Simulation of Hybrid Systems

Bert van Beek

Systems Engineering Group
Eindhoven University of Technology

HYCON PhD school Siena 2007
17 June 2007

Outline

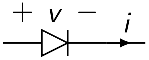
1. Dynamics and control ⇔ computer science world view
2. Simulation languages ⇔ verification formalisms
3. Chi language
4. Phenomena in hybrid simulation (languages)
5. Compositional Interchange Format (CIF)

Dynamics and control world view

- Predominantly continuous-time system
- Modeled by means of DAEs (differential algebraic equations), or by means of a set of trajectories
- Hybrid phenomena modeled by means of discontinuous functions and/or switched equations, possibly using extended solution concepts (Filippov, Utkin) leading to sliding modes
- Evolution of a hybrid system: value of each variable a (possibly *discontinuous*) function of time $\implies$ for each variable at each time point *one* value
- Examples: piecewise affine (PWA) systems, mixed logic dynamical (MLD) systems or linear complementarity (LC) systems

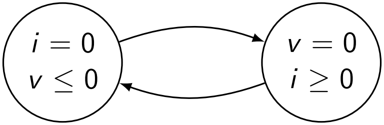
$$(i = 0 \wedge v \leq 0) \vee (v = 0 \wedge i \geq 0)$$

DAE model of a diode



Computer science world view

- Predominantly discrete-event system
- Modeled by means of (timed/hybrid) automaton, process algebra, Petri net, data flow languages, etc.
- Evolution of a hybrid system: sequence of time transitions and action transitions. Due to sequences of action transitions, a variable may have *multiple* values at the same time point
- Time transitions: for each variable *continuous* function of time
- Discontinuities are represented by actions

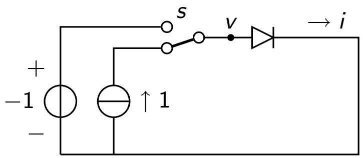


Automaton model of a diode

DAE model of diode-switch network

Switching between

- voltage source of -1 (diode blocks)
- current source of $+1$ (diode conducts)



Generalized differential algebraic equation models use predicates as "equations":

$$(\neg s \implies v = -1) \wedge (s \implies i = 1)$$

$$(i = 0 \wedge v \leq 0) \vee (v = 0 \wedge i \geq 0)$$

$$s = \begin{cases} \text{true} & \text{for } 0 \leq t < 2 \\ \text{false} & \text{for } t \geq 2 \end{cases}$$

Algebraic variables

Reduced DAE model of diode-switch network:

$$\begin{cases} s \wedge v = 0 \wedge i = 1 & \text{for } 0 \leq t < 2 \\ \neg s \wedge v = -1 \wedge i = 0 & \text{for } t \geq 2 \end{cases}$$

Values of v and i change discontinuously at time point 2.
Therefore they are *algebraic* variables.

Algebraic variables:

- Any function of time, possibly discontinuous
- No memory
- No derivative

Continuous variables

Continuous variables:

- Continuous function of time: $x(t) = x(0) + \int_0^t \dot{x}(s)ds$
- Memory
- Derivative may be used

Examples

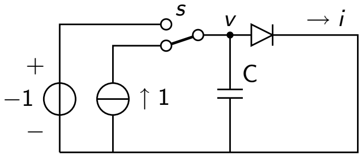
- Mass m, force F, velocity v:
  $F = m\dot{v}$, continuous variable v
- Tank with volume V, inflow Q_i, outflow Q_o:
  $\dot{V} = Q_i - Q_o$, continuous variable V
- Capacitor C, voltage v, current i:
  $i = C\dot{v}$, continuous variable v

DAE model of diode-switch-capacitor network

$$(\neg s \implies v = -1) \wedge (s \implies i = 1 - \mathrm{C}\dot{v})$$

$$(i = 0 \wedge v \leq 0) \vee (v = 0 \wedge i \geq 0)$$

$$s = \begin{cases} \mathrm{true} & \text{for } 0 \leq t < 2 \\ \mathrm{false} & \text{for } t \geq 2 \end{cases}$$



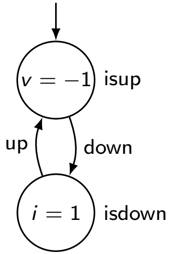
Equivalent specification:

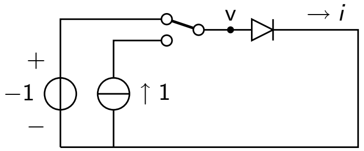
$$\begin{cases} s \wedge ((i = 0 \wedge \dot{v} = 1/\mathrm{C} \wedge v \leq 0) \vee (v = 0 \wedge i = 1)) & \text{for } 0 \leq t < 2 \\ \neg s \wedge v = -1 \wedge i = 0 & \text{for } t \geq 2 \end{cases}$$

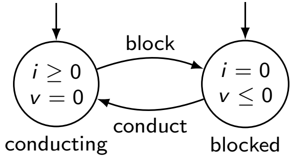
Problem: Now v must be a continuous variable, with derivative, and continuous behavior $\implies$ value of v cannot change discontinuously to -1 when switch opens!

