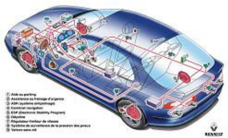
A unified formalism for complex systems architecture

Boris Golden

Defence for a PhD in Computer Science

May the 13^{th}, 2013

Our aim



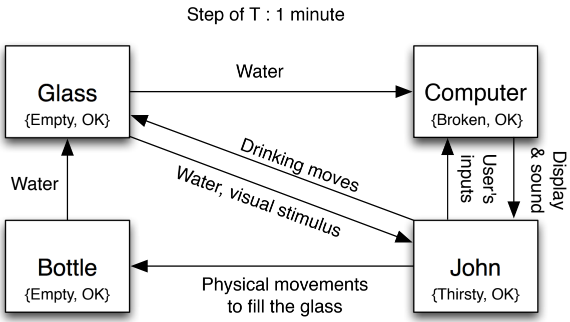
We want to define a **formal framework to model & reason on such "complex industrial" systems** characterized by:

- heterogeneous components (esp. both discrete & continuous)
- a huge quantity of which are integrated at multiple scales.

Towards a unified formalism

- Dedicated, **well-formalized tools exist to design specific types of systems** (physical, software, organizational)
- And the underlying approaches to model & design those various systems have **strong similarities at a certain level of abstraction**
- In Systems engineering (i.e. the discipline to design complex industrial systems with heterogeneous parts), **architectural models & methods deal with the "big picture", but lack a formal semantics** unifying all existing vertical formalisms
- $\rightarrow$ We propose a **unified formalism for these "complex" systems, by dealing with heterogeneity + multiscale**!

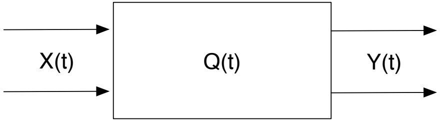
Systems approach is the basis of all specific systems design



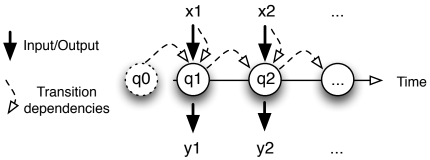
This intuitive graphical language allows to **describe all systems with the same concepts**: time, data, flow, box, state, behavior.

How we model systems

- A **functional** machine processing dataflows



- with **step by step transitions** to change state & output at predefined moments of time characteristic of the system:



- Systems can then be **integrated together** as Lego blocks.

Summary of this presentation

1. Dataflows
   - Time
   - Data
   - Dataflows
2. Systems
3. Integration operators
4. Architecture of systems

Time reference

A time reference is a unified & generic modeling of time:

Definition (Time reference)

A **time reference** is an infinite set T together with an internal law $+^T : T \times T \to T$ and a pointed subset $(T^+, 0^T)$ satisfying the following conditions:

- upon T^+: closure, initiality, left neutrality
- upon T: associativity, right neutrality, left cancellation, linearity

Example

$\mathbb{N}$, $\mathbb{R}$, $^*\mathbb{R}$ (set of nonstandard real numbers containing infinitesimal, standard & infinite real numbers).

Good news: time is linear!

Proposition (Total order on a time reference)

We can define a **total order** $\preceq^T$ *on* T *as follows:*

$$a \preceq^T b \Leftrightarrow \exists c \in T^+, \; b = a +^T c$$

Remark: this is a classical result.

Time scales

Sets of moments of a time reference (later used to define systems, both discrete & continuous):

Definition (Time scale)

A **time scale** is any subset $\mathbb{T}$ of a time reference T such that:

- $\mathbb{T}$ has a minimum $m^{\mathbb{T}} \in \mathbb{T}$ such that $0 \preceq m^{\mathbb{T}}$
- $\forall t \in T,\ \mathbb{T}_{t+} = \{t' \in \mathbb{T} | t' \succ t\}$ has a minimum $succ^{\mathbb{T}}(t)$
- $\forall t \in T \mid t \succ m^{\mathbb{T}}$, the set $\mathbb{T}_{t-} = \{t' \in \mathbb{T} | t' \prec t\}$ has a maximum $pred^{\mathbb{T}}(t)$
- the axiom of induction is verified on $\mathbb{T}$.

Expressivity of time scales

Example

A time scale on the time reference $\mathbb{R}^+$ can be any subset A such that: $\forall t, t' \in \mathbb{R}^+,\ |A \cap [t; t+t']|$ is finite.

