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. The Chi language
4. Phenomena in hybrid simulation

Dynamics and control world view

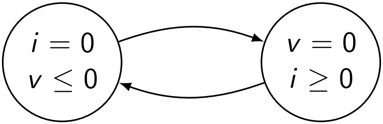
- Predominantly continuous-time system
- Modeled by means of DAEs (differential algebraic equations), or by means of a set of trajectories
- Hybrid phenomena modeled by means of discontinuous functions and/or switched equations, possibly using extended solution concepts (Filippov, Utkin) leading to sliding modes
- Evolution of a hybrid system: for each variable possibly discontinuous function of time
- 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
- Time transitions: for each variable *continuous* function of time
- Discontinuities are represented by actions

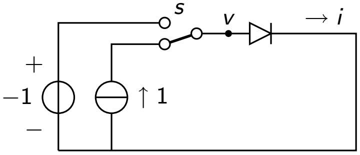


Automaton model of a diode

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
- Memory
- Derivative may be used

Examples

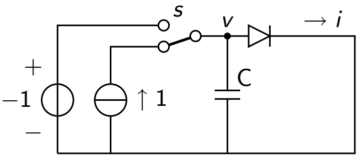
- Mass m, force F, velocity v:
  $F = m\dot{v}$, continuous variable v
- Tank with volume V, inflow Q_i, outflow Q_o:
  $\dot{V} = Q_i - Q_o$, continuous variable V
- Capacitor C, voltage v, current i:
  $i = C\dot{v}$, continuous variable v

DAE model of diode-switch-capacitor network

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

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

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



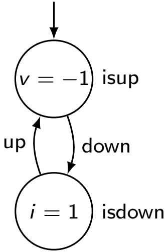
Equivalent specification:

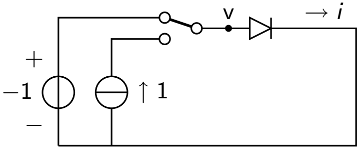
$$\begin{cases} s \wedge ((i = 0 \wedge \dot{v} = 1/\mathrm{C} \wedge v \leq 0) \vee (v = 0 \wedge i = 1)) & \text{for } 0 \leq t < 2 \\ \neg s \wedge v = -1 \wedge i = 0 & \text{for } t \geq 2 \end{cases}$$

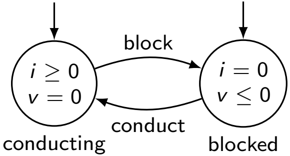
Problem: Now v must be a continuous variable, with derivative, and continuous behavior $\implies$ value of v cannot change discontinuously to -1 when switch opens!

Automaton model of diode-switch network (1)

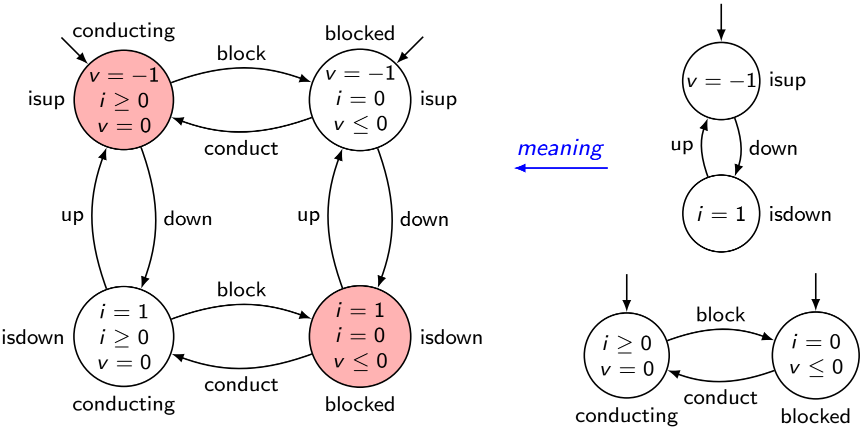
- All variables are continuous
- Mode switching: assume that value of a variable can change arbitrarily to satisfy invariant of next mode $\Longrightarrow$ automaton way of realizing behavior analogous to algebraic variable