Automaton model of diode-switch network (1)

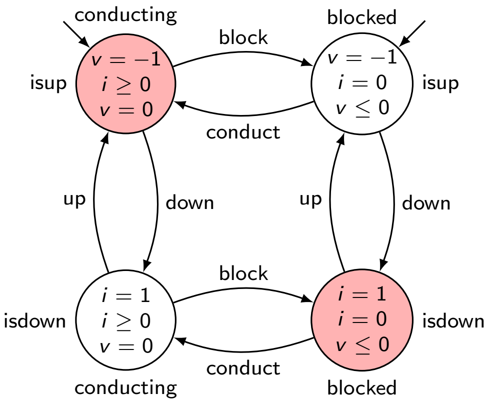
- All variables are *continuous*
- Mode switching: assume that value of a variable can change arbitrarily to satisfy invariant of next mode $\Longrightarrow$ continuous variables behave in a similar way as algebraic variables

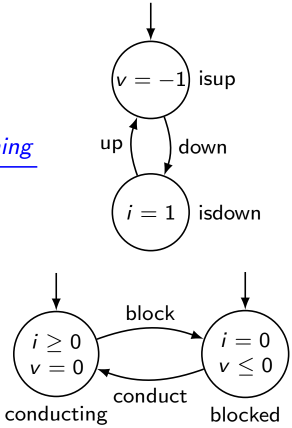




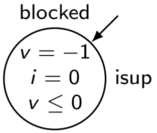


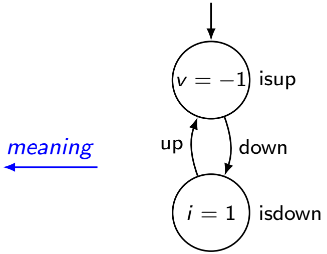
Interleaving parallel composition

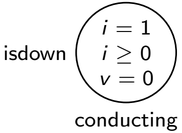


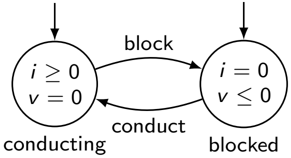


Simplified automaton



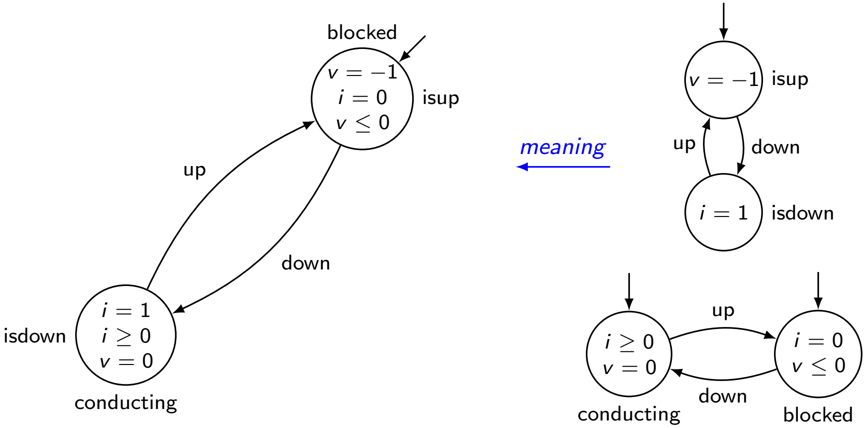






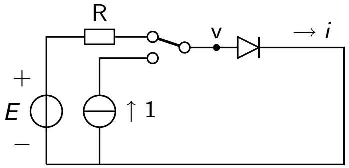
Synchronizing on common labels

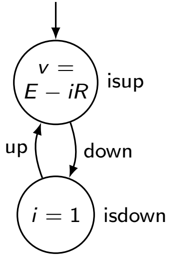
Change diode model labels such that they synchronize with switch model labels

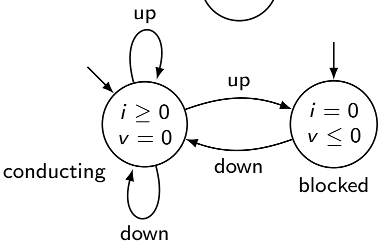


Automaton model of diode-switch network (2)

