HRML: a hybrid relational modelling language

He Jifeng

Hybrid Systems

- Systems are composed by continuous physical component and discrete control component
- The system state evoles over time according to interacting law of discrete and continuous dynamics.
  - For discrete dynamics, it changes state instantaneously and discontinuously.
  - During continuous transitions, its state is a continuous function of continuous time and varies according to a differential equation.
- Modelers mix discrete time reactive systems with continuous time ones.

Key issues

(1) to invent formal modeling techniques for hybrid systems using which one can easily model discrete and continuous behaviours. These techniques should also compositional and hierarchical and enable the user to uniformly model a complex system at different levels.

(2) to develop formal analysis, verification and synthesis techniques to support the architecture model of hybrid systems, and guarantee the correctness of refinement and combination of subsystem models, thus solving the constructivity problem of complex systems.

History

- Simulink: Explicit model made of ODEs
- Modelica: Implicit model made of DAEs
- Hybrid automata (Alur, Henzinger ,Tavermini)
- Phase transition system (Maler),
- Declarative control (Kohn),
- Extended state-transition system (Zhou)
- Hybrid action systems (R*ö*nkk*ö*)
- Differential Dynamic Logic (Platzer)

Modelling Languages

- Hybrid CSP (He, Zhou)
- Extended Guarded Command Language with Differential Equations (R*ö*nkk*ö*)
- Hybrid π-calculuds (Rounds and Song)
- SHIFT: Network of hybrid automata
- R-Charon: Reconfigurable systems

Our approach

We propose a hybrid relational modelling language, where

(1) the discrete transitions are modelled by assignment and output as zero time actions, while the continuous transitions of physical world are described by differential equations and synchronous constructs.
(2) The signal mechanism is used for describing interaction between system controller with physical device.
(3) Three types of guards are introduced to model the condition under which the system controller switches to a new mode.

Contents

1. Hybrid Relation calculs.
2. HRML: a hybrid modelling language
3. Laws of Hybrid Programs
4. Case study

Relation

A relation is a pair $(\alpha P, P)$, where P is a predicate containing no free variables other than in αP, and αP is a set of variable names:

$$\alpha P \;=\; in\alpha \cup out\alpha$$

where $in\alpha$ is a set of undashed variables standing for initial value

and $out\alpha$ is a set of dashed variables standing for final value.

Hybrid relation

A *hybrid relation* is a binary relation P where its alphabet αP is enlarged with a set $con\alpha$ of *continuous variables*, which are introduced to record the dynamic behaviour of physical coponents

$$\alpha P \;=\; in\alpha \cup con\alpha \cup out\alpha$$

Discrete variables

The discrete variables observable at the start of a hybrid program are the same as those observable at the end, in this case the output alphabet is obtained just by putting a dash on all the variables of the input alphabet:

$$out\alpha \;=\; \{x' \mid x \in in\alpha\}$$

Continuous variables

The continuous variables of are used to record dynamic behavior of the physical devices controlled by the program, and they are modelled as mappings from time to physical state of the devices.

$con\alpha$ is divided into two sets $own\alpha$ and $env\alpha$ which represent the set of continuous variables owned by P and the set of continuous variables accessble by P respectively.

Differential equation

Differential equation $DF =_{df} (F(v, \dot{v}) = 0)$ can be seen as a hybrid relation

$$in\alpha =_{df} \{t\}$$

$$out\alpha =_{df} \{t'\}$$

$$own\alpha =_{df} \{v\}$$

$$DF =_{df} (t \leq t') \wedge$$

$$\forall \tau \in [t, t') \bullet (F(v, \dot{v})(\tau) = 0)$$

Hybrid Relatin Calculus

- Sequential operators:
  - Choice
  - Conditional
  - Composition
- Parallel operators:
  - Disjoint parallel
  - Parallel by merge
- Recursion

Disjoint parallel

Let P and Q be hybrid relations with disjoint output alphabet and $con\alpha$. Deine their parallel composition $P \| Q$ by

$$P \parallel Q \ =_{df} \ (P \wedge Q)$$

where

$$in\alpha \ =_{df} \ in\alpha P \cup in\alpha Q$$

$$out\alpha \ =_{df} \ out\alpha P \cup out\alpha Q$$

$$con\alpha \ =_{df} \ con\alpha P \cup con\alpha Q$$

Parallel with shared output

A merge mechanism M is a pair $(x : \mathbf{Val}, op)$, where x is a variable of type **Val**, and op is a binary operator over Val.

