{Grd_{L,L'}\}_{L \in Loc, L' \in Loc}$ of **guard conditions** $Grd_{L,L'}$ in $\mathbf{R}^{Var}$
  - Family $\mathbf{Rst} = \{Rst_{L,L'}\}_{L \in Loc, L' \in Loc}$ of **reset functions** $Rst_{L,L'} : \mathbf{R}^{Var} \to \mathbf{R}^{Var}$
  - Family $\mathbf{Flow} = \{Flow_L\}_{L \in Loc}$ of **vector fields** $Flow_L : \mathbf{R}^{Var} \to \mathbf{R}^{Var}$
  - Family $\mathbf{Inv} = \{Inv_L\}_{L \in Loc}$ of **location invariants** Inv_L in $\mathbf{R}^{Var}$

Example: Water-level Monitor

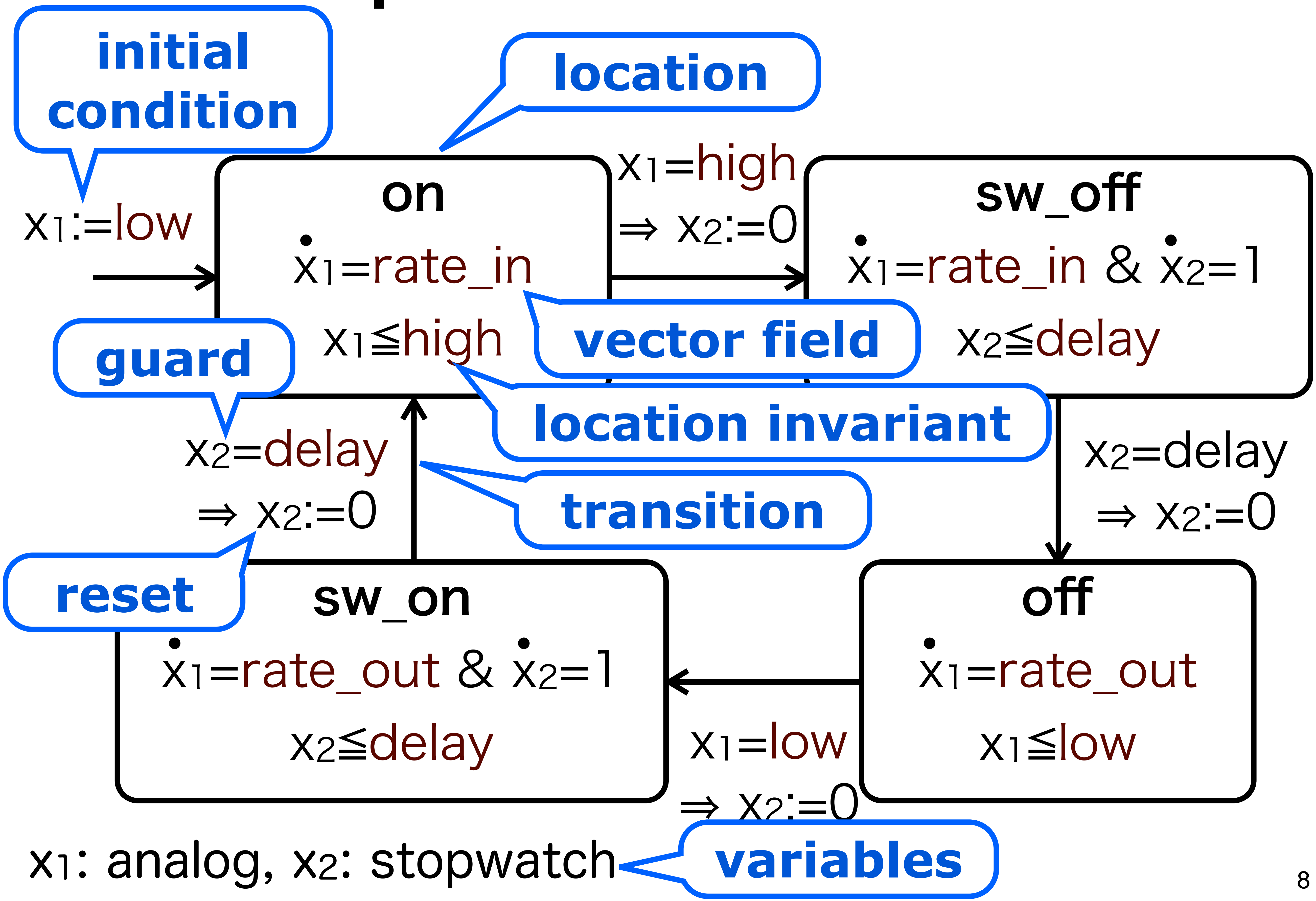


x_1: analog, x_2: stopwatch

Example: Water-level Monitor

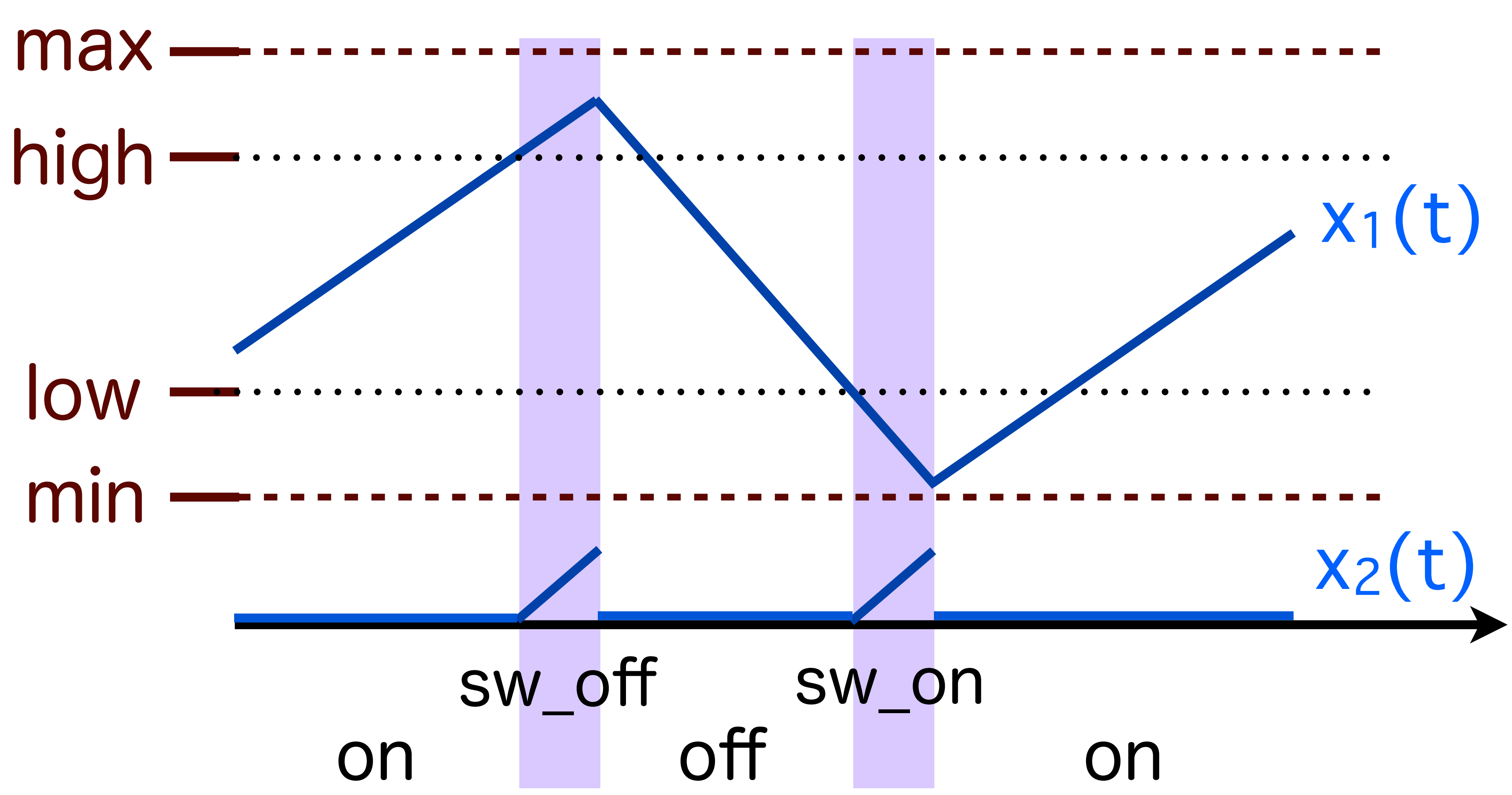


Example: Water-level Monitor



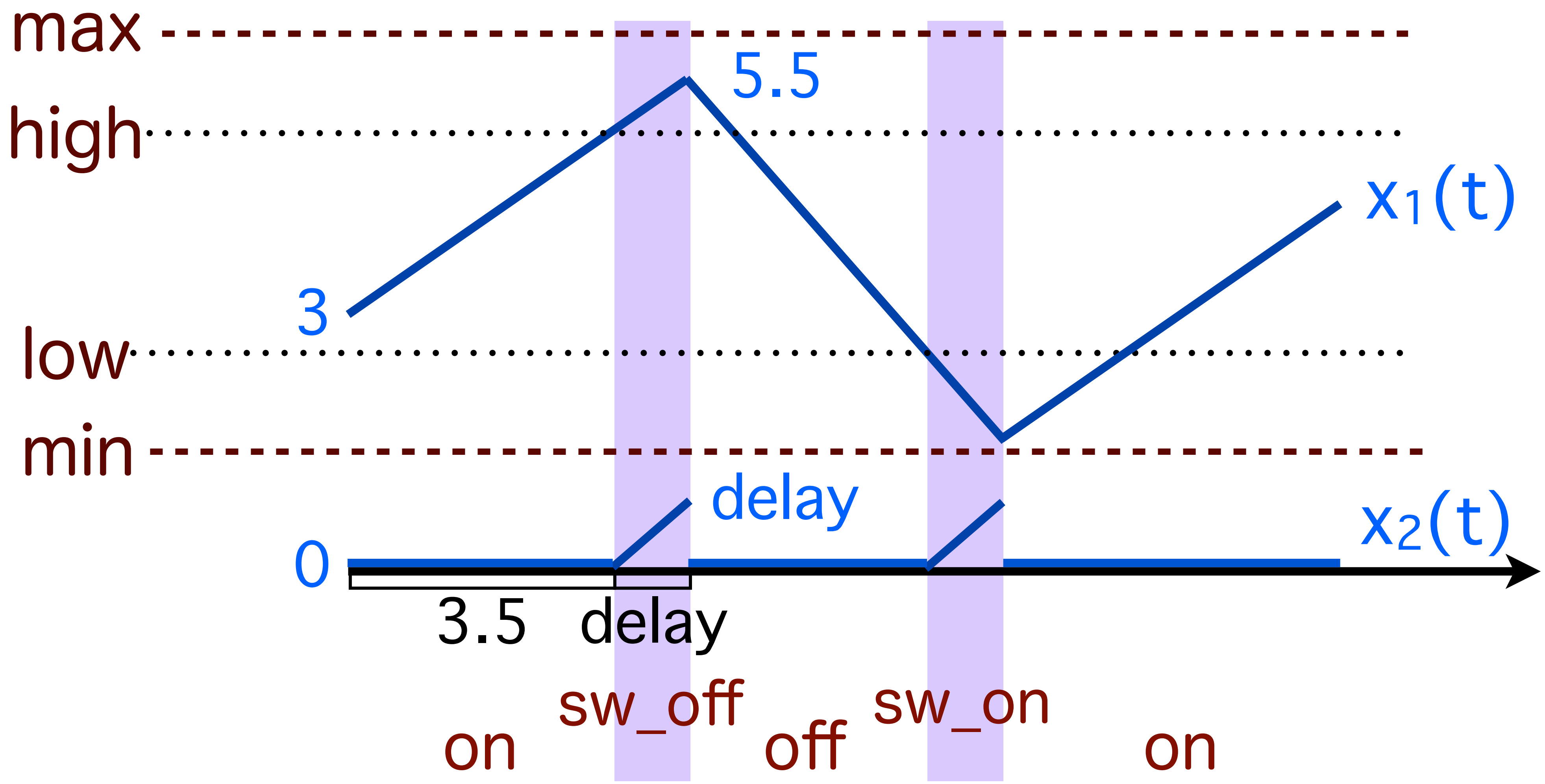
Execution of Water-level Monitor

- Two rates of the water flow: rate_in and rate_out
- The controller tries to turn on/off the pump when the water level reaches low/high
- It takes delay seconds to turn on/off the pump