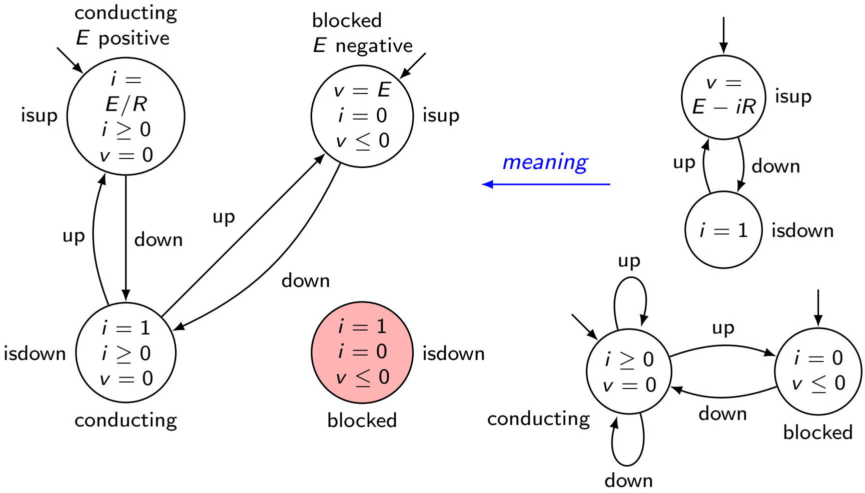
Voltage source E is positive or negative





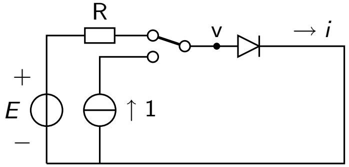


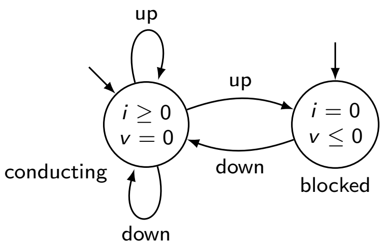
Synchronizing on common labels (2)



Conclusions automaton model of diode-switch network

- Automaton model of diode not a compositional way of modeling:
  Model of diode needs to know action names of automata models of switches.
- Automata mode switching only in case of actions.
- Incorrect behavior in case of time-dependent varying voltage or current source: no switching of mode of automaton!





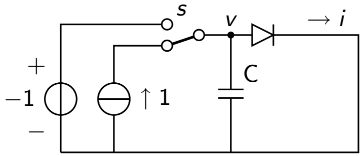
Combining the DC and CS world views (1)

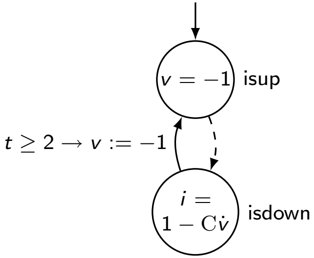
Combine the differential algebraic equations from the dynamics and control world view with automata

- Algebraic variables
- Continuous variables
- Hybrid phenomena may be modeled by means of discontinuous functions and/or switched equations, possibly using extended solution concepts (Filippov, Utkin) leading to sliding modes
- By means of actions, automata can change from one mode to another
- In each mode, variables (algebraic/dotted) can behave according to discontinuous functions of time

DC + CS model of diode-switch-capacitor network

Assume t denotes time





$(i = 0 \wedge v \leq 0) \vee (v = 0 \wedge i \geq 0)$

More general model: instead of assignment $v := -1$ on automaton edge, specify that the value of v may change arbitrarily.

E.g. $t \geq 2 \rightarrow \{v\} : \mathrm{true}$ (see last sheets on instantaneous equations).

Algebraic and continuous variables in simulation languages

- Distinction between algebraic and continuous variables usually implicit.
- No derivative $\implies$ algebraic
- Derivative $\implies$ continuous

Consider:

$$\begin{aligned} x < 0 &\implies \dot{x} = 1 \\ x \geq 0 &\implies x = 0 \end{aligned}$$

Is x continuous, or switching between continuous and algebraic? In many languages such models are not allowed. Use discontinuous right hand sides to approach the meaning:

$$\dot{x} = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{if } x \geq 0 \end{cases}$$

Simulation languages

- Ease of modeling $\Longrightarrow$ complex languages
- Verification not an issue, no formal semantics: (no verification)
- Languages specialize either in the discrete-event (DE) domain or in the continuous-time (CT) domain
- Hybrid languages usually DE$^+$ (E.g. Siman, Simple++) or CT$^+$ (E.g. Simulink, Modelica, EcosimPro)

Verification formalisms

- Ease of formal analysis $\Longrightarrow$ small languages with formal semantics
- Ease of modeling not an issue: cumbersome for modeling and simulation
- Examples for hybrid systems: PHAVer, HyTech; for timed systems: PROMELA, UPPAAL.