Examples

(1) $M1 = (x : Real, \mathbf{max})$ is a merge mechanism.

(2) $M3 = (x : L, \mathbf{glb})$, where L is a lattice, is a merge mechanism.

Parallel by merge

Let P and Q be hybrid relations with $x' \in out\alpha P \cap out\alpha Q$. We define their parallel composition equipped with the merge mechanism M, denoted by $P \parallel_M Q$, as follows:

$$
\begin{aligned}
P \parallel_M Q &=_{df} \exists m, n : \mathbf{Val} \bullet \\
&\qquad (P[m/x'] \wedge Q[n/x'] \wedge (x' = (m \mathbin{op} n))) \\
in\alpha &=_{df} in\alpha P \cup in\alpha Q \\
out\alpha &=_{df} out\alpha P \cup out\alpha Q \\
con\alpha &=_{df} con\alpha P \cup con\alpha Q
\end{aligned}
$$

Healthiness Conditions

The healthiness conditions of hybrid programs are closely related to the following features:

- Time
- Interaction mechanism
- Intermediate Observation
- Divergence

Introducing Time

Time variables t and t' are introduced in an alphabet of hybrid relation to record the start and complete time instants of a transition. a hybrid relation P has to meet the following condition

$$P(t, t') \;=\; P(t, t') \wedge (t \leq t')$$

We introduce a function **H1** to convert a hybrid relation into a healthy hybrid relation:

$$\mathbf{H1}(P) \;=_{df}\; P \wedge (t \leq t')$$

Interaction mechanism

A signal, denoted by its name, has two types of status, i.e., either presence or absence.

A signal is present if

(1) it is an input signal received from the environment, or

(2) it is emitted as the result of performing an output command.

For any signal s, we use a clock variable s.**clock** to record the time instants at which the signal s is present.

Healthiness condition of clock variable

$s.\mathbf{clock}$ has to be a subset of $s.\mathbf{clock}'$ since the latter may be added some time instants of $[t,\, t']$ at which the signal s is present. Thus, a hybrid relation is required to meet the following condition:

$$P \;=\; P \wedge \mathbf{inv}(s)$$

where

$\mathbf{inv}(s) \;=_{df}\; (s.\mathbf{clock} \subseteq s.\mathbf{clock}') \wedge (s.\mathbf{clock}' \subseteq (s.\mathbf{clock} \cup [t,\, t']))$

We introduce a function **H2** to convert a hybrid relation into a healthy hybrid relation:

$$\mathbf{H2}(P) \;=_{df}\; P \wedge \mathbf{inv}(s)$$

Introducing program status variables

We add st and st' to the output alphabet of a hybridrelation to describe the program status.

- $st = term$ indicates its sequential predecessor terminates successfully. As a result, the control passes to the hybrid program.
- $st = stable$ indicates the predecessor has not finished yet (for example, it is waiting for occurrences of some events). As a result, the hybrid program can not start its execution.
- $st = div$ indicates the predecessor enters a chaotic status, and can not be rescued by its environment.

Healthiness condition of st

A hybrid program has to keep idle until its sequential predecessor terminates successfully.

$$P \;=\; (\mathbf{H1} \circ \mathbf{H2})(P) \lhd st = term \rhd \mathbf{skip}$$

where

$$\mathbf{skip} \;=_{df}\; II_A \lhd (st \neq div) \rhd (\mathbf{H1} \circ \mathbf{H2})(\bot)$$

We define a mapping to convert a hybrid relation into a **HC3**-healthy one:

$$\mathbf{H3}(P) \;=_{df}\; (\mathbf{H1} \circ \mathbf{H2})(P) \lhd st = term \rhd \mathbf{skip}$$

Healthiness condition of st'

Once a hybrid program enters a divergent state, its future behaviour becomes uncontrollable. This requires it to meet the following condition:

$$P \;=\; P; \mathbf{skip}$$

Define

$$\mathbf{H4}(P) \;=_{df}\; P; \mathbf{skip}$$

Composition of healthy convertions Hi

Define

$$\mathbf{H} =_{df} (\mathbf{H1} \circ \mathbf{H2} \circ \mathbf{H3} \circ \mathbf{H4})$$

Theorem

P satisfies $\mathbf{HC1} - \mathbf{HC4}$ if and only if $P = \mathbf{H}(P)$

Theorem

(1) **H** is monotonic and idempotent.