Execution of HA

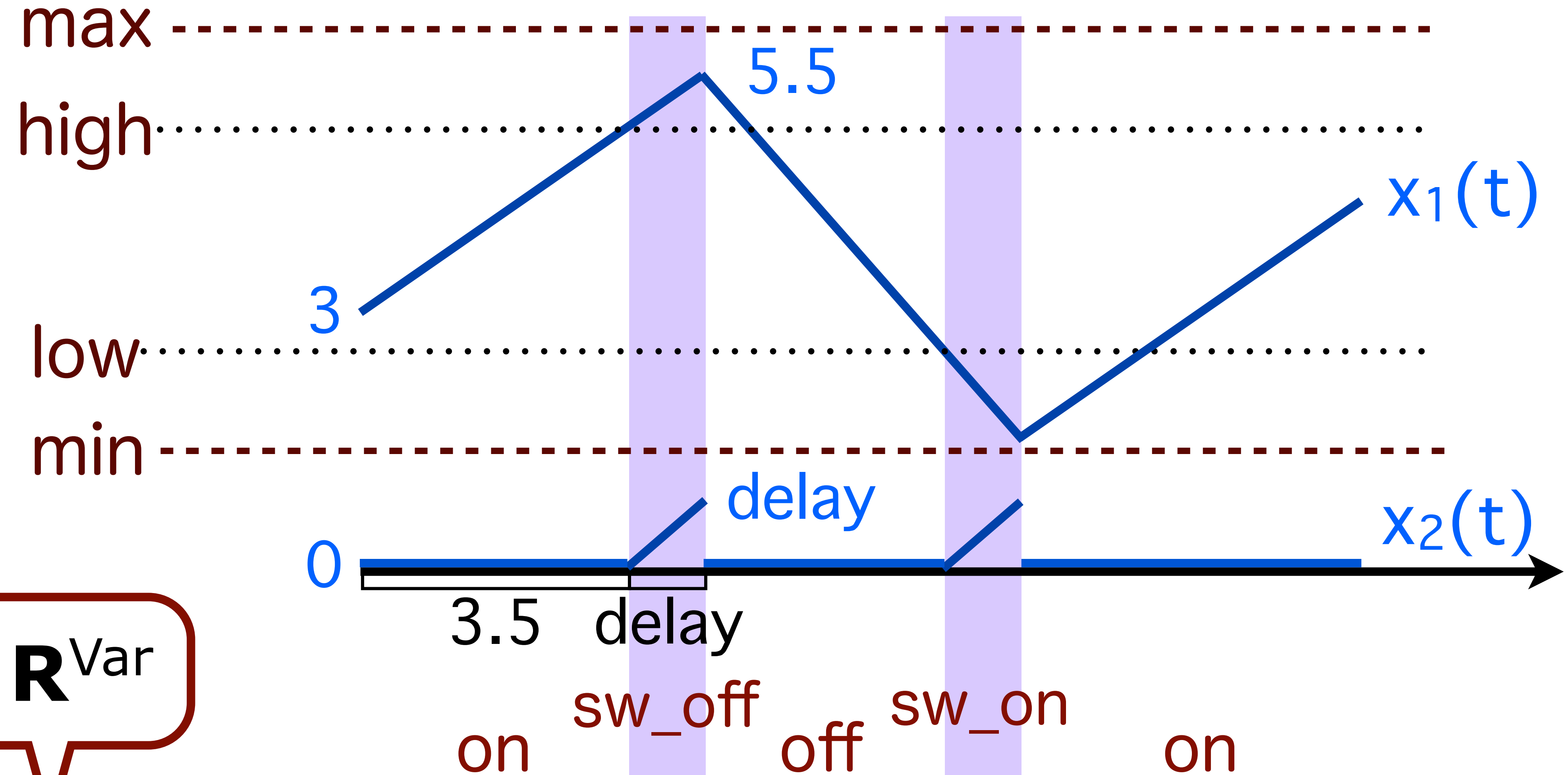
- (Finite- or infinite-length) **execution** of a HA is a valuation of variables as functions over time



$$\langle on,(3,0)\rangle \xrightarrow{3.5} \langle on,(high,0)\rangle \xrightarrow{0} \langle sw\_off,(high,0)\rangle \xrightarrow{delay}$$
$$\langle sw\_off,(5.5,delay)\rangle \xrightarrow{0} \cdots$$

Execution of HA

- (Finite- or infinite-length) **execution** of a HA is a valuation of variables as functions over time



$\text{Loc} \times \mathbf{R}^{\text{Var}}$

$\langle \text{on},(3,0)\rangle \xrightarrow{3.5} \langle \text{on},(\text{high},0)\rangle \xrightarrow{0} \langle \text{sw\_off},(\text{high},0)\rangle \xrightarrow{\text{delay}}$
$\langle \text{sw\_off},(5.5,\text{delay})\rangle \xrightarrow{0} \cdots$

continuous evolution for duration 3.5

discrete transition

Operational Semantics of HA

- Continuous evolution

$$\frac{t>0 \quad \phi(0)=v \quad \forall \tilde{t}\in[0,t]\ \frac{d\phi}{d\tilde{t}} = Flow_L(\phi(\tilde{t})) \wedge Inv_L[\phi(\tilde{t})]}{\langle L,v\rangle \xrightarrow{t} \langle L,\phi(t)\rangle}$$

- Discrete transition

$$\frac{Grd_{L1,L2}[v_1] \quad v_2 = Rst_{L1,L2}(v_1) \quad Inv_{L2}[v_2]}{\langle L_1,v_1\rangle \xrightarrow{0} \langle L_2,v_2\rangle}$$

Safety Verification of HA

- **Safety property** $\Box P$ means:
  - P holds initially, and is preserved by every continuous evolution and discrete transition
  - i.e. invariance
  - Example: $\Box(\min \leq x_1 \land x_1 \leq \max)$

Talk Outline

1. Hybrid Automata

2. Imp$_{HA}$ and
   Strongest Postcondition Calculus

3. Inductive Verification

4. Experimental Results

Modeling HA Executions with Programs

- **Given a HA, we consider a simple imperative language Imp_{HA}**
  - Program's states correspond to the states in the HA executions
  - Motivation: reuse of traditional program verification tools

