

Formal Techniques for Analyzing Hybrid Systems

Ashish Tiwari

Representing Dynamical Systems

How to solve the model checking problem $S \models \phi$?

Crucially depends on how we represent S

Recall that $S = (\mathbf{X}, \mathbf{F}, \text{Init})$

We need to represent the state space, the initial set Init , and the set of trajectories

Representing Systems: State Space

State space X

- State space can be defined as the set of all valuations of a set of variables
- We just need to declare variables X and their types

The initial states Init

- This is just a subset of the state space
- Can be represented by a formula over X

Representing Systems: Trajectories

The set of all possible trajectories is usually represented by **local rules**

That specify the **next** state(s) given the **current state** (and **input**, if any)

The notion of **next** is dependent on the time domain \mathbf{T}

When $\mathbf{T} = \mathbb{N}$, **next** is just (current time + 1)

From continuous time to discrete-time

When $\mathbf{T} = \mathbb{R}^{\geq 0}$, there is a **popular way** to map it to $\mathbf{T} = \mathbb{N}$
 $s \rightarrow s'$ if there exists a $t' > t$ s.t. $s' = F(t')$ and $s = F(t)$

From my current state s , the system can **nondeterministically** transition any state reachable in **any future time** instance

For example, if $\frac{dx}{dt} = 1$ is a clock, its discrete time semantics is:
 $x \rightarrow x'$ if $x' > x$

This is **time oblivious**

If we want to keep track of time elapsed, we can add a new state variable t and have

$(x, t) \rightarrow (x', t')$ if $x' > x$ and $x' - x = t' - t$

Representing Systems: Trajectories

Let us focus on time domain \mathbb{N} , and consider ways to specify a single step $s \rightarrow s'$ state transition

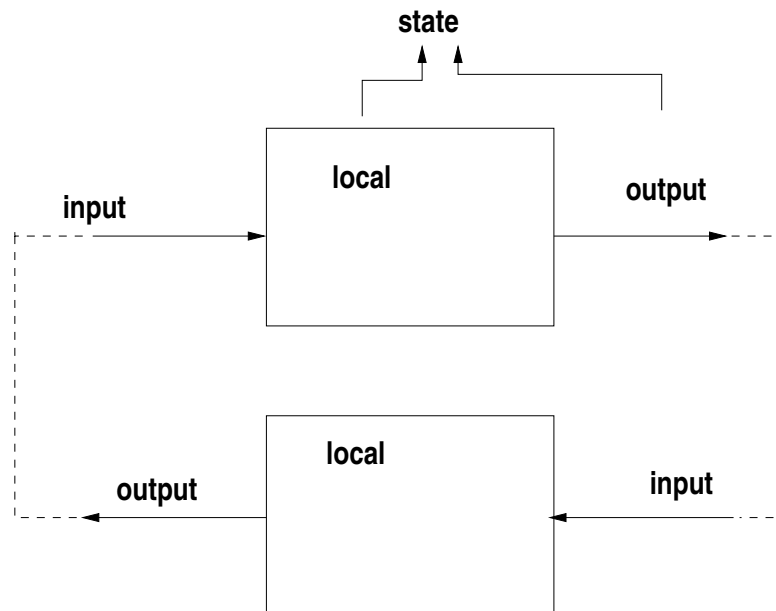
This can be given as a formula over a pair of state variables: $\phi(x, x')$

For open systems, this formula can depend on the input variables

For continuous time systems, the rule is a formula over (x, \dot{x})

Representing Systems: Modularly

We can represent dynamical systems succinctly by describing them as a composition of two or more smaller dynamical systems:



We need to think about **open** systems first

Open Dynamical Systems

Systems with inputs

Inputs do not change the state space of the system
state space still defined by the local and output variables

Inputs can influence the possible trajectories

Hence, rule $\phi(s, i, s')$ can depend on the input variables i

$s \rightarrow s'$ if there exists an input $\mathbf{i} \in \mathbf{I} : \phi(s, \mathbf{i}, s')$

Representing Systems: Modularly

State space of the composition = cross product of the two state spaces
= valuations of $L_1 \cup O_1 \cup L_2 \cup O_2$

Trajectories of the composition = ??
depends on the **compositional framework**

Two Compositional Frameworks

Synchronous:

- $(s_1, s_2) \rightarrow (s'_1, s'_2)$ if $s_1 \rightarrow s'_1$ **AND** $s_2 \rightarrow s'_2$
- Both subsystems make a transition simultaneously

Asynchronous:

- $(s_1, s_2) \rightarrow (s'_1, s'_2)$ if $s_1 \rightarrow s'_1, s'_2 = s_2$ **OR** $s_2 \rightarrow s'_2, s'_1 = s_1$
- One of the two subsystems makes a transition

If the two subsystems are wired to each other, the transitions on the subsystems should be consistent with the inputs (generated by the other subsystem)

Modeling in SAL

Discrete-time hybrid-space dynamical systems can be modeled in SAL

Every SAL **MODULE** describes a dynamical (sub)system

```
collatz: MODULE =  
BEGIN  
  OUTPUT x: NATURAL  
  INITIALIZATION x IN {v: NATURAL | True};  
  TRANSITION  
  [  
    x%2==0 --> x' = x/2;  
  ]  
  else --> x' = 3*x+1;  
]  
END;
```

As Two Subsystems

```
collatzEven: MODULE = BEGIN
  INPUT x: NATURAL
  OUTPUT y: NATURAL
  TRANSITION
  [ x%2==0 --> y' = x/2; [] y%2==0 --> y' = y/2; ]
END;
```

```
collatzOdd: MODULE = BEGIN
  INPUT y: NATURAL
  OUTPUT x: NATURAL
  TRANSITION
  [ y%2==1 --> x' = 3*y+1; ]
END;
```

... Composed Together

```
collatz: MODULE = collatzEven [] collatzOdd ;
```

Input and Output match up **by name** here; could apply outer **rename** operator to a module o.w.

All modules can be put within a context (file) `collatz.sal`:

```
collatz: CONTEXT = BEGIN
  collatzEven: MODULE = ...
  collatzOdd: MODULE = ...
  collatz: MODULE = ...
END
```

Temporal Properties in SAL

The open problem:

```
am_I_true: THEOREM
  collatz |- F( x = 1 );
```

Modeling Exercise

Discrete-time discrete-space dynamical system modeling a vending machine

- User can start with dollar amount in the range $[0, \dots, 5]$
- User can buy a cake for \$1
- User can buy an apple for \$0.75

The state space is given by the

- (1) amount of money remaining,
- (2) number of cakes bought, and
- (3) number of apples bought

SAL Example: Modeling

```
% Vending machine in SAL
vm: CONTEXT =
BEGIN

DollarAmount: TYPE = [0..5];
QuarterAmount: TYPE = [0..20];
CakeAmount: TYPE = [0..5];
AppleAmount: TYPE = [0..10];
```


SAL Example: Modeling

```
machine: MODULE =  
BEGIN  
  OUTPUT  
  d: DollarAmount, q: QuarterAmount, c: CakeAmount, a: AppleAmount  
  INITIALIZATION  
  d IN {v: DollarAmount | v <= 4};  
  q = 0; c = 0; a = 0;
```

SAL Example: Modeling Contd

```
TRANSITION
[
  get_c: d >= 1 --> d' = d-1; c' = c+1
[]
  get_a: d >= 1 --> d' = d-1; a' = a+1; q' = q+1
[]
  change: q >= 4 --> d' = d+1; q' = q-4
[]
  else -->
]
END;
```

SAL Example: Modeling Contd

```
prop: THEOREM
```

```
  machine |- NOT ( U(q <= 4, a >= 5) );
```

```
prop2: THEOREM
```

```
  machine |- G ( NOT ( a >= 3 AND c >= 2) );
```

```
END
```

Exercise: Is prop valid or false?

Exercise: Is prop2 valid or false?

SAL Example: Analysis

One can use the **sal symbolic model checker** to model check the two properties

```
sal-smc vm prop  
sal-smc vm prop2
```

The first gives a counter-example:

start with 4 dollar bills, change when you have 4 quarters

The second says "proved".

SAL Modeling: Other Aspects

- Modules can be composed:

```
Module = Module [] Module;  
Module = Module || Module;
```

- Modules can be parameterized: `Module(i: Index) = ...`

- And composed:

```
Module = ([] (i: Index) Module[i]) || Observer
```

- Modules can use 'helper' functions, datatypes as ARRAYS of ARRAYS of ...

SAL Analysis: Other Aspects

Sal Tools

- sal-smc: symbolic model checker
- sal-bmc: bounded model checker (converts to SAT)
- sal-inf-bmc: infinite bounded model checker (SMT)
- sal-path-finder: for simulating
- sal-bmc -i, sal-inf-bmc -i: prove by k-induction
- sal-emc: explicit state model checker
- sal-deadlock-checker: check for deadlocked states
- your script: sal-atg: Automated test generator