Overview of the Chi language (1)

- Suited to:
    - simulation
    - verification
    - code generation
- Integrates:
    - discrete-event modeling (CS world view: automata, process algebra)
    - continuous-time modeling, (DC world view: switched differential algebraic equations)
    - discrete-time modeling (DC world view: sampled systems)
- Formal compositional semantics
- Consistent equation semantics of Chi ensures that equations are *always consistent*, comparable to invariants of hybrid automata

Overview of the Chi language (2)

- Is a process algebra defined by means of:
  - atomic statements, e.g. assignment ($x := 2$), DAE ($\dot{x} = -x + 1$)
  - an orthogonal set of operators, e.g. sequential comp. (;) and parallel comp. ($\|$)

  that can be freely combined
- Core language small. Ease of use due to many syntactical extensions (all formally defined)
- Modular and hierarchical and scalable by means of process definition and process instantiation (reuse)
- Stochastic: definition of distributions and sampling

The Chi language definition (1)

A Chi model is of the following form:

```
model M(parameter declarations) =
|[ channel and variable declarations
:: p
]|
```

where *p* represents a process term (statement)

The Chi language definition (2)

	Process term	Meaning
p ::=	`skip`	internal action
|	x `:=` e	assignment
|	a `!` e	sending
|	a `?` x	receiving
|	`delay` e	delay statement
|	`eqn` u	equation
|	X	name of mode
|	b `->` p	guard operator
|	p `;` p	sequential composition
|	p `||` p	parallel composition
|	p `|` p	alternative composition
|	$*p$	infinite repetition
|	$[\![\, D :: p \,]\!]$	scope operator: declaration D of local variables / channels / mode definitions
|	$l_{\mathrm{p}}(\mathbf{x}_k\,,\ \mathbf{h}_m\,,\ \mathbf{e}_n)$	process instantiation

The Chi language definition (3)

Equation eqn *u*:

- Differential equation: $rde_1 = rde_2$
  rde_1 and rde_2 are real-valued expressions on variables and dotted variables
- E.g. `eqn dot x = -x + y`

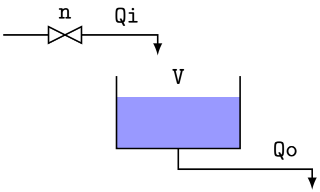
Also invariants inv *u* as in hybrid automata. E.g.:

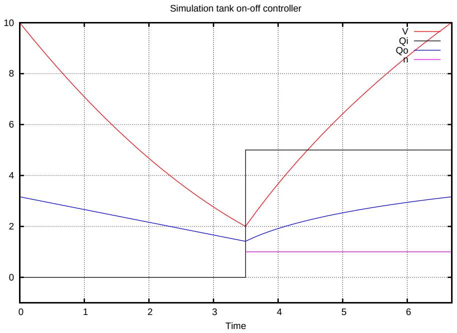
- `inv x >= 0`

Difference: equations normally do not restrict the duration of a solution, invariants do.
E.g.

- `eqn dot x = 1 || inv x >= 1`

Controlled tank system (1)



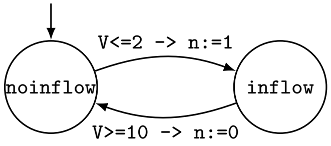


```
model ControlledTank() =
|[ var n: nat = 0, cont V: real = 10, alg Qi,Qo: real
:: eqn dot V = Qi - Qo
     ,    Qi = n * 5
     ,    Qo = sqrt(V)
|| *( V <= 2 -> n:= 1; V >= 10 -> n:= 0 )
]|
```

Controlled tank system (2)

Equivalent specification using modes, as in automata

```
model ControlledTank() =
|[ var n: nat = 0, cont V: real = 10
     , alg Qi,Qo: real
:: eqn dot V = Qi - Qo
     ,    Qi = n * 5
     ,    Qo = sqrt(V)
|| |[ mode noinflow =
        ( V <= 2  -> n:= 1; inflow )
    , mode inflow   =
        ( V >= 10 -> n:= 0; noinflow )
   :: noinflow
   ]|
]|
```



Current simulation tools for Chi

- Stand-alone symbolic simulator for hybrid and timed Chi (Python)
- S-function block hybrid Chi simulator for co-simulation in Matlab/Simulink
- Stand-alone simulator for timed Chi (C)
- For more info see: `se.wtb.tue.nl/sewiki` and `se.wtb.tue.nl/~vanbeek` (publications)

Note: slight changes of syntax in this presentation with syntax in current tools.