Modeling HA Executions with Programs

- **Given a HA, we consider a simple imperative language Imp_{HA}**
  - Program's states correspond to the states in the HA executions
  - Motivation: reuse of traditional program verification tools
- **We provide a notion of strongest-postcondition (SP) for each Imp_{HA} statements**
  - Cf. forward reachability analysis

Imp_{HA} Language

- Imp_{HA} is a simple **imperative language** for sketching an execution of the HA
  - `s := skip | s; s | evolve t | trans`

- Consider **implicit variables** $x_S = \langle x_L, x_V \rangle = \langle x_L, (x_1,\ldots,x_q) \rangle$ that express the "current state" in $Loc \times \mathbf{R}^{Var}$ of the HA execution
- **Program state** S is a map from variable names to program values
  - Example: $S = \{x_L \mapsto on, x_1 \mapsto 3.1, x_2 \mapsto 0\}$

Imp_{HA} Language

- Imp_{HA} is a simple **imperative language** for sketching an execution of the HA
  - `s := skip | s; s | evolve t | trans`

continuous evolution for the duration t

discrete transition

- Consider **implicit variables** $x_s = \langle x_L, x_v \rangle = \langle x_L, (x_1,\ldots,x_q) \rangle$ that express the "current state" in $Loc \times \mathbf{R}^{Var}$ of the HA execution
- **Program state** S is a map from variable names to program values
  - Example: $S = \{x_L \mapsto on, x_1 \mapsto 3.1, x_2 \mapsto 0\}$

Imp_{HA} Language

- Imp_{HA} is a simple **imperative language** for sketching an execution of the HA
  - `s := skip | s; s | evolve t | trans`

  continuous evolution for the duration t (evolve t)

  discrete transition (trans)

- Consider **implicit variables**
  $x_S = \langle x_L, x_V \rangle = \langle x_L, (x_1,\ldots,x_q) \rangle$

  that express the "current state" in $Loc \times \mathbf{R}^{Var}$ of the HA execution

Imp_{HA} Language

- Imp_{HA} is a simple **imperative language** for sketching an execution of the HA
  - `s := skip | s; s | evolve t | trans`

continuous evolution for the duration t

discrete transition

- Consider **implicit variables**
  $x_s = \langle x_L, x_v \rangle = \langle x_L, (x_1,\ldots,x_q) \rangle$

  that express the "current state" in $Loc \times \mathbf{R}^{Var}$ of the HA execution
- **Program state** S is a map from variable names to program values
  - Example: $S = \{x_L \mapsto on,\ x_1 \mapsto 3.1,\ x_2 \mapsto 0\}$

Strongest Postcondition Calculus

- **Lemma 3.** For any program s in Imp_{HA}, if the initial state satisfies P, the final state satisfies SP(P,s) with SP defined as follows:

$$SP(P, \texttt{skip}) := P$$

$$SP(P, s_1 ; s_2) := SP(SP(P, s_1), s_2)$$

$$SP(P, \texttt{evolve}\ t) := \exists \phi\ P[x_v \leftarrow \phi(0)] \wedge \phi(t) = x_v \wedge (\forall \tilde{t} \in [0,t]\ \frac{d\phi}{d\tilde{t}} = Flow_{xL}(\phi(\tilde{t})) \wedge Inv_{xL}[\phi(\tilde{t})])$$

$$SP(P, \texttt{trans}) := \exists \langle L', x_v' \rangle P[x_s \leftarrow \langle L', x_v' \rangle] \wedge Grd_{L',xL}[x_v'] \wedge x_v = Rst_{L',xL}(x_v') \wedge Inv_{xL}[x_v]$$

Strongest Postcondition Calculus

- **Lemma 3.** For any program s in Imp_{HA}, if the initial state satisfies P, the final state satisfies SP(P,s) with SP defined as follows:

$$SP(P, \texttt{skip}) := P$$

$$SP(P, s_1;s_2) := SP(SP(P, s_1), s_2)$$

$$SP(P, \texttt{evolve } t) := \exists\phi\ P[x_v \leftarrow \phi(0)] \wedge \phi(t)=x_v \wedge (\forall\tilde{t}\in[0,t]\ \frac{d\phi}{d\tilde{t}}=Flow_{xL}(\phi(\tilde{t})) \wedge Inv_{xL}[\phi(\tilde{t})])$$

$$SP(P, \texttt{trans}) := \exists\langle L', x_v'\rangle P[x_s \leftarrow \langle L', x_v'\rangle] \wedge Grd_{L',xL}[x_v'] \wedge x_v=Rst_{L',xL}(x_v') \wedge Inv_{xL}[x_v]$$

derived from the operational semantics of HA

Example: Water-level Monitor

1. Consider

$$SP((x_L=on \wedge x_1=low), \texttt{evolve}\ t) \Rightarrow x_1 \leq max$$

Example: Water-level Monitor

1. Consider

$$SP((x_L{=}on \wedge x_1{=}low), \texttt{evolve}\ t) \Rightarrow x_1 \leq max$$

2. From the definition of SP

$$(\exists\phi\ (x_L{=}on \wedge \phi_{x1}(0){=}low) \wedge$$
$$\phi(t){=}(x_1,x_2) \wedge (\forall\tilde{t}\in[0,t]\ \frac{d\phi}{d\tilde{t}}{=}Flow_{xL}(\phi(\tilde{t})) \wedge Inv_{xL}[\phi(\tilde{t})]))$$
$$\Rightarrow x_1 \leq max$$

Example: Water-level Monitor

1. Consider

$$SP((x_L{=}on \wedge x_1{=}low), \texttt{evolve}\ t) \Rightarrow x_1 \leq max$$

2. From the definition of SP

$$(\exists\phi\ (x_L{=}on \wedge \phi_{x1}(0){=}low) \wedge$$

$$\phi(t){=}(x_1,x_2) \wedge (\forall\tilde{t}\in[0,t]\ \frac{d\phi}{d\tilde{t}}{=}Flow_{xL}(\phi(\tilde{t})) \wedge Inv_{xL}[\phi(\tilde{t})]))$$

