A Unified Formalism for Complex Systems Architecture

Ph.D. in Computer Science

by

Boris Golden

Defended on May 13th 2013 in front of:

École Polytechnique - LIX

Acknowledgments

Thank you to **Daniel Krob**, who introduced me to the fascinating subject of systems architecture, from formal models to real industrial case studies!

Thank you to **Marc Aiguier**, who helped me make all this happen, and who has been supportive during the whole study!

Thank you to **Dan Frey** for inviting me to visit him at the Massachusetts Institute of Technology, opening new perspectives for my work!

Thank you to **Antoine Rauzy** who has always been providing stunning insights & feedbacks on my work!

Thank you to **Patrice Perny** for introducing me to the amazing world of multi-criteria optimization!

Thank you to **Yann Hourdel** with whom I have been working at LIX and sharing a lot at the end of my PhD!

Thank you to **Frédéric Boulanger**, **Sylvain Peyronnet** and **Marc Pouzet** for accepting to be the reporters of this PhD thesis!

Thank you to **Éric Goubault** for having accepted to be the President of the Jury for my defense!

Thank you to **my friends and family** who have provided so much support during my whole PhD!

And finally, *thank you* to my **Delphine** for her unconditional love & support!

Abstract

Complex industrial systems are typically artificial objects designed by men, involving a huge number of heterogeneous components (e.g. hardware, software, or human organizations) working together to perform a mission. In this thesis, we are interested in modeling the functional behavior of such systems, and their integration. We will model real systems as functional black boxes (with an internal state), whose structure and behaviors can be described by the recursive integration of heterogeneous smaller subsystems.

Our purpose is to give a unified and minimalist semantics for heterogeneous integrated systems and their integration. By “unified”, we mean that we propose a unified model of real systems that can describe the functional behavior of heterogeneous systems and that is closed under integration. By “minimalist” we mean that our formalization intends to provide a small number of concepts and operators to model the behaviors and the integration of complex industrial systems. Our work thus allows to give a relevant formal semantics to concepts and models typically used in Systems Engineering.

In this framework, the integration of real systems can be modeled as a recursive process consisting in alternating composition and abstraction, to build a target overall system from elementary systems recursively composed and abstracted at different levels. Based on these definitions of systems and their integration, a minimalist systems architecture framework allows to deal with requirements, systems structure and underspecification during the design.

In this thesis, we first define heterogeneous dataflows, and then systems as step-by-step machines transforming dataflows. Such systems can be integrated using operators, to build more complex systems, so that we can handle the two dimensions of the complexity (heterogeneity and integration). We then introduce a minimalist formalism for systems architecture to model requirements, underspecification & structure of systems through the design process. We finally open perspectives around systems optimization when fairness is required.

Keywords: Complex industrial systems, Systems modeling, Systems architecture, Systems semantics, Systems Engineering, Systems integration, Timed Mealy machine, Hybrid time, Non-standard analysis, Dataflows, Architecture framework, Transfer function, Requirements, Fair optimization.

Les systèmes industriels complexes sont des objets artificiels conçus par l'Homme, et constitués d'un grand nombre de composants hétérogènes (e.g. matériels, logiciels ou organisationnels) collaborant pour accomplir une mission globale. Dans cette thèse, nous nous intéressons à la modélisation du comportement fonctionnel de tels systèmes, ainsi qu'à leur intégration. Nous modéliserons donc les systèmes réels par le biais d'une approche de boîte noire fonctionnelle avec un état interne, dont la structure et le comportement fonctionnel peuvent être obtenus par l'intégration récursive de composants élémentaires hétérogènes.

Notre but est de donner une sémantique unifiée et minimaliste pour de tels systèmes, ainsi que pour leur intégration. Par "unifié", nous voulons dire que nous proposons un modèle permettant de représenter différents types de systèmes et qui reste clos sous les opérateurs d'intégration. Par "minimaliste", nous voulons dire que notre formalisme entend définir un petit nombre de concepts et d'opérateurs pour modéliser l'intégration des systèmes industriels complexes. Notre travail permet donc de donner une sémantique aux concepts et modèles typiquement utilisés en Ingénierie système.

Dans ce cadre, l'intégration de systèmes réels peut être modélisée comme un processus récursif consistant en une alternance de composition et d'abstraction, pour construire un système cible à partir de systèmes plus élémentaires récursivement connectés entre eux puis redéfinis à des niveaux supérieurs d'abstraction. En s'appuyant sur ces définitions, nous proposons un cadre d'architecture minimaliste qui permet d'exprimer des exigences, de décrire la structure d'un système et de rendre compte de la sous-spécification durant la phase de conception.

Dans cette thèse, nous définissions tout d'abord la notion de flots de données hétérogènes, puis de systèmes comme machines de transformation de flots de données de façon algorithmique. Ces systèmes peuvent ensuite être intégrés par le biais d'opérateurs, permettant de construire des systèmes plus complexes, et donc de rendre compte des deux dimensions principales de la complexité (l'hétérogénéité et l'intégration). Nous introduisons ensuite un formalisme pour l'architecture système, en modélisant les exigences, la sous-spécification et la structure des systèmes. Enfin, nous ouvrons des perspectives autour de l'optimisation entre systèmes lorsque l'équité est recherchée.

Mots-clés : Systèmes industriels complexes, Modélisation de système, Architecture système, Sémantique des systèmes, Ingénierie système, Intégration de systèmes, Machine de Mealy temporisée, Temps hybride, Analyse non-standard, Flot de données, Cadre d'architecture, Fonction de transfert, Exigences, Optimisation équitable.

Contents

Chapter 1

Introduction

1.1 Complex industrial systems

Industrial systems are typically artificial objects designed by men, involving heterogeneous components (mostly: hardware, software, and human organizations) working together to perform a mission. In this thesis, we are interested in modeling the functional behavior of such systems, and their integration. From a practical point of view, our aim is to give a unified formal semantics to the concepts manipulated on a daily basis by engineers from various fields working together on the design of complex industrial systems. Indeed, they need formal tools to reason on & model those systems in a unified & consistent way, with a clear understanding of the underlying concepts.

We will model complex industrial systems as heterogeneous integrated systems, since our work highlights two aspects of the complexity of such systems:

- the *heterogeneity* of systems, that can be naturally modeled following continuous or discrete time, and that are exchanging data of different types[1]. Several specialized fields are involved in the design of a complex industrial system, making it difficult to keep a unified vision of this system and to manage its design.
- the *integration* of systems, i.e. the recursive mechanism to build a system through the synthesis of smaller systems working together, and whose behaviors will be described at a more concrete level (i.e. a finer grain). There are many interrelations between a possibly huge number of components, and there are recursive levels of integration.

The concept of *complex systems* has led to various definitions in numerous disciplines (biology, physics, engineering, mathematics, computer science, etc).

[1]Data encompasses here all kinds of elements that can be exchanged between real objects. We distinguish three kinds of homogeneous systems: hardware/physical systems (transforming continuous physical parameters), software systems (transforming and managing discrete data), and human/organizational systems (organized through processes).

One speaks for instance of dynamical, mechanical, Hamiltonian, hybrid, embedded, concurrent or distributed systems (cf. [6, 8, 45, 53, 60]). A minimalist informal definition consistent with (almost) all those of the literature is that a *system* is "a set of interconnected parts forming an integrated whole", and the adjective *complex* implies that a system has "properties that are not easily understandable from the properties of its parts". In the mathematical formalization of *complex systems*, there are today two major approaches: the first one is centered on understanding how very simple but numerous elementary components can lead to complex overall behaviors (e.g. cellular automatas), the second one (that will also be ours) is centered on giving a precise semantics to the notion of system and to the integration of systems to build greater overall systems.

When mathematically apprehended, the concept of system (in the sense of this second approach) is classically defined with models coming from:

- control theory and physics, that deal with systems as partial functions (dynamical systems may also be rewritten in this way), called transfer functions, of the form:

  $$\forall t \in T,\ y(t) = F(x, q, t)$$

  where x, q and y are inputs, states and outputs dataflows, and where T stands for time (usually considered in these approaches as continuous (see [60, 5, 21]).

- theoretical computer sciences and software engineering, with systems that can be depicted by automaton-oriented formalisms equivalent to timed transition systems with input and output, evolving on discrete times generally considered as a universal predefined sequence of steps (see for instance [37, 10, 35, 26, 19, 18]). There is also a purely logical approach for representing discrete abstract systems. The core modelling language in this direction is probably Lustre [35, 20]. A reasoned overview of all these approaches can be found in [39].

However all these models do not easily allow to handle systems with heterogeneous time scales. The introduction of a more evolved notion of time within models involves many difficulties, mainly the proper definition of sequential transitions or the synchronization of different systems exchanging dataflows without synchronization of their time scales. Dealing with an advanced definition of time will typically imply to introduce infinity and infinitesimal (for instance with non-standard real numbers).

To address this challenge, the theory of hybrid systems was developed jointly in control theory (see [60, 66]) and in computer science (see [36, 6, 41]). A serious issue with this theory is however that the underlying formalism has some troubling properties such as the Zeno's paradox which corresponds to the fact that an hybrid system can change of state an infinite number of times within a finite time with the convergence of a series of decreasing durations which should be avoided in a robust modeling approach. Other interesting and slightly different attempts in the same direction can also be found in Rabinovitch and Trakhtenbrot (see [52, 61]) who tried to reconstruct a finite automata theory on the basis of a real time framework, or in [67]. Hybrid automatas (see [36]) are another classical model for representing abstract hybrid systems.

Moreover, none of these models address the integrative & architectural dimensions of complex industrial systems (an approach similar to ours on the structure of complex systems, but without the introduction of heterogeneous time, has been carried out in [3], using a coalgebraic formalism). There is therefore a great challenge on being able to unify in a same formal framework mathematical methods dealing with the definition & design of both continuous and discrete systems, and at the same time being able to define the integration & architecture of such systems in the same formalism. This will be at the center of our approach.

1.2 Systems Engineering

When dealing with complex industrial systems containing heterogeneous components, engineers face problems with the semantics of the models they work with to describe real systems when they involve a large number of heterogeneous elementary systems.

To build modern industrial systems, it has thus been necessary to create complex engineering models & processes being able to deal with a huge number of engineers coming from many different domains. Indeed, because of the size of such modern industrial systems (trains, planes, space shuttles, etc), their realization leads to major conceptual and technical difficulties. The reason is that it is difficult, or even impossible, for one single person to completely comprehend such systems globally. Although this designation is not precise nor universal, it is naturally associated with industrial systems whose design, industrialization and change lead to important and difficult problems of integration, directly related to both the huge number of basic components integrated at multiple levels, and the important scientific and technological heterogeneity of such systems (generally involving software, hardware/physical and human/organizational parts).
To manage the complexity of such systems, some methods have emerged during the last century. We can trace the origin of these methods to the Cold War where USA had to build defence systems for which the issues of data management, decisions and military riposte had been taken into account as a whole, in an integrated and consistent way, in order to ensure a short reaction time between a Soviet attack and an American counter-attack. From there, a body of knowledge called *Systems Engineering*, focused on the integration mastery of large industrial systems has progressively emerged since the 50's. Hence, Systems Engineering consists of a set of concepts, methods and good organizational and technical practices that the industry had to develop to be able to deal with the complexity of industrial systems (see [11, 42, 57, 62] for more details on this subject).

But in fact, Systems Engineering is "just" the application to engineering of a more general thought paradigm, called *systems approach* (also often referred to as *systems thinking*). Systems approach focuses on interactions between systems, and views such systems as black boxes described only through their functional behavior and their internal state. In systems approach, a system is thus a black box receiving and emitting flows, and characterized by an internal state. A system can itself be decomposed into a set of interconnected subsystems. It is therefore an observational modeling of systems. Systems approach also implies to step back with a high-level point of view seeing any system as being part of an overall greater system (i.e. a system is in interaction with other systems in its environment).

A key assumption of systems approach is thus to consider that any system is only in interaction with other systems, and that the behavior of any real system[2] can be explained within this framework. The main advantage of this approach is that it helps to understand how things influence one another within a whole (with often unexpected long-term or long-distance influences). All interactions between systems are captured by logical flows, figuring a unidirectional transmission of elements (which can be material, energetic or informational). "Logical" here means that it only models the exchange of data between elements, and not the way this exchange really occurs. For instance, the physical reality behind a flow between two real systems (like the delay or the rate of transmission of the network cable between two computers) will itself be modeled by a specific type of system called *interface* (it would here take into account the physical properties of the cable, when the flow would only account for the logical exchange of data).

The systems approach is a very powerful tool to model a lot of real-life objects and situations. We give a simple example to illustrate the main concepts of systems approach. Imagine an individual, John, sitting in front of its computer and using it. John has on its table a bottle of water and a glass he uses when he is thirsty. We want to model the overall real system[3] (which is here closed, i.e. not exchanging data with its environment). We should characterize the following objects at the right abstraction level:

- *systems*: those are all the objects involved. Here: glass, bottle, computer, John (the table will not be useful in our modeling).

- *states*: each system has a set of possible states. E.g. the glass can be 'not_empty' or 'empty'.

[2]In the literature, the real object and its model are often confused and both called *system*. When clarification is needed, we will call *real system* any object of the real world which behavior we want to explain as a transformation of flows of data. We will call *system* the mathematical object introduced to model real systems.

[3]As every model, it is not exhaustive since focused on given aspects of the real system considered to meet the (implicit) goals of the modeling.

- *data*: each system can receive or send data. E.g. the glass can send water, or a visual stimulus (indicating the level of water in the glass).
- *flows*: each system has inputs and outputs that allow it to exchange data with other systems. E.g. the bottle can send water to the glass (when the bottle is not empty). But the glass may send water to the computer, even if it is not its intended use.
- *behaviors* (functions and states): each system will have specific behaviors that will make it generate outputs and change its states, according to time and inputs. E.g. when the glass is empty and receives water, it becomes not_empty. But if it receives a drinking move, it sends water to John and becomes empty.
- *time* of the overall system: a time scale of description has to be chosen to be consistent with the model. E.g. a step of 1 minute for the time scale can be a consistent choice here.

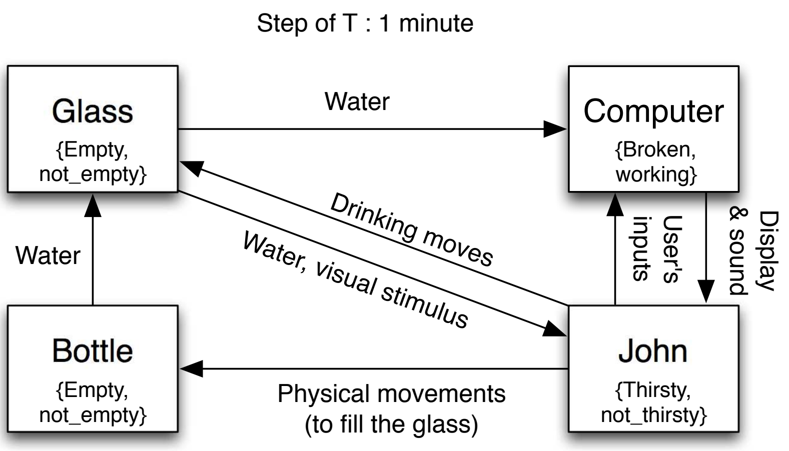


Figure 1.1: Example of an elementary systemic modeling

All the concepts introduced here are informal. We will give a semantics to all the objects of this intuitive "graphical" language used in systems approach.

1.3 What is systems architecture?

1.3.1 A definition

Systems Architecture is a generic discipline to handle systems (existing or to be created), in a way that supports reasoning about the structural properties of these objects.

Depending on the context, Systems Architecture can in fact refer to:

- the architecture of a system, i.e. a *model* to describe/analyze a system
- architecting a system, i.e. a *method* to define the architecture of a system
- a body of knowledge for "architecting" systems while meeting business needs, i.e. a *discipline* to master systems design.

At this point, we can only say that the "architecture of a system" is (similarly to the one of a building) a global model of a real system consisting of:

- a structure
- properties (of various elements involved)
- relationships (between various elements)
- behaviors & dynamics
- multiple views of elements (complementary and consistent).

We will not describe here the numerous issues raised (at every level of a company: corporate strategy, marketing, product definition, engineering, manufacturing, operations, support, maintenance, etc) by the design and management of complex industrial systems. But these issues can be summarized as:

- going from local to global, i.e. mastering integration and emergence
- building an invariable architecture in a moving environment.

In this context, Systems Architecture is a response to the conceptual and practical difficulties of the description and the design of complex industrial systems. Systems Architecture helps to describe consistently and design efficiently complex systems such as:

- an industrial system (the original meaning of Systems Architecture)
- an IT infrastructure (Enterprise Architecture)
- an organization (Organizational Architecture)
- a business (Business Architecture).

Systems Architecture will often rely on a tool called an *architecture framework*, i.e. a reference model to organize the various elements of the architecture of a system into complementary and consistent predefined views allowing to cover all the scope of Systems Architecture. Famous architecture frameworks are for instance: DoDAF, MoDAF or AGATE[4].

Finally, Systems Architecture will consider any system with a socio-technical approach (even when dealing with a "purely" technical system). In particular, during the design (or transformation) of a system, the systems in the scope of this design (or transformation) can be divided in two separated systems in interaction:

[4]A good overview of these frameworks can be found on Wikipedia:
http://en.wikipedia.org/wiki/Enterprise_Architecture_framework

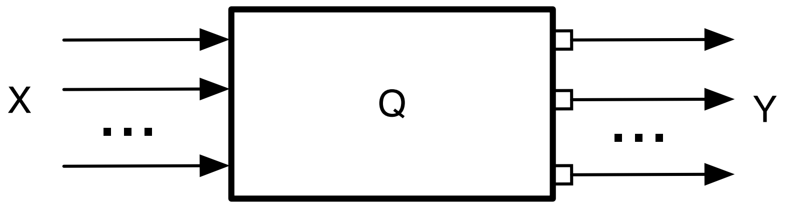
- the *product*, i.e. the system being designed or transformed
- the *project*, i.e. the socio-technical system (teams, tools, other resources and their organization following strategies & methods) in charge of the design or transformation of the product.

1.3.2 Fundamental principles

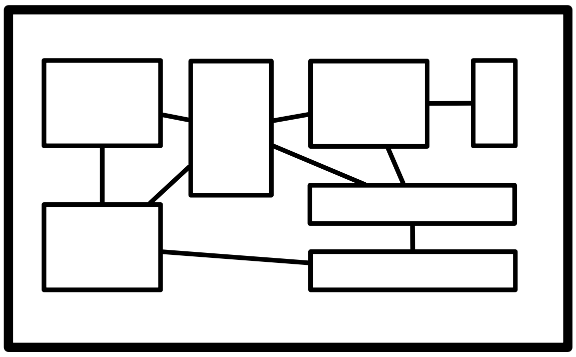
Whatever the type of system and the acception considered (model, method or discipline), Systems Architecture is based on 9 fundamental principles:

Thinking with a systemic approach

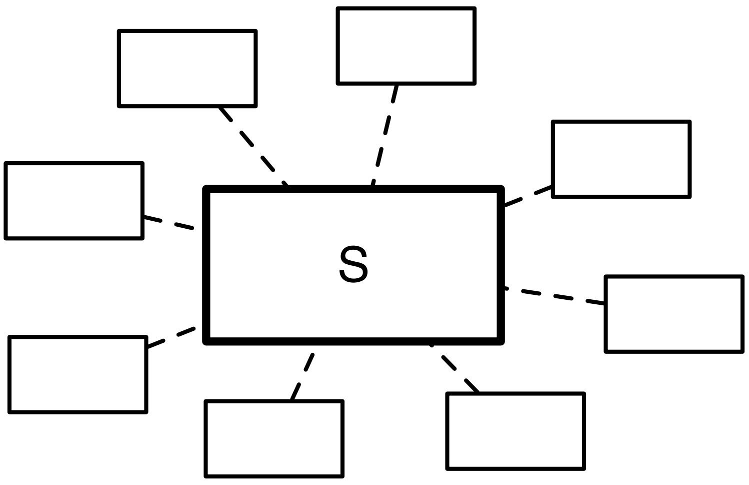
1. the objects of the reality are modeled as systems (i.e. a box performing a function and defined by its perimeter, inputs, outputs and an internal state). *Ex: a mobile phone is a system which takes in input a voice & keystrokes and outputs voices & displays. Moreover, it can be on, off or in standby. Overall, the phone allows to make phone calls (among other functions).*



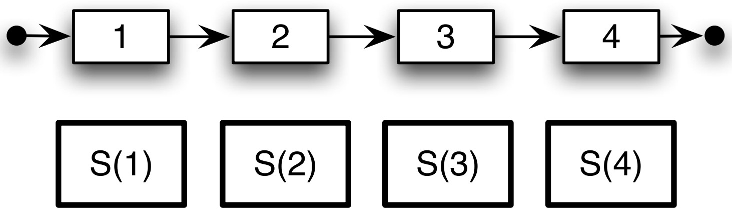
2. a system can be broken down into a set of smaller subsystems, which is less than the whole system (because of emergence). *Ex: a mobile phone is in fact a screen, a keyboard, a body, a microphone, a speaker, and electronics. But the phone is the integration of all those elements and cannot be understood completely from this set of separate elements.*



3. a system must be considered in interaction with other systems, i.e. its environment. *Ex: a mobile phone is in interaction with users, relays (to transmit the signal), repairers (when broken), the ground (when falling), etc. All these systems constitute its environment and shall be considered during its design.*

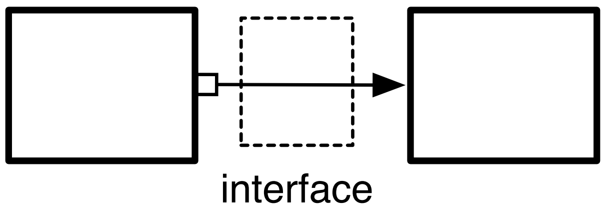


4. a system must be considered through its whole lifecycle. *Ex: a mobile phone will be designed, prototyped, tested, approved, manufactured, distributed, sold, used, repaired, and finally eventually recycled. All these steps are important (and not only the moment when it is used).*

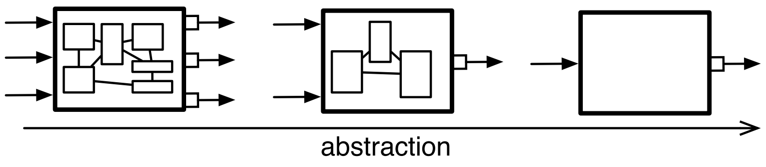


Reasoning according to an architecture paradigm

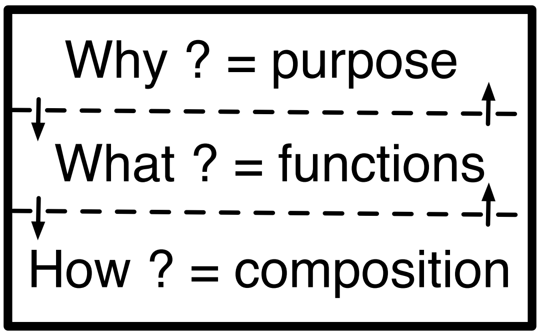
5. a system can be linked to another through an interface, which will model (when needed) the properties of the way they are linked in the reality. *Ex: when phoning, our ear is in direct contact with the phone, and there is therefore a link between the two systems (the ear and the phone). However, there is a hidden interface: the air! The properties of the air may influence the link between the ear and the phone (imagine for example if there is a lot of outside noise).*



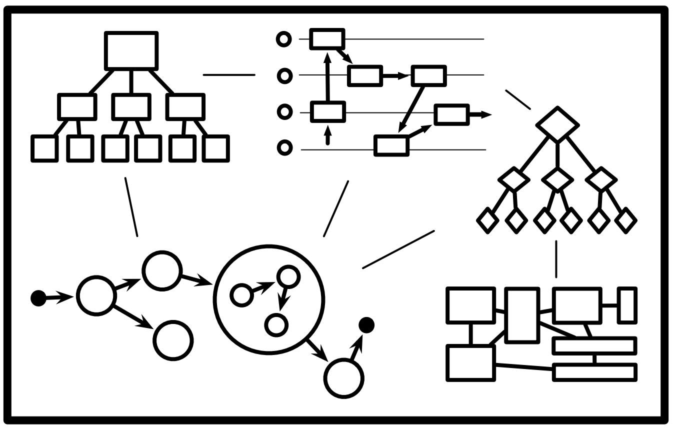
6. a system can be considered at various abstraction levels, allowing to consider only relevant properties and behaviors. *Ex: do you consider your phone as a device to make phone calls (and other functions of modern phones), a set of material and electronics components manufactured together, or a huge set of atoms ? All these visions are realistic, but they are just at different abstraction levels, whose relevancy will depend on the purpose of the modeling.*



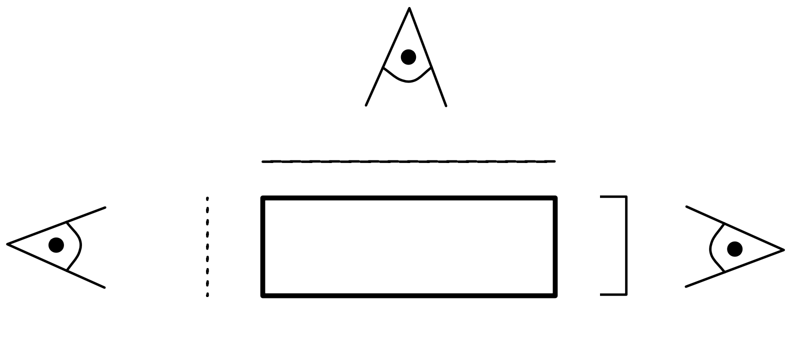
7. a system can be viewed according to several layers (typically three at least: its purpose, its functions, and its construction). *Ex: a phone is an object whose purpose is to accomplish several missions for its environment: making phone calls, being a fashionable object, offering various features of personal digital assistants, etc. But it is also a set of functions organized to accomplish these missions (displaying on the screen, transmitting signal, delivering power supply, looking for user inputs, making noise if necessary, etc). And finally, all these functions are implemented through physical components organized to perform these functions.*



8. a system can be described through interrelated models with given semantics (properties, structure, states, behaviors, datas, etc). *Ex: from the point of view of properties, the phone is a device expected to meet requirements like "a phone must resist to falls from a height of one meter". But a phone will also change state: when a phone is off and that the power button is pressed, the phone shall turn on. Function dynamics of the phone are also relevant: when receiving a call, the screen will display the name and the speaker will buzz, but if the user presses no button the phone will stop after 30 seconds... This will typically be described with diagrams in modeling languages like UML or SysML.*



9. a system can be described through different viewpoints corresponding to various actors concerned by the system. *Ex: marketers, designers, engineers (in charge of software, electronics, acoustics, materials, etc), users, sales, repairers... All these people will have different visions of the phone. When the designer will see the phone as an easy-to-use object centered on the user, the engineer will see it as a technological device which has to be efficient and robust. A marketer may rather see it as a product which must meet clients' needs and market trends to be sold. All these visions are important and define the system in multiple and complementary ways.*



1.4 Towards a unified formalism

Various models & methods have been developed during the last decades to help modeling and designing specific types of systems. The idea introduced by Daniel Krob in [12] is that all those approaches share strong similarities at a certain level of abstraction, and could thus be formalized in a unified framework helping to deal with both the "big picture" and the more vertical models of systems. It also means synthesizing the "fundamental" characteristics of what is called *systems architecting*, i.e. the application of the systems approach to the design of complex industrial systems.