Nondeterminism

Models may be nondeterministic in several ways:

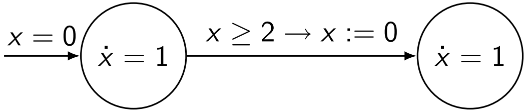
- Nondeterministic choice between actions
  E.g. `x:=1 | x:=2`
- Nondeterministic choice between multiple solutions of equations
  E.g. equation $x^2 = 1$
- Nondeterministic choice between time passing and execution of an action
  E.g. "Execute assignment `x:= 1` at any time point between now and two time units later"

Nondeterminism is often used in verification to prove that a control system behaves correctly in an non-deterministic environment, where inputs/behavior cannot be predicted.

How do simulators deal with non-determinism?

- Solutions of systems of equations usually required to be unique
- Choice between actions usually enforced by implicit or explicit priorities
  E.g. Stateflow in Simulink has explicit priority scheme for actions, where graphical position of a block / mode determines its priority
- Non-determinism may be completely forbidden (e.g. Modelica)
- Some simulators may run in different modes (e.g. Chi simulator):
  - Mode where each choice is resolved by user interaction (e.g. choice between actions, length of time step, choice between action or time passing)
  - Free running mode, where simulator makes the choice
- Enforce *urgency* to give priority to actions over time passing

Urgency


$$\begin{tikzcd} {} \arrow[r, "x=0"] & \boxed{\dot{x}=1} \arrow[r, "x \geq 2 \to x := 0"] & \boxed{\dot{x}=1} \end{tikzcd}$$

Using an urgency semantics, the assignment to x is executed *as soon as* the guard becomes true ($x = 2$).

Without urgency, the assignment can be executed when the guard becomes true, but also *at any time point* after that.

Most simulation languages use an urgency or “triggering guard” semantics. Chi language can express both urgency and non-urgency.

Strict inequalities in urgent guards

Continuous variable x and discrete variable y:

$$\begin{tikzcd} {} \arrow[r, "x = 0 \wedge y = 0"] & \dot{x} = 1 \arrow[r, "x > 2 \rightarrow y := x"] & \cdots \end{tikzcd}$$

What is the value of y after execution of the assignment $y := x$?
This depends on the semantics / meaning of the model:

- The formal mathematical meaning is that the statement cannot be executed: there is no smallest value of x that is bigger than two
- In some simulation languages the guard $x > c$ may be interpreted as for example:
  - $x \geq c + \varepsilon_\mathrm{a}$, with $\varepsilon_\mathrm{a} > 0$
  - $x \geq c(1 + \varepsilon_\mathrm{r})$, with $\varepsilon_\mathrm{r} > 0$
  - $x \in [a, b]$, with $a > 2$, $b > 2$, $b > a$

Abstract simulation algorithm

Assumptions:

- Equations have a unique solution
- Urgency: execution of actions has priority over time passing

Generalized and simplified algorithm for hybrid systems simulation:

1. Execute actions until no more can be executed. In the case of choice between actions, choose the action with the highest priority; otherwise let the simulator make a choice
2. Solve equations for the continuous and algebraic variables, until the time point that one or more actions become enabled, or until the end time of the simulation
3. Unless the end point of the simulation has been reached or the model has terminated, go back to 1.

Algebraic loops (1)

Actual simulation algorithms are much more complex, particularly in the case of algebraic loops.

A system of algebraic equations has an algebraic loop if there is a circular dependency in the equations and algebraic variables (assuming values of discrete and continuous variables as known).

```
model NoLoop1() =
|[ alg y,z: real
:: eqn y + z = 2
     ,     z = 0
]|
```

```
model AlgebraicLoop1() =
|[ alg y,z: real
:: eqn y + z = 2
     , y - z = 0
]|
```

```
model NoLoop2() =
|[ var n: real = 1.0
 , cont x: real = 1.0
 , alg y,z: real
:: eqn dot x = -x + y
     ,     y = n*x
     ,     z = y + 2
]|
```

```
model AlgebraicLoop2() =
|[ var n: real = 1.0
 , cont x: real = 1.0
 , alg y,z: real
:: eqn dot x = -x + y
     ,     y = n*x - z
     ,     z = y + 2
]|
```