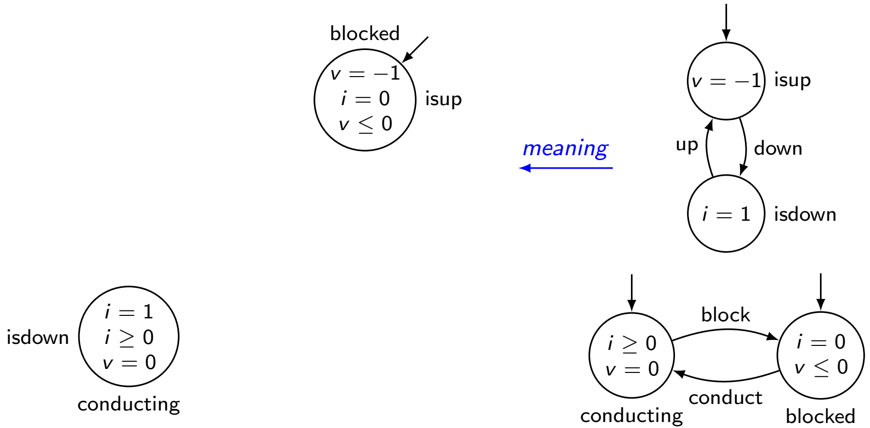




Interleaving parallel composition

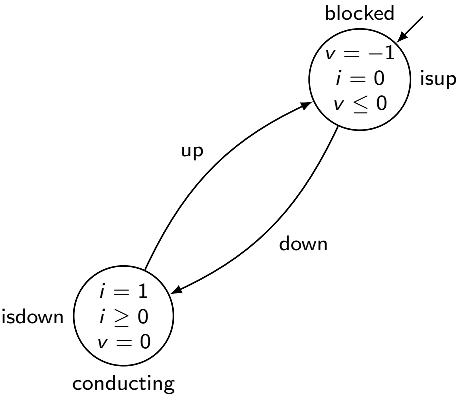


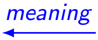
Simplified automaton

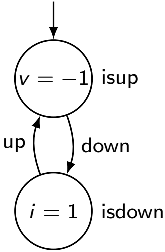


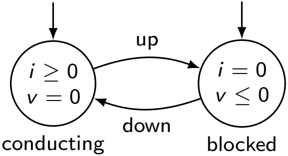
Synchronizing on common labels

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



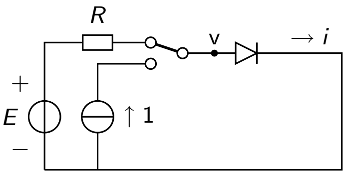


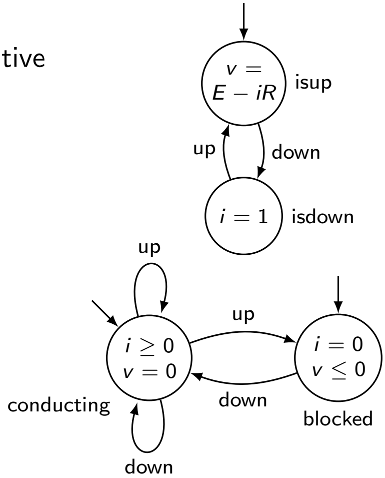




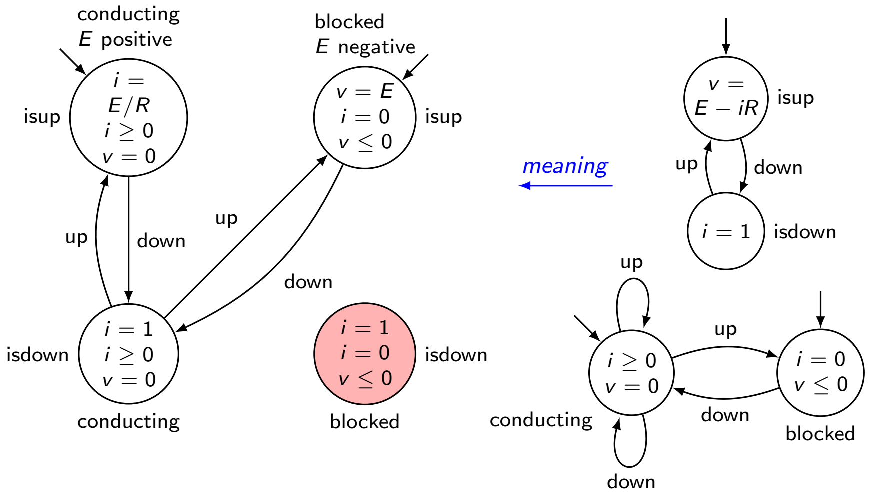
Automaton model of diode-switch network (2)

Voltage source E is positive or negative



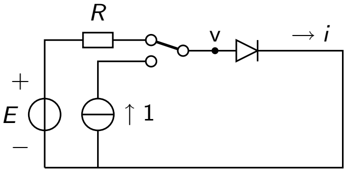


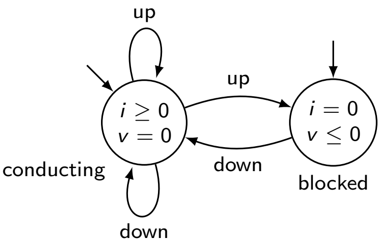
Synchronizing on common labels (2)



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 of current source: no switching of mode of automaton!