From a practical point of view, our aim is to give a unified formal semantics to the concepts manipulated on a daily basis by engineers from various fields working together on the design of complex industrial systems. Indeed, they need formal tools to reason on & model those systems in a unified & consistent way, with a clear understanding of the underlying concepts[5].

Of course, we do not intend to replace existing frameworks dedicated to specific systems, as such frameworks are much more accurate when dealing with the design of homogeneous systems. Our approach has strong benefits when dealing with heterogeneous systems at the right level of integration. Our aim is to give a unified formal semantics to the concepts manipulated on a daily basis by engineers from various fields working together on the design of complex industrial systems. Indeed, they need formal tools to reason on & model those systems in a unified & consistent way, with a clear understanding of the underlying concepts.

The purpose of the present work is thus to contribute to a unified formal framework for complex systems modeling & architecture. We will model the observational behavior of any real system through a functional machine processing dataflows (for related work on dataflow networks, see [37, 19, 18, 17]) in a way that can be encoded by timed transitions for changing states and outputs in instantaneous reaction to the inputs (comparable with timed Mealy machines [46]). We show that our formalization makes it possible to model all kinds of real systems (physical, software and human/organizational), which is necessary in Systems Engineering.

An underlying assumption of our approach is that each system has its own rhythm, and that this rhythm cannot be changed by an interaction with another system, nor cause sampling problems when two systems of different time scales are integrated together. This means that each system somehow has a set of characteristic predefined moments of transitions that are generally based on its internal mechanisms seen at a certain level of abstraction.

Overall, our approach towards systems modeling & architecture is simple:

- we use a unified framework where continuous & discrete times are handled

[5]Note thus that our purpose is not to define executable models, or an actionable formalism with a dedicated language that can be used on the field by engineers. It clearly differentiates our approach from many existing languages like Lustre[35], Simulink[68] or Altarica[7].

in a unified way

- we separate the behavior of systems that can be observed (outputs and states) and their structure (how a system is built from elementary components)
- we view all behaviors as "algorithmic". We define and model all objects so that a system behavior can be explained as a step-by-step transformation of dataflows
- we model systems structure following a "Lego paradigm". We explain the architecture of systems through the integration of smaller building blocks, themselves, modeled as systems
- we consider that only three actions are possible during the design process: abstracting a system, composing together a set of systems, and verifying if a system behavior respects a set of requirements[6].

We generalize and extend the approach of the previous work in [12] (where a unified model for continuous and discrete systems was defined by using non-standard infinitesimal and finite time steps) by dealing with time, data, and synchronization axiomatically, and by introducing integration operators that were introduced in [30] for discrete systems only. Our purpose is to give a unified and minimalist semantics for heterogeneous integrated systems and their integration. By "unified", we mean that we propose a unified model of real systems that can describe the functional behavior of heterogeneous systems and that is closed under integration. By "minimalist" we mean that our formalization intends to provide a small number of concepts and operators to model the behaviors and the integration of complex industrial systems. Our work thus allows to give a relevant formal semantics to concepts and models typically used in Systems Engineering, where semi-formal modeling is well-spread.

To build new systems from existing systems, we will formalize two operators that play a crucial role when modeling or designing real systems:

1. *Composition* operators, and
2. *Abstraction* operator.

Composition operators consist in building a larger system by aggregating together smaller systems and connecting together some of the inputs and outputs of those systems. As to the abstraction, it allows to define from a composition of systems a more abstract system that will itself be integrated in more global ones. In fact, abstraction aims at structuring systems at many levels of description,

[6]Requirements are used in Systems Engineering to define expected properties of systems, especially regarding their behavior. They are thus logical properties on a system.

from the most concrete to the most abstract one. The purpose of the abstraction operator is also to make easier the description of systems at more abstract levels.

In this framework, *integration* of real systems can be modeled as a building a target multiscale system from elementary systems recursively composed and abstracted at different levels.

Based on these definitions of systems and their integration, we are able to define a minimalist systems architecture framework to deal with requirements, systems structure, and underspecification during the design process.

1.5 Structure of this manuscript

For a better understanding, this manuscript should be better read chapter after chapter. Hence, we progressively build our framework. First, we define heterogeneous dataflows, and then systems as step-by-step machines transforming dataflows. Such systems can be integrated using operators, to build more complex systems, so that we can handle the two dimensions of the complexity (heterogeneity and integration). We then introduce a minimalist formalism for systems architecture to model requirements, underspecification & structure of systems through the design process. We finally open perspectives around systems optimization when fairness is required.

Chapter 2, *Heterogeneous dataflows*, introduces formal definitions of time, data and dataflows. Our unified definition of time allows to deal uniformly with both continuous and discrete times, while our definition of data allows to handle heterogeneous data having specific behaviors. This makes it possible to define heterogeneous dataflows with generic synchronization mechanisms allowing to mix dataflows together. The deliverable is a unified and well-formalized definition of heterogeneous dataflows with properties that will be later needed to define & integrate systems.

Chapter 3, *Systems*, defines a system as a mathematical object characterized by coupled functional and states behaviors upon a time scale. This is a definition modeling a real system as a black box with observable functional behavior and an internal state (similarly to a timed Mealy machine). This definition is expressive enough to capture, at some level of abstraction, the functional behavior of any real industrial system with sequential transitions. We also express the functional behavior of systems via transfer functions transforming dataflows and show the equivalence. The deliverable is a unified definition of a system (viewed as a functional black box) and a proof of its equivalence with transfer functions.

Chapter 4, *Integration operators*, provides formal operators to integrate such systems. Those operators make it possible to compose systems together (i.e. interconnecting inputs and outputs of various systems) and to abstract a system (i.e. change the level of description of a system in term of granularity of all dataflows). We show that these operators are consistent with the natural definitions of such operators on transfer functions. The deliverable is a set of integration operators that are proven to be consistent and whose expressivity allows to model systems integration. Chapters 2, 3 & 4 have been published as a journal article *Complex Systems Modeling II: A minimalist and unified semantics for heterogeneous integrated systems* [30] in *Applied Mathematics and Computation (Elsevier), 2012.*

Chapter 5, *A logic for requirements*, provides a minimalist logic to express requirements on systems. We first introduce an equivalent definition of systems using coalgebraic models. Based on these models, we define logical requirements to express properties on the observable behavior of systems. This chapter has been published as an article: *An adequate logic for heterogeneous systems* [4] at the *18th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2013).*

Chapter 6, *Towards a framework for systems architecture*, defines a formal framework to deal with heterogeneous integrated systems during the systems design process. Two main problems are addressed: how to deal with the underspecification of systems during their design process, and how to formalize the structure of a system. We consider a minimalist design process, consisting of requirements analysis and systemic recursion. We introduce the notion of views that allow to formalize the set of interrelated models used in practice to describe a more or less specified system at any step of the design process. We then introduce formal definitions of the internal structure of a system. The deliverable is a minimalist formal framework for systems architecture along the design process, with a generic framework to model underspecification and the structure of systems. This chapter has been accepted and presented at the *3rd International Workshop on Model Based Safety Assessment (IWMBSA'2013)* under the title *A minimalist formal framework for systems architecting* [31].

Chapter 7, *Fair assignments between systems*, finally explores the combinatorial optimization between subsystems, when fairness is required (for example to spread risk or cost between various units that should be assigned functions). Optimization in systems architecture is key to take design decisions during the process. Our core idea is to iterate a classical partial fairness decision criteria. The deliverable is a new, original model for fair multi-agent optimization that can be relevant to optimize design decisions during a systems design process. This chapter intends to open new perspectives and is fairly independent from the rest of the manuscript. It has been published as a paper *Infinite order Lorenz dominance for fair multiagent optimization* [32] at the International Conference *Autonomous Agents and Multi-Agent Systems 2010.*

Part I

Systems modeling

Chapter 2

Heterogeneous dataflows

We introduce formal definitions of time, data and dataflows. Our unified definition of time allows to deal uniformly with both continuous and discrete times, while our definition of data allows to handle heterogeneous data having specific behaviors. This makes it possible to define heterogeneous dataflows with generic synchronization mechanisms allowing to mix dataflows together. The deliverable is a unified and well-formalized definition of heterogeneous dataflows with properties that will be later needed to define & integrate systems.

2.1 Time

Most of the challenges raised by a unified definition of complex (industrial) systems are coming from time. Indeed, real systems are naturally defined according to various times, that can typically be discrete or continuous. We must therefore be able to define:

- a unified model of time encompassing continuous and discrete times to later introduce a unified definition of heterogeneous systems,
- the mixture of various time scales to integrate such systems.

Unifying both discrete and continuous times is a complicated issue (see [14] for an exhaustive survey on the subject). To reach this purpose, we propose to extend & axiomatize the approach developed in [12] where discrete and continuous times have been unified homogeneously by using techniques of non-standard analysis [48, 54, 27]. We introduce a more generic approach and deal with time axiomatically, that is by expressing the minimal properties that both time references and time scales have to satisfy. That allows to consider in a same uniform framework many different times: usual ones such as $\mathbb{N}$ and $\mathbb{R}$, or more specific ones such as the non-standard real numbers $^*\mathbb{R}$, or the VHDL time (see below).

2.1.1 Time reference

A time reference is a universal time in which all systems will be defined. It captures the intuition we have of time: a linear quantity composed of ordered moments, pairs of which define durations. Such a modeling of "time" will be common to all the systems we want to integrate together.

Definition 2.1.1 (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^+:*
  - $\forall a, b \in T^+,\ a +^T b \in T^+$ *closure (Δ_1)*
  - $\forall a, b \in T^+,\ a +^T b = 0^T \Longrightarrow a = 0^T \wedge b = 0^T$ *initiality (Δ_2)*
  - $\forall a \in T^+,\ 0^T +^T a = a$ *left neutrality (Δ_3)*
- *upon T:*
  - $\forall a, b, c \in T,\ a +^T (b +^T c) = (a +^T b) +^T c$ *associativity (Δ_4)*
  - $\forall a \in T,\ a +^T 0^T = a$ *right neutrality (Δ_5)*
  - $\forall a, b, c \in T,\ a +^T b = a +^T c \Longrightarrow b = c$ *left cancellation (Δ_6)*
  - $\forall a, b \in T,\ \exists c \in T^+,\ (a +^T c = b) \vee (b +^T c = a)$ *linearity (Δ_7)*

Elements of T are *moments* whilst elements of T^+ are *durations* (i.e. distances between moments). Any duration can be considered as a moment, by setting a conventional origin.

The properties given upon T and T^+ are constraints that catch the intuitive view that the time elapses linearly by adding successively durations between them.

Proposition 1 (Total order on a time reference) *We can define a total order $\preceq^T$ (later written $\preceq$ for convenience) on T as follows:*

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

Proof This is a classical result using Δ_2, Δ_4, Δ_5, Δ_6 and Δ_7 (time references are similar to specific semigroups, cf [24]).

$\preceq^T$ is reflexive by Δ_5, transitive by Δ_4, total by Δ_7. To be a total order, it should moreover be antisymmetric. Suppose that $a \preceq^T b$ and $b \preceq^T a$. Thus, $\exists c, d \in T^+$ such that $b = a +^T c$ and $a = b +^T d$. But then: $a = b +^T d = (a +^T c) +^T d = a +^T (c +^T d)$ by associativity Δ_4. But by Δ_5, $a +^T 0 = a = a +^T (c +^T d)$, and so by Δ_6, $c +^T d = 0^T$. As $c, d \in T^+$, we have by Δ_2 that $c = d = 0^T$ and finally by Δ_5 again $a = b$. $\preceq^T$ is antisymmetric and is therefore a total order. □

Moreover, we can remark that Δ_1 ensures that any element of T greater than an element of T^+ will be in T^+, and Δ_3 ensures that 0^T is the minimum of T^+, so that the set of durations has natural properties according to $\preceq^T$ and can be understood as "positive" elements of T.

Example 1 In [12], the time reference is the set of non-standard real numbers $^*\mathbb{R}$ defined as the quotient of real numbers $\mathbb{R}$ under the equivalence relation $\equiv \subseteq \mathbb{R}^{\mathbb{N}} \times \mathbb{R}^{\mathbb{N}}$ defined by:

$$(a_n)_{n\geq 0} \equiv (b_n)_{n\geq 0} \Longleftrightarrow m(\{n \in \mathbb{N} | a_n = b_n\}) = 1$$

where m is an additive measure that separates between each subset of $\mathbb{N}$ and its complement, one and only one of these two sets being always of measure 1, and such that finite subsets are always of measure 0. The obvious zero element of $^*\mathbb{R}$ is $(0)_{n\geq 0}$, $^*\mathbb{R}^+$ is its positive part taken here as durations, and the internal law $+$ is defined as the usual addition on $\mathbb{R}^{\mathbb{N}}$, i.e.:

$$(a_n)_{n\geq 0} + (b_n)_{n\geq 0} = (a_n + b_n)_{n\geq 0}$$

$^*\mathbb{R}$ satisfies all the conditions of Definition 2.1.1 and is a well-defined time reference. Observe also that $^*\mathbb{R}$ has as subset, the set of non-standard integers $^*\mathbb{Z}$ (and subsequently $^*\mathbb{N}$) where infinite numbers are all numbers having absolute value greater that any $n \in \mathbb{N}$. $\Diamond$

Some authors, e.g. [38], add commutativity and Archimedean properties in the definition of a time reference. Commutativity is intuitive and the Archimedean property excludes Zeno's paradox. However, they are not always satisfied by standard models of time, as in the VHDL time used in some programming languages.

Example 2 The VHDL time [13] $\mathcal{V}$ is given by a pair of natural numbers (both sets of moments and durations are similar): the first number denotes the "real" time, the second number denotes the step number in the sequence of computations that must be performed at the same time – but still in a causal order. Such steps are called "δ-steps" in VHDL (and "micro-steps" in StateCharts). The idea is that when simulating a circuit, all independent processes must be simulated sequentially by the simulator. However, the real time (the time of the hardware) must not take these steps into account. Thus, two events e_1, e_2 at moments $(a, 1), (a, 2)$ respectively will be performed sequentially (e_1 before e_2) but at a same real time a. The VHDL addition is defined by the following rules:

$$(r' \neq 0) \Longrightarrow (r, d) + (r', d') = (r + r', d')$$
$$(r' = 0) \Longrightarrow (r, d) + (r', d') = (r, d + d')$$

where r, r', d and d' are natural numbers and $+$ denotes the usual addition on natural numbers. Clearly, the internal law $+$ above is not commutative, nor Archimedean: we may infinitely follow a δ-branch by successively adding δ-times.[1] $\Diamond$

2.1.2 Time scale

Time references give the basic expected properties of the set of all moments. Now, we want to define time scales, i.e. sets of moments of a time reference that will be used to define a system (systems can indeed have various paces & origin dates).

A time scale will later be used to define step-by-step behavior of system, which makes it necessary to define it as a sequence of moments.

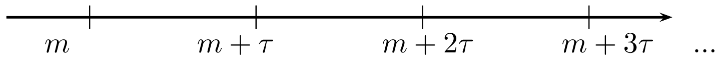
Definition 2.1.2 (Time scale) *A* **time scale** *is any subset $\mathbb{T}$ of a time reference T such that:*

- *$\mathbb{T}$ has a minimum $m^{\mathbb{T}} \in \mathbb{T}$*
- *$\forall t \in T,\ \mathbb{T}_{t+} = \{t' \in \mathbb{T} \mid t \prec t'\}$ has a minimum called $succ^{\mathbb{T}}(t)$*
- *$\forall t \in T$, when $m^{\mathbb{T}} \prec t$, the set $\mathbb{T}_{t-} = \{t' \in \mathbb{T} \mid t' \prec t\}$ has a maximum called $pred^{\mathbb{T}}(t)$*
- *the principle of induction[2] is true on $\mathbb{T}$.*

The set of all time scales on T is noted $Ts(T)$.

A time scale is defined so that it will be possible to make recursive constructions on it, and to locate any moment of the time reference between two moments of a time scale. A time scale necessarily has an infinite number of moments. In fact, a time scale is expected to comply with the Peano axioms[3], excepted that the $succ^{\mathbb{T}}$ and $pred^{\mathbb{T}}$ are defined for moments of T and not only $\mathbb{T}$.[4]
This is not equivalent: a simple counter-example on time reference $\mathbb{R}^+$ can show it is possible to have *pred* and *succ* properly defined for moments of the subset $\mathcal{T} = \{1 - \frac{1}{2^n} \text{for } n \in \mathbb{N}\} \cup \{1 + \frac{1}{2^n} \text{for } n \in \mathbb{N}\}$ whereas moment 1 has no *pred* or *succ* in $\mathcal{T}$. This fundamental property prevents Zeno's effect on any time scale.

Most of time scales (discrete and continuous) used when modeling real systems can be defined as unified regular time scales of step τ and of minimum m:



[1]This is not the intended use of VHDL time, however: VHDL computations should perform a finite number of δ-steps.

[2]For $A \subset \mathbb{T}$, $\left(m^{\mathbb{T}} \in A \ \&\ \forall t \in A, succ^{\mathbb{T}}(t) \in A\right) \Rightarrow A = \mathbb{T}$.

[3]It can be easily checked that the above conditions imply Peano axioms.

[4]These specific properties will be necessary to prove that time scales are closed under finite union.

Example 3 By using results of non-standard analysis, continuous time scales can then be considered in a discrete way. Following the approach developed in [12] to model continuous time by non-standard real numbers, a regular time scale can be $^*\mathbb{N}\tau$ where $\tau \in {}^*\mathbb{R}^+$ is the step, $0 \in {}^*\mathbb{N}\tau$ and $\forall t \in {}^*\mathbb{N}\tau$, $succ^{^*\mathbb{N}\tau}(t) = t + \tau$. This provides a discrete time scale for modeling classical discrete time (when the step is not infinitesimal) and continuous time (when the step is infinitesimal). $\diamondsuit$

Example 4 In the VHDL time $\mathcal{V}$, the internal law induces a lexicographic ordering on $\mathbb{N} \times \mathbb{N}$. Thus, let $\mathcal{W} \subset \mathcal{V}$ such that: $\forall a \in \mathbb{N},\ \exists N_a \in \mathbb{N},\ \forall (a, b) \in \mathcal{W}, b \leq N_a$ (i.e. there are only a finite number of steps at each moment of time in $\mathcal{W}$). Then $\mathcal{W}$ is a time scale in the VHDL time. $\diamondsuit$

Example 5 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. $\diamondsuit$

We have shown that we can accommodate heterogeneous times with our definitions. We introduce a fundamental proposition allowing to unify different time scales, which will be necessary for systems integration (when the systems involved do not share the same time scales). Overall, our definition of time will be suitable for heterogeneous integrated systems.

Proposition 2 (Union of time scales) *A finite union of time scales (on the same time reference T) is still a time scale.*

Proof The proof for two time scales is enough.

Let $\mathbb{T}_1, \mathbb{T}_2$ be two time scales on T. Let $\mathbb{T} = \mathbb{T}_1 \cup \mathbb{T}_2$. We want to prove that $\mathbb{T}$ is a time scale.
$\mathbb{T}$ is a subset of T. Note that $\mathbb{T}$ has a minimum $min\big(m^{\mathbb{T}_1}, m^{\mathbb{T}_2}\big)$, and that $\forall t \in T$, the *succ* and *pred* functions can be obviously defined by:

- $succ^{\mathbb{T}}(t) = min\big(succ^{\mathbb{T}_1}(t), succ^{\mathbb{T}_2}(t)\big)$
- when $t \succ m^{\mathbb{T}}$, $pred^{\mathbb{T}} = max\big(pred^{\mathbb{T}_1}(t), pred^{\mathbb{T}_2}(t)\big)$ [5]

We need to prove that the induction principle holds on $\mathbb{T}$. This can be proved by using a lemma: if $m^{\mathbb{T}} \in A$ & $\forall t \in A, succ^{\mathbb{T}}(t) \in A$ then $\forall t \in \mathbb{T}_i,\ t \in A \Rightarrow succ^{\mathbb{T}_i}(t) \in A$ for $i = 1, 2$. This lemma is proved using the principle of induction in $\mathbb{T}_i$ on intervals of successive elements of $\mathbb{T}_i$ in $\mathbb{T}$:

Let $P(t)$ be a proposition such that: $\forall t \in \mathbb{T},\ P(t) \Rightarrow P\big(succ^{\mathbb{T}}(t)\big)$ (L0). We want to show that: $\forall t \in \mathbb{T}_1,\ P(t) \Rightarrow P\big(succ^{\mathbb{T}_1}(t)\big)$ (L1). Let $t \in \mathbb{T}_1$ and $t' = succ^{\mathbb{T}_1}(t)$. If $t' = succ^{\mathbb{T}}(t)$, then (L1) is true for this t by (L0). Else: let $t_a = succ^{\mathbb{T}}(t)$ and $t_b = pred^{\mathbb{T}}(t')$. Then $t_a, t_b \in \mathbb{T}_2$ and $succ^{\mathbb{T}}$ (by construction

[5]for convenience of writing, we assume that if $pred^{\mathbb{T}_i}$ is not well defined for its argument, its value is $m^{\mathbb{T}}$.

equals to $succ^{\mathbb{T}_2}$ since there is no moment of $\mathbb{T}_1$ between t_a and t_b) defines, by induction in $\mathbb{T}_2$, a set of successive elements of $\mathbb{T}_2$ between t_a and t_b. If $P(t)$ is true, then $P(t_a)$ is true by (L0).

The principle of induction which is true in $\mathbb{T}_2$ can be applied to the set $[t_a, t_b]$ of moments of $\mathbb{T}_1$, so that $P(t_b)$ is true, and finally $P(t') = P\big(succ^{\mathbb{T}}(t_b)\big)$ is true by (L0).

What means that $P(t') = P\big(succ^{\mathbb{T}_1}(t)\big)$ is true. Finally, we have shown that: (L0) $\Rightarrow$ (L1). We can show the same property (L'1) on $\mathbb{T}_2$. Applying independently the principle of induction for proposition P on $\mathbb{T}_1$ and on $\mathbb{T}_2$, we have that: if $P(m^{\mathbb{T}})$ is true, P is true on $\mathbb{T}_1$ and P is true on $\mathbb{T}_2$. The proposition P is true on their union $\mathbb{T}$. Therefore, the principle of induction is true on $\mathbb{T}$.

Finally, $\mathbb{T} = \mathbb{T}_1 \cup \mathbb{T}_2$ satisfies the principle of induction, and thus $\mathbb{T}$ is a time scale on T. $\square$

Remark 1 *It is easy to show that the union of an infinite number of time scales can define a set of moments that is not a time scale. We recall the example* $\mathcal{T} = \{1 - \frac{1}{2^n} \textit{for } n \in \mathbb{N}\} \cup \{1 + \frac{1}{2^n} \textit{for } n \in \mathbb{N}\}$ *where moment* 1 *has no pred or succ in* $\mathcal{T}$. *Let define an infinite sequence of time scales:* $\forall n \in \mathbb{N}$, $\mathbb{T}_n = \{1 - \frac{1}{2^n}\} \cup \{1 + \frac{1}{2^n}\} \cup \mathbb{N}$. *The union* $\mathbb{T} = \bigcup_{n \in \mathbb{N}} \mathbb{T}_n$ *is not a time scale since the moment* 1 *has no pred or succ in* $\mathbb{T}$.

In our approach of time, even if the time reference is for instance $\mathbb{R}$, we do not set a discrete time reference, so that although all time scales on $\mathbb{R}$ are discrete and isomorphic to $\mathbb{N}$, they are not constrained by a predefined universal discrete time. Thus, our approach allows to define finer time scales for modeling real systems (which can work at different rhythms or with shifts of phase) than considering a given discrete time reference (as done for instance with reactive systems [43], where a universal sequential clock sets a given granularity that cannot be further refined[6]).

2.2 Data

Another challenge to address to model complex systems is the heterogeneity of data (modeling any element that can be exchanged between real systems) and of their synchronization between different time scales. We introduce datasets that will be used for defining data carried by dataflows.

2.2.1 Datasets

Definition 2.2.1 (ϵ-alphabet) *A set D is an **ϵ-alphabet** if $\epsilon \in D$. For any set B, we can define an ϵ-alphabet by $\overline{B} = B \cup \{\epsilon\}$.*

[6]However, our approach brings more complexity when modeling real systems.