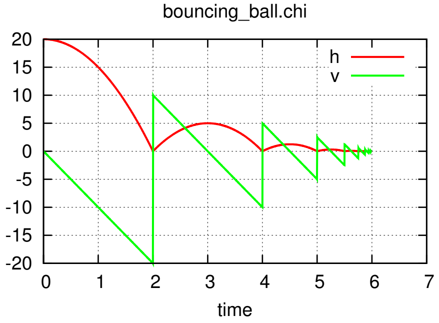
Algebraic loops (2)

```
model AlgebraicLoop1() =
|[ var n: real = 2.0, alg y,z: real
:: eqn y + z = n
     , y - z = 0
]|
```

How do hybrid systems simulators deal with algebraic loops?

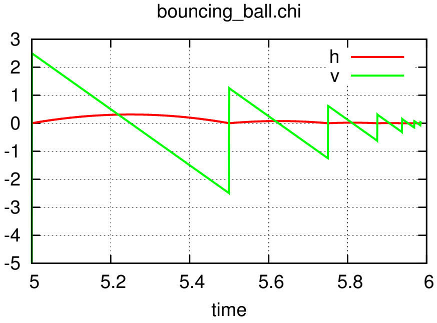
- Many don't
- Algebraic loops may be solved symbolically (`eqn y = n/2, z = n/2`)
- Algebraic loops may be solved with specialized numerical solvers
- For nonlinear dynamics, numerical solvers may require an "initial guess" close to the solution to ensure convergence of the algorithm

Simulation phenomena: the bouncing ball



```
model BounceBall() =
|[ cont h: real = 20.0
      , v: real = 0.0
:: eqn dot h = v
     , dot v = -10
|| |[ mode fall = ( h <= 0 -> v:= -0.5 * v; rise )
    , mode rise = ( v <= 0 -> skip; fall )
   :: fall
   ]|
]|
```

Accumulation point and zeno behavior



- Simulation will not proceed beyond time point 6, which is an accumulation point
- In theory there will be an infinite number of events before time point 6 is reached (zeno behavior)
- Unless special measures are taken, numerical simulation of zeno behavior may lead to erroneous results

Proper bouncing ball model

- The symbolic Chi simulator can proceed until the numerical machine accuracy is reached
- A number of bounces approaching infinity does not conform to reality
- A proper model ensures that bouncing eventually stops, and certainly before simulation errors occur; e.g. when bounce height becomes too low
- Bouncing can be stopped in two ways:
  - By adding a force counteracting gravity, so that time may still progress
  - By terminating the model (no more actions possible and no more time passing)

Proper bouncing ball model: force counteracting gravity

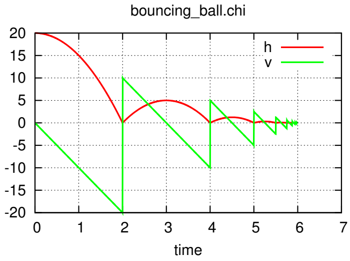
```
model BounceBall() =
|[ cont h: real = 20.0
      , v: real = 0.0
 , var g: real = 10.0, F: real = 0.0
:: eqn dot h = v
     , dot v = -g + F
|| |[ mode fall   = ( h <= 0 -> v:= -0.5 * v; rise )
    , mode rise   = ( v <= 0 -> skip
                             ; ( h >= 0.01 -> skip; fall
                               | h <  0.01 -> h:= 0.0; F:= g
                                              // terminate state machine
                               )
                    )
   :: fall
   ]|
]|
```

Proper bouncing ball model: complete termination

```
model BounceBall() =
|[ cont h: real = 20.0
      , v: real = 0.0
:: |[ mode fall   = ( eqn dot h = v, dot v = -10
                    | h <= 0 -> v:= -0.5 * v; rise
                    )
    , mode rise   = ( eqn dot h = v, dot v = -10
                    | v <= 0 -> skip
                              ; ( h >= 0.01 -> skip; fall
                                | h <  0.01 -> h:= 0.0    // terminate model
                                )
                    )
   :: fall
   ]|
]|
```

State event detection using symbolic solvers

```
model BounceBall() =
|[ cont h: real = 20.0
      , v: real = 0.0
:: eqn dot h = v
     , dot v = -10
|| |[ mode fall = ( h <= 0 -> v:= -0.5 * v; rise )
    , mode rise = ( v <= 0 -> skip; fall )
   :: fall
   ]|
]|
```