(2) Healthy hybrid relations form a complete lattice L.

Closure of healthy hybrid relations

Theorem

(1) $\mathbf{H}(P) \sqcap \mathbf{H}(Q) = \mathbf{H}(P \sqcap Q)$

(2) $\mathbf{H}(P) \lhd b \rhd \mathbf{H}(Q) = \mathbf{H}(P \lhd b \rhd Q)$

(3) $\mathbf{H}(P); \mathbf{H}(Q) = \mathbf{H}(P; \mathbf{H}(Q))$

(4) If P and Q lie in the complete lattice L, then so does $(P \|_M Q)$

where the merge mechanism
$M =_{df} (st : \{term, stable, div\}, \mathbf{glb})$.

HRML: a hybrid relational modelling language

$$
\begin{array}{rcl}
AP & ::= & \mathbf{skip} \,|\, \mathbf{chaos} \,|\, \mathbf{stop} \,|\, x := e \,|\, !s \,|\, \mathbf{delay}(\delta) \\
EQ & ::= & R(v,\, \dot{v}) \,|\, EQ \,\mathbf{init}\, v_0 \,|\, EQ \| EQ \\
P & ::= & AP \,|\, P \sqcap P \,|\, P; P \,|\, P \lhd b(x) \rhd P \,|\, P \| P \,| \\
 & & EQ \,\mathbf{until}\, g \,|\, \mathbf{when}(G) \,|\, \mu X \bullet P(X) \\
 & & \mathbf{timer}\, c \bullet P | \ \mathbf{signal}\, s \ \bullet \ P \\
g & ::= & \mathbf{skip} \,|\, s \,|\, test \,|\, g \cdot g \,|\, g + g \\
test & ::= & \mathbf{true} \,|\, v \geq e \,|\, v \leq e \,|\, test \wedge test \,|\, test \vee test \\
G & ::= & g \& P \,|\, G [\!] G
\end{array}
$$

Alphabet of HRML programs

The alphabet of an $HRML$ program P of $HRML$ consists of the following components

$$\alpha P =_{df} \mathbf{in}\alpha P \cup \mathbf{out}\alpha P \cup \mathbf{con}\alpha P$$

where $\mathbf{con}\alpha P = \mathbf{own}\alpha P \cup \mathbf{env}\alpha P$, and $\mathbf{own}\alpha$ comprises two types of continuous variables:

$$\mathbf{con}\alpha =_{df} \mathbf{phy}\alpha \cup \mathbf{timer}\alpha$$

to specify the physical devices and timers owned by P respectively.

Elements of the input alphabet

$\mathbf{in}\alpha P$ denotes the set of input variables of P

$$\mathbf{in}\alpha P \ =_{df} \ \{st, t, count\} \cup PVar \cup ClockVar$$

- *count* (of the type non-negative reals) describes the emitting order of the signals that occur in the same time instant.
- $PVar$ denotes the set of program variables.
- $ClockVar$ is the set of clock variables:

  $$ClockVar \ =_{df} \{s.clock \mid s \in InSignal \cup OutSignal\}$$

  where $s.clock$ records the time instant t and the emit order *count* at which signal s occurs.

Atomic commands

1. Assignment:

it is used to model the discrete change. The execution of $x := e$ assigns the value of e to variable x instantaneously

$$(x := e) \ =_{df} \ \mathbf{H}(II_{\mathbf{in}\alpha}[e/x])$$

2. Output:

$!s$ emits signal s, and then terminates immediately.

$$!s \ =_{df} \ \mathbf{H}(II_{\mathbf{in}\alpha}[(s.clock \cup \{(t,\, count)\})/s.clock])$$

Laws of output

Theorem

(1) $!s_1; !s_2 \;=\; !s_2; !s_1$

(2) $!s; !s \;=\; !s$

(3) $!s; (x := e) \;=\; (x := e); !s$

Guard

Let g be a guard g, the boolean function

$$g.\textbf{fired} : Intervel \rightarrow Time \rightarrow Bool$$

is used to specify its status over the time interval $[t,\, t']$.