The elements of an ϵ-alphabet are called *data* and ϵ is a universal blank symbol ϵ accounting for the absence of data (as the blank symbol in a Turing machine)[7]. An ϵ-alphabet can have an infinite number of data. A system dataset (also called dataset) is an ϵ-alphabet with the description of the behavior of the data:

Definition 2.2.2 (System dataset) *A* ***system dataset*** *is a pair* $\mathcal{D} = (D, \mathcal{B})$ *such that:*

- D *is an* ϵ*-alphabet*
- $\mathcal{B}$*, called* ***data behavior****, is a pair* (r, w) *with* $r : D \to D$ *and* $w : D \times D \to D$ *such that*[8]*:*
  - $r(\epsilon) = \epsilon$ $\qquad (R1)$
  - $r\big(r(d)\big) = r(d)$ $\qquad (R2)$
  - $r\big(w(d, d')\big) = r(d')$ $\qquad (R3)$
  - $w\big(r(d'), d\big) = d$ $\qquad (W1)$
  - $w\big(w(d, d'), r(d')\big) = w(d, d')$ $\qquad (W2)$

Remark 2 *We will sometimes use interchangeably the* ϵ*-alphabet and the dataset in our definitions & examples if it is more convenient and when there is no ambiguity.*

$\mathcal{B}$ will be useful to synchronize dataflows defined on different time scales (see Projection below). Data behaviors can be understood as the functions allowing to read and write data in a "virtual" 1-slot[9] buffer defining how this synchronization occurs at each moment of time:

- when a buffer is read, what is left (depending on the nature of data, it can partially vanish)
- when a new data is written (second parameter of w), knowing the current content of the buffer (first parameter of w), what is the new content of the buffer (depending on the nature of data and the new incoming data, it can be partially or totally modified).

In this context, the conditions on r and w can be understood as follows:

- (R1): reading an empty buffer (i.e. containing ϵ) results in an empty buffer
- (R2): reading the buffer once or many times results in the same content of the buffer

[7]basically, introducing this blank means that a flow "transmitting nothing" at a given moment will be coded as a flow transmitting ϵ at this moment.

[8]These axioms give a relevant semantics and are necessary to define consistent projections of dataflows on time scales.

[9]1-slot means that the buffer can contain only one data. This data will be used to compute the value of a dataflow at any moment of a time scale, to be able to synchronize a dataflow with any possible time scale.

- (R3): reading a buffer in which a data has just been written results in the same content whatever the initial content of the buffer was before writing the data
- (W1): when the buffer has just been read, the new data erases the previous one
- (W2): when the buffer has just been written with a data, it will not be modified if it is again written with the result of the reading function on this same data[10]
- we also have by (R1) + (W1): $w(\epsilon, d) = d$ (W3). When an empty buffer is written with a new data, the buffer contains this new data.

2.2.2 Implementation of standard data behaviors

There are two classical examples of data behaviors when modeling real systems:

Example 6 [**Persistent data behavior**] In this case, data cannot be consumed by a reading, and every writing erases the previous data (this data behavior was the only one used in [12]):

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

$\diamondsuit$

Example 7 [**Consumable data behavior**] In this case, data is consumed by a reading, and every writing (excepted when it is ϵ) erases the previous data:

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

$\diamondsuit$

We give a less classical example of data behavior that can be used to represent the ability to accumulate data received (what can be meaningful when data are written more frequently than read). It is important to notice that the buffer is still a 1-slot buffer and that all accumulated data will be consumed entirely by a single reading[11].

Example 8 [Accumulative data behavior] Let A be a non-empty set and $D = \mathcal{P}(A)$ be the set of subsets of A. We consider that $\epsilon = \emptyset$, so that D is an

[10]This rule will ensure that a dataflow projected on a finer time scale is equivalent to the initial dataflow.

[11]Modeling another kind of reading shall be modeled by buffers in the system itself, this is not the purpose of these "virtual" buffers dedicated to synchronization of data between different time scales.

ϵ-alphabet[12]. In this case, data is consumed by a reading, and every writing is added (using internal law of D, here $\cup$) to the previous data:

$$r(d) = \epsilon \text{ and } w(d, d') = d \cup d'$$

$\Diamond$

It is straightforward to check that these examples are compliant with the axioms of data behaviors.

Remark 3 *The same real data can be modeled using different behaviors: for instance, an electric current might be measured by a number of electrons at each step of a time scale (consumable behavior, data expressed as a natural number), or by a continuous flow of electrons (persistent behavior, data expressed as a real number in Amperes). Thus, a data behavior is not an intrinsic property of the real data it models, but a modeling choice.*

2.3 Dataflows

The dataflows will be used to describe variables of systems (inputs, outputs and states). We also define the synchronization of dataflows between time scales (to be able to properly define the integration of systems with different time scales).

2.3.1 Definition

In what follows, $\mathcal{D}$ will stand for a dataset of ϵ-alphabet D with behaviors (r_D, w_D). A dataflow is a flow defined at the moments of a time scale carrying data of a dataset. It will be used to define the evolution of states, inputs and outputs of a system.

Definition 2.3.1 (Dataflow) *Let $\mathbb{T}$ be a time scale. A* **dataflow** *over $(\mathcal{D},\mathbb{T})$ is a mapping $X : \mathbb{T} \to D$.*

Definition 2.3.2 (Sets of dataflows) *The set of all dataflows over $(\mathcal{D},\mathbb{T})$ is noted $\mathcal{D}^{\mathbb{T}}$. The set of all dataflows over $\mathcal{D}$ with any possible time scale on time reference T is noted $\mathcal{D}^T = \bigcup_{\mathbb{T} \in Ts(T)} \mathcal{D}^{\mathbb{T}}$.*

2.3.2 Operators

We introduce an operator making it possible to project a dataflow on any time scale. The mechanism to compute the resulting dataflow correspond to the idea that there is an intermediary buffer which stores or outputs the values of the initial dataflow so that they can be read according to the new time scale. The projection of a dataflow on a time scale makes it possible to synchronize a

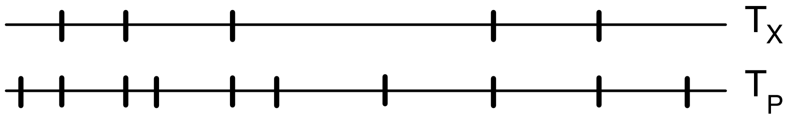
[12] We can extend this example to any unital magma of identity element ϵ.

dataflow between two different time scales, with the rule that a data arriving at t will be read at the first next moment on the time scale of projection (the computation of this synchronization only requires a 1-slot virtual buffer and data behaviors). It will be essential when composing together systems using different time scales to define the properties of the exchange of data.

Definition 2.3.3 (Projection of a dataflow on a time scale) *Let X be a dataflow over ($\mathcal{D}$, $\mathbb{T}_X$) and $\mathbb{T}_P$ be a time scale. Let $\mathbb{T} = \mathbb{T}_X \cup \mathbb{T}_P$. Let $\mathbb{T}'_P = succ^{\mathbb{T}}(\mathbb{T}_P)$*[13]*. We define recursively the buffer function $b : \mathbb{T} \to D$ by* [14]*:*

- *(P1) if $t \in \mathbb{T}_X \setminus \mathbb{T}'_P$, $b(t) = w\big(b(pred^{\mathbb{T}}(t)), X(t)\big)$*
- *(P2) if $t \in \mathbb{T}'_P \setminus \mathbb{T}_X$, $b(t) = r\big(b(pred^{\mathbb{T}}(t))\big)$*
- *(P3) if $t \in \mathbb{T}_X \cap \mathbb{T}'_P$, $b(t) = X(t)$*
- *(P4) if $t \in \mathbb{T}_P \setminus (\mathbb{T}_X \cup \mathbb{T}'_P)$, $b(t) = b(pred^{\mathbb{T}}(t))$*

*The **projection** $X_{\mathbb{T}_P}$ **of** X **on** $\mathbb{T}_P$ is then the dataflow over ($\mathcal{D}$, $\mathbb{T}_P$) defined by setting $X_{\mathbb{T}_P}(t) = b(t)$ for every $t \in \mathbb{T}_P$.*



The semantics of each part of the definition is the following: (P1) occurs when a new data is received, and when the data on the buffer has not been read at the previous step so does not need to be processed with the reading function. (P2) occurs when no new data is received, and when the data on the buffer has been read at the previous step and so needs to be processed in the buffer with the reading function. (P3) occurs when a new data is received, and when the data on the buffer has been read at the previous step, so that the content of the buffer is the new data, by condition (W1). Finally, (P4) occurs when no new data is received, and when the data on the buffer has not yet been read, so that nothing changes.

Remark 4 *It is straightforward to show that the projection of a dataflow on its own time scale does not alter the dataflow (as only P3 occurs): $X_{\mathbb{T}_X} = X$*

We define equivalent dataflows as dataflows that cannot be distinguished by any projection. It is useful to be able to compare data flows defined on different time scales (but that will have exactly the same behavior for any given system).

[13]Corresponding to moments such that a data has been read at the previous moment and shall be marked as "read".

[14]By convention, $b\big(pred^{\mathbb{T}}(m^{\mathbb{T}})\big) = \epsilon$, which makes it simpler to define the rules of projection without making a special case when $t = m^{\mathbb{T}}$.

Definition 2.3.4 (Equivalent dataflows) *Two dataflows X and Y are* ***equivalent*** *(noted $X \sim Y$) if, and only if:*

$$\text{for any time scale } \mathbb{T} \text{ on } T,\ X_{\mathbb{T}} = Y_{\mathbb{T}}$$

Example 9 *Let X be a dataflow over $(\mathcal{D}, \mathbb{T})$ where $\mathcal{D}$ as a persistent data behavior and $\mathbb{T}$ is a regular time scale of step τ and minimum m. Let $\mathbb{T}'$ be the regular time scale of step $\tau/2$ and minimum m. Let X' be the dataflow over $(\mathcal{D}, \mathbb{T}')$ defined by: $\forall t \in \mathbb{T},\ X'(t) = X'(t+\tau) = X(t)$. Then, $X \sim X'$. The intuition behind this result is that X' has 2 moments for each moment of X but carry the same persistent data, so that they cannot be distinguished.*

Definition 2.3.5 (Equivalent dataflows as far as) *Two dataflows X and Y are* ***equivalent as far as*** *$t_0 \in T$ (noted $X \sim_{t_0} Y$) if, and only if:*

$$\text{for any time scale } \mathbb{T} \text{ on } T,\ \text{for all } t \preceq t_0 \text{ in } \mathbb{T},\quad X_{\mathbb{T}}(t) = Y_{\mathbb{T}}(t)$$

2.3.3 Consistency of dataflows

We introduce two propositions insuring the consistency of our definition of dataflows. These propositions will be useful to prove the consistency of operators we define in Chapter 4. The first proposition ensures that the way we define the projection of a data flow on a finer time scale is consistent with the equivalence relationship we have naturally defined on dataflows:

Proposition 3 (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, we have:*

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

Proof The proof uses the properties of the data behaviors and the principle of induction on time scales to show that: $\forall \mathbb{T} \in Ts(T),\ X_{\mathbb{T}} = (X_{\mathbb{T}_P})_{\mathbb{T}}$, and thus $X \sim X_{\mathbb{T}_P}$ (by definition of $\sim$). Let $\mathbb{T}$ be a time scale. We note $P = X_{\mathbb{T}_P}$ and want to prove that $X_{\mathbb{T}} = P_{\mathbb{T}}$.

Let b_X be the buffer (defined for moments of $\mathbb{T}_X \cup \mathbb{T}$) used for defining the projection of X on $\mathbb{T}$ and b_P be the buffer (defined for moments of $\mathbb{T}_P \cup \mathbb{T}$) used for defining the projection of P on $\mathbb{T}$. We will prove by induction on $\mathbb{T}$ that: $\forall t \in \mathbb{T},\ b_X(t) = b_P(t)$.

(a) First, we want to prove that the equality is true for moments before $m^{\mathbb{T}_X}$. Let $t_0 \in \mathbb{T}$ with $t_0 \prec m^{\mathbb{T}_X}$ (we will suppose without loss of generality[15] that $m^{\mathbb{T}} \prec m^{\mathbb{T}_X}$).

- $X_{\mathbb{T}}(m^{\mathbb{T}}) = \epsilon$ by (P0) + (P4) to initialize the buffer.
- for P:

[15]We may add if necessary a smaller initial moment to $\mathbb{T}$.

  - if $m^{\mathbb{T}} \prec m^{\mathbb{T}_P}$, then $P_{\mathbb{T}}(m^{\mathbb{T}}) = \epsilon$ by (P0) and (P4)
  - else: we have $\forall t \in \mathbb{T}_P \mid t \prec m^{\mathbb{T}_X}$, $P(t) = X_{\mathbb{T}_P}(t) = \epsilon$ by (P0) and (P4) since $m^{\mathbb{T}} \prec m^{\mathbb{T}_X}$. As $w(\epsilon, \epsilon) = \epsilon$ by (W3) and $r(\epsilon) = \epsilon$ by (R1), the buffer till $m^{\mathbb{T}}$ in the projection of P on $\mathbb{T}$ is alway equal to ϵ and we have $P_{\mathbb{T}}(m^{\mathbb{T}}) = \epsilon$.

- finally, $X_{\mathbb{T}}(m^{\mathbb{T}}) = P_{\mathbb{T}}(m^{\mathbb{T}})$.
- we can extend the proof by induction to any $t_0 \in \mathbb{T}$ with $t_0 \prec m^{\mathbb{T}_X}$ since $\forall t \in \mathbb{T}_P$ such that $t \prec m^{\mathbb{T}_X}$, $P(t) = \epsilon$.

(b) For the case where $t = m^{\mathbb{T}_X}$, we have $b_X(t) = b_P(t) = X(m^{\mathbb{T}_X})$, which will allow to initiate the induction.

(c) Then, we want to prove that the induction hypothesis can be used on $\mathbb{T}$. Let $t_0 \in \mathbb{T}$ with $t_0 \succeq m^{\mathbb{T}_X}$ such that $b_X(t_0) = b_P(t_0)$ and $t_1 = succ^{\mathbb{T}}(t_0)$. We want to prove that $b_X(t_1) = b_P(t_1)$. Let $A =]t_0; t_1] \cap \mathbb{T}_P$ [16] and $B =]t_0; t_1] \cap \mathbb{T}_X$ (we have $B \subseteq A$ since $\mathbb{T}_X \subseteq \mathbb{T}_P$). Let $t_x = pred^{\mathbb{T}_X}\big(succ^{\mathbb{T}_X}(t_0)\big)$[17]. Three cases are to be considered:

- (1.1) if $B = \emptyset$ and $A = \emptyset$
  - for $b_X(t_1)$: by (P2) we have $b_X(t_1) = r\big(b_X(t_0)\big)$
  - for $b_P(t_1)$: by (P2) we have $b_P(t_1) = r\big(b_P(t_0)\big) = r\big(b_X(t_0)\big)$
  - and so $b_X(t_1) = b_P(t_1)$.
- (1.2) if $B = \emptyset$ and $A \neq \emptyset$, two subscases shall be considered
  - (1.2.1) if $t_x = t_0$, then
    * for $b_X(t_1)$: by (P2) we have $b_X(t_1) = r\big(b_X(t_0)\big)$. According to the situation: by (P3) $b_X(t_0) = X(t_0) = w\big(\epsilon, X(t_0)\big)$ or by (P1) $b_X(t_0) = w\big(\ldots, X(t_0)\big)$. Anyway, $b_X(t_0) = w\big(\ldots, X(t_0)\big)$, and by (R3) we have $b_X(t_1) = r\big(X(t_0)\big)$.
    * for $b_P(t_1)$: $\forall t \in A$, $P(t) = r\big(X(t_0)\big)$, using (P2) and (R2) in the definition of P as the projection of X on $\mathbb{T}_P$. So, by (P1) we have $b_P(m^A) = P(m^A) = r\big(X(t_0)\big)$. As $w\big(r(d'), d\big) = d$ by (W1), we have applying (P1) for all $t \in A$, $b_P(t) = r\big(X(t_0)\big)$. If $t_1 \notin A$, we apply (P4) and get: $b_P(t_1) = r\big(X(t_0)\big)$, and the result is the same if $t \in A$.
    * and so $b_X(t_1) = b_P(t_1)$.
  - (1.2.2) if $t_x \prec t_0$, then

[16] This notation is intended to capture the elements of $\mathbb{T}_P$ greater than t_0 and less than or equal to t_1.

[17] t_x is the latest moment of T_X before or at t_0. It exists since $t_0 \succeq m^{\mathbb{T}_X}$.

  * for $b_X(t_1)$: by (P2) we have $b_X(t_1) = r\big(b_X(t_0)\big)$. But $b_X(t_0)$ can be, by (P2) and (P4), expressed recursively as $b_X(t_0) = r\big(b_X(t_X)\big)$. Whatever the situation, as in (1.2.1) we can write $b_X(t_x) = w\big(\ldots, X(t_x)\big)$ and so by (R3) we have $b_X(t_0) = r\big(b_X(t_X)\big) = r\big(X(t_x)\big)$.
  * for $b_P(t_1)$: the proof is exactly the same as in (1.2.1).
  * and so $b_X(t_1) = b_P(t_1)$.

- (1.3) if $B \neq \emptyset$ then $A \neq \emptyset$. We show as in (1.2) that $b_X(m^B) = b_P(m^B)$, and we conclude recursively that $b_X(t_1) = b_P(t_1)$ using (W2).

(d) Finally, by induction, we have $\forall t \in \mathbb{T},\ b_X(t) = b_P(t)$ (begining the induction at $t = m^{\mathbb{T}_X}$, the anterior t being handled by the first case (a)).

Hence, $X_{\mathbb{T}} = P_{\mathbb{T}} = (X_{\mathbb{T}_P})_{\mathbb{T}}$ for any time scale $\mathbb{T}$. Thus $X \sim X_{\mathbb{T}_P}$ (by definition of $\sim$) which proves our result. $\square$

The second proposition proves the consistency of the projection when dealing with a successive refinement of time scale for a dataflow, so that using an intermediate finer time scale does not alter the resulting projected dataflow:

Proposition 4 (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:*

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

Proof $(X_{\mathbb{T}_P})_{\mathbb{T}}$ and $X_{\mathbb{T}}$ share the same time scale $\mathbb{T}$, so we need to prove that they have the same value at each moment of $\mathbb{T}$. This can be proven by induction in a very similar way to the previous proof. $\square$

Chapter 3

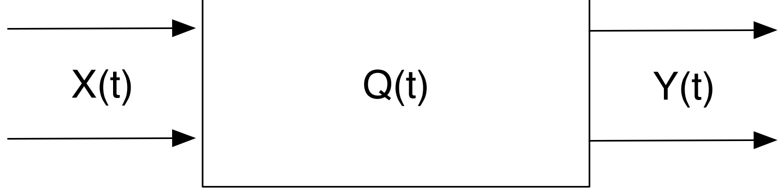
Systems

We define a system as a mathematical object characterized by coupled functional and states behaviors, together with a timescale. We thus model a real system as a black box with an observable functional behavior and an internal state (similarly to a timed Mealy machine). This definition is expressive enough to capture, at some level of abstraction, the functional behavior of any real system with sequential transitions. We also express the functional behavior of systems via transfer functions transforming dataflows and we show the equivalence of the two formalisms. The deliverable is a unified definition of a system (viewed as a functional black box) and a proof of its equivalence with transfer functions.

3.1 Systems

3.1.1 Definition

We view real systems as functional black boxes, transforming inputs into outputs, and characterized by an internal state. A system can thus be semi-formally represented as follows:



We define a system as a mathematical object (figuring a functional black box with an internal state[1]), characterized by coupled functional and states behaviors (defining step by step transitions for changing state and output in

[1]The properties of this internal state are decisive when studying computability.

instantaneous reaction to inputs). The system transitions occur only at given moments that form the time scale of the system.

Definition 3.1.1 (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 a time scale called the time scale of the system,*
- $Input = (In, \mathcal{I})$ *and* $Output = (Out, \mathcal{O})$ *are datasets, called input and output datasets,*
- S *is a nonempty* ϵ*-alphabet* [2]*, called the* ϵ*-alphabet of states,*
- q_0 *is an element of* S*, called initial state,*
- $\mathcal{F} : In \times S \times \mathbb{T}_s \to Out$ *is a function called functional behavior,*
- $\mathcal{Q} : In \times S \times \mathbb{T}_s \to S$ *is a function called states behavior.*

$(Input, Output)$ *is called the signature of* $\int$*.*

Our definition of a system can be understood as a timed Mealy machine, i.e. a Mealy machine [46] where we have introduced time[3], and where the set of states is not supposed to be finite (what is a key point from the point of view of computability, but outside of the scope of this work).

$\mathbb{T}_s$ represents the moments of “life” of the system, i.e. the moments where state and output can change in the behavior of the system, and where input is read by the system.

Input and *Output* respectively model the data that the system receives as inputs and emits as outputs.

S models the possible states of the system, this state being all information “inside” the system, allowing to define its instantaneous behavior according to inputs and time. q_0 is a distinguished state which will be the one used to compute the initial state of the system at $m^{\mathbb{T}_s}$.

$\mathcal{F}$ and $\mathcal{Q}$ compute respectively the output and the current state of a system, from its last defined state, its current input and the moment of time considered. The input can therefore have an instantaneous influence on the output, which is the most generic definition and is useful to model systems processing without delay the data they receive (e.g. interfaces between system that can just add random noises to their input). Those principles of transitions are similar to the ones of a Mealy machine (note that since we will consider the behavior of a system within time, Moore & Mealy paradigms are not equivalent as Moore paradigm does not allow an instantaneous influence of an input on an output).

Introducing time in the transition functions is necessary so that the system has information about time to make transitions only at moments on its time

[2]Defining S as an ϵ-alphabet (therefore containing ϵ) and not just as a set will make it possible to define a dataflow of states, what will later be convenient.

[3]The introduction of time is a fundamental difference and brings a lot of complexity.

scale. Defining the system just as a sequential behavior on its time scale (which is only *one* possible time scale in the time reference) without knowledge of time would make it difficult to compose meaningfully this system with another system having a sequential behavior on another time scale, so that the composition can still be expressed as a system. Indeed, when we will later define the "product" of 2 systems with different time scales as a new system, it will require to define them on a shared time scale, therefore disrupting the initial time scale of each system and making it impossible to define a step by step behavior of the resulting system without knowledge of time (or without introducing tricky & artificial states to "count" the number of moments). This is a fundamental characteristic of our model.

We will now define the step-by-step execution of a system within time, allowing to transform an input dataflow into an output dataflow, while defining a state dataflow.

3.1.2 Execution

Given an input dataflow, a system has a deterministic behavior that can be summarized through its output dataflow and its state dataflow that describes its output and its state at each moment of its time scale. The system reads its input at each moment of its time scale[4] ; if data arrive at moments outside of the system time scale, they will be, by definition, synchronized with the time scale of the system to match its rhythm, through a dataflow projection.

Definition 3.1.2 (Execution of a system) *Let $\int$ be 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[5]:*
  - $Q(m^{\mathbb{T}_s}) = \mathcal{Q}\big(\tilde{X}(m^{\mathbb{T}_s}), q_0, m^{\mathbb{T}_s}\big)$
  - $\forall t \in \mathbb{T}_s,\ Q\big(succ^{\mathbb{T}_s}(t)\big) = \mathcal{Q}\big(\tilde{X}(succ^{\mathbb{T}_s}(t)), Q(t), succ^{\mathbb{T}_s}(t)\big)$
- *$Y \in Out^{\mathbb{T}_s}$ is defined by:*
  - $Y(m^{\mathbb{T}_s}) = \mathcal{F}\big(\tilde{X}(m^{\mathbb{T}_s}), q_0, m^{\mathbb{T}_s}\big)$
  - $\forall t \in \mathbb{T}_s,\ Y\big(succ^{\mathbb{T}_s}(t)\big) = \mathcal{F}\big(\tilde{X}(succ^{\mathbb{T}_s}(t)), Q(t), succ^{\mathbb{T}_s}(t)\big)$

X, Q and Y are respectively input, state and output dataflows.

[4]Buffers of the system can be defined inside the system itself if we want for instance to model a delayed reading of the inputs.

[5]The ϵ-alphabet of states S is associated with a persistent behavior, since the state of a system at any moment of the time reference can be obtained by considering its last defined state.

The current state of a system at a given step is defined as the state resulting from the transition at this step, and the input has thus an instantaneous influence on the state at moments of the time scale of the system. In term of modeling semantics, it is more meaningful, as the system exists not only on its time scale but in the whole time reference. Observing the state of the system between 2 steps must allow to see its new state (it would not be meaningful to have to wait until the next step of the system to have its state updated). In the case of classical Mealy machines, there is no such phenomenon since the time is implicitly defined as a universal sequence of steps (and the exact time of occurrence of transitions is not important to define the sequence of outputs).

We notice that because of this instantaneous influence of the input on the state, the state of the system at $m^{\mathbb{T}_s}$ is not q_0 but the state q_1 (computed with $\mathcal{Q}$ from q_0 and the first input).

Thus, the transitions of a system within time can be represented as follows:

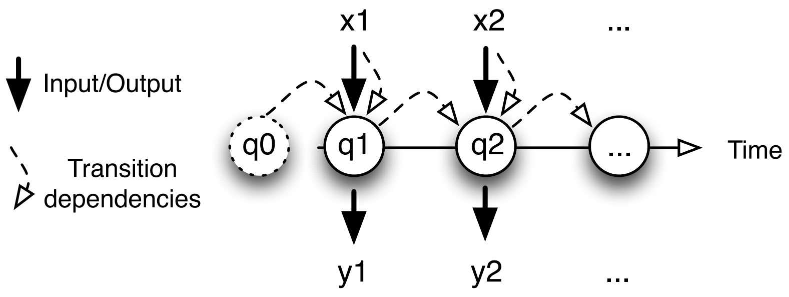


Figure 3.1: State transitions of a system within time (with dependencies)

3.1.3 Examples & expressivity

We give examples of modeling in our formalism of the three elementary kinds of real systems mentioned in the introduction (physical, software and human), as well as a hybrid system.

Example 10 (Software system) A software system can be modeled as a Turing machine with input and output, whose transitions are made following a time scale. In our model, the state of a system contains the memory of the Turing machine, its logical state, and its RW-head's position.

We consider a classical Turing machine with input and output. Let Q^{Tur} be the finite, nonempty set of logical states of the Turing machine, q_{Tur0} be the initial logical state, Σ be the ϵ-alphabet of internal tape symbols, In and Out be the sets of input and output, and $\delta : Q^{Tur}\times\Sigma\times In \to Q^{Tur}\times\Sigma\times Out\times\{-1,0,1\}$ the transition function (separated into 4 projections δ_1, δ_2, δ_3, δ_4 respectively on Q_{Tur}, Σ, Out and $\{-1,0,1\}$).