Example

A regular time scale can be $^*\mathbb{N}\tau$ where $\tau \in\ ^*\mathbb{R}^+$ is the step, $0 \in\ ^*\mathbb{N}\tau$ and $\forall t \in\ ^*\mathbb{N}\tau$, $succ^{^*\mathbb{N}\tau}(t) = t + \tau$.

Property (unification of discrete & continuous time scales)

In the last example, we can thus define **both discrete and continuous time scales in a unified formalism**, depending on whether τ is infinitesimal or finite!

Time scales are a good definition of time for systems!

Because time scales:

1. **allow recursive definitions** (for dataflow transformation)
2. **unify discrete & continuous time** (e.g. within $^*\mathbb{R}$)
3. **can be mixed together** (for systems integration):

Proposition (Finite union of time scales)

A finite union of time scales is still a time scale.

Datasets

We define the data that will be manipulated by systems. A dataset is an alphabet of symbols together with a "data behavior":

Definition (Dataset)

A **dataset** is a 2-tuple $\mathcal{D} = (D, \mathcal{B})$ such that:

- D is a set containing a special blank ϵ
- $\mathcal{B} = (r, w)$ where $r : D \to D$ and $w : D \times D \to D$ verify

$$r(\epsilon) = \epsilon \qquad (R1)$$
$$r\big(r(d)\big) = r(d) \qquad (R2)$$
$$r\big(w(d, d')\big) = r(d') \qquad (R3)$$
$$w\big(r(d'), d\big) = d \qquad (W1)$$
$$w\big(w(d, d'), r(d')\big) = w(d, d') \qquad (W2)$$

Data behaviors give a meaningful semantics to data

Example (Persistent data behavior)

In this case, data cannot be consumed by a reading, and every writing erases the previous data (e.g. what the screen of my phone displays):

$$r(d) = d \quad \text{and} \quad w(\_, d) = d$$

Example (Consumable data behavior)

In this case, data is consumed by a reading, and every writing (excepted when it is ϵ) erases the previous data (e.g. my phone itself as an object):

$$r(d) = \epsilon \quad \text{and} \quad w(d, d') = \begin{cases} d & \text{if } \; d' = \epsilon \\ d' & \text{else} \end{cases}$$

Datasets are a good definition of data for systems!

Because we want to handle the following properties of data:

- they **carry information**
- they can have **different modeling semantics** (e.g. persistent vs consumable) to handle heterogeneity of data
- we want to be able to give a **consistent synchronization** of data between different time scales

Dataflows are flows of data at moments of a time scale

Definition (Dataflow)

A **dataflow** over $(\mathcal{D},\mathbb{T})$ is a mapping $X : \mathbb{T} \to D$. The set of all dataflows over $(\mathcal{D},\mathbb{T})$ is noted $\mathcal{D}^{\mathbb{T}}$.

A dataflow can be observed from any time scale:

Definition (Projection of a dataflow on a time scale)

The **projection** $X_{\mathbb{T}_P}$ **of** X **on** $\mathbb{T}_P$ is the dataflow on $(\mathcal{D}, \mathbb{T}_P)$ induced (following the data behaviors) by X on $\mathbb{T}_P$.

Equivalent dataflows cannot be discriminated by any projection:

Definition (Equivalence of dataflows as far as)

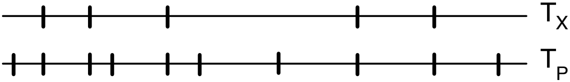
X and Y are **equivalent as far as** $t_0 \in T$ (noted $X \sim_{t_0} Y$) iif:
$\forall \mathbb{T} \in Ts(T),\ \forall t \in \mathbb{T} \mid t \preceq t_0,\quad X_{\mathbb{T}}(t) = Y_{\mathbb{T}}(t)$

Consistency of dataflow projections

Proposition (Equivalence of projection on a finer time scale)

Let X be a dataflow on $(\mathcal{D}, \mathbb{T}_X)$ and let $\mathbb{T}_P$ be a time scale such that $\mathbb{T}_X \subseteq \mathbb{T}_P$. Then:

$$X \sim X_{\mathbb{T}_P}$$



Proposition (Equivalence of projections on nested time scales)

Let X be a dataflow and let $\mathbb{T} \subseteq \mathbb{T}_P$ be two nested time scales. Then, we have:

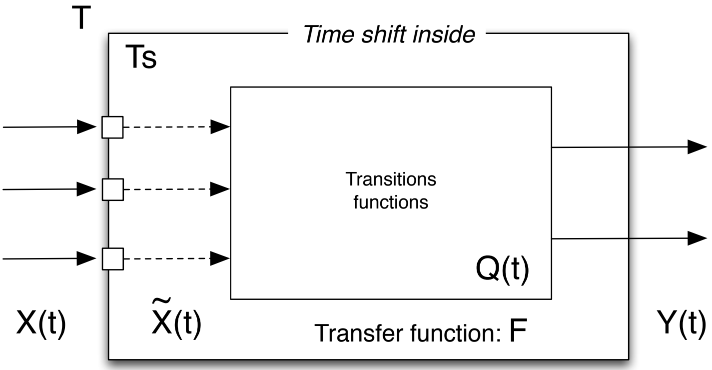
$$(X_{\mathbb{T}_P})_{\mathbb{T}} = X_{\mathbb{T}}$$

Dataflows are a good definition of systemic flows!

Because they have the following properties:

- they **capture the heterogeneity** of time and data
- the dataflow equivalence ensures a **consistent definition of systems** by preventing modeling artefacts
- they will ensure a **consistent definition of systems integration** thanks to the ability to project dataflows between time scales

Representation of a system



Definition of a system

Definition (System)

A **system** is a 7-tuple $\int = (\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, \mathcal{Q})$ where

- $\mathbb{T}_s$ is the time scale of the system
- $Input = (In, \mathcal{I})$ and $Output = (Out, \mathcal{O})$ are respectively input and output datasets
- S is the non-empty set of states
- q_0 is the initial state of the system
- $\mathcal{F} : In \times S \times \mathbb{T}_s \to Out$ is the functional behavior
- $\mathcal{Q} : In \times S \times \mathbb{T}_s \to S$ is the states behavior.

- Behavior functions contain $\mathbb{T}_s$ for integration consistency.
- Inputs can have an instantaneous influence on state & output.

Step by step execution within time

Definition (Execution of a system)

Let $X \in In^T$ be an input dataflow for $\int$ and $\tilde{X} = X_{\mathbb{T}_s}$. The **execution of $\int$ on the input dataflow** X is the 3-tuple (X, Q, Y) where

- $Q \in S^{\mathbb{T}_s}$ is recursively defined by:
  - $Q(m^{\mathbb{T}_s}) = \mathcal{Q}\big(\tilde{X}(m^{\mathbb{T}_s}), q_0, m^{\mathbb{T}_s}\big)$
  - $\forall t \in \mathbb{T}_s,\ Q\big(succ^{\mathbb{T}_s}(t)\big) = \mathcal{Q}\big(\tilde{X}(succ^{\mathbb{T}_s}(t)), Q(t), succ^{\mathbb{T}_s}(t)\big)$
- $Y \in Out^{\mathbb{T}_s}$ is defined by:
  - $Y(m^{\mathbb{T}_s}) = \mathcal{F}\big(\tilde{X}(m^{\mathbb{T}_s}), q_0, m^{\mathbb{T}_s}\big)$
  - $\forall t \in \mathbb{T}_s,\ Y\big(succ^{\mathbb{T}_s}(t)\big) = \mathcal{F}\big(\tilde{X}(succ^{\mathbb{T}_s}(t)), Q(t), succ^{\mathbb{T}_s}(t)\big)$

Remark: inputs are read only at the moments of its time scale.
Subtility: the initial state of the system is computed using q_0.

Transfer functions are a semantics of systems execution

Definition (Transfer function)

A function $F : Input^T \rightarrow Output^{\mathbb{T}_s}$ is a **transfer function** of time scale $\mathbb{T}_s$ on signature $(Input, Output)$ if, and only if it is causal:

$$\forall X, Y \in Input^T,\ \forall t \in T,\ (X_{\mathbb{T}_s} \sim_t Y_{\mathbb{T}_s}) \Rightarrow (F(X) \sim_t F(Y))$$

Theorem (Transfer function of a system)

Let $\int$ be a system. The couple of dataflows (X,Y) resulting from all possible executions of $\int$ induce a **unique transfer function** $F_{\int}$.