For any $\tau \in [t,\, t']$

$$g.\textbf{fired}([t,\, t'])(\tau) = true$$

indicates the guard g is ignited at the time instant τ.

Laws of guard

Define $(g = h) =_{df} (g.\mathbf{fired} = h.\mathbf{fired})$

Theorem

(1) $(\text{Guard}, +, \cdot, \mathbf{false}, \mathbf{true})$ forms a Boolean algebra.

(2) $\cdot$ has **false** as its zero.

(3) $+$ has **true** as zero.

Corollary $g + (g \cdot h) = g$

Order

We say g is weaker than h (denoted by $g \leq h$), if the ignition of h can fire g immediately:

$$g \leq h \ =_{df} \ h = (h \cdot g)$$

Theorem $\leq$ is a partial order.

Theorem $g \leq h$ iff $g = (g + h)$

Ignition of guards

We introduce the following boolean function

$$g.\textbf{triggered} : \mathit{Interval} \rightarrow \mathit{Bool}$$

to identify the cases when the guard g is only fired at the endpoint of the interval

$$g.\textbf{triggered}([t, t']) \ =_{df} \ \left(\begin{array}{l} g.\textbf{fired}([t, t'])(t') \wedge \\ \\ \forall \tau \in [t, t') \bullet \neg g.\textbf{fired}([t, t'])(\tau) \end{array} \right)$$

To specify those cases when the guard g remains idle we introduce the boolean function $g.\textbf{inactive}$

$$g.\textbf{inactive}([t, t']) \ =_{df} \ \forall \tau \in [t, t'] \bullet \neg g.\textbf{fired}([t, t'])(\tau)$$

Properties of triggered*and*inactive

Theorem

(1) $(g1 + g2).\mathbf{triggered} \;=$

$$\left(\begin{array}{l} g1.\mathbf{triggered} \wedge (g2.\mathbf{triggered} \vee g2.\mathbf{inactive}) \vee \\ \\ g2.\mathbf{triggered} \wedge (g1.\mathbf{triggered} \vee g1.\mathbf{inactive}) \end{array} \right)$$

(2) $(g1 + g2).\mathbf{inactive} = \; g1.\mathbf{inactive} \wedge g2.\mathbf{inactive}$

when statement

The program $\mathbf{when}(g_1 \& P_1 [\!] .... [\!] g_n \& P_n)$ waits for one of its guards to be fired, then selects a program P_i with the ignited guard to be executed.

$\mathbf{when}(g_1 \& P_1 [\!] .... [\!] g_n \& P_n) =_{df}$

$$\mathbf{H}(st' = stable \wedge II_{C \cup \{count\}} \wedge \mathbf{time-passing} \wedge \bigwedge_{1 \leq k \leq n} (g_k.\mathbf{inactive}))$$

$$\vee$$

$$\bigvee_{1 \leq i \leq n} \mathbf{H} \left(\begin{array}{l} st' = term \wedge II_{C \cup PVar} \wedge \mathbf{time-passing} \wedge \\ \mathbf{update}(count,\, g_i) \wedge \\ g_i.\mathbf{triggered} \wedge \bigwedge_{k \neq i} (g_k.\mathbf{triggered} \vee g_k.\mathbf{inactive}) \end{array} \right) ; P_i$$

when statememt

where

$$C =_{df} \{s.clock \mid s \in OutSignal\}$$

$$\begin{aligned} \mathbf{update}(count,\, g) =_{df}\ & (count' = count) \\ & \lhd g \cap InSignal = \emptyset \rhd \\ & (count' > \mathbf{max}(count,\, \mathbf{index}(g)) \end{aligned}$$

$$\mathbf{index}(g) =_{df} \mathbf{max}(\{0\} \cup \{\pi_2(\mathbf{last}(s.clock')) \mid s \in g\})$$

Laws of when statement

If a **when** construct comprises a **skip** guard, then other guarded branches can be selected only when their corresponding guards are fired immediately after the **when** statement starts its execution.