$$\Rightarrow x_1 \leq max$$

3. We can solve the ODE into a closed form and simplify as

$$\forall\tilde{t}\in[0,t]\ Inv_{xL}[(low + rate_{in}\,\tilde{t},\ x_2)]$$

$$\Rightarrow low + rate_{in}\,t \leq max$$

Talk Outline

1. Hybrid Automata

2. Imp$_{HA}$ and
   Strongest Postcondition Calculus

3. Inductive Verification

4. Experimental Results

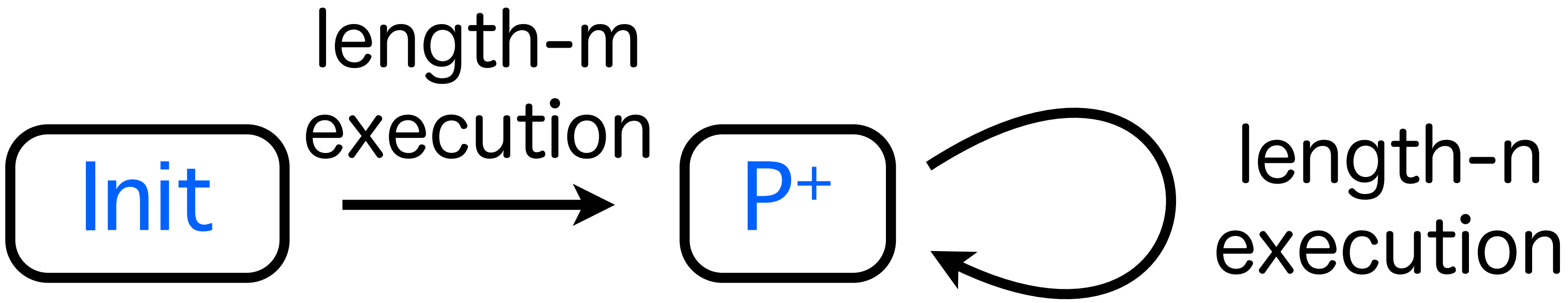
Inductive Verification

- Encoding **safety obligation for all the** (infinitely-many) **executions** of the HA into a **bounded number of verification conditions**

Inductive Verification

- Encoding **safety obligation for all the** (infinitely-many) **executions** of the HA into a **bounded number of verification conditions**

- **Induction strategy**
  - Inspired by deductive program verification technique for handling loops
  - **Construction of a *lasso*-shaped structure**
    - Assume a loop invariant P^+



Induction Strategy (Simplest Case)

- **Theorem 1.** Given a predicate P^+ such that $P^+ \Rightarrow P$, the following inference rule is correct

$$\frac{\begin{array}{ll} VC_0: & \text{Init} \Rightarrow P^+ \\ VC_1: & \forall t \geq 0\ SP(P^+, \texttt{evolve t}) \Rightarrow P \\ VC_{-1}: & \forall t \geq 0\ SP(P^+, \texttt{evolve t; trans}) \Rightarrow P^+ \end{array}}{HA \models \Box P}$$

Induction Strategy (Simplest Case)

- **Theorem 1.** Given a predicate P^+ such that $P^+ \Rightarrow P$, the following inference rule is correct

$$\frac{\begin{array}{ll} VC_0: & Init \Rightarrow P^+ \\ VC_1: & \forall t \geq 0 \; SP(P^+, \texttt{evolve } t) \Rightarrow P \\ VC_{-1}: & \forall t \geq 0 \; SP(P^+, \texttt{evolve } t; \texttt{trans}) \Rightarrow P^+ \end{array}}{HA \models \Box P}$$

- **Proof sketch**
  - VC_0 checks that all the initial states satisfy P^+
  - VC_{-1} verifies that all the two step executions

$$\sigma_i \xrightarrow{t_{i+1}} \sigma_{i+1} \xrightarrow{0} \sigma_{i+2}$$

    from a state σ_i that satisfies P^+ evolve for a duration t_{i+1} to a state σ_{i+2} that again satisfies P^+
  - VC_1 ensures that P was not broken during the continuous evolution

Induction Strategy (Simplest Case)

- **Theorem 1.** Given a predicate P^+ such that $P^+ \Rightarrow P$, the following inference rule is correct

$$
\frac{
\begin{array}{ll}
VC_0: & Init \Rightarrow P^+ \\
VC_1: & \forall t \geq 0\ SP(P^+, \texttt{evolve}\ t) \Rightarrow P \\
VC_{-1}: & \forall t \geq 0\ SP(P^+, \texttt{evolve}\ t; \texttt{trans}) \Rightarrow P^+
\end{array}
}{HA \models \Box P}
$$

- **Proof sketch**
  - VC_0 checks that all the initial states satisfy P^+
  - VC_{-1} verifies that all the two step executions

$$
(P^+)\ \sigma_i \xrightarrow{t_{i+1}} \sigma_{i+1} \xrightarrow{0} \sigma_{i+2}\ (P^+)
$$

    from a state σ_i that satisfies P^+ evolve for a duration t_{i+1} to a state σ_{i+2} that again satisfies P^+
  - VC_1 ensures that P was not broken during the continuous evolution

Example: Water-level Monitor

- Verification is possible with the inference rule where the loop invariant is set as

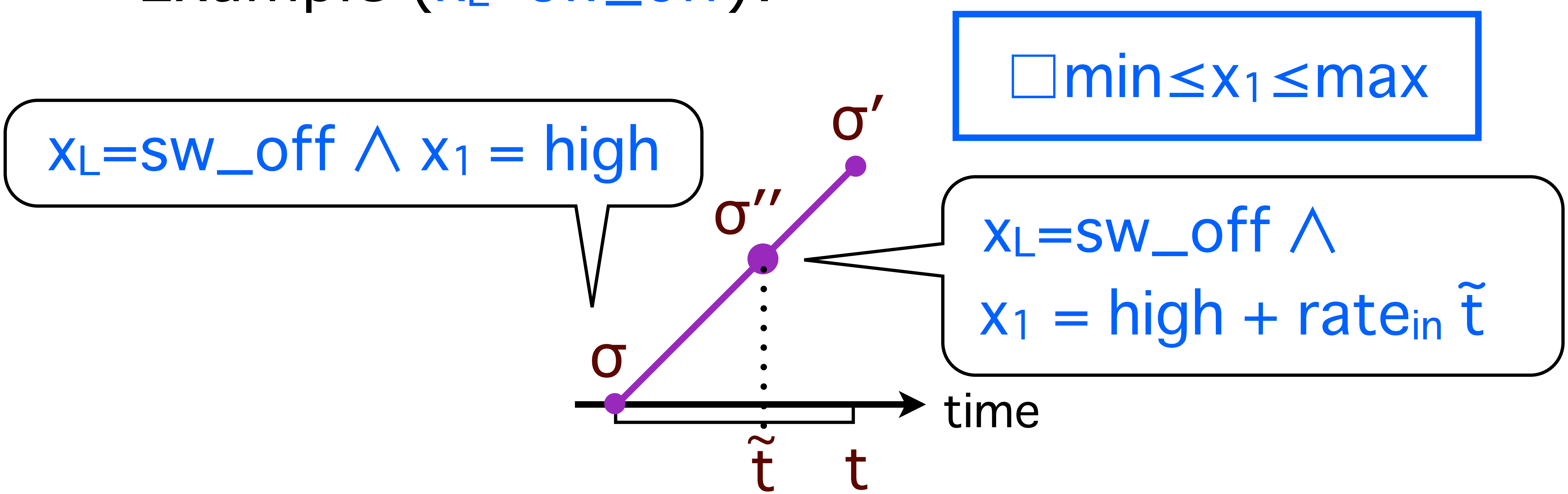
  $P^{+} \equiv x_L = sw\_off \Rightarrow x_1 = high \wedge x_L = sw\_on \Rightarrow x_1 = low$