Remark: In practice, transfer functions are extremely difficult to specify, since they are a function of dataflows themselves. But the correspondence between systems & transfer functions is key to prove the consistency of our work.

Example (Physical system)

Any Hamiltonian system can be modeled as a system in our framework. E.g. the water tank.

Example (Software system)

We define a Turing machine with inputs and outputs as a system.

Example (Human system)

We can model a human as a system, to define the meaningful states & behavior at high-level, so that they can be taken into account during the design (e.g. pilot: alive, asleep, dead).

Expressivity of our model

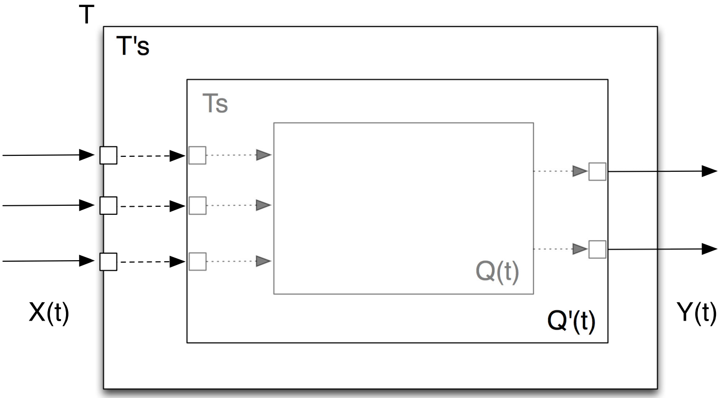
Our definition of **a system can model key real systems types** relevant in systems engineering: physical, software and human.

What is integration?

Building multiscale systems from a set of elementary systems by recursive application of composition and abstraction operators:

- *Composition* (divided in Product and Feedback) consists in aggregating systems together in an overall greater system where some inputs and outputs of the various systems have been interconnected.
- *Abstraction* allows to "zoom out" from a system to define a more abstract system that can itself be recursively integrated.

Representation of the extension of a system



Extension to a finer time scale

The extension operator makes it possible to define a finite number of systems on a shared time scale.

Definition (Extension of a system)

Let $\mathbb{T} \in Ts(T)$ be a time scale such that $\mathbb{T}_s \subseteq \mathbb{T}$. The **extension of $\int$ to $\mathbb{T}$** is the new system

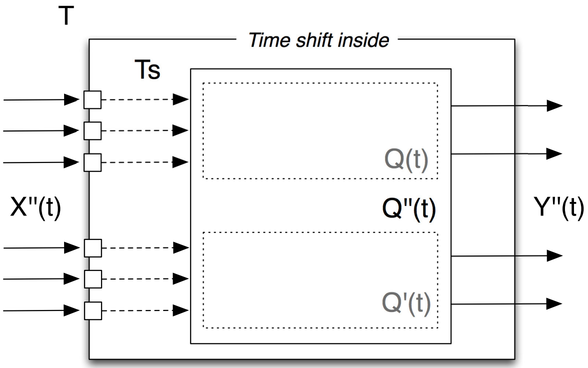
$$\int = (\mathbb{T}, Input, Output, S \times In \times Out, (q_0, \epsilon, \epsilon), \tilde{\mathcal{F}}_{\mathbb{T}}, \tilde{\mathcal{Q}}_{\mathbb{T}})^a$$

[a] $\tilde{\mathcal{F}}_{\mathbb{T}}$ and $\tilde{\mathcal{Q}}_{\mathbb{T}}$ are technical functions extending $\mathcal{F}$ and $\mathcal{Q}$ to finer time scales.

Theorem: Equivalence of a system by extension

Let $\int$ be a system and $\int'$ be its extension to a finer time scale. Then S and S' have **equivalent** transfer functions: $F_{\int} \sim F_{\int'}$.

Representation of the product of 2 systems



Remark: the product on datasets naturally induces the definition of **multiple inputs and outputs**.