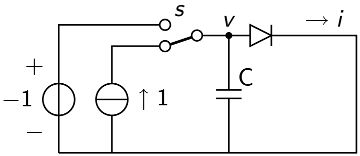


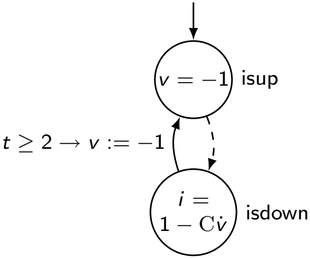
Combining the DC and CS world views (1)

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 can behave according to discontinuous functions of time

DC + CS model of diode-switch-capacitor network





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

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 only 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 instead:

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

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)

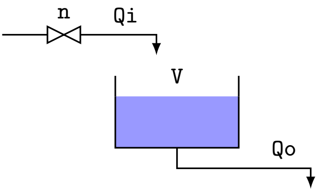
	Process term	Meaning
$p ::=$	`skip`	internal action
$\mid$	x `:=` e	assignment
$\mid$	a `!` e	sending
$\mid$	a `?` x	receiving
$\mid$	`delay` e	delay statement
$\mid$	`inv` u	invariant (equations)
$\mid$	X	name of mode
$\mid$	b `->` p	guard operator
$\mid$	p `;` p	sequential composition
$\mid$	p `||`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

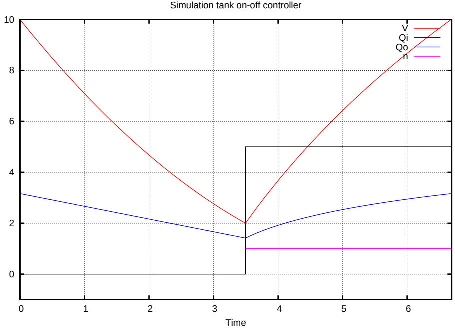
The Chi language definition (3)

Invariant inv u:

- Differential equation: $rde_1 = rde_2$
  rde_1 and rde_2 are real-valued expressions on variables and dotted variables
  E.g. $\dot{x} = -x + y$
- Other predicates
  E.g. $x \geq 0,\ y = 2x + 2,\ \mathrm{true},\ \mathrm{false}$

Controlled tank system (1)



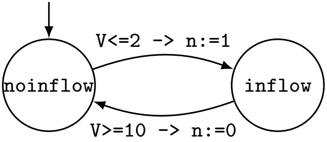



```
model ControlledTank()=
|[ var n: nat = 0, cont V: real = 10, alg Qi,Qo: real
:: inv dot V = Qi - Qo
     ,    Qi = n * 5
     ,    Qo = sqrt(V)
|| *( V <= 2 -> n:= 1; V >= 10 -> n:= 0 )
]|
```

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
:: inv 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
   ]|
]|
```

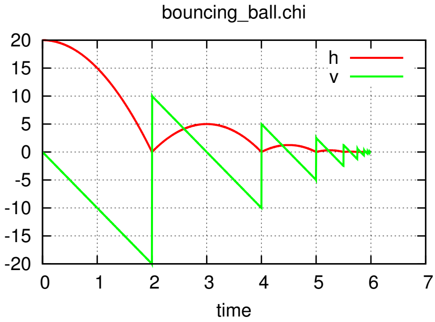


Simulation tools for Chi

- 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)
- See se.wtb.tue.nl/sewiki/

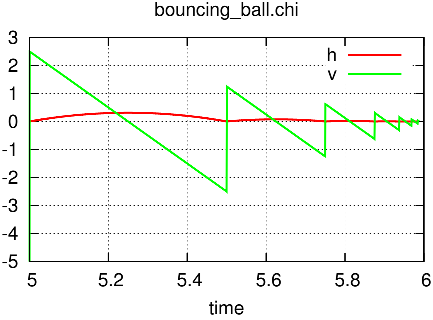
Note: slight changes of syntax in this presentation with syntax in current tools.

Simulation phenomena: the bouncing ball




```
model BounceBall() =
|[ cont h: real = 20.0
      , v: real = 0.0
:: inv 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 proper model terminates when bouncing is no longer realistic (height becomes too low)

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

State event detection (1)

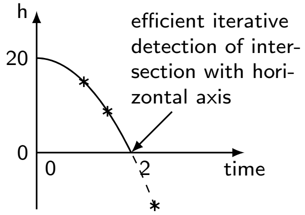
- Numerical solvers solve differential algebraic equations at discrete time points
- Interval between discrete time points is the *step size*
- Step size can be *fixed* or *variable*

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`

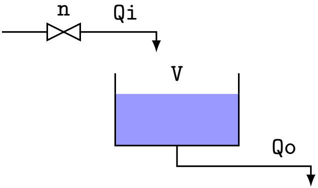
State event detection (2)

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
:: inv 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
:: inv 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
:: inv 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
:: inv 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