Example: Water-level Monitor

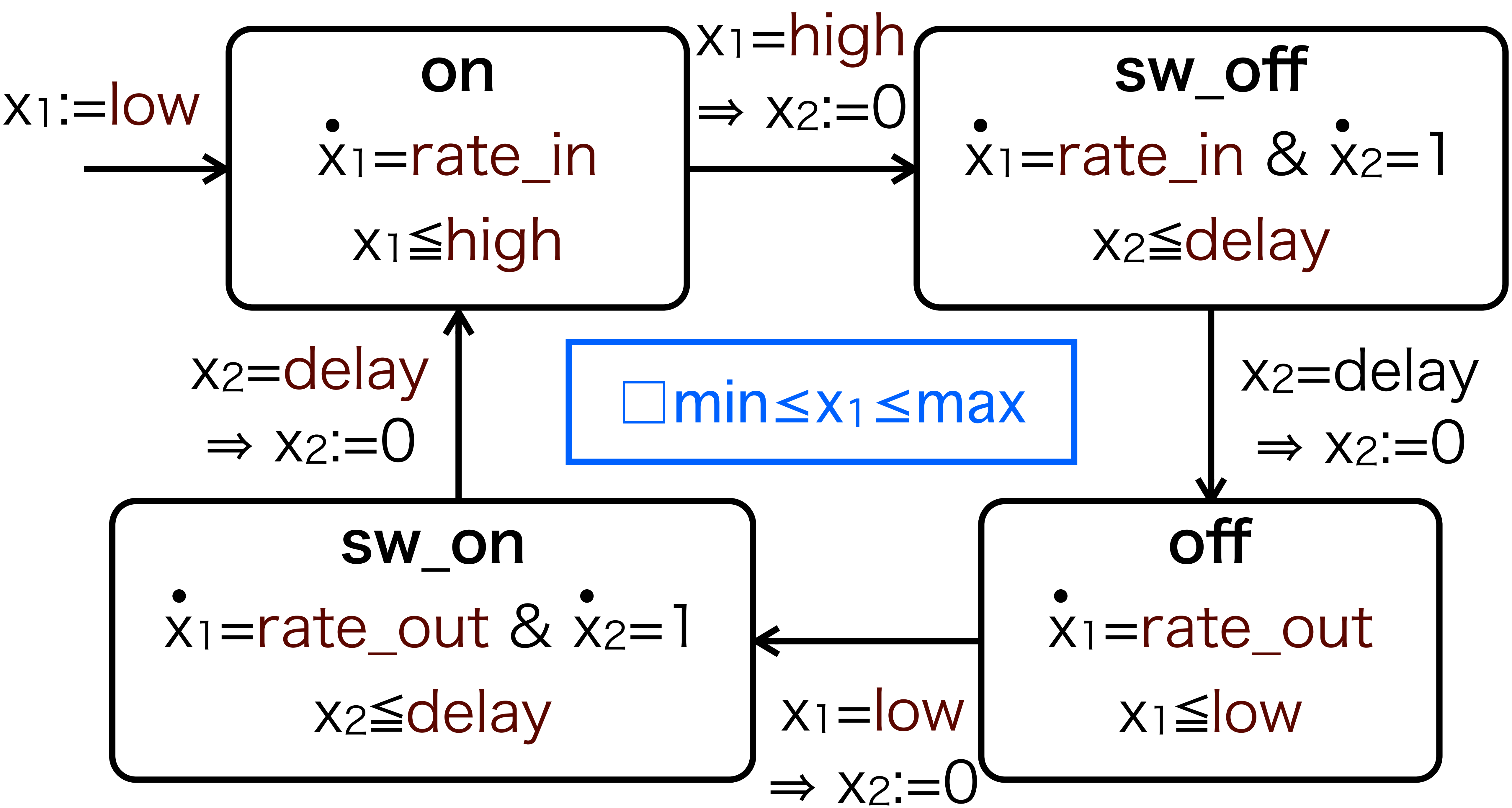
- Verification is possible with the inference rule where the loop invariant is set as

  $P^{+} \equiv x_L = sw\_off \Rightarrow x_1 = high \wedge x_L = sw\_on \Rightarrow x_1 = low$

- This loop invariant is necessary to verify the safety during the continuous evolution
  - Example ($x_L = sw\_off$):



Example: Water-level Monitor



Induction Strategy (Unrolled Case)

- **Theorem 2.** Given a predicate P^+ such that $P^+ \Rightarrow P$, consider the following m+n+2 SPs...

$$SP_1 \equiv SP(\text{Init}, \texttt{evolve}\ t_1)$$

$$SP_2 \equiv SP(SP(SP_1, \texttt{trans}), \texttt{evolve}\ t_2)$$

$$\vdots$$

$$SP_m \equiv SP(SP(SP_{m-1}, \texttt{trans}), \texttt{evolve}\ t_m)$$

$$SP_0 \equiv SP(SP_m, \texttt{trans})$$

$$SP_{m+1} \equiv SP(P^+, \texttt{trans}), \texttt{evolve}\ t_1)$$

$$SP_{m+2} \equiv SP(SP(SP_{m+1}, \texttt{trans}), \texttt{evolve}\ t_2)$$

$$\vdots$$

$$SP_{m+n} \equiv SP(SP(SP_{m+n-1}, \texttt{trans}), \texttt{evolve}\ t_n)$$

$$SP_{-1} \equiv SP(SP_{m+n}, \texttt{trans})$$

Induction Strategy (Unrolled Case)