L1 $\mathbf{when}((\mathbf{skip}\&P) \parallel (g\&)Q \parallel G)$

$= \ \mathbf{when}((\mathbf{skip}\&P) \parallel ((\mathbf{skip} \cdot g)\&Q) \parallel G)$

Corollary

$\mathbf{when}((\mathbf{skip}\&P) \parallel (g\&P) \parallel G) \ = \ \mathbf{when}((\mathbf{skip}\&P) \parallel G)$

Laws of when statement

Once a guard is fired, so does the same guard in its guarded **when**-construct.

L2 $\mathbf{when}(((g \cdot h_1) \& \mathbf{when}((g \cdot h_2) \& P \parallel G_1)) \parallel G_2)$

$= \mathbf{when}(((g \cdot h_1) \& \mathbf{when}(((\mathbf{skip} + g) \cdot h_2) \parallel G_1)) \parallel G_2)$

Corollary If $g \leq h$, then
$\mathbf{when}(h \& \mathbf{when}(g \& P \parallel G_1) \parallel G_2) = \mathbf{when}(h \& \mathbf{when}(\mathbf{skip} \& P \parallel G_1) \parallel G_2)$

Laws of concurrency

An input signal can ignite the corresponding guard in the **when** and **until** statements.

L1 $(!s; P) \| (R \,\mathbf{until} (s + g)\,;\, Q) \;=\; (!s; P) \| Q$

L2 $(!s; P) \| \mathbf{when}((s \& Q) \,[\!]\, G)$

$=\;\; (!s; P) \| \mathbf{when}((\mathbf{skip} \& Q) \,[\!]\, G)$

Corollary

$(!s; P) \| \mathbf{when}(s \& Q) \;=\; (!s; P) \| Q$

Laws of concurrency

when statements are closed under concurrent composition.

L3 Let $P = \mathbf{when}((g_1 \& P_1) \,[\!]\, ... \,[\!]\, (g_n \& P_n))$

and $Q = \mathbf{when}((h_1 \& Q_1) \,[\!]\, ... \,[\!]\, (h_m \& Q_m))$. Then

$$P \| Q \;=\; \mathbf{when} \left(\begin{array}{l} (g_1 \& (P_1 \| Q)) \,[\!]\, ... \,[\!]\, (g_n \& (P_n \| Q)) \,[\!] \\ (h_1 \& (P \| Q_1)) \,[\!]\, ... \,[\!]\, (h_m \& (P \| Q_m)) [\!] \\ [\!]_{i,\,j} ((g_i \cdot h_j) \& (P_i \| Q_j)) \end{array} \right)$$

Corollary

$\mathbf{when}(g \& P) \| \mathbf{when}(g \& Q) \;=\; \mathbf{when}(g \& (P \| Q))$

Introducing signal

The first theorem demonstrates how to introduce signals as an interaction mechanism to link a physical device with a controller

Theorem

Let $l < m < n$. If $R \sqsupseteq (\dot{v} > 0)$ then

$$(R \,\mathbf{init}\, m \,\mathbf{until}\, (v \geq n)) \;=$$

$$\mathbf{sig}\, s,\, u \bullet \left(\begin{array}{l} (R \,\mathbf{init}\, m \,\mathbf{until}\, s) \\ \\ \parallel \mathbf{when}((v \geq n)\&!s \,[\![\, (v \leq l)\&!u) \end{array} \right)$$

Introduce signal

Theorem

Let $l < m < n$. If $R \Rightarrow (\dot{v} < 0)$ then

$(R \,\mathbf{init}\, m \,\mathbf{until}\, (v \leq l))$

$$= \mathbf{signal}\, s,\, u \bullet \left(\begin{array}{l} (R \,\mathbf{init}\, m \,\mathbf{until}\, u) \\ \\ \parallel \mathbf{when}(((v \geq n)\&!s) \,[\!]\, ((v \leq l)\&!u)) \end{array} \right)$$

Adjust the sampling rate

The following theorem is used to reduce the sampling rate of the controller by estimating the change speed of physical state

Theorem Let $l < m < n$.