Product

Definition (Product of systems)

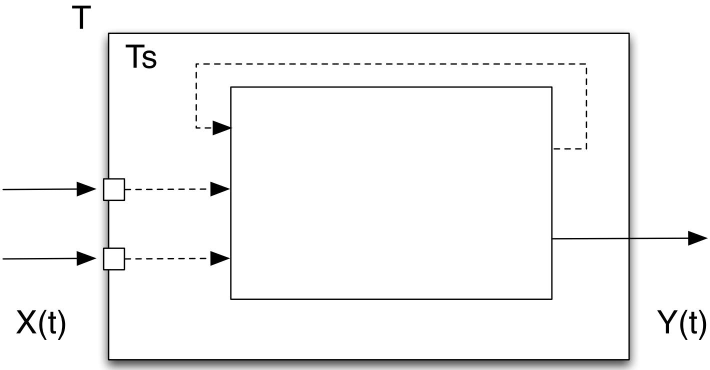
The **product** $\mathcal{S}_1 \otimes \cdots \otimes \mathcal{S}_n$ is the system

- $Input = Input_1 \otimes \cdots \otimes Input_n$ (and idem for Output)
- $S = S_1 \times \cdots \times S_n$ and $q_0 = ({q_0}_1, \ldots, {q_0}_n)$
- $\mathcal{F}((x_1, \ldots, x_n), (q_1, \ldots, q_n), t) = (\mathcal{F}_1(x_1, q_1, t), \ldots, \mathcal{F}_n(x_1, q_1, t))$
- $\mathcal{Q}((x_1, \ldots, x_n), (q_1, \ldots, q_n), t) = (\mathcal{Q}_1(x_1, q_1, t), \ldots, \mathcal{Q}_n(x_1, q_1, t))$

Theorem: Consistency of the product of systems

The transfer function of the product is **equivalent** to the usual product of the transfer functions: $F_{\int 1 \otimes \cdots \otimes \int_n} \sim F_{\int_1} \otimes \cdots \otimes F_{\int_n}$

Representation of a feedback



Feedback (constructive definition)

Definition (Feedback of a system)

When there is no instantaneous influence of dataset D from the input to the output, the **feedback of D in $\int$** is the system $\int_{fb(D)} = \big(\mathbb{T}_s, (In, \mathcal{I}'), (Out, \mathcal{O}'), S, q_0, \mathcal{F}', \mathcal{Q}'\big)$ where

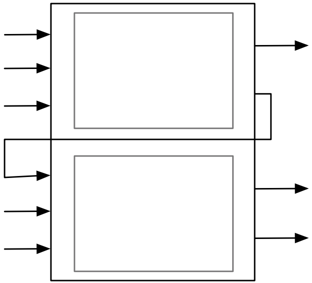
- we note $d_{x,q,t} = \mathcal{F}\big((\epsilon, x), q, t\big)_D$
- $\mathcal{I}'$ is the restriction of $\mathcal{I}$ to In, and $\mathcal{O}'$ of $\mathcal{O}$ to Out
- $\mathcal{F}'(x \in In, q \in S, t) = \mathcal{F}\big((d_{x,q,t}, x), q, t\big)_{Out}$
- $\mathcal{Q}'(x \in In, q \in S, t) = \mathcal{Q}\big((d_{x,q,t}, x), q, t\big)$

Theorem: Consistency of the feedback on systems

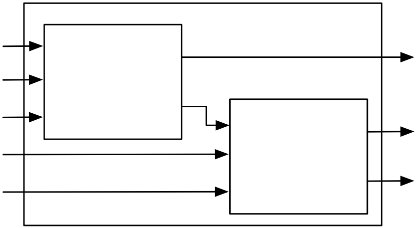
The transfer function of the feedback of a system **equals** the usual feedback of the transfer function of this system: $F_{\int_{fb(D)}} = fb_{(F_{\int}, D)}$

Sequential composition from product and feedback

Any **sequential composition of n systems** can be easily obtained from a finite sequence of product and feedback operators:



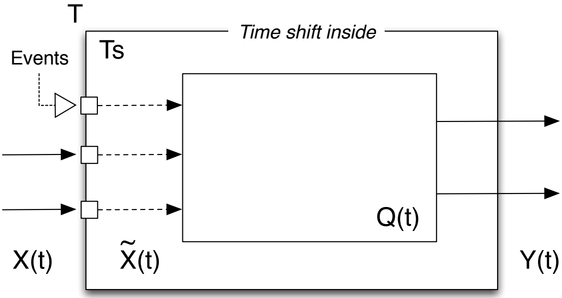
$\approx$



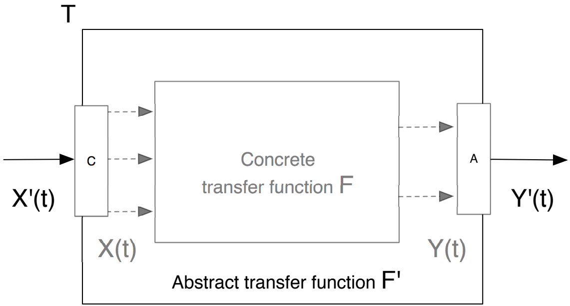
Modeling nondeterministic systems with an oracle