- **Theorem 2 (cont).** Then, the following inference rule is correct

$$\begin{array}{ll}
VC_1: & \forall t_1 \geq 0\ SP_1 \Rightarrow P \\
VC_2: & \forall t_1, t_2 \geq 0\ SP_2 \Rightarrow P \\
 & \vdots \\
VC_m: & \forall t_1..t_m \geq 0\ SP_m \Rightarrow P \\
VC_0: & \forall t_1..t_m \geq 0\ SP_0 \Rightarrow P^+ \\
\hdashline
VC_{m+1}: & \forall t_1 \geq 0\ SP_{m+1} \Rightarrow P \\
 & \vdots \\
VC_{m+n}: & \forall t_1..t_n \geq 0\ SP_{m+n} \Rightarrow P \\
VC_{-1}: & \forall t_1..t_n \geq 0\ SP_{-1} \Rightarrow P^+ \\
\hline
\end{array}$$

$$HA \models \Box P$$

Induction Strategy (Unrolled Case)

- **Proof sketch**
  - VC_0 checks that all the initial states σ_0 reach a state σ_{2m} that satisfy P^+ after m steps

$$\sigma_0 \xrightarrow{t_1} \cdots \xrightarrow{0} \sigma_{2m}$$

  - VC_{-1} verifies that all the n step executions

$$\sigma_{2m} \xrightarrow{t_1} \cdots \xrightarrow{0} \sigma_{2m+2n}$$

    from a state σ_{2m} that satisfies P^+ reaches a state σ_{2m+2n} that again satisfies P^+
  - Other VCs i.e. $VC_1,\ldots,VC_{m+n}$ ensure that P was not broken before/after a discrete transition and during a continuous evolution

Induction Strategy (Unrolled Case)

- **Proof sketch**
  - VC_0 checks that all the initial states σ_0 reach a state σ_{2m} that satisfy P^+ after m steps

    $$\text{Init} \quad \sigma_0 \xrightarrow{t_1} \cdots \xrightarrow{0} \sigma_{2m} \quad P^+$$

  - VC_{-1} verifies that all the n step executions

    $$\sigma_{2m} \xrightarrow{t_1} \cdots \xrightarrow{0} \sigma_{2m+2n}$$

    from a state σ_{2m} that satisfies P^+ reaches a state σ_{2m+2n} that again satisfies P^+
  - Other VCs i.e. $VC_1,\ldots,VC_{m+n}$ ensure that P was not broken before/after a discrete transition and during a continuous evolution

Induction Strategy (Unrolled Case)

- **Proof sketch**
  - VC_0 checks that all the initial states σ_0 reach a state σ_{2m} that satisfy P^+ after m steps

$$\text{Init} \rhd \; \sigma_0 \xrightarrow{t_1} \cdots \xrightarrow{0} \sigma_{2m} \; \lhd P^+$$

  - VC_{-1} verifies that all the n step executions

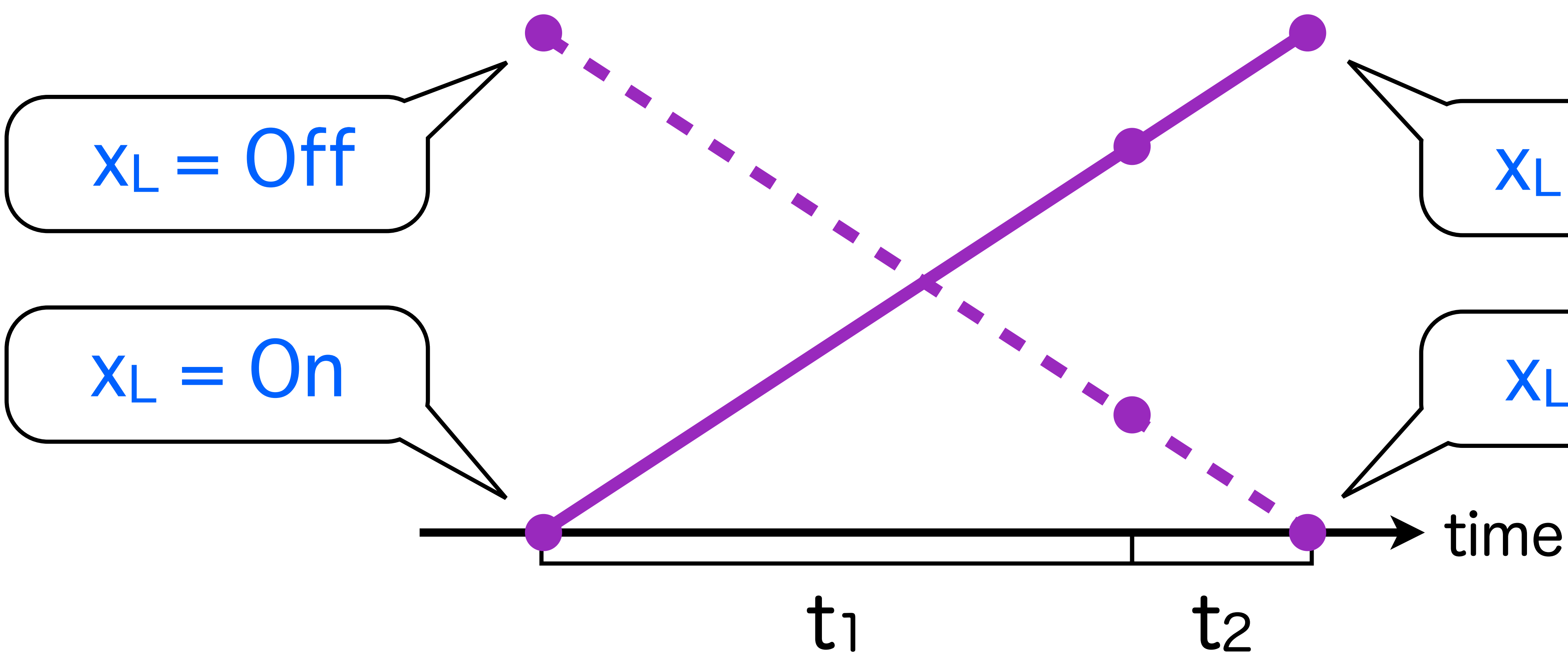
$$P^+ \rhd \; \sigma_{2m} \xrightarrow{t_1} \cdots \xrightarrow{0} \sigma_{2m+2n} \; \lhd P^+$$

    from a state σ_{2m} that satisfies P^+ reaches a state σ_{2m+2n} that again satisfies P^+
  - Other VCs i.e. $VC_1,\ldots,VC_{m+n}$ ensure that P was not broken before/after a discrete transition and during a continuous evolution