We define the system $\int = (\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, \mathcal{Q})$ simulating this Turing machine by:

- $\mathbb{T}_s = \mathbb{N}$
- $Input = (In, \mathbb{I})$ where $\mathbb{I}$ is any data behavior on dataset In
- $Output = (Out, \mathbb{O})$ where $\mathbb{O}$ is any data behavior on dataset Out
- $S = \Sigma^{\mathbb{Z}} \times \overline{Q^{Tur}} \times \overline{\mathbb{Z}}$ [6]
- $q_0 = (\epsilon, q_{Tur0}, 0)$
- $\mathcal{F}\big(x, (tape, q_{Tur}, i), t\big) = \delta_3(k)$ where $k = q_{Tur}, tape[i], x \in Q^{Tur} \times \Sigma \times In$
- $\mathcal{Q}\big(x, (tape, q_{Tur}, i), t\big) = \big(tape\big[i \leftarrow \delta_2(k)\big], \delta_1(k), i + \delta_4(k)\big)$ [7]

This system simulates the initial Turing machine. Thus, our model contains Turing-like models of software systems.[8]

This system is also a good illustration of why introducing t in the behavior functions of a system is necessary. If we were defining this Turing machine in the time reference $\mathbb{R}$, the resulting system should "know" at which moments of time it should make a transition, which would require a knowledge of time (it can here be understood as a clock). $\Diamond$

Example 11 (Human system) A basic example of a human system may be an individual, John, in the context of its work. In our modeling, John has two states ("normal" or "tired"). He can receive requests (by phone) from its colleagues (he must answer them by "Yes" or "No") and can also receive energy (when eating for example, what makes him normal if he was tired). Lastly, John can become tired after receiving too many requests from its colleagues. John is a very helpful guy always ready to help people, but when he is tired, he only helps urgent requests.

In the scope of our story, John can be modeled as the following system:

- we choose $\mathbb{T}_s = \mathbb{N}$ (each unit of time being a second)[9]
- $Input = (In, \mathcal{I})$ with $In = \{Urgent\_request, Request, \epsilon\} \times \{Energy, \epsilon\} \times \{Too\_many, \epsilon\}$ and $\mathcal{I}$ being the consumable behavior associated to In. Too_many is an event to model nondeterministic behaviors of the system at this abstraction level[10]

[6]In $\Sigma^{\mathbb{Z}}$, ϵ is the sequence $(\epsilon)_{\mathbb{Z}}$.

[7]Where $tape\big[i \leftarrow x\big]$ means replacing in the sequence $tape$ the i^{th} symbol with x, and $\delta_i(k)$ is the i^{th} element of n-uple k.

[8]This example makes it clear that what we call Q in our model of systems is actually the product of the logical state, the head position and the internal memory in a Turing machine.

[9]The choice of the time scale will be especially important when composing this system with other systems having their own time scales. The hidden assumption here is that John cannot receive more than one phone request each second.

[10]The nondeterminism allows to express in this high-level modeling the fact that John will become tired when having received "too many" requests (what cannot be expressed precisely at this abstraction level).

- $Output = (Out, \mathcal{O})$ with $Out = \{Yes, No, \epsilon\}$ and $\mathcal{O}$ being the consumable behavior
- $S = \{Tired, Normal, \epsilon\}$ and $q_0 = Normal$
- $\mathcal{F}\big((x_1, x_2, e), q, t\big) = \begin{cases} Yes & \text{if } x_1 = \mathit{Request} \;\&\; q = Normal \\ & \text{or } x_1 = Urgent\_request \\ No & \text{if } x_1 = \mathit{Request} \;\&\; q = Tired \\ \epsilon & \text{else (i.e. } x_1 = \epsilon) \end{cases}$
- $\mathcal{Q}\big((x_1, x_2, e), q, t\big) = \begin{cases} Tired & \text{if } e = Too\_many \\ & \text{or } q = Tired \;\&\; x_2 \neq \mathit{Energy} \\ Normal & \text{else} \end{cases}$

The main benefit of such a modeling is to be able to explain at high level what the meaningful states and behavior of a person can be, so that they will be taken into account in the design process of other parts of the system. ♢

Example 12 (Physical system) It has been proved that any Hamiltonian system can be modeled within the framework introduced in [12]. As our definition of system generalizes this previous work[11], we will recall a simplified example of a Water Tank given in [12], which is a well-known example of the hybrid systems and control theory literature.

We work in the time reference $^*\mathbb{R}$ of nonstandard real numbers. Let us fix first some regular continuous time scale $\mathbb{T}$ with infinitesimal time step τ. We consider a water tank where water arrives at a variable rate $w_i(t) \geq 0$ (with $t \in \mathbb{T}$) through one single pipe. The water leaves through an output pipe (of maximal throughput capacity C) at rate $w_o(t)$ (with $t \in \mathbb{T}$) controlled by a valve whose position is given by $v(t) \in [0, 1]$ (with $t \in \mathbb{T}$), 0 and 1 modeling respectively here the fact that the valve is closed or open. The valve is controlled by a sensor measuring the level $l(t)$ of water in the tank, which aims at keeping this level in a given interval $[L_1, L_2]$ (the initial water tank level L_0 belongs therefore to this interval). The water tank can be modeled as a system, taking on input the current values of the incoming water flow $w_i(t)$ and the position v(t) of the valve and sending on its output the corresponding output water flow $w_o(t)$ and water level $l(t)$ according to the following equations:

$$w_o(0) = C\,V_0, \quad w_o(t+\tau) = C\,v(t) \ \text{ for every } t \in \mathbb{T}^*,$$
$$l(0) = L_0, \quad l(t+\tau) = l(t) + (w_i(t) - w_o(t))\,\tau \ \text{ for every } t \in \mathbb{T}^* \ .$$

The input and output spaces of the system are thus $In_T = \overline{[0, C]} \times \overline{[0, 1]}$ and $Out_T = \overline{[0, C]} \times \overline{[L_1, L_2]}$. This illustrates the modeling of a simple physical system in our framework. Modeling of more complex physical systems can be found in [12]. ♢

[11] As seen in Chapter 2, non-standard time scales defined in [12] are still time scales in our new model.

Example 13 (Hybrid system) We define a classical system modeling the physical behavior of a lamp, with a continuous output signal representing the output lamp brightness, evolutions of which have an inertia and can be described by differential equations. The lamp can be modeled as a system, taking on input the information on whether the button is pressed or released $b(t)$, and sending on its output the new output lamp brightness energy $y(t)$, while having an internal state $\big(m(t), s(t)\big)$ composed of a mode ("On" or "Off") and the current brightness of the lamp. The input of the system will be $B = \{\rho, \pi\}$ (where ρ and π respectively model the fact that the button is either pressed or released), the outputs will be $Y = \mathbb{R}^+$ and the states will be $M \times S = \{On, Off\} \times \mathbb{R}^+$. To model such a system, we work in the time reference $^*\mathbb{R}$ of nonstandard real numbers and set a continuous regular time scale $\mathbb{T}$ with infinitesimal time step τ. We then define the following transition functions (one for each part of the state, and one for the output):

$$m(0) = Off$$

$$m(t+\tau) = \begin{cases} Off & \text{if } (m(t) = On \text{ and } b(t+\tau) = \rho) \text{ or} \\ & (M(t) = Off \text{ and } b(t+\tau) = \pi) \\ On & \text{if } (m(t) = On \text{ and } b(t+\tau) = \pi) \text{ or} \\ & (m(t) = Off \text{ and } b(t+\tau) = \rho) \end{cases}$$

$$s(0) = 0$$

$$s(t+\tau) = \begin{cases} s(t) + (e - s(t))\tau k & \text{if } m(t+\tau) = On \\ s(t) - (e - s(t))\tau k & \text{if } m(t+\tau) = Off \end{cases}$$

$$y(t) = s(t)$$

where e is the maximal brightness of the lamp, and k is a parameter to express the speed of evolution of the lamp brightness within time after a mode shift (therefore modeling the inertia of the lamp). Note that in the recursive definition of the output, $s(t)$ replaces the term $y(t)$ of the differential equation, since the part S of the state of the lamp is used to store the current brightness of the lamp[12].

$\diamondsuit$

3.2 Transfer functions

Functional behaviors (between inputs and outputs) of systems induce "causal" functions transforming dataflows, i.e. functions whose behavior is deterministic and do not depend on data received in the future[13].

[12]It is the usual way to transform a differential equation into transition functions, as it is not possible to define the new output based on the previous output.

[13]Indeed, switching from defining a step by step behavior to a transformation of dataflows makes it necessary to introduce this constraint, as a function transforming dataflow might

3.2.1 Definition

Definition 3.2.1 (Transfer function) *Let Input and Output be two datasets and let $\mathbb{T}_s$ be a time scale. A function $F : Input^T \to Output^{\mathbb{T}_s}$ is a **transfer function** on time scale $\mathbb{T}_s$ of signature $(Input, Output)$ if, and only if it is causal, i.e.:*

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

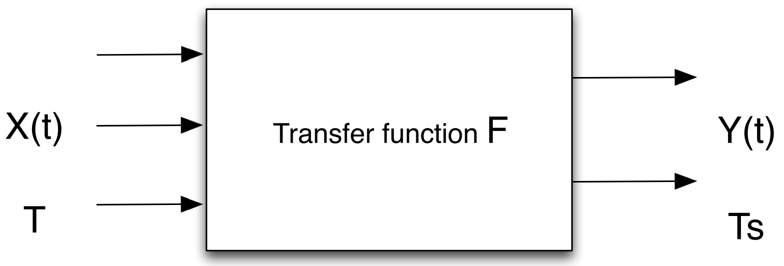


Figure 3.2: A transfer function

Corollary 1 *A transfer function cannot distinguish equivalent dataflows:*

$$X \sim Y \Rightarrow F(X) = F(Y)$$

Proof $X \sim Y$ implies that $X_{\mathbb{T}_s} = Y_{\mathbb{T}_s}$. So that $\forall t \in T,\ X_{\mathbb{T}_s} \sim_t Y_{\mathbb{T}_s}$, which by definition of a transfer function means that $F(X) \sim_t F(Y)$, so that $F(X) = F(Y)$ as they share the same time scale $\mathbb{T}_s$. □

A transfer function is a classical and "universal" representation (see [60, 21]) of any functional behavior (i.e. an object that receives and sends data within time). We will show that every system induces a transfer function, and later that integration operators defined on systems are consistent with the corresponding integration operators that can naturally be defined on transfer functions.

We define equivalent transfer functions as transfer functions which cannot be distinguished in term of dataflow transformation:

Definition 3.2.2 (Equivalence of transfer functions) *Let F_1 and F_2 be two transfer functions sharing the same signature. F_1 and F_2 are **equivalent** (noted $F_1 \sim F_2$) if, and only if:*

$$\forall X \in Input^T,\ F_1(X) \sim F_2(X)$$

induce non-causal behavior where the resulting dataflow would be different at a given moment when a future value in the input dataflow is changed!

Remark 5 *There is a particular transfer function: the only one from $\emptyset$ to $\emptyset$*[14]*, which is called closed transfer function (corresponding to closed systems, i.e. systems without interactions with their environment). However, it is important to realize that a system with a closed transfer function isn't empty, since its internal state contains information describing the temporal evolution of the system*[15]*. Moreover, it may be possible to decompose a closed system into subsystems with non-closed transfer functions.*

3.2.2 Transfer function of a system

A unique transfer function can be associated with any system. It describes the functional behavior of this system.

Theorem 1 (Transfer function of a system) *Let $\int$ be a system. There exists a unique*[16] *transfer function $F_\int$, called* ***the transfer function of*** *$\int$ and such that: for all input dataflow X of $\int$, $F_\int(X)$ is the output dataflow in the execution of $\int$.*

Proof Let $F_\int : In^T \to Out^{\mathbb{T}_s}$ be the function defined by setting for every $X \in In^T$, $F_\int(X)$ as the unique output dataflow $Y \in Out^{\mathbb{T}_s}$ corresponding to the input dataflow X in the execution of $\int$. We want to prove that it is causal. Let $X, Y \in In^T$ be two input dataflows for the system $\int$. If $X \sim_{t_0} Y$ for some $t_0 \in T$, then by definition we have $X_{\mathbb{T}_s}(t) = Y_{\mathbb{T}_s}(t)$ for every $t \preceq t_0$ in $\mathbb{T}_s$. The execution of a system for an input dataflow only depends on the projection of this dataflow on the time scale of the system. By definition of the execution of a system, the output of this system at t only depends on inputs received until t (included), and so $F_\int(X) \sim_{t_0} F_\int(Y)$. Thus, $F_\int$ is causal. $\square$

A system can then be represented as in Figure 3.3 (where the white squares on the left account for “virtual” buffers projecting the input dataflow on the time scale $\mathbb{T}_s$ of the system).

[14]this function is unique, and it is an initial object in the category of sets

[15]moreover, the very first step of the 3-layer architectural analysis (operational, functional, constructional) of a system consists in finding a closed greater system containing the system to be sure to capture all interactions between the system and its environment

[16]Unique on time scale $\mathbb{T}_s$, and up to equivalence on all time scales.

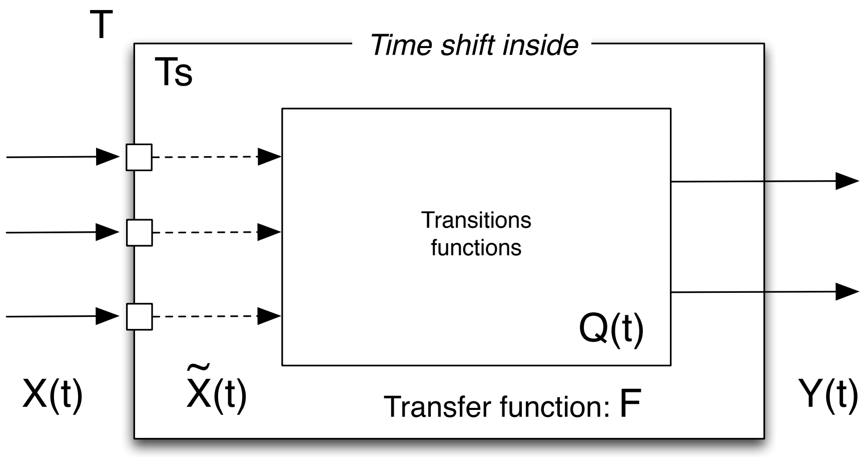


Figure 3.3: A system

In practice, nontrivial transfer functions are very difficult to specify as they are not a step by step definition of dataflow transformation, but a function of dataflows themselves. But we will use the correspondence between systems and transfer functions for a key matter: proving the consistency of the integration operators we will define in the following chapter.

Remark 6 *The relationship between systems and transfer functions can also be conversely understood: a system is the step-by-step algorithmic specification of a transfer function, and its states are the equivalence classes induced by the equivalence relationship defined on prefix dataflows such that two prefix dataflows X_1 and X_2 are equivalent if the output dataflows resulting from the transformation of any dataflow beginning with X_i are equivalent for the part of the dataflow occurring from the last moment of X_i.*

Chapter 4

Integration operators

We now introduce formal operators to integrate systems[1]. These operators make it possible to compose systems together (i.e. interconnecting inputs and outputs of various systems) and to abstract a system (i.e. changing the level of description of a system in term of granularity of dataflows). We show that these operators are consistent with the natural definitions of those operators on transfer functions, and that our definition of system is closed under those operators. The deliverable is a set of integration operators that are proven to be consistent and whose expressivity allows to model systems integration.

4.1 Composition

Composition consists in aggregating systems together in an overall greater system where some inputs and outputs of the various systems have been interconnected (it is thus directly related to the definition of the internal structure of the system). Composition requires to have a definition of the synchronization of dataflows between the different time scales of the systems considered, what has been introduced in Chapter 2. An important point is the transmission of data between systems is instantaneous through the composition operators (any delay is a modeling choice that should be defined through an intermediary system called interface).

[1]We will consider in what follows that all time scales are defined within the same time reference T. Note that this could be a restrictive assumption when considering systems like GPS satellites that leverage general relativity principles, what would typically require different time references in our formalism.

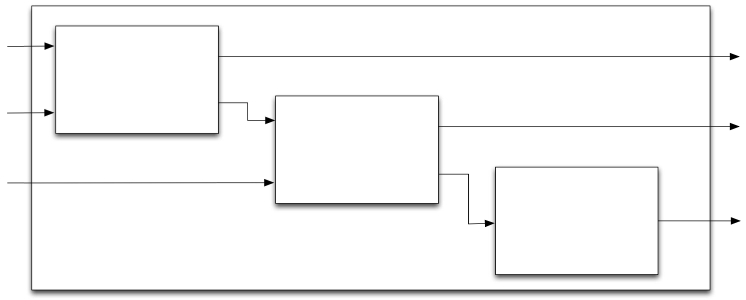


Figure 4.1: Composition of systems

We define two operators for basic systems composition: the *product* (allowing to define a new overall system from a set of systems, without interconnecting them[2]) and the *feedback* (allowing to define a new system by interconnecting an input and an output of the same system). Dividing composition into two steps allows to distinguish between the aggregation of systems, and the interconnections within the new overall system, and makes it easier to prove theorems by induction.

4.1.1 Timed extension

We first introduce a "technical" operator called *extension* that will facilitate the definition of the product by allowing to define on a shared time scale a finite[3] number of systems. The extension concentrates all technical difficulties (which are resulting from the introduction of time) in defining the composition of systems.

We define *instantaneous behaviors transition functions* associated with a system. They will make it possible to extend the transition functions of a system to any moment of time, by the introduction of virtual extension buffers for input and output in the state (so that the new state is augmented with a data of input and a data of output, memorizing respectively the data to be received and emitted, in a similar way that we have defined the projection for data flows).

Definition 4.1.1 (Instantaneous behaviors transition functions) *Let $\int = (\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, \mathcal{Q})$ be a system. We note w_i the writing function for Input and r_i, r_o the reading functions for Input and Output. Writing $x' = w_i(b_i, x)$* [4]*, we define the* ***instantaneous behaviors transition functions*** *of a system:*

[2]In spite of its appearing simplicity, defining the product is not straightforward as all systems do not share the same time scale.

[3]Since it is not possible, in a generic time reference, to ensure that an infinite union of time scales will still be a time scale, in particular because of Zeno's paradox.

[4]x' thus corresponds to the data to be read on the input.

$$\begin{array}{rcl} \tilde{\mathcal{F}} : In \times (S \times In \times Out) \times T & \longrightarrow & Out \\ \big(x, (q, b_i, b_o), t\big) & \longmapsto & \left\{ \begin{array}{ll} \mathcal{F}\big(x', q, t\big) & \textit{if } \ t \in \mathbb{T}_s \\ r_o(b_o) & \textit{else} \end{array} \right. \end{array}$$

and

$$\begin{array}{rcl} \tilde{\mathcal{Q}} : In \times (S \times In \times Out) \times T & \longrightarrow & S \times In \times Out \\ \big(x, (q, b_i, b_o), t\big) & \longmapsto & \left\{ \begin{array}{ll} \mathcal{Q}(x', q, t), r_i(x'), \mathcal{F}(x', q, t) & \textit{if } \ t \in \mathbb{T}_s \\ \big(q, x', r_o(b_o)\big) & \textit{else} \end{array} \right. \end{array}$$

The new transition functions $\tilde{\mathcal{F}}$ and $\tilde{\mathcal{Q}}$ are defined for every moment of the time reference T and work with extended states containing virtual extension buffers allowing to synchronize inputs and outputs with the time scale of the system. These new transition functions can be restricted to any time scale $\mathbb{T}$, noted $\tilde{\mathcal{F}}_{\mathbb{T}}$ and $\tilde{\mathcal{Q}}_{\mathbb{T}}$.
The extension of a system consists in defining it on a finer time scale (making it possible to define a finite number of systems on a shared time scale, i.e. the union of their time scales).

Definition 4.1.2 (Extension of a system) *Let $\int$ be a system of time scale $\mathbb{T}_s$. Let $\mathbb{T}$ be a time scale such that $\mathbb{T}_s \subseteq \mathbb{T}$. The **extension of $\int$ to** $\mathbb{T}$ is the new system*[5]*:*

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

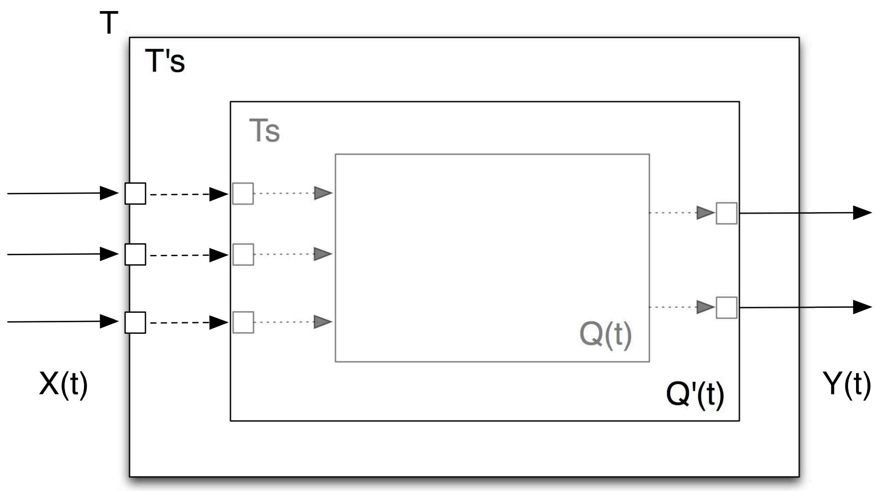


Figure 4.2: Extension of a system

[5] $S \times In \times Out$ is considered as an ϵ-alphabet whose blank symbol is $(\epsilon, \epsilon, \epsilon)$.

If a system has been extended several times, the input dataflow will be projected in several sequential virtual extension buffers (as much as the dataflows gets into the different layers of the nested extended systems) till reaching the initial time scale of the system.

Theorem 2 (Equivalence of a system by extension) *Let $\int$ be a system and $\int_{\mathbb{T}}$ be its extension to a finer time scale. Then $\int$ and $\int_{\mathbb{T}}$ have equivalent transfer functions:*

$$F_{\int} \sim F_{\int_{\mathbb{T}}}$$

Moreover, the state dataflows in their execution are equivalent when projected on the initial ϵ-alphabet of states S.

Proof Let X be an input dataflow for $\int$ and $\int_{\mathbb{T}}$. The execution of $\int$ will be defined on the projected dataflow $X_{\mathbb{T}_s}$ and the execution of $\int_{\mathbb{T}}$ will be defined on the projected dataflow $X_{\mathbb{T}}$. But $\tilde{\mathcal{F}}_{\mathbb{T}}$ and $\tilde{\mathcal{Q}}_{\mathbb{T}}$ are defined to simulate the following behavior during their execution: they project the dataflow $X_{\mathbb{T}}$ on the time scale $\mathbb{T}_s$, then compute the transitions for the system $\int$ and finally project the output dataflow of time scale $\mathbb{T}_s$ on the finer time scale $\mathbb{T}$. Thus, $\int_{\mathbb{T}}$ will in fact compute $F_{\int_{\mathbb{T}}}(X) = \big(F_{\int}\big((X_{\mathbb{T}})_{\mathbb{T}_s}\big)\big)_{\mathbb{T}}$, which by Proposition 4 is in fact $F_{\int}(X_{\mathbb{T}_s})_{\mathbb{T}}$. But as $\mathbb{T}_s \subseteq \mathbb{T}$, by Proposition 3, $F_{\int}(X_{\mathbb{T}_s})_{\mathbb{T}} \sim F_{\int}(X_{\mathbb{T}_s})$. But since $F_{\int}(X_{\mathbb{T}_s}) = F_{\int}(X)$, we finally have that $F_{\int} \sim F_{\int_{\mathbb{T}}}$.

The proof for the equivalence of the state dataflows (projected on the initial ϵ-alphabet of states S) is straightforward as S is associated with a persistent behavior. $\square$

4.1.2 Product

We now define the product of transfer functions and of systems, and show that they are mutually consistent. We first define the associative product $\otimes$ of datasets.

Definition 4.1.3 (Product of datasets) *Let $\mathcal{D}_1 = \big(D_1, (r_1, w_1)\big)$ and $\mathcal{D}_2 = \big(D_2, (r_2, w_2)\big)$ be two datasets. $\mathcal{D}_1 \otimes \mathcal{D}_2 = \big(D, (r, w)\big)$ is a new dataset called* ***product of** $\mathcal{D}_1$ **and** $\mathcal{D}_2$* *and defined by*[6]*:*

- $D = D_1 \times D_2$ [7]
- $r\big((d_1, d_2)\big) = \big(r_1(d_1), r_2(d_2)\big)$
- $w\big((d_1, d_2), (d_1', d_2')\big) = \big(w(d_1, d_1'), w(d_2, d_2')\big)$

This product of datasets is associative and allows to define an associative product of dataflows.

[6]It is easy to show that the new reading and writing functions comply with the axioms of a data behavior.

[7](ϵ, ϵ) is considered as the blank symbol ϵ.

Definition 4.1.4 (Product of dataflows) *Let X be a dataflow on ($\mathcal{D}_X$, $\mathbb{T}_X$) and Y be a dataflow on ($\mathcal{D}_Y$, $\mathbb{T}_Y$). The **product** $X \otimes Y$ **of** X **and** Y is the dataflow on ($\mathcal{D}_X \otimes \mathcal{D}_Y$, $\mathbb{T}_X \cup \mathbb{T}_Y$) defined by:*

$$\forall t \in \mathbb{T}_X \cup \mathbb{T}_Y,\ X \otimes Y(t) = \big(X_{\mathbb{T}_X \cup \mathbb{T}_Y}(t), Y_{\mathbb{T}_X \cup \mathbb{T}_Y}(t)\big)$$

We define the projection of a dataflow on a dataset, allowing to consider only a part of the aggregated datasets of the dataflow.