Example (Abstraction can bring nondeterminism to a model)

A glass with solidity $s \in \{0, \ldots, 100\}$, where s decrease at each impact i **"becomes" nondeterministic** (in reaction to i) when described as *broken* for $s = 0$ and *OK* for $s \in \{1, \ldots, 100\}$.



Representation of the abstraction operator



Remark: the abstraction is a **"zoom out"** of datasets (considering higher level datas for inputs, outputs and states, and eventually merging different dataflows), time (considering intervals of time instead of moments) and thus overall behavior.

Abstraction of a system

Definition (Abstraction of a system)

Let $\int = (\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, \mathcal{Q})$ be a system.
$\int' = (\mathbb{T}_a, Input_a \otimes \mathcal{E}, Output_a, S_a, q_a0, \mathcal{F}_a, \mathcal{Q}_a)$ is an **abstraction of** $\int$ for input and output abstractions (A_i, A_o) iif:

$$\exists A_q : S^{\mathbb{T}_s} \to {S_a}^{\mathbb{T}_a}, \text{ forall execution } (X, Q, Y) \text{ of } \int, \exists E \in \mathcal{E}^{\mathbb{T}_a}, \\ \big(A_i(X_{\mathbb{T}_s}) \otimes E, A_q(Q), A_o(Y)\big) \text{ is an execution of } \int'.$$

Conversely, $\int'$ is a concretization of the system $\int$.

Theorem: Consistency of the abstraction of a system

The transfer function of an abstraction of a system **equals** the corresponding abstraction of the transfer function of this system.

These operators are good to model systems integration!

- They **encompass key integration operators**: composition & abstraction
- It ensures a **consistent integration of heterogeneous systems**
- It makes it possible to **recursively integrate systems** since our definition of systems is closed under those operators.

Definition (Systemic signature)

A **systemic signature** is a 4-tuple $(X, Y, Q, \mathbb{T})$ where X, Y and Q are datasets (respectively called *input values*, *output values* and *states*) and $\mathbb{T}$ is a time scale.

Definition (Requirement)

A **requirement** on $(X, Y, Q, \mathbb{T})$, is a logical formula (e.g. using temporal logics) expressing properties on the behavior of any system of systemic signature $(X, Y, Q, \mathbb{T})$. The set of all possible requirements on this systemic signature is noted $Req(X, Y, Q, \mathbb{T})$.

Example (Expected property on the behavior of a system)

The system can be expected to be "alive", meaning here that a non blank input read at instant t must instantly result in a non blank output or a modification of the internal state.

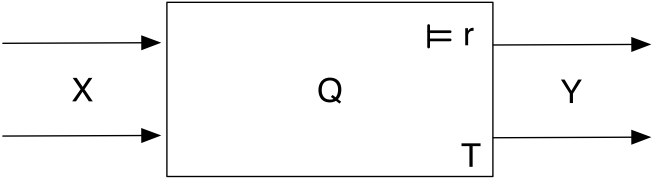
An underspecified system

Definition (Box)

A **box** is a 5-uplet $(X, Y, Q, \mathbb{T}, r)$ where:

- $(X, Y, Q, \mathbb{T})$ is a systemic signature
- $r \in Req(X, Y, Q, \mathbb{T})$

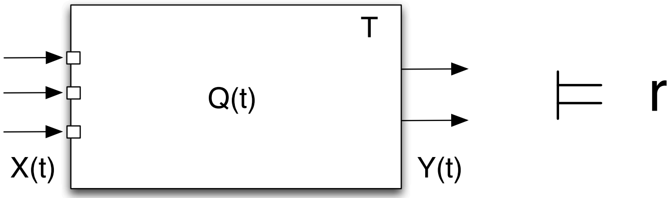
We note $BB(X, Y, Q, \mathbb{T})$ the set of boxes on $(X, Y, Q, \mathbb{T})$.