Example: Water-level Monitor

- Verification is also possible with the inference rule for $m=0$ and $n=2$ where the loop invariant is set as

$$P^{+} \equiv x_L = on \vee x_L = off$$



Algorithm for Inductive Verification

- Following algorithm generates VCs for $(m,n) \in [0, m_{max}] \times [1, n_{max}]$ and discharges the VCs

Input: HA; P; $m_{max} \in \mathbb{N}_{\geq 0}$; $n_{max} \in \mathbb{N}_{>0}$

Output: $true$: $HA \models \Box P$; $false$: cannot decide $\Box P$ within $m_{max} + n_{max}$ steps

for $m \in \{0, \ldots, m_{max}\}$; $n \in \{1, \cdots, n_{max}\}$ **do**
    $P^+ := P$
    while $P^+ \not\equiv false$ **do**
        if $\neg \forall i \in \{0, \ldots, m\}$ $\mathsf{Validate}(VC_i)$ **then**
            break
        end if
        if $\exists j \in \{m+1, \ldots, m+n, -1\}$ $\neg\mathsf{Validate}(VC_j)$ **then**
            $P^+ := P^+ \wedge \mathsf{Learn}(VC_j)$
        else
            return $true$
        end if
    end while
end for
return $false$

Algorithm for Inductive Verification

- Following algorithm generates VCs for $(m,n) \in [0, m_{max}] \times [1, n_{max}]$ and discharges the VCs

Input: HA; P; $m_{max} \in \mathbb{N}_{\geq 0}$; $n_{max} \in \mathbb{N}_{>0}$

Output: $true$: $HA \models \Box P$; $false$: cannot decide $\Box P$ within $m_{max} + n_{max}$ steps

for $m \in \{0, \ldots, m_{max}\}$; $n \in \{1, \cdots, n_{max}\}$ **do**
    $P^+ := P$
    while $P^+ \not\equiv false$ **do**
        if $\neg \forall i \in \{0, \ldots, m\}$ Validate(VC_i) **then**
            break
        end if
        if $\exists j \in \{m+1, \ldots, m+n, -1\}$ $\neg$Validate(VC_j) **then**
            $P^+ := P^+ \wedge$ Learn(VC_j)
        else
            return $true$
        end if
    end while
end for
return $false$

initialize P^+

loop invariant generation

Loop Invariant Generation

- When verification of
  $VC_i : \forall t_1..t_i \geq 0 \, (SP(P^+ \wedge s) \Rightarrow P)$
  fails, we can generate a loop invariant using a **quantifier elimination (QE)** method

$$QE(\forall x_s \, \forall t_1..t_i \, VC_i)[x_0 \leftarrow x_s]$$

Loop Invariant Generation

- When verification of
  $VC_i : \forall t_1..t_i \geq 0\ (SP(P^+ \wedge s) \Rightarrow P)$
  fails, we can generate a loop invariant using a **quantifier elimination (QE)** method

$$QE(\forall x_s\ \forall t_1..t_i\ VC_i)[x_0 \leftarrow x_s]$$

- This process is hard in general
  - VC_i should be simplified by assuming a class of problems
    - ✳Otherwise QE might not succeed
  - Generated invariant might be too large. They should be simplified manually
    - ✳Two strategies for the simplification

Implementation

- **We have implemented the verification algorithm using Mathematica** (i.e. algebraic formula manipulation system)
  - ODEs are solved into closed forms with **DSolve**
  - **Validate** is implemented using built-in functions, **FullSimplify**, **Reduce**, and **FindInstance**
  - **Learn** is implemented using a built-in function **Resolve** that performs QE

Implementation

- **We have implemented the verification algorithm using Mathematica** (i.e. algebraic formula manipulation system)
  - ODEs are solved into closed forms with **DSolve**
  - **Validate** is implemented using built-in functions, **FullSimplify**, **Reduce**, and **FindInstance**
  - **Learn** is implemented using a built-in function **Resolve** that performs QE

- **Several optimizations**
  - Separation of formulas wrt HA locations
  - Reutilization of simplification process of sub-formulas

Talk Outline

1. Hybrid Automata

2. Imp_{HA} and
   Strongest Postcondition Calculus

3. Inductive Verification

4. Experimental Results

Experimental Results

Problem	#Loc #Var	m/n	Our method	MC tools	KeYmaera
WLM	4/2	0/1	0.85s	N/A	1.8s
Gas burner	2/3	4/2	2.22s	0.004s	N/A
Temp. ctrl	4/3	1/1	2.82s	0.012s	N/A
Bouncing ball	1/2	0/1	0.49s	N/A	0.9s
ETCS	2/3	0/1	4.48s	N/A	3.1s
Highway 9	10/9	0/2	0.22s	0.22s	N/A
Highway 19	20/19	0/2	3.64s	N/A	N/A

Comparison with Other Tools

- **MC tools**
  - **HyTech** [Henzinger+ 96] and **PHAVer** [Frahse 02]
  - Solve three problems quite efficiently (Ex.2,3,6)
  - Cannot handle instances with uncertain parameters (Ex.1,4)
  - Some scaling issues (Ex.6,7)
- **KeYmaera** [Platzer+ 08]
  - Handles various *hybrid programs* automatically
  - However, does not succeed on most of programs translated from HA (Ex.2,3,6,7)
    - ✳Models should be annotated manually
    - ✳Otherwise, users need to interact with underlying theorem prover

Conclusion

- **Automated logical analytic method for a large class of linear and nonlinear HA**
  - Algorithmic verification with SP calculus and limited derivation rules,
    i.e., induction and loop unrolling
  - Loop invariant generation guided by the response from the decision process
  - Promising experimental results with several HA
- **Future work**
  - Automation of generation process of efficient loop invariants
  - Support for larger class of HA,
    e.g., with unsolvable ODEs, parallel composition