Definition 4.1.5 (Projection of a dataflow on a dataset) *Let $\mathcal{D} = \mathcal{D}_1 \otimes \mathcal{D}_2$ be a dataset. Let $X \in \mathcal{D}^T$ be a dataflow of time scale $\mathbb{T}_X$. The **projection of** X **on** $\mathcal{D}_i$ ($i = 1, 2$) is the dataflow $X_{\mathcal{D}_i}$ on $(\mathcal{D}_i, \mathbb{T}_X)$ defined by:*

$$\forall t \in \mathbb{T}_X,\ X_{\mathcal{D}_i}(t) = d_i \text{ where } X(t) = (d_1, d_2) \in D_1 \otimes D_2.$$

We can now define the product of transfer functions:

Definition 4.1.6 (Product of transfer functions) *Let $F_1 : Input_1{}^T \to Output_1{}^{\mathbb{T}_1}$ and $F_2 : Input_2{}^T \to Output_2{}^{\mathbb{T}_2}$ be two transfer functions. The **product of** F_1 **and** F_2 is the function $F_1 \otimes F_2 : (Input_1 \otimes Input_2)^T \to (Output_1 \otimes Output_2)^{\mathbb{T}_1 \cup \mathbb{T}_2}$ defined by:*

$$\forall X \in (Input_1 \otimes Input_2)^T,\ F_1 \otimes F_2(X) = F_1(X_{Input_1}) \otimes F_2(X_{Input_2})$$

We have a straightforward and useful property of the product of datasets and dataflows:

Proposition 5 (Decomposability of dataflows) *Let $\mathcal{D} = \mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n$ be a product of n datasets. Let $X \in \mathcal{D}^T$ be a dataflow. Then:*

$$X = X_{\mathcal{D}_1} \otimes \cdots \otimes X_{\mathcal{D}_n}$$

We also introduce a Lemma used in the proof of the next proposition.

Lemma 1 (Decomposability of dataflows for equivalence as far as) *Let $\mathcal{D} = \mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n$ be a dataset. Let $X, Y \in \mathcal{D}^T$ be two dataflows. Then:*

$$\forall t \in T:\ \big(X \sim_t Y\big) \Leftrightarrow \big(\forall i,\ X_{\mathcal{D}_i} \sim_t Y_{\mathcal{D}_i}\big)$$

We can then show that this product defines a transfer function and is associative:

Proposition 6 (Closure and associativity of the product) *$F_1 \otimes F_2$ is a transfer function and $\otimes$ on transfer functions is associative.*

Proof Let $\mathcal{D} = \mathcal{D}_1 \otimes \mathcal{D}_2$ be a dataset. Let $X, Y \in \mathcal{D}^T$ be two dataflows. Then by the previous lemma: $\forall t \in T:\ \big(X \sim_t Y\big) \Leftrightarrow \big(\forall i,\ X_{\mathcal{D}_i} \sim_t Y_{\mathcal{D}_i}\big)$. Since transfer functions are causal: $\forall t \in T:\ \big(X \sim_t Y\big) \Rightarrow \big(\forall i,\ F_i(X_{\mathcal{D}_i}) = F_i(Y_{\mathcal{D}_i})\big)$ and thus $F_1 \otimes F_2(X) = F_1 \otimes F_2(Y)$, which means that $F_1 \otimes F_2$ is a transfer function. □

We finally define the product of n systems (sharing the same time scale) as the new system resulting from the aggregation of those systems (called "subsystems" of the new system)[8].

Definition 4.1.7 (Product of systems sharing the same time scale) *Let $(\int^i)_i = (\mathbb{T}_s, Input_i, Output_i, S_i, q_{0i}, \mathcal{F}_i, \mathcal{Q}_i)_i$ be n systems of time scale $\mathbb{T}_s$. The* ***product*** *$\int^1 \otimes \cdots \otimes \int^n$ is the system $\big(\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, \mathcal{Q}\big)$ where:*

- *$Input = Input_1 \otimes \cdots \otimes Input_n$ and $Output = Output_1 \otimes \cdots \otimes Output_n$*
- *$S = S_1 \times \cdots \times S_n$ and $q_0 = (q_{01}, \ldots, q_{0n})$*
- *$\mathcal{F}\big((x_1, \ldots, x_n), (q_1, \ldots, q_n), t\big) = \big(\mathcal{F}_1(x_1, q_1, t), \ldots, \mathcal{F}_n(x_1, q_1, t)\big)$*
- *$\mathcal{Q}\big((x_1, \ldots, x_n), (q_1, \ldots, q_n), t\big) = \big(\mathcal{Q}_1(x_1, q_1, t), \ldots, \mathcal{Q}_n(x_1, q_1, t)\big)$*

The product is easily generalized to systems with different time scales thanks to the extension: we first extend the systems to the union of their time scales (which is a time scale since there are a finite number of time scales), and then we make the product following our definition of the product for systems defined on a shared time scale:

Definition 4.1.8 (Product of systems) *Let $(\int^i)_i = (\mathbb{T}_i, Input_i, Output_i, S_i, q_{0i}, \mathcal{F}_i, \mathcal{Q}_i)_i$ be n systems. Let $\mathbb{T} = \mathbb{T}_1 \cup \cdots \cup \mathbb{T}_n$ be the union of their time scales. The* ***product*** *$\int^1 \otimes \cdots \otimes \int^n$ is the system $(\int^1)_{\mathbb{T}} \otimes \cdots \otimes (\int^n)_{\mathbb{T}}$*

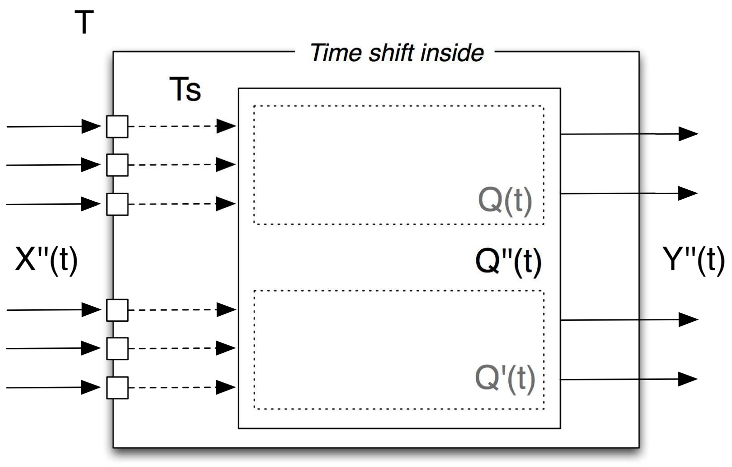


Figure 4.3: Product of systems

[8]Defining the product of n systems (and not just 2) is useful to give a semantics to the notion of subsystem, since defining it only for 2 would make it necessary to make sequential products of 2 systems to simulate the product of n systems, therefore making it difficult to define the notion of "subsystem" of a given system.

Theorem 3 (Consistency of the product of systems) *The transfer function of the product of n systems $(\int_i)$ is equivalent to the product of their transfer functions:*

$$F_{\int_1 \otimes \cdots \otimes \int_n} \sim F_{\int_1} \otimes \cdots \otimes F_{\int_n}$$

Proof If the n systems share the same time scale, the proof is straightforward by definition of the product of systems (and the equivalence is in fact an equality). If not: we consider the extension of the n systems to the union of their time scales. By Theorem 2, the corresponding transfer functions are equivalent, and finally the equivalence stated in this Theorem is straightforward. $\square$

4.1.3 Feedback

The feedback consists in defining a new system by connecting one of the output of a system to one of the input of the same system, when it is possible (e.g. the input and the output share the same dataset[9]). However, it is not always possible to feedback an output on an input of same dataset: to define recursively the feedback of a system in a constructive way and express it as a new system with transition functions, it is necessary to establish the noninstantaneous influence of the input on the output concerned.

The feedback introduces indeed troubles because it is no more possible, when computing the output of a system, to consider the input, since both are connected together in a loop. To define the feedback, it is therefore necessary to establish the existence[10] and unicity[11] of the mathematical object resulting from the feedback of one output to one input of a given system.

We first introduce a natural definition of the feedback on transfer functions as a fixed point of dataflows, and show later that it captures the constructive feedback we define on systems.

Definition 4.1.9 (Feedback of a transfer function) *Let F be a transfer function of time scale $\mathbb{T}$ on signature $(D \otimes A, D \otimes B)$, such that: $\forall X \in A^T,\ \exists!\, Y_X \in D^{\mathbb{T}_s},\ F\big(Y_X \otimes X\big)_D = Y_X$. The* ***feedback of*** *F* ***on*** *D is the new transfer function $fb_{(F,D)}$ of time scale $\mathbb{T}_s$ on signature (A, B) defined by:*

$$\forall X \in A^T,\ fb_{(F,D)}(X) = F\big(Y_X \otimes X\big)_B$$

This definition of the feedback captures formally the notion of feedback introduced above.

Proposition 7 (Equivalence of feedback on a finer time scale) *Let F be a transfer function and $F_{\mathbb{T}}$ be an extension of F to a finer time scale $\mathbb{T}$ (i.e.*

[9] Sharing only the same ϵ-alphabet is not enough since having different data behaviors would make different resulting feedbacked systems according to the extension considered for the initial system.

[10] i.e. in spite of the chicken/egg problem induced by the loop, it is possible to define a consistent dataflow of the loop, and therefore we can describe a consistent evolution of the new system with the input and the output connected being equals at any time

[11] i.e. there is only one consistent evolution of the initial system with the feedback

$\mathbb{T}_F \subset \mathbb{T}$). Then, $fb_{(F,D)}$ exists if, and only if $fb_{(F_{\mathbb{T}},D)}$ exists, and in this case we have:

$$fb_{(F,D)} \sim fb_{(F_{\mathbb{T}},D)}$$

Proof As $F \sim F_{\mathbb{T}}$ and as the feedbacked input and output share the same data behaviors, Y_X for F will work for $F_{\mathbb{T}}$ by considering $\big(Y_X\big)_{\mathbb{T}}$, and conversely. □

We now define the feedback of a system in a constructive way:

Definition 4.1.10 (Feedback of a system) *Let $\int = \big(\mathbb{T}_s, (D \times In, \mathcal{I}), (D \times Out, \mathcal{O}), S, q_0, \mathcal{F}, \mathcal{Q}\big)$ be a system such that there is no instantaneous influence of dataset D from the input to the output*[12]*, i.e. $\forall t \in \mathbb{T}_s, \forall x \in In$, $\forall d \in D$, $\mathcal{F}\big((d,x),q,t\big)_D = \mathcal{F}\big((\epsilon,x),q,t\big)_D$. The **feedback of** D **in** $\int$ is the system $\int_{FB(D)} = \big(\mathbb{T}_s, (In, \mathcal{I}'), (Out, \mathcal{O}'), S, q_0, \mathcal{F}', \mathcal{Q}'\big)$ with:*

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

where $d_{x,q,t}$ stands for $\mathcal{F}\big((\epsilon,x),q,t\big)_D$.

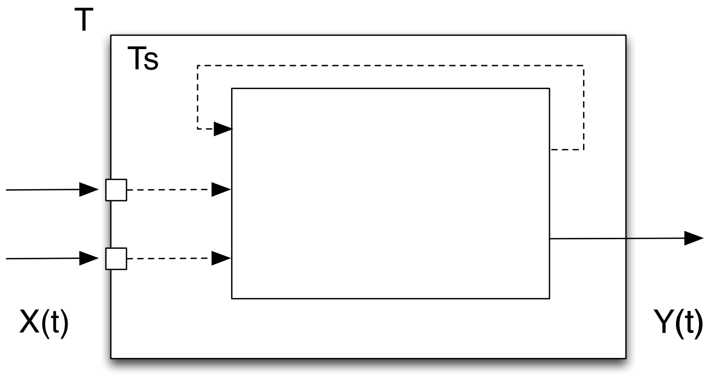


Figure 4.4: Feedback of a system

A well-spread practice in Systems Engineering is to feedback a system with an interface to model properties of the real link (delay of transmission, errors, etc).

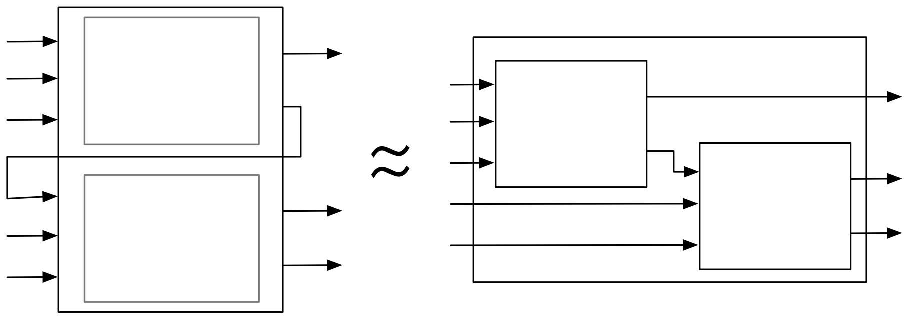
Theorem 4 (Consistency of the feedback on systems) *The transfer function of the feedback of a system (when it exists) is the feedback of the transfer function of this system:*

$$F_{\int_{FB(D)}} = fb_{(F_{\int},D)}$$

[12]As explained informally before, this condition makes it possible to define a unique feedback, in a constructive way, i.e. without having to solve a fixed point equation that could lead to zero or multiple solutions whose construction is not mechanistic.

Proof We easily show by induction that the feedbacked dataflow constructed in the definition of a feedbacked system is a fixed point for the initial transfer function. □

Together, product & feedback allow to compose 2 systems together by interconnecting their inputs & outputs:



4.2 Abstraction

The abstraction allows to define from a system a more abstract system, so that it can be integrated into more global ones. Abstraction allows to consider the right systemic level to describe a system, according to modeling needs, and is thus a fundamental tool to deal with the complexity of systems by hiding unnecessary low-level details related to the behavior of the system. It helps people to better understand a system and makes easier the formal analysis by working on abstraction of systems (see [25] for abstract interpretation which is a well-known example of abstraction), but it can also be necessary for computational complexity matters or mathematical reasons.

The abstraction can be understood as a “zoom out” from the point of view 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 overall behavior. For instance, a computer may be considered as an electronic device with electrical signals. However, we generally abstract this electronic device into a more abstract device able to process complex data as emails.

Abstraction is fundamental for Systems Engineering: it allows to consider the right systemic level to describe a system on given aspects. It is a fundamental tool to deal with the complexity of systems. Abstraction makes it possible to simplify time, states, data, multiple inputs/outputs, and behaviors of a system. It is especially necessary when a system is hard to describe because it is resulting from the composition of numerous subsystems. The reverse process, i.e. *concretization*, allows to describe first the system from a high level point of view, and progressively to detail the analysis and the decomposition of the dataflows and behaviors of the system (what is generally correlated with decomposition, i.e. the inverse operator of composition to break down a system into subsystems).

Before defining abstraction, we need to explain how to deal in a simple way with nondeterminism in our framework, since abstraction can bring nondeterminism to originally deterministic systems, and we want to ensure the closure of our definition of systems under all integration operators.

4.2.1 Nondeterminism

Nondeterministic behaviors can be modeled in a system in the minimalist way that follows: one of the input $\mathcal{E}$ can be used as an oracle (or a dataflow of *events*), i.e. an input giving information to the system to make its transitions (functional and states).

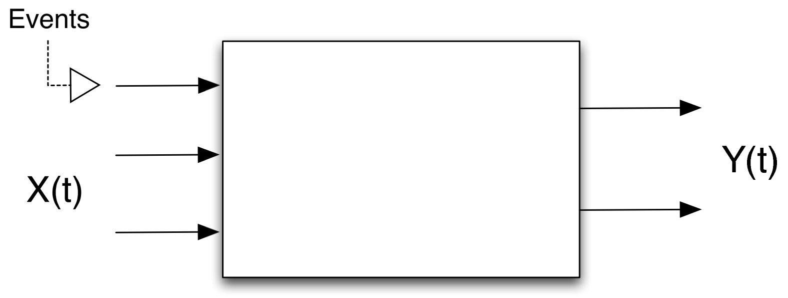


Figure 4.5: A nondeterministic system within our formalism

It can simulate classical nondeterminism of Mealy machines (where functional and states behaviors have their value in nonempty subsets of the target datasets of their deterministic version) by indicating at each step which element to chose within this subset (so that the nondeterministic behavior is simulated by a dataflow of events within a deterministic system). This flow can also be understood as the formalization of the imperfection/underspecification of any deterministic model[13]. It is therefore possible to take into account this imperfection by considering that transitions can be influenced by specific events carried by the oracle.

This kind of "events dataflow" typically corresponds to the events used in States diagrams in modeling languages like SysML (where an event as "the water tank is full" will in fact correspond to expressing in a deterministic way a nondeterministic event that cannot be computed from the current input or state of the system).

Note that we are only focusing on a minimalist introduction of nondeterministic behaviors corresponding to the one needed to be able to properly define abstraction of systems, which generates such behaviors..

[13]Note that this imperfection & underspecification can be on purpose, to define a simpler system. In practice, the possibility to model deterministic behaviors of a real system is often restricted by the limited grain of description of the real system state observed.

4.2.2 Abstraction

The abstraction of a dataflow consists in defining a new dataflow on a more abstract dataset and on a more abstract time scale (typically with a larger step).

Definition 4.2.1 (Abstraction/concretization of dataflows) *An **abstraction** of dataflows is a surjective function $A : {D_c}^{\mathbb{T}_c} \to {D_a}^{\mathbb{T}_a}$ which is causal:*

$$\forall X, Y \in {D_c}^{\mathbb{T}_c},\ \forall t \in T,\ \big(X \sim_t Y\big) \Rightarrow \big(A(X) \sim_t A(Y)\big)$$

*The associated **concretization** is the function $C : {D_a}^{\mathbb{T}_a} \to \mathcal{P}\big({D_c}^{\mathbb{T}_c}\big)$ defined by $C(X) = A^{-1}(\{X\})$.*

We remark that an abstraction/concretization of dataflows is in fact a partition of the concrete dataflows whose elements are indexed by the abstract dataflows.

Example 14 We can take the example of a computer whose LAN connection is described by an input dataflow of bits on a regular time scale of step 10^{-6} sec, i.e. $D_c = \{0, 1, \epsilon\}$ and $\mathbb{T}_c = \tau\mathbb{N}$ with $\tau = 0.000001$. We can abstract this dataflow to a higher-level dataflow on $D_a = \{email, file, picture, video, html, \epsilon\}$ on time scale $\mathbb{T}_a = \tau'\mathbb{N}$ with $\tau' = 0.01$. ◊

The abstraction of a transfer function is a new transfer function working on abstract dataflows, with nondeterministic behaviors modeled by events dataflows (explained below in Example 15).

Definition 4.2.2 (Abstraction of a transfer function) *Let $F : Input^T \to Output^{\mathbb{T}_s}$ be a transfer function. Let $A_i : Input^{\mathbb{T}_s} \to {Input_a}^{\mathbb{T}_a}$ be an abstraction for input dataflows and $A_o : Output^{\mathbb{T}_s} \to {Output_a}^{\mathbb{T}_a}$ an abstraction for output dataflows. The **abstraction of** F for input and output abstractions (A_i, A_o) with events $\mathcal{E}$ is the new transfer function*

$$F_a : (Input_a \otimes \mathcal{E})^T \to {Output_a}^{\mathbb{T}_a}$$

defined by:

$$\forall X \in Input^T,\ \exists E \in \mathcal{E}^{\mathbb{T}_a},\ F_a\big(A_i(X_{\mathbb{T}_s}) \otimes E\big) = A_o\big(F(X)\big)$$

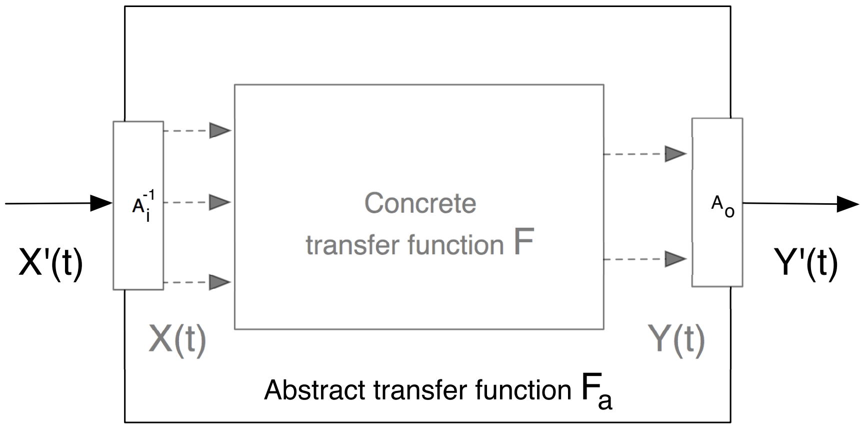

X'(t)
A_i^{-1}
Concrete transfer function F
A_o
Y'(t)
X(t)
Y(t)
Abstract transfer function F_a

Figure 4.6: Abstraction of a transfer function

Thus, the following diagram commutes (we dismiss events here):

$$\begin{tikzcd}
Input^{\mathbb{T}_s} \arrow[r, "F"] \arrow[d, "A_i"'] & Output^{\mathbb{T}_s} \arrow[d, "A_o"] \\
{Input_a}^{\mathbb{T}_a} \arrow[r, "F_a"'] & {Output_a}^{\mathbb{T}_a}
\end{tikzcd}$$

We now define the abstraction of a system.

Definition 4.2.3 (Abstraction of a system) *Let $\int = \big(\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, \mathcal{Q}\big)$ be a system. $\int' = \big(\mathbb{T}_a, Input_a \otimes \mathcal{E}, Output_a, S_a, q_{a0}, \mathcal{F}_a, \mathcal{Q}_a\big)$ is an abstraction of $\int$ for input and output abstractions (A_i, A_o) if, and only if: $\exists A_q : S^{\mathbb{T}_s} \to {S_a}^{\mathbb{T}_a}$, for all execution (X, Q, Y) of $\int$, $\exists E \in \mathcal{E}^{\mathbb{T}_a}$, $\big(A_i(X_{\mathbb{T}_s}) \otimes E, A_q(Q), A_o(Y)\big)$ is an execution of $\int'$. Conversely, $\int'$ is a concretization of the system $\int$.*

Indeed, an abstraction consists in abstracting inputs, states and outputs dataflows in the execution of a system, and to define on these abstract dataflows a new system that will have abstract behaviors corresponding to the initial behaviors of the initial system. A good abstraction will be based on dataflows abstraction which will define consistent transitions in the abstract system for states and outputs. However, nondeterministic behaviors (modeled by events dataflow E) will generally appear in the abstract system. It is a consequence of regrouping states and input/output data in more abstract ϵ-alphabets, making it impossible to express the abstract behaviors as deterministic transitions on those ϵ-alphabets (for instance, one abstract data may correspond to several concrete data sometimes resulting in several behaviors of the concrete system, and the same may occur for the states). The abstraction of a deterministic system may thus result in nondeterministic behaviors, what does not mean that the real system modeled is nondeterministic. Actually, determinism is not an intrinsic property of a system, it depends on the level of description of its behavior. It is thus important to understand that determinism is a property of a model, not of a real object, and that various consistent models modeling the same real system may not be all deterministic or nondeterministic.

Example 15 [Nondeterministic behaviors of the abstraction of systems] We consider a glass whose state is described by an integer between 0 and 100 modeling the solidity of the glass (0 means broken). This glass can receive physical forces which lower its solidity till it is broken. At this level, the glass is described as a deterministic system. If we consider an abstraction of this model, we may consider the glass has being broken or not (two states) and receiving a shock (i.e. a sequence of physical forces) or nothing. When the glass, not broken yet, receives a shock, it will sometimes become broken, and sometimes remain not broken, depending of the previously received shocks. Therefore, at this level of abstraction, the glass has nondeterministic behaviors (since a shock may break it, with parameters that cannot be explained at this abstraction level). $\Diamond$

Theorem 5 (Consistency of the abstraction of a system) *The transfer function of the abstraction of a system is the abstraction of the transfer function of this system.*

Proof The proof is straightforward since the abstraction of a system is defined exactly as abstracting the transfer function of the initial system. $\square$

4.3 Integration of systems

The integration is a recursive mechanism to build a system through the synthesis of smaller subsystems working together.

4.3.1 Composition & abstraction

The *integration* of systems in our framework consists in composing together a finite set of systems, with product (P) and feedback (F), then applying the abstraction (A) to describe the resulting system at a more abstract level, and repeating those steps recursively till reaching the target overall system. We believe that the recursive *integration* of real systems (as done in Systems Engineering) can be modeled consistently as the corresponding integration of systems in our framework, using only P/F/A. We thus introduce a modeling postulate:

Postulate 1 (Real integration can be modeled with P/F/A) *Any real system $\int^r$ resulting from the "real" integration of elementary real systems $(\int_i^r)$ can be consistently modeled as a system $\int$ resulting from recursive applications of operators P/F/A on the elementary systems $(\int_i)$ (modeling the elementary real systems $(\int_i^r)$).*

One can remark that we only provided operators to integrate systems together. In reality, systems design involves mixing both bottom-up and top-down approaches. However, the same operators still hold, as the top-down approach can be interpreted as finding the right subsystems that, integrated together, are equivalent to the higher level system.

We will introduce in the Chapter 6 a formalism to capture the internal structure of a system induced by those integration operators.

4.3.2 Example

We now give an example of an integrated system resulting from the integration of elementary systems (i.e. components). We will here work on simple transfer functions that can be naturally expressed as systems.

Example 16 [Integrated system] It has been shown in the paper [12] that any Hamiltonian system can be modeled within the framework introduced in this paper. We recall the example of a Water Tank given in [12] and already introduced in Chapter 3. We will show how to define a model of a water tank from the integration of elementary systems.

We work in the time reference $^*\mathbb{R}$ of nonstandard real numbers. Let us fix first some regular continuous time scale $\mathbb{T}$ with infinitesimal time step τ that will be used to define all the systems of our example.