If $R \sqsupseteq (0 < \dot{v} \leq r)$ and $\delta < (n - m)/r$, then

$(R \,\mathbf{init}\, m \,\mathbf{until}\, (v \geq n))$

$$= \mathbf{signal}\, s,\, u \bullet \left(\begin{array}{l} (R \,\mathbf{init}\, m \,\mathbf{until}\, s) \\ \| \left(\begin{array}{l} \mathbf{delay}(\delta); \\ \\ \mathbf{when}(((v \geq n) \& !s) \,[\!]\, ((v \leq l) \& !u)) \end{array} \right) \end{array} \right)$$

Water Tank

The system is used to control the water level in a tank by switching on a control valve, The water level will rise whenever the valve is open, otherwise it will drop down. Assume that the rising and dropping phases are governed by the following differential equations:

$$Rise \ =_{df} \ (\dot{h} = f(h))$$

$$Drop \ =_{df} \ (\dot{h} = -g(h))$$

where $f(h) > 0$ and $g(h) > 0$.

Requirement

The goal is to maintain the water level between L and M units. Assume that initially the water level is M and the control valve is open. Such a requirement can be formalised as an **until** statement.

$$Req \ =_{df} \ (h \leftarrow M); (Goal \, \mathbf{until} \, false)$$

where $Goal \ =_{df} \ (L \leq h \leq H) \ \wedge \ (\dot{h} \neq 0)$

Design 1

To deliver a refinement of Req by investigating the rising up and falling down phases of the water level separately Let
$L < M^- < M < M^+ < H$. Define

$$Up_1 \;=_{df}\; (L < h < H) \wedge (0 < \dot{h} < r)\,\mathbf{until}\,(h \geq M^+)$$

$$Down_1 \;=_{df}\; (L < h < H) \wedge (l < \dot{h} < 0)\,\mathbf{until}\,(h \leq M^-)$$

where we assume that the variations of water level are bounded

$$r \;=_{df}\; \mathbf{sup}\{f(h) \mid L \leq h \leq M\}$$

$$l \;=_{df}\; \mathbf{inf}\{-g(h) \mid L \leq h \leq M\}$$

Define $Design_1 \;=_{df}\; (h \leftarrow M); (Up_1; Down_1)^*$

Theorem $Req \sqsubseteq Design_1$

Design 2

To refine Up_1 and $Down_1$ by introducing the differential equations *Rise* and *Drop*:

$$Up_2 \quad =_{df} \quad Rise\,\textbf{until}\,(h \geq M^+)$$

$$Down_2 \quad =_{df} \quad Drop\,\textbf{until}\,(h \leq M^-)$$

Let $Design_2 \;=_{df}\; (h \leftarrow M); (Up_2; Down_2)^*$

Theorem $Design_1 \sqsubseteq Design_2$

Design 3

To construct a control program to monitor the variations in water level, and emit signals to alternate the transition modes of the tank. Let

$$Up_3 \;=_{df}\; Rise\,\mathbf{until}\,off$$

$$Down_3 \;=_{df}\; Drop\,\mathbf{until}\,on$$

and define

$$Ctrl \;=_{df}\; C^*$$

where $C \;=_{df}\; (\mathbf{delay}(\rho); \mathbf{when}(h \geq M^+ \& !off \| h \leq M^- \& !on))$

The delay command $\mathbf{delay}(\rho)$ of C is inserted to avoid the reignition of consecutive **when** statements.

Design 3

The water tank can then be described by the hybrid program $Tank$

$$Tank(x) \ =_{df} \ (h \leftarrow x); WL^*$$

where $WL \ =_{df} \ (Up_3; Down_3)$

and the parameter x denotes the initial water level.

Define $Design_3 \ =_{df} \ \mathbf{signal}\, on, off \bullet (Tank(M) \| Ctrl)$

Theorem $Design_2 \sqsubseteq Design_3(M)$

Conclusion

The objectives of this work are:

- to invent formal modeling techniques for hybrid systems using which one can easily model discrete and continuous behaviours. These techniques should also compositional and hierarchical and enable the user to uniformly model a complex system at different levels, thus solving the modeling problem of complex systems.
- to develop formal analysis, verification and synthesis techniques to support the above architecture model of hybrid systems, and guarantee the correctness of refinement and combination of subsystem models using the above modeling techniques, thus solving the constructivity problem of complex systems.