A box induces a set of corresponding systems

Definition (Realization of a box)

Let $B = (X, Y, Q, \mathbb{T}, r)$ be a box. A **realization** of B is any system S of systemic signature $(X, Y, Q, \mathbb{T})$ such that $S \vDash r$. When such a system exists, B is said to be *realizable*.



These are good definitions to handle underspecification!

We can now deal with underspecification:

- a systemic signature only **specifies the systemic variables**
- a box **specifies the variables & behavior constraints**
- a system is the **algorithmic specification** of a box.

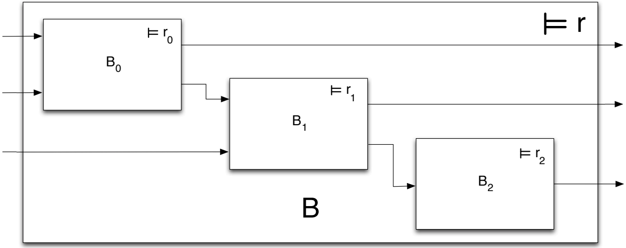
Views define nested boxes in a consistent way

Definition (View)

A **view** is a pair $\left(B, (B_0, \ldots, B_{n-1}, C)\right)$:

- B is a box
- $(B_0, \ldots, B_{n-1}, C)$ is a refinement of B.

A view can be realized by a *consistent* (n+1)-tuple of systems.



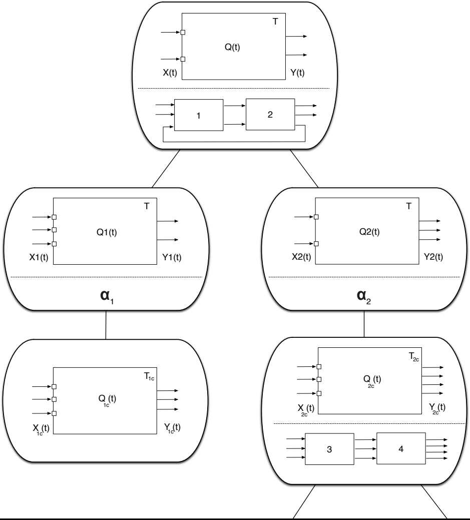
Multiscale systems are the realization of multiscale views

Definition (Multiscale system)

A **multiscale system** is a tree where:

- all leaves are labelled with a system
- internal nodes with an even depth are labelled with a pair (S, C), where S is a system and C is a composition plan
- internal nodes with an odd depth are labelled with a pair (S, α), where S is a system and α is an abstraction function
- for each even node (S, C) of children $(S_0, \_), \ldots, (S_{n-1}, \_)$; we have: $S = C(S_0, \ldots, S_{n-1})$
- for each odd node (S, α), its unique child $(S', \_)$ is such that: $S = \alpha(S')$.

Multiscale systems = systems with an internal structure!



Synthesis of our architecture framework

- Heterogeneous dataflows
  - **Time** = time reference + time scale
  - **Data** = dataset + data behavior
  - **Dataflow** = time + data
- Systems
  - **System** = functional behavior + state behavior + time scale
  - **Transfer function** = causal transformation of dataflows
  - **Execution of a system** = transfer function
- Integration of systems
  - **Composition** = extension + product + feedback
  - **Abstraction** = change of systemic level + nondeterminism
  - **Integration operators** = composition + abstraction
- Architecture
  - **Box** = signature + requirement
  - **View** = box + structure
  - **Multiscale system** = system + structure

Whole manuscript published in 4 articles

- Chapters 2, 3, 4: **A minimalist and unified semantics for heterogeneous integrated systems** in *Applied Mathematics and Computation (Elsevier), 2012*
- Chapter 5: **An adequate logic for heterogeneous systems** at the *18th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2013)*.
- Chapter 6: **A minimalist formal framework for systems architecting** at the *3rd International Workshop on Model Based Safety Assessment (IWMBSA'2013)*
- Chapter 7: **Infinite order Lorenz dominance for fair multiagent optimization** at the International Conference *Autonomous Agents and Multi-Agent Systems 2010*.

Some perspectives to continue this work

- Confronting this formalism to **real industrial cases**
- **Correctness-by-construction** (bottom-up preservation of properties)
- Formalizing the **link with synchronous languages** (e.g. Lustre, Simulink)
- Integrating **events** in our definition of system (e.g. Altarica).