We shall therefore consider the system consisting of a water tank where water arrives at a variable rate $w_i(t) \geq 0$ (with $t \in \mathbb{T}$) through one single pipe. The water leaves through another (output) pipe at rate $w_o(t)$ (with $t \in \mathbb{T}$) controlled by a valve (see Figure 4.7-*a*) whose position will be given by $v(t) \in [0, 1]$ (with $t \in \mathbb{T}$), 0 and 1 modeling respectively here the fact that the valve is closed or open. The initial position at time $t = 0$ of the valve is equal to some constant $v(0) = V_0$. The maximal throughput capacity of the output pipe is C, thus its actual throughput at each moment $t \in \mathbb{T}$ is $C\,v(t)$. The valve is controlled by a sensor measuring the level $l(t)$ of water in the tank, which aims at keeping this level in a given interval $[L_1, L_2]$ (the initial water tank level L_0 belongs therefore to this interval). For simplicity, we assume that there is always enough water in the tank to saturate the output pipe and that the incoming flow does not exceed the output pipe's capacity, i.e. that one has always $\max(w_i(t),\ t \in \mathbb{T}) \leq C$.

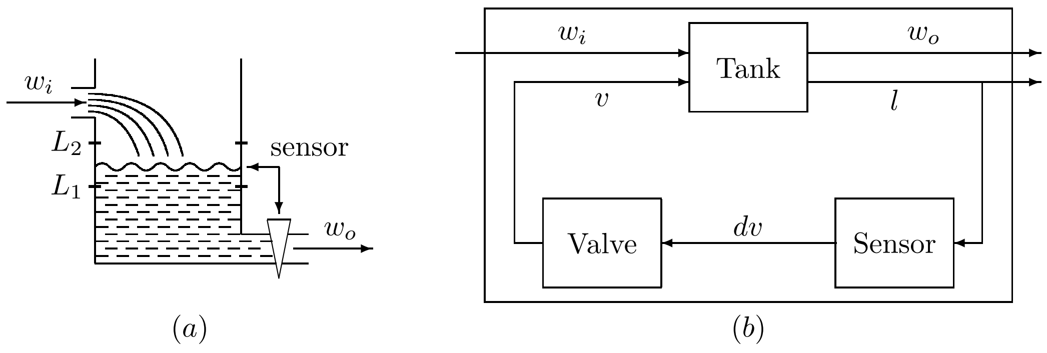


Figure 4.7: The Water Tank (a) and an associated simplified system (b).

The overall system has the input space $In = \overline{[0, C]}$ (incoming flow rate) and the output space $Out = \overline{[0, C]} \times \overline{[L_1, L_2]}$ (output flow rate and current water level in the tank). It can be modeled as the composition of the three following systems (see Figure 4.7-b) working on datasets of persistent behaviors:

1. The *tank* transfer function, T, taking on input the current values of the incoming water flow $w_i(t)$ and the position v(t) of the valve and sending on its output the corresponding output water flow $w_o(t)$ and water level $l(t)$ according to the following equations:

$$w_o(0) = C\,V_0,\quad w_o(t+\tau) = C\,v(t) \ \text{ for every } t \in \mathbb{T}^*,$$
$$l(0) = L_0,\quad l(t+\tau) = l(t) + (w_i(t) - w_o(t))\,\tau \ \text{ for every } t \in \mathbb{T}^* .$$

   The input and output spaces of T are then $In_T = \overline{[0, C]} \times \overline{[0, 1]}$ and $Out_T = \overline{[0, C]} \times \overline{[L_1, L_2]} \times \overline{[L_1, L_2]}$ (one of the output is duplicated).

2. The *sensor* transfer function, S, taking on input the water level $l(t)$ and sending on its output a valve position adjustment $dv(t)$ defined from some given equation as follows [14] :

$$dv(0) = 0,\quad dv(t+\tau) \ = \ \operatorname{sign}\left(l(t) - \frac{L_1 + L_2}{2}\right)\tau \ \text{ for every } t \in \mathbb{T}^* .$$

   The input and output spaces of S are then $In_S = \overline{[L_1, L_2]}$ and $Out_S = \overline{\{-\tau, 0, \tau\}}$.

3. The *valve* transfer function, V, taking on input the adjustment $dv(t)$ and providing on its output $v(0) = V_0$ at time $t = 0$ and the value $v(t)$ given by the following formula:

$$v(t+\tau) = \begin{cases} v(t) + dv(t) & \text{if } v(t) + dv(t) \in [0, 1]\,, \\ v(t) & \text{if } v(t) = 0 \wedge dv(t) = -\tau \text{ or } v(t) = 1 \wedge dv(t) = \tau, \end{cases}$$

[14] Where sign denotes the function from $\mathbb{R}$ into $\{-1, 0, 1\}$ defined by setting $\operatorname{sign}(0) = 0$, $\operatorname{sign}(x) = 1$ for every $x > 0$ and $\operatorname{sign}(x) = -1$ for every $x < 0$.

for $t \in \mathbb{T}^*$. The input and output spaces of V are then $In_V = \overline{\{-\tau, 0, \tau\}}$ and $Out_V = \overline{[0,1]}$.

As one can see on Figure *b*, the transfer function of the full tank water system can be obtained from:

- the product $P = T \otimes S \otimes V$ of signature $\big(\overline{[0,C]} \times \overline{[0,1]} \times \overline{[L_1,L_2]} \times \overline{\{-\tau,0,\tau\}}, \overline{[0,C]} \times \overline{[L_1,L_2]} \times \overline{[L_1,L_2]} \times \overline{\{-\tau,0,\tau\}} \times \overline{[0,1]}\big)$
- the feedback $F_1 = fb_{(P,\overline{[L_1,L_2]})}$ of signature $\big(\overline{[0,C]}\times\overline{[0,1]}\times\overline{\{-\tau,0,\tau\}}, \overline{[0,C]}\times \overline{[L_1,L_2]} \times \overline{\{-\tau,0,\tau\}} \times \overline{[0,1]}\big)$
- the feedback $F_2 = fb_{(F_1,\overline{\{-\tau,0,\tau\}})}$ of signature $\big(\overline{[0,C]}\times\overline{[0,1]}, \overline{[0,C]}\times\overline{[L_1,L_2]}\times \overline{[0,1]}\big)$
- the feedback $F_3 = fb_{(F_2,\overline{[0,1]})}$ of signature $\big(\overline{[0,C]}, \overline{[0,C]} \times \overline{[L_1,L_2]}\big)$, resulting in the final overall target system.

$\Diamond$

Part II

Systems architecture

Chapter 5

A logic for requirements

Requirements are a key part of systems architecture. They are used to define expected properties of systems behaviors[1], and can be defined as a logical formula using systemic variables & time. In this chapter, we introduce an example of a logic to model such requirements, based on a coalgebraic definition of systems. Defining the logic on coalgebras makes it possible to introduce classic results (like bissimulation and adequacy) and makes the subsequent proofs easier.

5.1 A coalgebraic definition of systems

We first introduce another way to define systems via coalgebras. This definition is just the expression using coalgebraic formalism & notations of our functional definition of systems introduced in Chapter 3. We will use it to define a logic on coalgebraic systems.

5.1.1 Preliminaries

We will use many terms and notations from the coalgebraic theory. To help the intuition, we recall in this subsection some coalgebraic notions that will be used in this chapter. A coalgebra can be seen as an abstraction of transition systems (i.e. automatas) of all kinds. A coalgebra consists of a set S equipped with a transition function $\alpha : S \to \mathcal{F}(S)$ where $\mathcal{F} : Set \to Set$ is an endofunctor on the category Set of sets, defining the signature of the co-algebra α. Hence, α provides the set of states S with some structures. Unlike algebraic operations that enable to recursively build complex objects from basic objects given by signatures, coalgebraic operations are means to observe system states. More formally, we have:

Definition 5.1.1 (Coalgebra) *Let $\mathcal{F} : Set \to Set$ be an endofunctor, called* **signature functor**. *A* **coalgebra** *for $\mathcal{F}$ or $\mathcal{F}$-***coalgebra**, *is any pair (S, α) where:*

[1] We are thus dealing here with what is generally called *functional* requirements.

- *S is a set, elements of which are called* **states**,
- *$\alpha : S \to \mathcal{F}(S)$ is a mapping, called* **transition mapping**.

Coalgebras are well-adapted to define infinite data structures such as streams and infinite lists over an alphabet A, as well as all kinds of automatas. Hence, coalgebras for the signature functors $A \times \_ : S \mapsto A \times S$ and $A \times \_ + 1 : S \mapsto A \times S + 1$ are respectively streams and infinite lists over an alphabet A, whilst coalgebras for the signature functors $O \times \_^I : S \mapsto O \times S^I$, $(O \times \_)^I : S \mapsto (O \times S)^I$ and $(2^{\_})^I : S \mapsto (2^S)^I$ are respectively Moore, Mealy and nondeterministic automatas.

Definition 5.1.2 (Coalgebra morphism) *Let (S, α) and (S', α') be two coalgebras over a same signature functor $\mathcal{F}$. A* **coalgebra morphism** *is a mapping $f : S \to S'$ such that the following diagram commutes:*

$$\begin{tikzcd}
S \arrow[r, "f"] \arrow[d, "\alpha"'] & S' \arrow[d, "\alpha'"] \\
\mathcal{F}(S) \arrow[r, "\mathcal{F}(f)"'] & \mathcal{F}(S')
\end{tikzcd}$$

Given a signature functor $\mathcal{F}$, $\mathcal{F}$-coalgebras and coalgebra morphisms clearly form a category noted $CoAlg(\mathcal{F})$.

An important notion in the categorical theory of coalgebras is the characterization of the final coalgebra that contains all systems behaviors (i.e. traces).

Definition 5.1.3 (Final coalgebra) **A final $\mathcal{F}$-coalgebra** (Γ, π) *is a $\mathcal{F}$-coalgebra such that for every $\mathcal{F}$-coalgebra (S, α) there is a unique coalgebra morphism $!_\alpha : (S, \alpha) \to (\Gamma, \pi)$, that is:*

$$\begin{tikzcd}
S \arrow[r, "!_\alpha"] \arrow[d, "\alpha"'] & \Gamma \arrow[d, "\pi"] \\
\mathcal{F}(S) \arrow[r, "\mathcal{F}(!_\alpha)"'] & \mathcal{F}(\Gamma)
\end{tikzcd}$$

A final coalgebra, when it exits[2], is unique up to isomorphism. A final coalgebra can be seen as a maximal representation of a system, that is which contains all system behaviors.

[2]It has been shown in [55] that every signature functor that is built from both constant and identity functors and is closed under sum and product of functors, function space functor ($F(S) = S^A$) and finite powerset functor ($F(X) = 2^X$) has a final coalgebra.

5.1.2 Transfer functions via coalgebras

Given a timescale, let us define the mapping $\_^{\mathbb{T}} : T \to \mathbb{T}$ that from $d \in T$ yields the latest $d' \in \mathbb{T}$ such that $d \succeq d'$. Hence, $d^{\mathbb{T}}$ is defined by: $\begin{cases} d & \text{if } d \in \mathbb{T} \\ pred^{\mathbb{T}}(d) & \text{otherwise} \end{cases}$

The following definition means that dataflows can be observed at any instant of time although their values only change at instants in their time scale.

Definition 5.1.4 (Snapshots) *Let T be a time reference and $\mathbb{T} \subseteq T$ be a time scale. Let f be a $\mathbb{T}$-dataflow over A and let $d \in T$ be an instant. The* **snapshot of f at time** *d, denoted $f :: d$, is the element $f(d^{\mathbb{T}})$ of A.*

Similarly to Rutten's work in [56], we will show that the behavior of systems can be characterized by causal functions mapping infinite inputs to infinite outputs sequences. Hence, observable behaviors of systems are given by causal transfer functions (equivalent to the transfer functions defined in Chapter 3):

Definition 5.1.5 (Transfer function) *Let In and Out be two datasets denoting, respectively, the values in input and in output. Let T be a time reference. Let $\mathbb{T} \subseteq T$ be a time scale. A function $\mathcal{F} : In^T \to Out^{\mathbb{T}}$ is a* **transfer function** *if, and only if it is causal, that is:*

$$\forall d \in T, \forall f, g \in In^T, (\forall d' \preceq d \in T, f :: d' = g :: d') \Rightarrow \mathcal{F}(f) :: d = \mathcal{F}(g) :: d$$

Then, let us define the mapping $\_{}_{\mathbb{T}} : T \to \mathbb{T}$ that from $d \in T$ yields the least $d' \in \mathbb{T}$ such that $d' \succeq d$. Hence, $d_{\mathbb{T}}$ is defined by: $\begin{cases} d & \text{if } d \in \mathbb{T} \\ succ^{\mathbb{T}}(d) & \text{otherwise} \end{cases}$

We introduce the technical notions of derivative dataflow and derivative function that will be useful to build final systems.

Definition 5.1.6 (Derivative dataflow) *Let T be a time reference. Let $\mathbb{T} \subseteq T$ be a time scale. Let f be a $\mathbb{T}$-dataflow over a set A. Let $d \in T$ be a moment. The $\mathbb{T}$-dataflow f_d* **derivative** *of f* **at** *d is defined by:*

- $\forall d' \prec d_{\mathbb{T}} \in \mathbb{T}, f_d(d') = f(d')$
- $\forall d' \succeq d_{\mathbb{T}} \in \mathbb{T}, f_d(d') = f(succ^{\mathbb{T}}(d'))$

Definition 5.1.7 (Derivative function) *Let $\mathcal{F} : In^T \to Out^{\mathbb{T}}$ be a transfer function. For every input $i \in In$ and every time $d \in \mathbb{T}$, we define the* **derivative function** *$\mathcal{F}_{(i,d)} : In^T \to Out^{\mathbb{T}}$ for every dataflow $f : \mathbb{T}' \to In$ with $\mathbb{T}' \subseteq T$ by:*

$$\mathcal{F}_{(i,d)}(f) = \mathcal{F}((i,d) : f)_d$$

where $(i,d) : f : \mathbb{T}' \to In$ is the dataflow defined from f as follows:

- $\forall d' \prec d_{\mathbb{T}'} \in \mathbb{T}', (i,d) : f(d') = f(d')$
- $(i,d) : f(d_{\mathbb{T}'}) = i$

- $\forall d' \succ d_{\mathbb{T}'} \in \mathbb{T}', (i,d) : f(d') = f(pred^{\mathbb{T}'}(d'))$

Proposition 8 *For every transfer function $\mathcal{F} : In^T \to Out^{\mathbb{T}}$, $\mathcal{F}_{(i,d)}$ is a transfer function.*

Proof Let $d_1 \in T$, let $f, g \in In^T$ such that for every $d_2 \preceq d_1 \in T$, $f :: d_2 = g :: d_2$. By construction, we also have that $((i,d) : f) :: d_2 = ((i,d) : g) :: d_2$. As $\mathcal{F}$ is causal, we can then write that $\mathcal{F}((i,d) : f) :: d_1 = \mathcal{F}((i,d) : g) :: d_1$.
Here, two cases have to be considered:

1. $d_1 \neq d$. In this case, we can directly conclude by $\mathcal{F}((i,d) : f)_{|d} :: d_1 = \mathcal{F}((i,d) : g)_{|d} :: d_1$.
2. $d_1 = d$. By construction, $((i,d) : f) :: d = ((i,d) : g) :: d$, and $((i,d) : f) :: succ^{\mathbb{T}}(d) = ((i,d) : g) :: succ^{\mathbb{T}}(d)$ since $f :: d_1 = g :: d_1$. We can then conclude $\mathcal{F}((i,d) : f) :: succ^{\mathbb{T}}(d) = \mathcal{F}((i,d) : g) :: succ^{\mathbb{T}}(d)$, i.e $\mathcal{F}((i,d) : f)_{|d} :: d_1 = \mathcal{F}((i,d) : g)_{|d} :: d_1$.

$\square$

5.1.3 Systems as coalgebras

We now rewrite our definition of systems from Chapter 3 using our coalgebraic formalism:

Definition 5.1.8 (Systems) *Let In and Out be two datasets denoting, respectively, the values in inputs and outputs. Let T be a time reference. A* **system** *$\mathcal{S}$ is defined by a coalgebra (S, α) for the signature $\mathcal{H} = (Out \times \_)^{In \times \mathbb{T}} : Set \to Set$ where $\mathbb{T} \subseteq T$ is the time scale of $\mathcal{S}$, together with a distinguished element q_0 denoting the initial state of the system $\mathcal{S}$. A system $\mathcal{S}$ is called a* **pre-system** *when its initial state is removed.*

Any deterministic Mealy machine can be represented in our formalism. Indeed, given a Mealy machine (S, α) with $\alpha : S \to (Out \times S)^{In}$, we can define the equivalent pre-system $\mathcal{S} = (S, \alpha')$ over the signature $(Out \times \_)^{In \times \omega}$ by: $\forall n < \omega, \forall i \in In, \forall q \in Q, \alpha'(q)(i,n) = \alpha(q)(i)$. More generally, all examples of systems from Chapter 3 can be easily translated to our coalgebraic formalism.

In the following, given a system $((S, \alpha), q_o)$ over a signature $\mathcal{H} = (Out \times \_)^{In \times \mathbb{T}}$, we will note $\alpha(q)(i,d)_1$ (resp. $\alpha(q)(i,d)_2$) the resulting output value (resp. resulting state) of the pair $\alpha(q)(i,d)$.

Definition 5.1.9 (Category of systems) *Let $\mathcal{S} = ((S, \alpha), q_0)$ and $\mathcal{S}' = ((S', \alpha'), q'_0)$ be two systems over $\mathcal{H}$. A* **system morphism** *$h : \mathcal{S} \to \mathcal{S}'$ is a coalgebra homomorphism $h : (S, \alpha) \to (S', \alpha')$ such that $h(q_0) = h(q'_0)$.*
We note $Sys(\mathcal{H})$ (resp. $PSys(\mathcal{H})$) the **category of systems** *(resp.* **of pre-systems***) over $\mathcal{H}$.*

Below, we give a classical result over category of systems: the existence of a terminal system. This last point will be useful to give a trace model to systems via transfer functions.

Theorem 6 *Let $\mathcal{H} = (Out \times \_)^{In \times \mathbb{T}}$ be a signature. Let Γ be the set of all transfer functions $\mathcal{F} : In^T \to Out^{\mathbb{T}}$. Let $\pi : \Gamma \to (Out \times \Gamma)^{In \times \mathbb{T}}$ defined for every $\mathcal{F} : In^T \to Out^{\mathbb{T}}$ and every $i \in In$ and every $d \in \mathbb{T}$ by $\pi(\mathcal{F})(i,d) = (\mathcal{F}((i,d) : f)(d), \mathcal{F}_{(i,d)})$ where $f \in In^T$ is arbitrary*[3]*. Then, the pre-system (Γ, π) is the final coalgebra in $PSys(\mathcal{H})$, that is for every pre-system (S, α) there exists a unique homomorphism $!_\alpha : (S, \alpha) \to (\Gamma, \pi)$.*

Proof For every pre-system (S, α), we define the function $!_\alpha : S \to \Gamma$ which for every $q \in S$ associates the transfer function $!_\alpha(q) : In^T \to Out^{\mathbb{T}}$ defined as follows. Let $d \in \mathbb{T}$, and let $(m^{\mathbb{T}}, d_1, \ldots, d_n)$ such that for every i, $0 \leq i < n$, $d_{i+1} = succ^{\mathbb{T}}(d_i)$ with $d_0 = m^{\mathbb{T}}$ and $d_n = d$. Then, for every $f : \mathbb{T}' \to In \in In^T$, $!_\alpha(q)(f)(d)$ equals:

$$\alpha(\alpha(\ldots(\alpha(\alpha(q)(f :: m^{\mathbb{T}}, m^{\mathbb{T}})_2)(f :: d_1, d_{1\mathbb{T}})_2) \\ (f :: d_2, d_{2\mathbb{T}})_2 \ldots)(f :: d_{n-1}, d_{n-1\mathbb{T}})_2)(f :: d_n, d_{n\mathbb{T}})_1$$

It is not very difficult to verify that $!_\alpha(q)$ is causal, and the function $!_\alpha$ is a homomorphism which is further unique. □

We call the transfer function $!_\alpha(q)$ above the **behavior** of q, and then $!_\alpha(q_0)$ will be the behavior of the system $((S, \alpha), q_0)$. This result is the coalgebraic equivalent of Theorem 1.

Conversely, given a transfer function $\mathcal{F} \in \Gamma$, we can build the minimal system $<\mathcal{F}>$ the behavior of which is $\mathcal{F}$ as follow:

- $\mathcal{F}$ is the initial state.
- $<\mathcal{F}>$ is the set of transfer functions of Γ that contains $\mathcal{F}$ and is closed under transitions in (Γ, π) for any inputs and times.
- $\alpha_{\mathcal{F}} : <\mathcal{F}> \to (Out \times <\mathcal{F}>)^{In \times \mathbb{T}}$ is the mapping that associates to every $\mathcal{F}' \in <\mathcal{F}>$, every $i \in In$ and every $d \in \mathbb{T}$ the pair $\pi(\mathcal{F}')(i, d)$.

Proposition 9 *$<\mathcal{F}>$ is the minimal system (i.e. it has the smallest number of states) such that $!_{\alpha_{\mathcal{F}}}(\mathcal{F}) = \mathcal{F}$.*

Proof The fact that $!_{\alpha_{\mathcal{F}}}(\mathcal{F}) = \mathcal{F}$ follows from the identity Id_Γ which is the unique homomorphism over Γ. To show that $<\mathcal{F}>$ is minimal, let us consider a system $((S, \alpha), q_0)$ such that $!_\alpha(q_0) = \mathcal{F}$. Let us define $<q_0>$ the subsystem $((S', \alpha'), q'_0)$ of $((S, \alpha), q_0)$ as follow:

- $q'_0 = q_0$
- S' is the set of states of S that contains q_0 and is closed under transitions in (S, α) for any inputs and times.

[3]This is correct because transfer functions are causal.

- $\alpha' : S' \to (Out \times S')^{In \times \mathbb{T}}$ is the mapping that associates to every $q \in S'$, every $i \in In$ and every $d \in \mathbb{T}$ the pair $\alpha(q)(i,d)$.

As $!_\alpha$ is a homomorphism, it directly follows that the image $!_\alpha(S)$ is a sub-presystem of (Γ, π). Moreover, by definition we have that $!_\alpha(S) = !_\alpha(< q_0 >) = <!_\alpha(q_0)> = <\mathcal{F}>$ whence we can conclude that $<\mathcal{F}>$ is the minimal system such that $!_{\alpha_\mathcal{F}}(\mathcal{F}) = \mathcal{F}$. □

By Theorem 6 and Proposition 9, we will talk about systems and transfer functions indifferently.

5.2 A logic for system requirements

We present in this section a logic $\mathcal{L}$ whose the interpretation will be over systems. $\mathcal{L}$ is a slight extension of μ-calculus to input and output values and times. The interest of μ-calculus is its greats increase in expressive power. Indeed, it includes many of modal logics commonly used in verification of reactive and distributed systems. We will then use this logic to define requirements expressing properties on the behavior of systems.

5.2.1 Definition

The logic $\mathcal{L}$ being an extension of μ-calculus that is known to subsume most of modal and temporal logics, it will allow to express standard properties over systems such as reactiveness, liveness, safety, etc. Now, time being explicit in our framework, $\mathcal{L}$ must also allow to express both real-time properties on the production time of output values from input ones, and properties on the input or output value reading from both dataflow and moment. This requires a language to express time expressions. By our axiomatization, it is natural to define such expressions as first-order terms with variables over the mono-sorted first-order signature $\Sigma = (F, R)$ where the set of function names $F = \{succ^1, pred^1, +^2\} \cup \{d^0 | d \in T\}$ and the set of predicates $R = \{\prec^2, \preceq^2\}$ for T a time reference. [4]
Hence, time terms will be elements in the set $T_\Sigma(V)$ which is the set of all terms freely generated from the signature Σ and a set V of time variables. A model for Σ or Σ-model is any first-order structure $(\mathbb{T}, succ^\mathbb{T}, pred^\mathbb{T}, +^\mathbb{T}, \prec^\mathbb{T}, \preceq^\mathbb{T})$ where $\mathbb{T} \subseteq T$ is a time scale, $+^\mathbb{T} : (d, d') \mapsto (d +^T d')_\mathbb{T}$, $\prec^\mathbb{T} = \prec^T_{|\mathbb{T}}$ and $\preceq^\mathbb{T} = \preceq^T_{|\mathbb{T}}$. In the following, this model will simply note $\mathbb{T}$ when this does not raise ambiguity.
Given an interpretation of variables $\iota : V \to T$ and a time term $t \in T_\Sigma(V)$ variables of which are among $\{x_1, \ldots, x_n\}$, the evaluation of t for ι in $\mathbb{T}$, noted $[\![t]\!]^\mathbb{T}_\iota$, is the evaluation of $\iota(t)$ by interpreting *succ*, *pred*, $+$, $\prec$ and $\preceq$ by $succ^\mathbb{T}$, $pred^\mathbb{T}$, $+^\mathbb{T}$, $\prec^\mathbb{T}$ and $\preceq^\mathbb{T}$, respectively, and every constant d by $d_\mathbb{T}$. [5]

[4]The exponents attached to function and predicate names indicate their arity. Hence, any moment $d \in T$ is considered as a constant function.

[5]$\iota(t)$ is the term obtained from t by replacing every variable x_i by its value $\iota(x_i)$ taken as a constant.

In the next definition, we need a set of supplementary variables, called fixed point variables, to express formulas in μ-calculus that denote recursion on states. To differentiate these variables with those in V, we will denote in the following variables in V by the letters $x, x', x_1, x_2, \dots, y, y', y_1, y_2, \dots$ whilst fixed point variables will be denoted by $\overline{x}, \overline{x}', \overline{x}_1, \overline{x}_2, \dots, \overline{y}, \overline{y}', \overline{y}_1, \overline{y}_2, \dots$.

Definition 5.2.1 (Input and output terms) *Let T be a time reference. Let V be a set of time variables. Let $\mathcal{H} = (Out \times \_)^{In \times \mathbb{T}}$ be a signature with $\mathbb{T} \subseteq T$.* **Input terms** *(resp.* **output terms***) over $\mathcal{H}$ are all inputs in In (resp. outputs in Out) and all expressions of the form $\_ ::_{In} t$ (resp. $\_ ::_{Out} t$) where $t \in T_\Sigma(V)$.*

Input expressions $\_ ::_{In} t$ (resp. $\_ ::_{Out} t$) denote the content of input (resp. output) dataflows at the moment t.