- Symbolic solvers derive a symbolic solution of the set of equations until the *exact* time point of a state event (urgent guard becomes true)
- Symbolic solvers work only for simple, well conditioned, dynamics; e.g. constant derivatives $\dot{x} = c$, or bouncing ball with $g = 10$ instead of $g = 9.81$
- Biggest problem is not the symbolic solution of complex dynamics, but the symbolic solution of state events

Numerical solvers

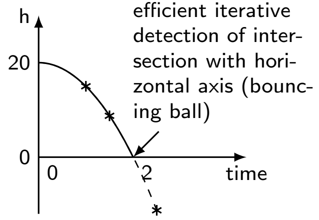
- Numerical solvers solve differential algebraic equations at *discrete time points only*
- Between discrete time points linear or polynomial interpolation (using multiple discrete points) is used
- Solutions are usually given with a predefined absolute and relative accuracy; e.g. calculated value of x is c means that error can be in the order of $(c + \varepsilon_{\mathrm{a}})(1 + \varepsilon_{\mathrm{r}})$
- Interval between discrete time points is the *step size*
- Step size can be *fixed* or *variable*

State event detection using numerical solvers

State event detection by means of zero crossing detection / root finding:

- Convert the state condition to a *root function* that calculates the value of the variable minus the threshold
- When the root function crosses zero, the state event has been located
  E.g. `V >= 2 -> n:= 1` leads to root function returning `V - 2`

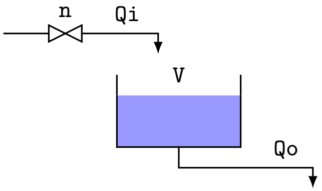
State event detection (3)



Efficient state event detection / zero crossing detection / root finding:

- Solve the system of equations until *beyond* the threshold crossing, keeping the dynamics *unchanged*
- Iteratively approach the exact point of threshold crossing (root function returns zero)
- When the exact time point of the zero crossing (`h = 0`) has been located, change the dynamics (e.g. at time point 2, execute the assignment `v:= -0.5 * v`)

Zero crossing detection problems (1)



First empty the tank, then fill it:

```
model ControlledTank()=
|[ var n: nat = 0, cont V: real = 10, alg Qi,Qo: real
:: eqn dot V = Qi - Qo
     ,     Qi = n * 5
     ,     Qo = sqrt(V)
|| V <= 0 -> n:= 1; V >= 10 -> n:= 0
]|
```

State event detection for `V <= 0` leads to taking the *root of a negative number* because of equation `Qo = sqrt(V)`!

Zero crossing detection problems (2)

Solution, conditional expression / discontinuous right hand side:

```
model ControlledTank()=
|[ var n: nat = 0, cont V: real = 10, alg Qi,Qo: real
:: eqn dot V = Qi - Qo
     ,    Qi = n * 5
     ,    Qo = ( V >= 0 -> sqrt(V) | V < 0 -> 0 )
|| V <= 0 -> n:= 1; V >= 10 -> n:= 0
]|
```

Alternative syntax in other languages:

```
Qo = if V >= 0 then sqrt(V) else 0
```

Simulation without zero crossing detection

If zero crossing detection in solver can be switched off (e.g. Matlab/Simulink, Modelica), or if zero crossing detection is not implemented:

- Variable step size numerical solver will decrease step size when approaching the discontinuity
  $\Longrightarrow$ large number of smaller and smaller steps when approaching the discontinuity
- Fixed step size solver will overstep the discontinuity
  $\Longrightarrow$ big numerical error near discontinuity

Time events

Time events are easy for hybrid simulators:

- explicitly specified (absolute timing)
- or calculated by addition of time and intervals (relative timing)

```
model ControlledTank()=
|[ var n: nat = 0
 , cont V: real = 10, alg Qi,Qo: real
:: eqn dot V = Qi - Qo
     ,    Qi = n * 5
     ,    Qo = sqrt(V)
|| time >= 2 -> n:= 1; delay 5; n:= 0
]|
```

- Absolute timing: at time point 2, the valve is opened
  `time >=2 -> n:= 1`
- Relative timing: 5 units of time after that, the valve is closed
  `delay 5; n:= 0`

Instantaneous equations (1)

Discontinuities using actions (computer science approach):

- assignments
- instantaneous equations / action predicates / jump predicates

Instantaneous equations $W : r$

- more general than assignments
- predicate r relates values of variables before and after action
- W: set of variables that may change
  (often not explicitly specified, but derived from r)

Examples instantaneous equations

- $\{x\} : x = 1$ means $x := 1$
- $\{x\} : x = x^{-} + 1$ means $x := x + 1$

Instantaneous equations (2)

Values of x before and after an action in different languages:

- x^- and x, or x^- and x^+
- `old x` and `x`, or `pre(x)` and `post(x)`

Example colliding bodies:

```
model collision(m0,m1,c: real) =
|[ cont x0: real := 0.0, x1: real := 1.0
      , v0: real := 0.0, v1: real := 0.0
:: eqn dot x0 = v0, dot v0 = 1
     , dot x1 = v1, dot v1 = 0
|| x0 >= x1 ->
     {v0,v1}:          v0 - v1 = -c * (old v1 - old v0),
             m0 * v0 + m1 * v1 = m0 * old v0 + m1 * old v1
]|
```

Newton's collision rule and conservation of momentum at collision

The compositional interchange format (CIF)

Developed in HYCON WP3 along with the *interchange format for piecewise linear systems*.

Purpose:

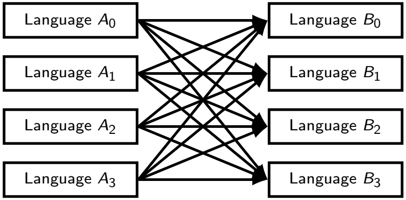
- Establish inter-operability of a wide range of tools by means of model transformations to and from the interchange formats
- Avoid the implementation of many bi-lateral translators between specific formalisms
- Provide a generic modeling formalism and simulator for a wide range of hybrid systems (CIF in particular)

If you develop tools for hybrid system analysis, the interchange formats developed in HYCON could be useful:

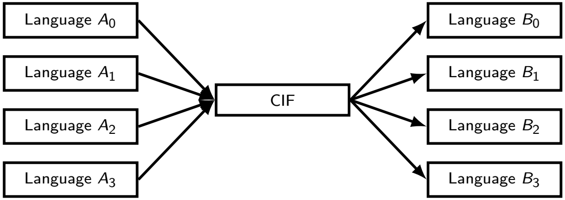
- `www.ist-hycon.org` (WP3)
- for the CIF also: `se.wtb.tue.nl/~vanbeek` (publications)

Examples of translations

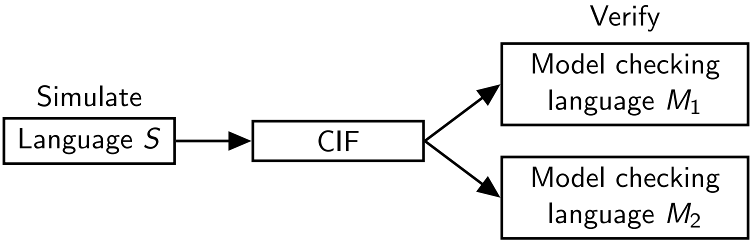
Without the CIF:

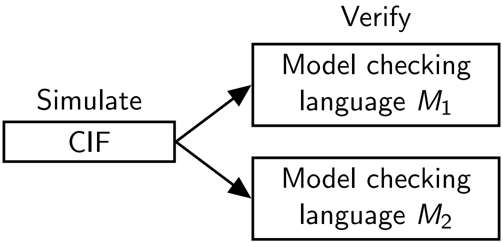


With the CIF:



Examples of applying the CIF





Requirements of the CIF

- Formal and compositional semantics, based on (hybrid) transition systems, allowing property preserving model transformations
- Concepts based on mathematics, independent of implementation aspects such as equation sorting, and numerical equation solving algorithms
- Support arbitrary differential algebraic equations (DAEs), including fully implicit equations, higher index systems, algebraic loops, steady state initialization, switched systems such as piecewise affine systems, and DAEs with discontinuous right hand sides

Requirements of the CIF

- Support a wide range of urgency concepts, such as used in hybrid automata, including ‘urgency predicates’, ‘deadline predicates’, ‘triggering guard semantics’, and ‘urgent actions’
- Support parallel composition with synchronization by means of shared variables, shared actions, and CSP channels
- Support hierarchy and modularity to allow definition of parallel modules and modules that can contain other modules (hierarchy), and to allow the definition of variables and actions as being local to a module, or shared between modules

Three formats of the CIF

Abstract:	Facilitates mathematical definition of the formal semantics CIF is defined as an automaton: $(X, X_{\mathrm{i}}, \mathrm{dtype}, V, v_0, \mathrm{init}, \mathrm{flow}, \mathrm{inv}, \mathrm{urgent}, L, E)$ and operators on the automata for parallel composition, variable and action abstraction, and urgent action definition
Concrete:	Provides user friendly syntax, usable for modeling directly in the CIF. Extensions for automaton definition and instantiation. Semantics defined by means of formal translations to the abstract format (in progress)
Transfer:	Facilitates the file generation and parsing process. E.g. XML format