Definition 5.2.2 (System requirements formulas) *Let T be a time reference. Let V be a set of time variables. Let X be a set of fixed points variables. Let $\mathcal{H} = (Out \times \_)^{In \times \mathbb{T}}$ be a signature with $\mathbb{T} \subseteq T$.* **System requirements formulas** *are defined as follows:*

$$\varphi := \Theta |\ E = E' \mid i \downarrow_t o |\ \overline{x} \mid [(i,t)]\varphi | \neg\varphi | \varphi_1 \wedge \varphi_2 | \exists x.\varphi | \nu\overline{x}.\psi$$

where Θ is a first-order formula built over the signature $\Sigma = (F, R)$ and the set of variables V, E and E' are either both input terms or both output terms, $i \in In$, $t \in T_\Sigma(V)$, $x \in V$, $\overline{x} \in X$ and ψ is a formula that may contain occurrences of the variable $\overline{x}$ provided that every free occurrences of $\overline{x}$ occurs positively in ψ, i.e. within the scope of an even number of negations [6]*.*
A formula φ is **closed** *when every time variable x and every fixed point variable $\overline{x}$ are in the scope of a quantifier $\exists x$ and an operator $\nu\overline{x}$, respectively.*

Intuitively, atoms of the form $E = E'$ check the content of input or output dataflows at different moments, and atoms of the form $i \downarrow_t o$ stand for output formula and check that it is possible to produce the output o after performing the input i at the moment t. A formula of the form $[(i,t)]\varphi$ stands for a state formula, and states that after performing an input i at the moment t, all reachable states satisfy φ. Finally, a formula of the form $\nu\overline{x}.\psi$ stands for a formula that expresses a recursion on states and is defined semantically as a function with fixpoints. More precisely, a formula φ of the logic can be semantically defined by a function $f_\varphi : \mathcal{P}(S)^n \to \mathcal{P}(S)$ where occur freely the fixed point variables $\overline{x}_1, \dots, \overline{x}_n$ in φ, that given n subsets $S_1, \dots, S_n$ yields the set of states that satisfy φ. Therefore, a formula φ of the form $\nu\overline{x}.\psi$ that can be seen as a "looping", denotes the greatest fixpoint of the function $f_\varphi : \mathcal{P}(S) \to f_\psi$ (see below). It is well-known that such a fixpoint exists when f is monotonic on $\mathcal{P}(S)$. The condition that every free occurrences of $\overline{x}$ occurs positively in ψ, ensures monotonicity [16].

The least fixpoint operator μ is obtained standardly:

[6]The notions of free and bound variables are usual where ν is the only binding operator.

$$\neg\nu\overline{x}.\varphi \Leftrightarrow \mu\overline{x}.\neg\varphi'$$

where φ' is the formula obtained from φ by substituting $\neg\overline{x}$ for $\overline{x}$ in all free occurrences of $\overline{x}$ in φ.

Standardly, the universal quantifier is defined:

$$\neg\exists.\varphi \Leftrightarrow \forall x.\neg\varphi$$

5.2.2 Examples of requirements

We have defined a logic on systems that is expressive enough to model functional requirements.

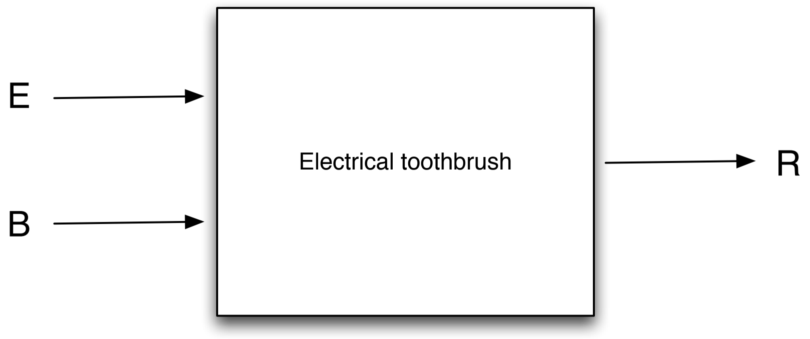
We propose to model a very simplified toothbrush viewed as a system:

Example 17 [Toothbrush] We set $\mathbb{R}$ as our time reference and work on the regular time scale $\mathbb{N}_\tau$ where the step τ stands for a hundredth of second.

The toothbrush has 2 input flows, $\mathcal{B}$ and $\mathcal{E}$, modeling respectively the button to control the toothbrush and the electricity coming by the power supply of the toothbrush.

The input $\mathcal{B}$ can take two values: 1 or 0, according to the state of the button (pressed or released). The input $\mathcal{E}$ can also take the two values 1 or 0, according to the presence or not of electricity allowing to supply the toothbrush.

Our toothbrush is modeled with one single output $\mathcal{R}$ figuring the rotation of the head designed to brush the teeth. The output $\mathcal{R}$ can take values in $\{0, 1, 2, 3, 4\}$ according to the speed of rotation (0 meaning no rotation and 4 being the highest speed of the head).



An oversimplified specification of the transfer function of the toothbrush can be the following:

- the toothbrush has 5 states: 0, 1, 2, 3, 4
- whatever the state of the system:
  - when $\mathcal{E} = 0$, then $\mathcal{R} = 0$ and the system returns to the state 0.

  - when $\mathcal{E} = 1$ and $\mathcal{B} = 0$ at any moment, then at the next step the state decreases of 1 (or 0 if it was 0), and $\mathcal{R}$ takes the value corresponding to the state.
  - when $\mathcal{E} = 1$ and $\mathcal{B} = 1$ at any moment, then at the next step the state increases of 1 (or remains to 4 if it was 4), and $\mathcal{R}$ takes the value corresponding to the state.

More formally, the system $\mathcal{S} = ((S, \alpha), q_0)$ is:

- $S = \{0, 1, 2, 3, 4\}$
- $q_0 = 0$
- $\alpha : S \to (Out \times S)^{In \times \mathbb{N}_\tau}$ with $Out = \{0, 1, 2, 3, 4\}$ and $In = \{0, 1\} \times \{0, 1\}$ is defined by: $\forall q \in \{0, 1, 2, 3, 4\}, \forall d \in \mathbb{N}_\tau$
  - $\alpha(q)((0, \_), d) = (0, 0)$
  - $\alpha(q)((1, 0), d) = \begin{cases} (q-1, q-1) & \text{if } q \in \{1, 2, 3, 4\} \\ (0, 0) & \text{if } q = 0 \end{cases}$
  - $\alpha(q)((1, 1), d) = \begin{cases} (q+1, q+1) & \text{if } q \in \{0, 1, 2, 3\} \\ (4, 4) & \text{if } q = 4 \end{cases}$

◊

We now propose to express the following requirement, insuring that the toothbrush reacts quickly to a pressure on its button: *(Reactiveness) when the button is pressed and that there is electricity powering, the toothbrush modifies its output within* 0.1 *second except if it is already at the highest speed. In this case, it remains to this highest speed while the button is pressed.* This requirement can be expressed as follows in our framework:

$$\_ ::_{In}\ x = (1, 1) \wedge \neg(\_ ::_{Out}\ x = 4) \Rightarrow \exists y.\ y \leq 10\ \wedge\ \neg(\_ ::_{Out}\ (x + y) = \_ ::_{Out}\ x)$$

$$\_ ::_{In}\ x = (1, 1) \wedge \_ ::_{Out}\ x = 4 \Rightarrow (\forall y. y \succeq x \Rightarrow \begin{cases} \mu\overline{x}. \_ ::_{In}\ y = (\_, 0) \\ \vee \\ ([((1, 1), y)]\overline{x} \wedge \_ ::_{Out}\ succ(y) = 4)) \end{cases}$$

We propose to model another requirement, imposing that the toothbrush would stop rapidly if the electricity supply is stopped: *(Inertia) when there is no power supply, the toothbrush's head stops its rotation within 1 second.* This requirement can be expressed as follows:

$$\_ ::_{In}\ x = (0, \_) \Rightarrow \exists y.\ y \leq x + 100\ \wedge\ \_ ::_{Out}\ y = 0$$

We introduce a last requirement constraining the speed of the toothbrush when it is working: *(Performance) when the toothbrush is used, the speed of rotation must be* 1*,* 2*,* 3 *or* 4. This requirement can be expressed as follows:

$$\_ ::_{In} x = (1,1) \Rightarrow \bigvee_{1 \leq i \leq 4} \_ ::_{Out} x = i$$

These 3 requirements are typical properties expected to be verified from systems, and illustrate the expressivity of our logic.

5.2.3 Adequacy of the logic

This subsection is not necessary for our framework, but proves an important property of our logic: adequacy.

We define the notion of bisimulation over which we will show the adequateness of our logic. Systems being defined by using coalgebraic notations, we define bisimulations for systems following notations in [2, 55]. Hence, a bisimulation between two systems is a transition structure respecting relation between sets of states.

Definition 5.2.3 (Bisimulation) *Let $\mathcal{S}_1 = ((S_1, \alpha_1), q_0^1)$ and $\mathcal{S}_2 = ((S_2, \alpha_2), q_0^2)$ be two systems over a signature $\mathcal{H} = (Out \times \_)^{In \times \mathbb{T}}$. A subset $R \subseteq S_1 \times S_2$ is a* **bisimulation** *if, and only if $(q_0^1, q_0^2) \in R$ and there exists a mapping $\alpha_R : R \to \mathcal{H}(R)$ such that both projections from R to S_1 and S_2 are coalgebra morphisms:*

$$\begin{tikzcd}
S_1 \arrow[d, "\alpha_1"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\alpha_R"] & S_2 \arrow[d, "\alpha_2"] \\
\mathcal{H}(S_1) & \mathcal{H}(R) \arrow[l, "\mathcal{H}(\pi_1)"'] \arrow[r, "\mathcal{H}(\pi_2)"] & \mathcal{H}(S_2)
\end{tikzcd}$$

$\mathcal{S}_1$ and $\mathcal{S}_2$ are said **bisimilar** *if, and only if there exists a bisimulation between them.*

All the basic facts on bisimulations remain true in our framework. Among others, the greatest bisimulation between $\mathcal{S}_1$ and $\mathcal{S}_2$, noted $\sim_{\mathcal{S}_1,\mathcal{S}_2}$ or simply $\sim$ when the context is clear, exists and is defined as the union of all bisimulations between $\mathcal{S}_1$ and $\mathcal{S}_2$.

Definition 5.2.4 (Input output terms evaluation) *Let $\mathcal{H} = (Out \times \_)^{In \times \mathbb{T}}$ with $\mathbb{T} \subseteq T$. Let V be a set of time variables. Let $\iota : V \to T$ be a time variable valuation. Let $f : \mathbb{T}' \to In$ be an input dataflow in In^T. Let E be an input term over $\mathcal{H}$. The* **evaluation** *of E for f, noted $[\![E]\!]^{\mathbb{T}'}_{(\iota,f)}$ is defined on the structure of E as follows:*

- $[\![i]\!]^{\mathbb{T}'}_{(\iota,f)} = i$ *for* $i \in In$
- $[\![\_ ::_{In} t]\!]^{\mathbb{T}'}_{(\iota,f)} = f([\![t]\!]^{\mathbb{T}'}_{\iota})$.

Let $g : \mathbb{T} \to Out$ be an output dataflow. Let E be an output term over $\mathcal{H}$. The **evaluation** *of E for g, noted $[\![E]\!]^{\mathbb{T}}_{(\iota,g)}$ is defined on the structure of E as follows:*

- $[\![o]\!]^{\mathbb{T}}_{(\iota,g)} = o$ *for* $o \in Out$
- $[\![\_ ::_{Out} t]\!]^{\mathbb{T}}_{(\iota,g)} = g([\![t]\!]^{\mathbb{T}}_{\iota})$.

Classically, the semantics of μ-calculus formulas is standardly defined by associating to each formula φ the set of states for which φ is true[7]. This kind of semantics can be easily extended to our logic equivalently to Definition 5.2.5 just below. However, Definition 5.2.5 is a more classical definition of satisfaction $\models$ defined as a binary relation between systems and formulas.

Definition 5.2.5 (Satisfaction) *Let $\mathcal{H} = (Out \times \_)^{In \times \mathbb{T}}$ be a signature with $\mathbb{T} \subseteq T$. Let $\mathcal{S} = ((S, \alpha), q_0)$ be a system over $\mathcal{H}$. Let φ be a formula over $\mathcal{H}$. For every valuation $\lambda : X \to \mathcal{P}(S)$, every interpretation of variables $\iota : V \to T$, every state $q \in S$ and every input dataflow $f : \mathbb{T}' \to In \in In^T$. $\mathcal{S}$* **satisfies** *for f, q, ι and λ the formula φ, noted $\mathcal{S} \models_{f,q,\iota,\lambda} \varphi$ if, and only if:*

- *if φ is a first-order formula Θ over Σ, then $\mathcal{S} \models_{f,q,\iota,\lambda} \Theta$ iff $\mathbb{T} \models_{\iota} \Theta$.*
- *if φ is an atom of the form $E = E'$ where E and E' are input terms, then $\mathcal{S} \models_{f,q,\iota,\lambda} E = E'$ iff $[\![E]\!]^{\mathbb{T}'}_{(\iota,f)} = [\![E']\!]^{\mathbb{T}'}_{(\iota,f)}$.*
- *if φ is an atom of the form $E = E'$ where E and E' are output terms, then $\mathcal{S} \models_{f,q,\iota,\lambda} E = E'$ iff $[\![E]\!]^{\mathbb{T}}_{(\iota,!_\alpha(q)(f))} = [\![E']\!]^{\mathbb{T}}_{(\iota,!_\alpha(q)(f))}$.*
- *if φ is an atom of the form $i \downarrow_t o$, then $\mathcal{S} \models_{f,q,\iota,\lambda} i \downarrow_t o$ iff $\alpha(q)(i, [\![t]\!]^{\mathbb{T}}_{\iota})_1 = o$.* [8]
- *if φ is a fixed point variable $\overline{x}$, then $\mathcal{S} \models_{f,q,\iota,\lambda} \overline{x}$ iff $q \in \lambda(\overline{x})$.*
- *if $\varphi = [(i,t)]\varphi'$, then $\mathcal{S} \models_{f,q,\iota,\lambda} [(i,t)]\varphi'$ iff $\mathcal{S} \models_{(i,[\![t]\!]^{\mathbb{T}'}_{\iota}):f,q',\iota,\lambda} \varphi'$ and $q' = \alpha(q)(i, [\![t]\!]^{\mathbb{T}}_{\iota})_2$.*
- *if $\varphi = \nu\overline{x}.\psi$, then $\mathcal{S} \models_{f,q,\iota,\lambda} \nu\overline{x}.\psi$ iff $\exists S' \subseteq S$, $q \in S'$ and $\forall q' \in S'$, $\mathcal{S} \models_{f,q',\iota,\lambda[S'/\overline{x}} \psi$.*
  Here, $\lambda[S'/\overline{x}]$ is the valuation such that $\lambda[S'/\overline{x}](\overline{x}) = S'$ and $\lambda[S'/\overline{x}](\overline{x}') = \lambda(\overline{x}')$ for every $\overline{x}' \neq \overline{x}$.
- *propositional connectors and first-order quantifier are handled as usual.*

$\mathcal{S}$ **satisfies** *a formula φ, noted $\mathcal{S} \models \varphi$, if, and only if for every $f \in In^T$, every $\iota : V \to T$ and every valuation $\lambda : X \to \mathcal{P}(S)$, $\mathcal{S} \models_{f,q_0,\iota,\lambda} \varphi$.*

[7]returning equally to define a function $f_\varphi : \mathcal{P}(S)^n \to \mathcal{P}(S)$ as previously where n is the set number of free fixed point variables in φ.

[8]Similarly, we could also write that $\mathcal{S} \models_{f,q,\iota,\lambda} \varphi$ iff $!_\alpha(q)((i, [\![t]\!]^{\mathbb{T}'}_{\iota}) : f)([\![t]\!]^{\mathbb{T}}_{\iota}) = o$.

From Definition 5.2.5, it is obvious to show that for every closed formula φ, every state $q \in S$ and every input dataflow $f \in In^T$,

$$\forall \lambda : X \to \mathcal{P}(S), \forall \iota : V \to T, \mathcal{S} \models_{f,q,\iota,\lambda} \varphi \Leftrightarrow \mathcal{S} \models_{f,q,\emptyset} \varphi$$

where $\emptyset : X \to \mathcal{P}(S)$ is the valuation that associates to every $\overline{x} \in X$ the emptyset $\emptyset$.

Let us show that $\mathcal{L}$ is expressive enough to characterize bisimilarity, that is two systems $\mathcal{S}_1$ and $\mathcal{S}_2$ are bisimilar when they are elementary equivalent and vice versa, where **elementary equivalence** means that:

$$\forall \varphi, \mathcal{S}_1 \models \varphi \Leftrightarrow \mathcal{S}_2 \models \varphi$$

Theorem 7 *Let $\mathcal{S}_1 = ((S_1, \alpha_1), q_0^1)$ ad $\mathcal{S}_2 = ((S_2, \alpha_2), q_0^2)$ be two systems over $(Out \times \_)^{In \times \mathbb{T}}$. Then, $\mathcal{S}_1$ and $\mathcal{S}_2$ are elementary equivalent if, and only if they are bisimilar.*

Proof To prove the **only if** implication, let us suppose that $q_0^1 \sim q_0^2$. Let $\lambda_2 : X \to \mathcal{P}(S_2)$. Let us define $\lambda_1 : X \to \mathcal{P}(S_1)$ by:

$$\lambda_1(\overline{x}) = \{q_1 | \exists q_2 \in \lambda_2(\overline{x}), q_1 \sim q_2\}$$

It is quite obvious to show by structural induction on formulas that for every φ,

$$\mathcal{S}_1 \models_{f,q_0^1,\iota,\lambda_1} \varphi \Leftrightarrow \mathcal{S}_2 \models_{f,q_0^2,\iota,\lambda_2} \varphi$$

We can apply the same reasoning from any valuation $\lambda_1 : X \to \mathcal{P}(S_1)$.

For the converse (the **if part**), let us define the relation $\equiv \subseteq S_1 \times S_2$ as follows: $q \equiv q'$ iff for every $f \in In^T$, every $\iota : V \to T$ and every $\lambda : X \to \mathcal{P}(S_1)$,

$$\forall \varphi, \mathcal{S}_1 \models_{f,q,\iota,\lambda} \varphi \Leftrightarrow \mathcal{S}_2 \models_{f,q',\iota,\lambda'} \varphi$$

where $\lambda' : X \to \mathcal{P}(S_2)$ is the mapping that associates the set $\{q' | \exists q \in \lambda(\overline{x}), q \equiv q'\}$ to each $\overline{x} \in X$. Let us show that $\equiv \subseteq \sim$. Let us suppose that $q \equiv q'$. By definition, this means for every $i \in In$ and every $d \in T$ that $\alpha_1(q)(i, d_\mathbb{T})_1 = \alpha_2(q')(i, d_\mathbb{T})_1$. It remains to prove that $\alpha_1(q)(i, d_\mathbb{T})_2 \equiv \alpha_2(q')(i, d_\mathbb{T})_2$. Let us suppose the opposite. This means there exists a formula ψ, a dataflow $f : \mathbb{T}' \to In \in In^T$, a variable interpretation $\iota : V \to T$ and a valuation $\lambda : X \to \mathcal{P}(S)$ such that $\mathcal{S}_1 \models_{f,\alpha_1(q)(i,d_\mathbb{T})_2,\iota,\lambda} \psi$ and $\mathcal{S}_2 \not\models_{f,\alpha_2(q')(i,d_\mathbb{T})_2,\iota,\lambda'} \psi$. By definition of satisfaction, ψ can be considered as a formula that does not contain first-order formulas and atoms of the form $E = E'$ because their satisfaction does not bring into play states. Therefore, we can write equivalently that $\mathcal{S}_1 \models_{(i,d_{\mathbb{T}'}):f,\alpha_1(q)(i,d_\mathbb{T})_2,\iota,\lambda} \psi$ and $\mathcal{S}_2 \not\models_{(i,d_{\mathbb{T}'}):f,\alpha_2(q')(i,d_\mathbb{T})_2,\iota,\lambda'} \psi$, whence we conclude $\mathcal{S}_1 \models_{f,q,\iota,\lambda} [i, d]\psi$ and $\mathcal{S}_2 \not\models_{f,q',\iota,\lambda'} [i, d]\psi$ what is not possible as $q \equiv q'$. The same reasoning can be carried out for $\equiv^{-1}$. □

This result proves the adequacy of the logic we have defined, giving it the minimal expected properties to be a "good" logic for system requirements.

Chapter 6

Towards a framework for systems architecture

We have introduced a unified model for heterogeneous integrated systems and their integration, and a formalism to express requirements on systems. Our definition of system captures the behavior of a real system that can be observed (functional and states behavior, called together *systemic behavior*). However, our definition of a system has two fundamental limitations from a systems architecture point of view: it requires to fully specify the functional and states behaviors, and it gives no information on what the system is composed of[1].

We thus introduce a formal framework to deal with the architecture of systems during the design process. The two main limitations above are addressed:

- how to deal with the underspecification of systems during the design process? (i.e. the fact that we are unable to define completely a system during its design, whose precise goal is to define the system completely, so we are constantly working on a partly defined object)
- how to formalize the internal structure of a system? (i.e. how to formally express the fact that a system is in fact a multiscale object obtained through the recursive integration of smaller systems).

We consider a minimalist design process, consisting of requirements analysis and systemic recursion. The only possible actions are:

1. breaking a system into a set of smaller systems (and conversely composing together a set of systems)
2. “concretizing” a system into a finer grain system (and conversely “abstracting” a system)

[1]the integration operators we have introduced do not allow to model the structure of a system, as they do not “remember” the systems they have integrated and just define the resulting system.

3. expressing requirements on a system (and conversely checking if a system verifies such requirements).

We introduce the notion of views that allow to formalize the set of interrelated models used in practice to describe a more or less specified system at any step of the design process[2]. We then introduce formal definitions to be able to define the internal structure of a system. The deliverable is a minimalist formal framework for systems architecture along the design process.

6.1 Handling underspecification

We will now give formal definitions of objects used during the systems architecting process to take into account the need to handle underspecified systems while defining their expected behavior. Indeed, one hardly manipulates fully-specified systems during the design phase[3].

Definition 1 (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.*

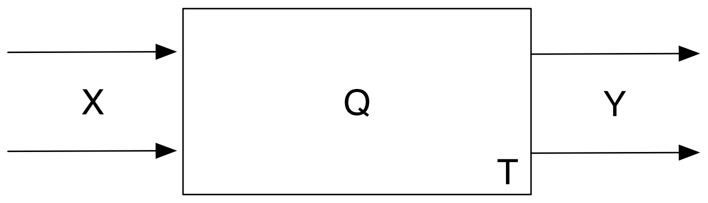


Figure 6.1: Illustration of a systemic signature

Remark 7 *A system naturally induces a systemic signature.*

We introduce a generic definition of system requirements, used to describe logical properties on the functional and states behavior a system (and whose a formalization on coalgebraic systems has been introduced in the previous chapter):

Definition 2 (Requirement) *Let* $(X, Y, Q, \mathbb{T})$ *be a systemic signature. A* **requirement** *on* $(X, Y, Q, \mathbb{T})$*, is a logical formula 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* $Req(X, Y, Q, \mathbb{T})$*.*

[2]We do not deal here with the concept of *layers* of a system that allow to describe the same system at a given level through different perspectives (e.g. functional and physical layers).

[3]This problem is for example addressed in B-method through the refinement of an abstract machine [1].

During the design process, most of the time we deal with objects that are not fully specified. The typical object manipulated at high-level during the design phase to model a system is a fuzzy description of this system through a systemic signature, together with a set of expected properties. We thus define a notion that captures such underspecified objects:

Definition 3 (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 of systemic signature* $(X, Y, Q, \mathbb{T})$.

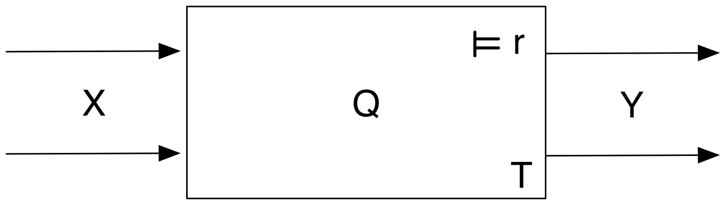


Figure 6.2: Illustration of a box

Remark 8 *We only consider one requirement in our definition of a box, which is equivalent to a finite set of requirements.*

A box induces a set of systems whose systemic signature matches the one from the box, and that verify the requirement of the box:

Definition 4 (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* $S \models r$ [4]. *When such a system exists,* B *is said to be realizable.*

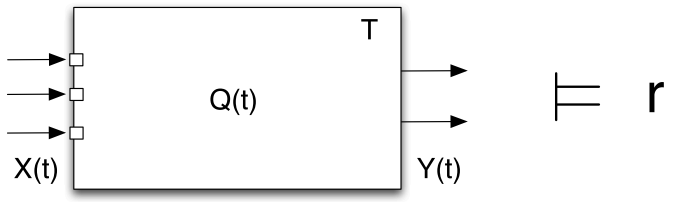


Figure 6.3: Illustration of the realization of a box

[4]which means that the functional and states behaviors of the system S verify the requirement r.

Remark 9 *One of the challenge of systems design, in this framework, is to be able to define only realizable boxes at each level. In practice, this is an iterative process with trial & error. Note that, in real life, the requirement associated with a box will constrain its behavior, but also express constraints related to cost, time, feasability & other business metrics.*

A key property in systems architecture is the ability to change the granularity of description of a model through mechanisms of abstraction and concretization. We naturally extend such mechanisms from systems to boxes:

Definition 5 (Concretization of a box) *Let $B_c \in BB(X_c, Y_c, Q_c, \mathbb{T}_c)$ (called concrete box). Let $B_a \in BB(X_a, Y_a, Q_a, \mathbb{T}_a)$ (called abstract box). Let $\alpha : (X_c, Y_c, Q_c, \mathbb{T}_c) \to (X_a, Y_a, Q_a, \mathbb{T}_a)$ be an abstraction mechanism. We say that B_c **concretizes** B_a **via** α if and only if: for any system S_c that is a realization of B_c, $\alpha(S_c)$ is a realization of B_a*[5]*.*

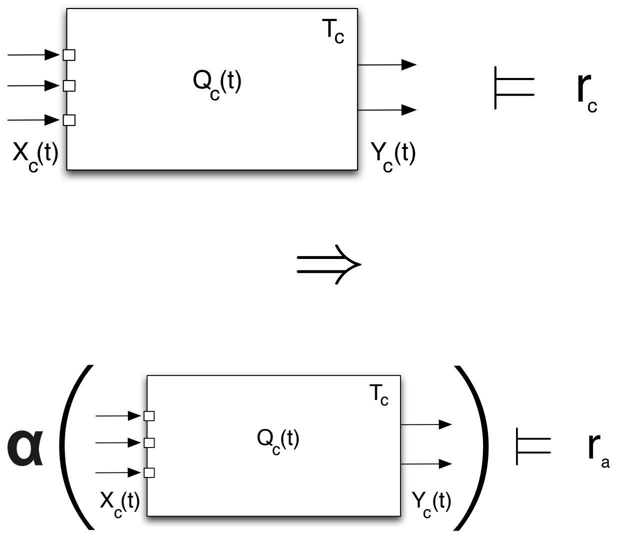


Figure 6.4: Illustration of the concretization of a box

We can now deal with underspecification:

- a systemic signature is the most underspecified object
- a box is a systemic signature, with expected properties on the systemic behavior
- a system is the algorithmic specification of a box.

[5]This (only) means that $\alpha(S_c)$ verifies the requirement of B_a. Still, is a very strong property and means that the requirement on the concrete system is "strong" enough to ensure that the requirement of its abstraction will be verified.

We will now introduce a formalism to deal with the structure of systems. This formalism will also model the fact that, during the design process, real systems are in fact modeled using boxes.

6.2 Modeling recursive structure

The goal of this section is to deal with the structure of more or less specified objects through the design process. We have defined integration operators for systems and transfer functions, but those operators do not allow to model the internal structure. We thus first introduce an object that formalizes a finite sequence of products and feedbacks on a finite set of systems:

Definition 6 (Composition plan) *Let $S_0, \dots, S_{n-1}$ be n systems. A* **composition plan** *for $S_0, \dots, S_{n-1}$ is a set $C \subset \{0, \dots, n-1\}^2$ of pairs where:*

- $\forall \ \big((a,b),(c,d)\big) \in C^2, \ \big[(a \neq c) \wedge (b \neq d)\big] \vee \big[(a = c) \wedge (b = d)\big]$
- $\forall \ (a,b) \in C$*, output Y_a of S_a and input X_b of S_b have the same dataset.*

More informally, C is a set of links between outputs and inputs of $S_0, \dots, S_{n-1}$ such that each input (resp. output) is linked to at most one output (resp. input). We thus write $C(S_0, \dots, S_{n-1})$ for the system resulting from the composition of these n systems according to C.

Remark 10 *The definition of a composition plan for systems can easily be extended to boxes (using their systemic signatures) and systems with multiple inputs and outputs.*

A key element in systems architecture is the ability to refine a system by breaking it down into smaller subsystems. We introduce a definition that make it possible to express such process on boxes, working on a single time scale:

Definition 7 (Refinement of a box) *Let $B \in BB(X, Y, Q, \mathbb{T})$. For all $i \in \{0, \dots, n-1\}$, let $B_i \in BB(X_i, Y_i, Q_i, \mathbb{T})$. Let C be a composition plan for $B_0, \dots, B_{n-1}$. $(B_0, \dots, B_{n-1}, C)$ is a* **refinement** *of B iff the systemic signature of $C(B_0, \dots, B_{n-1})$ is $(X, Y, Q, \mathbb{T})$.*

We give a first formalization to the systemic recursion by combining a box together with a refinement:

Definition 8 (View) *A* ***view*** *is a pair $\big(B, (B_0, \dots, B_{n-1}, C)\big)$:*

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

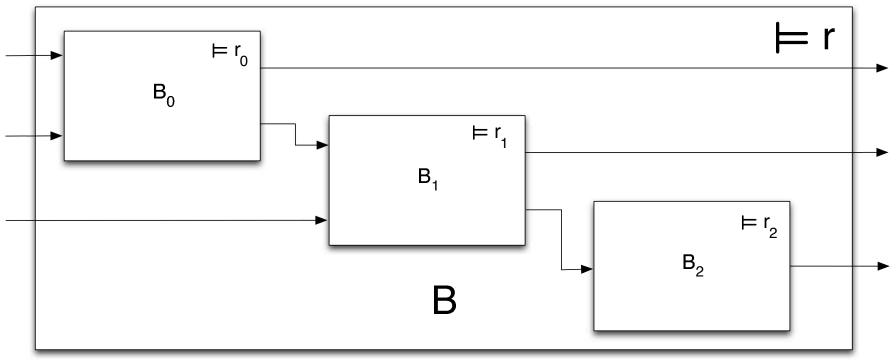


Figure 6.5: Illustration of a view

Remark 11 *The time reference is unique in a view. Changes of time reference are only possible through abstraction/concretization mechanisms.*

A view is thus a formal object that models the ability to refine a box through a set of interrelated boxes. We then naturally extend the notion of concretization of a box to a view:

Definition 9 (Concretization of a box by a view) *Let $V = \big(B_c, \_, \_\big)$ be a view. V is a* **concretization** *of a box B_a via an abstraction α iff B_c concretizes B_a via α.*

As for boxes, the existence of systems realizing the back boxes of a view in a consistent way is key[6]:

Definition 10 (Realization of a view) *Let $V = \big(B, (B_0, \ldots, B_{n-1}, C)\big)$ be a view. A* **realization** *of V is any realization $S_0, \ldots, S_{n-1}$ of $B_0, \ldots, B_{n-1}$ such that $C(S_0, \ldots, S_{n-1})$ is a realization of B. In this case, $C(S_0, \ldots, S_{n-1})$ is called composition of $S_0, \ldots, S_{n-1}$ according to V, and V is said to be realizable.*

However, a view only captures one level of systemic recursion. We thus introduce a new object capturing a finite number of systemic recursions and allowing the use of abstractions, so that at each systemic level, all systems resulting from the composition of lower level subsystems are brought to a common level of abstraction through individual abstractions:

Definition 11 (Multiscale view) *A* **multiscale view** *W is a tree such that:*

- *every node of W is labeled with a view*

[6]It is very difficult in a design process to define such views because many interrelated elements must be consistent. That's a reason why the design process is iterative in practice, and why verification is needed after integration (no correctness-by-construction). Note also that their is no "mechanistic" bottom-up implication: it is possible (and even likely!) to have realizations of the boxes of a view whose composition is not a realization of the main box.

- *every edge e of W from a parent node $V_p = \big(\_, (B_0, \ldots, B_{n-1}, \_)\big)$ to a child node V_c is labeled with a pair (k, α) where:*
  - *$k \in \{0, \ldots, n-1\}$ is called the index of the edge e*
  - *α is an abstraction such that V_p concretizes B_k via α*
- *for a parent node $V_p = \big(\_, (B_0, \ldots, B_{n-1}, \_)\big)$, there is at most one edge of index $k \in \{0, \ldots, n-1\}$.*

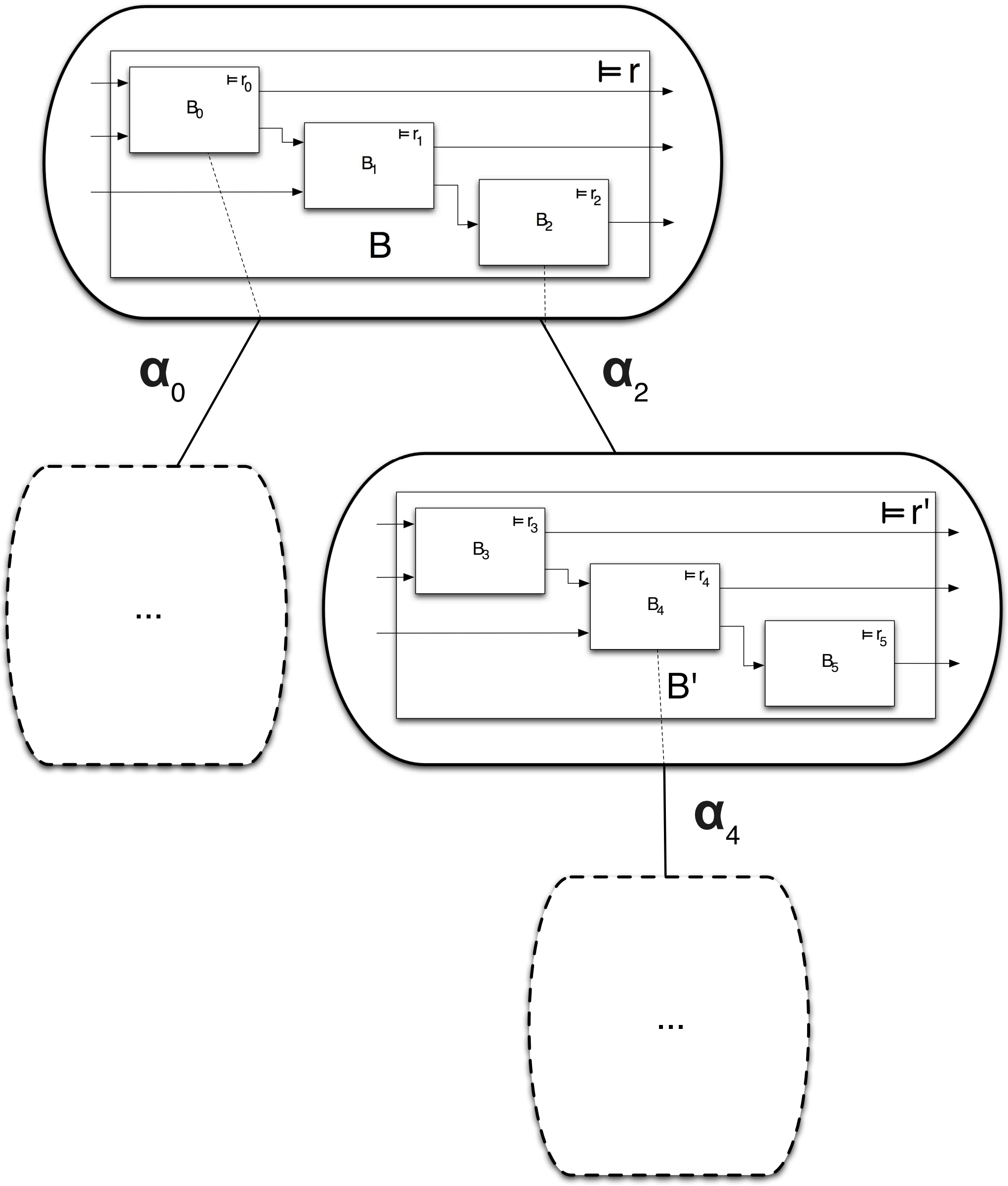


Figure 6.6: Illustration of a multiscale view

Remark 12 *For any node N from a multiscale view W, the subtree of W with root N naturally induces a new multiscale view.*

Thus, a multiscale view is a tree of views, such that a box in a view is either concretized (by another multiscale view), or "free":

Definition 12 (Free box) *Let W be a multiscale view. Let V be a view labeling a node of W. A* **free box** *of W is any box B of V such that: B is not concretized by any child of V in W. We write $freebox(W)$ for the finite sequence of free boxes of W, enumerated in depth-first order.*

Remark 13 *For any view V labeling a leaf of a multiscale view W, all the boxes of V are free boxes of W.*

Definition 13 (Valuation) *Let W be a multiscale view. Let V be a view labeling a node of W and let B be a box of V. The* **valuation** *$v_W(B) \in \mathbb{N}$ of B in W is defined as follows:*

- *$v_W(B) = 1$ when B is a free box of W*
- *$v_W(B) = \big|freebox(W_B))\big|$ else, where W_B is the multiscale view concretizing B in W.*

To define a system from a multiscale view, the free boxes need to be specified:

Definition 14 (Integration tree according to a multiscale view) *Let W be a multiscale view and let $freebox(W) = B_0^f, \dots, B_{n-1}^f$ be its free boxes, such that all free boxes of W are realizable. Let $S_0, \dots, S_{n-1}$ be n systems respectively realizing $B_0^f, \dots, B_{n-1}^f$. The* **integration tree** *of $(S_0, \dots, S_{n-1})$ according to W, which we note $\mathcal{I}\big(W, (S_0, \dots, S_{n-1})\big)$, is a tree (whose nodes are labelled with systems) recursively defined as follows:*

let $V = \big(B, (B_0, \dots, B_{p-1}, C)\big)$ be the label of the root node of W. $\forall i \in \{0, \dots, p\}$, let $x_i = \sum_{j=0}^{i-1} v_W(B_j)$[7]. $\forall i \in \{0, \dots, p-1\}$, we define M_i as:

- *when $B_i \in freebox(W)$, M_i is a single node labeled with S_{x_i}*
- *when $B_i \notin freebox(W)$, we define $M'_i = \mathcal{I}\big(W_i, (S_{x_i}, \dots, S_{x_{i+1}-1})\big)$[8], where W_i is the multiscale view concretizing B_i in W following α_i. M_i is a tree consisting in a root node labeled by $\alpha_i\big(root(M'_i)\big)$ and with a unique child M'_i.*

[7] as the free boxes are in depth-first order, x_i represents the index of the first free box of W which is a free box of B_i

[8] such tree is well-defined because: 1) the free boxes are enumerated in depth-first order 2) the recursion is applied to a strictly smaller multiscale view (it ensures that the recursion will terminate).

We then define $\mathcal{I}\big(W, (S_0, \dots, S_{n-1})\big)$ as the tree composed of p subtrees $M_0, \dots, M_{p-1}$ children of a root labelled by the composition of the systems labeling the roots of $M_0, \dots, M_{p-1}$, i.e. $C\big(root(M_0), \dots, root(M_{p-1})\big)$.

The integration tree is consistent iff the system labeling each node verifies the requirement of its associated box.

We can now define the system resulting from the integration of a sequence of systems following a multiscale view:

Definition 15 (Integration according to a multiscale view) *Let W be a multiscale view and let $freebox(W) = B_0^f, \dots, B_{n-1}^f$ be its free boxes, such that all free boxes of W are realizable. Let $S_0, \dots, S_{n-1}$ be n systems respectively realizing $B_0^f, \dots, B_{n-1}^f$. The* **integration according to** *W of $S_0, \dots, S_{n-1}$ is the system labeling the rood node of $\mathcal{I}\big(W, (S_0, \dots, S_{n-1})\big)$. An integration is consistent iff the corresponding integration tree is consistent.*

As for views, the existence of systems realizing a multiscale view in a consistent way is key:

Definition 16 (Realization of a multiscale view) *Let W be a multiscale view. Let $S_0, \dots, S_{n-1}$ be n systems. $(S_0, \dots, S_{n-1})$ is a* **realization** *of W if it is a consistent integration according to W. In this case, W is said to be realizable.*

We finally introduce a model of systems where the structure is described. A *multiscale system* describes the structure of a system in terms of successive compositions and abstractions allowing to build a system from a set of elementary systems:

Definition 17 (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 $(S_0, \_), \dots, (S_{n-1}, \_)$; we have: $S = C(S_0, \dots, S_{n-1})$*
- *for each odd node (S, α), its unique child $(S', \_)$ is such that: $S = \alpha(S')$.*

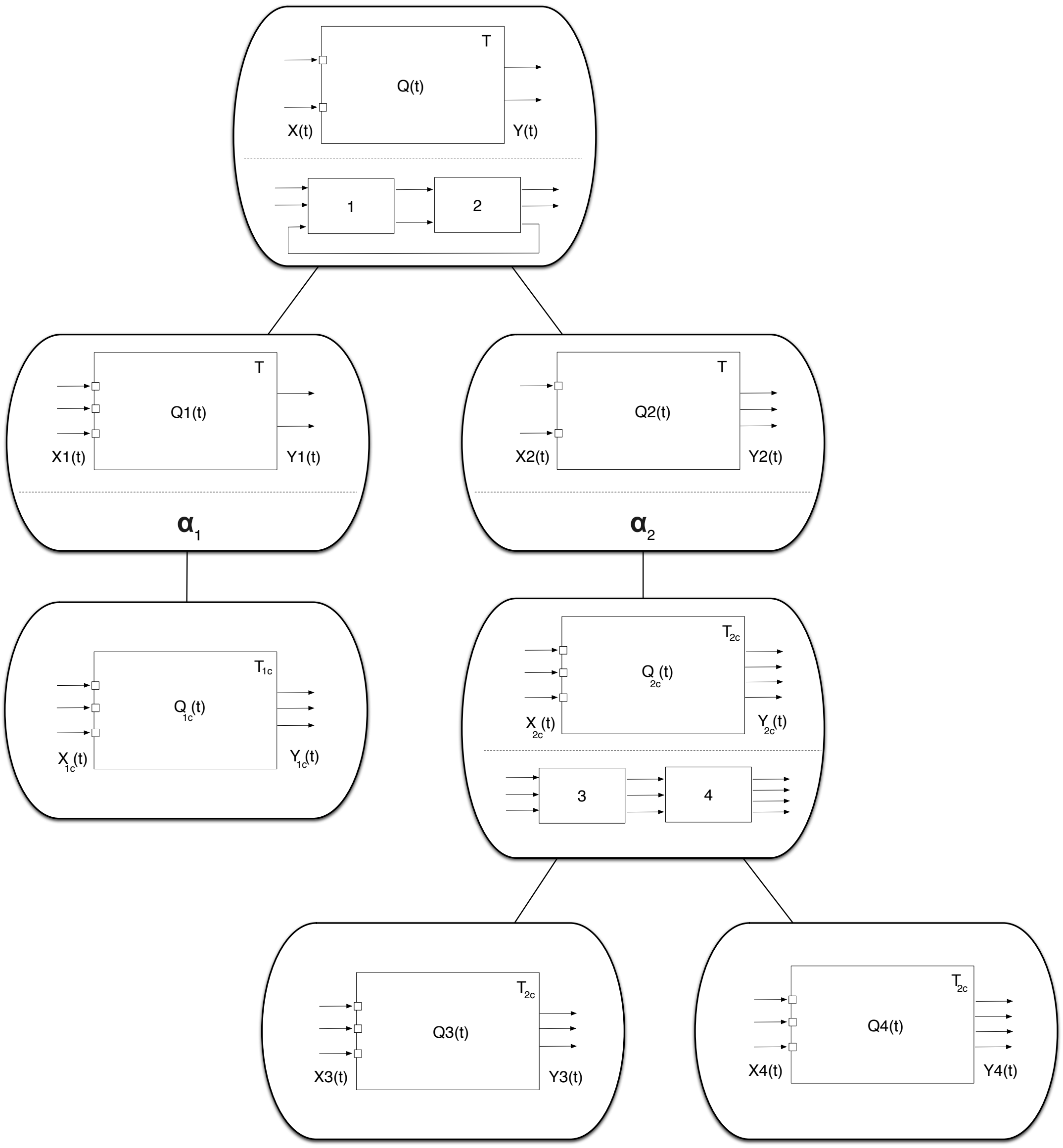


Figure 6.7: Illustration of a multiscale system

Remark 14 *The integration of a sequence of systems according to a multiscale view naturally induces a multiscale system.*

We thus have introduced a formalism allowing to define underspecified systems (with boxes), to define a recursive structure on them (through multiscale views), and finally to define fully specified systems with a recursive structure (through multiscale systems). Altogether, they form a minimalist framework for systems architecture in our formalism.

Chapter 7

Fair assignments between systems

This chapter intends to open new perspectives and is fairly independent from the rest of the manuscript. Optimization is a very important topic in systems architecture. We will introduce a model that allows to compute fair assignments, what can be very relevant to take structural decisions during the design process on how to allocate functions to different subsystems, or what is the best systemic decomposition to maintain an optimal homogeneity between subsystems, when fairness of a given quantity is required (for example to spread risk or cost between various subsystems).

We study fair assignment problems in decision contexts involving multiple agents (e.g. systems). In such problems, each agent (or system) has its own evaluation of costs and we want to find a fair compromise solution between individual points of views. Lorenz dominance is a standard decision model used in Economics to refine Pareto dominance while favoring solutions that fairly share happiness among agents. In order to enhance the discrimination possibilities offered by Lorenz dominance, we introduce here a new model called *infinite order Lorenz dominance*. We establish a representation result for this model using an ordered weighted average with decreasing weights. Hence we exhibit some properties of infinite order Lorenz dominance that explain how fairness is achieved in the aggregation of individual preferences. Then we explain how to solve fair assignment problems of m items to n agents, using infinite order Lorenz dominance and other models used for measuring inequalities. We show that this problem can be reformulated as a 0-1 nonlinear optimization problem that can be solved, after a linearization step, by standard LP solvers. We provide numerical results showing the efficiency of the proposed approach on various instances of the paper assignment problem.

The deliverable is a new, original model for fair multiagent optimization that can be relevant to optimize design decisions during a systems design process when fairness is required.

7.1 Introduction

Fairness of decision procedures is often considered as an important issue in decision problems involving multiple agents. Although not always formalized precisely, this normative principle generally refers to the idea of favoring solutions that fairly share happiness or dissatisfaction among agents. More formally, when comparing two cost vectors x and y (one component by agent), claiming that "x is more fair than y" bears to the vague notion that the components of x are "less spread out" or "more nearly equal" than the components of y are. This intuitive notion leaves room for many different definitions. The field has been explored by mathematicians who developed a formal theory of majorization [44] and by economists who studied the axiomatic foundations of inequality measures (for a synthesis see [47, 58]).

This body of knowledge has now a significant impact in computer sciences where many optimization problems require to incorporate the idea of fairness or equity in the definition of objectives. Let us mention for example multiagent job-shop scheduling problems, knapsack sharing problems, equitable approaches to location problems [49], fair bandwidth assignment, or any other resource allocation problem. This is also the case in the field of Artificial Intelligence where the notions of fairness and envy-freeness appear in various multiagent problems such as fair division of indivisible goods and combinatorial auctions [15, 22], paper assignment problems [33], marriage problems in social networks [29].

Example 1 *Let us consider a simple fair division problem where 5 items must be assigned to 5 agents. Every item is assigned to exactly one agent and each agent is assigned exactly one item. We want to find an assignment that fairly shares costs between agents, the costs being given by the following matrix of general term c_{ij} representing the cost of assigning item j to agent i:*

$$C = \begin{pmatrix} 5 & 8 & (4) & 9 & \mathbf{7} \\ \mathbf{1} & (3) & 2 & 7 & 8 \\ (3) & 9 & \mathbf{2} & 9 & 5 \\ 10 & 1 & 3 & (\mathbf{3}) & 4 \\ 5 & \mathbf{1} & 7 & 7 & (3) \end{pmatrix}$$

Any solution to this problem is a permutation that can be characterized by a square matrix Z of size 5 containing boolean variables z_{ij} where $z_{ij} = 1$ if and only if item j is assigned to agent i, Z having exactly one 1 in each row and column. To solve this multiagent assignment problem using standard optimization techniques, we could be interested in minimizing the average level of dissatisfaction among individuals, or equivalently the sum of individual dissatisfactions where the dissatisfaction of agent i is defined by $x_i = \sum_j c_{ij} z_{ij}$. This amounts to minimizing the linear function $\sum_i \sum_j c_{ij} z_{ij}$, a classical matching problem which can be solved in polytime with the Hungarian method. Here the optimal solution is given by setting to 1 variables z_{ij} corresponding to costs c_{ij} in bold in the C matrix. The associated dissatisfaction vector is given by $(7, 1, 2, 3, 1)$ which yields 14 as overall cost. However, this solution does not seem very fair. Although the average cost is below 3, one agent receives 7 whereas another gets 1.

If we consider now another permutation given by numbers into brackets in the cost matrix, we get a much preferable dissatisfaction profile regarding equity. For a slightly higher overall cost (16), we indeed obtain a significantly better balanced dissatisfaction profile: $(4, 3, 3, 3, 3)$. This solution actually minimizes the dissatisfaction of the least satisfied agent (min-max criterion) and the solution is here fully satisfactory. However, focusing on the least satisfied agent is not always convenient. It provides a pessimistic view on agents' satisfactions; moreover it is not very discriminating since multiple solutions remain equivalent from a worst case analysis point of view, even if they offer different perspectives to all but the least satisfy agent. The worst case can even mask very different situations as shown by this second example:

Example 2 *We consider an assignment problem with the following cost matrix:*

$$C' = \begin{pmatrix} 9 & 10 & (9) & 9 & \mathbf{10} \\ \mathbf{1} & (4) & 2 & 7 & 8 \\ (4) & 9 & \mathbf{2} & 9 & 5 \\ 10 & 1 & 3 & (\mathbf{2}) & 4 \\ 5 & \mathbf{1} & 7 & 7 & (4) \end{pmatrix}$$

Here, the optimal solution obtained with respect to the min-max criterion is given by numbers into brackets in the matrix. The associated dissatisfaction vector is $(9, 4, 4, 2, 4)$. However, in this case, the min-max solution might not be the best one. We could prefer sacrificing the least satisfied agent (who is apparently difficult to satisfy) so as to get better costs for the other agents. Hence vector $(10, 1, 2, 2, 1)$ that derives from positions in bold in matrix C' should be preferred to the previous one.

These examples show that simple objectives like min-sum or min-max are not perfectly suited to fair optimization problems. We will propose a more sophisticated model that attaches more importance to least satisfied agents without forgetting the other agents. It is based on an extension of a partial dominance concept known as *Lorenz Dominance* in Social Choice Theory and used for the measurement of inequalities. Our aim is to introduce this model and its main properties, and then elaborate a computationally efficient procedure using this model to generate fair solutions in multiagent assignment problems. For application purpose, we will consider one to one assignment problems as in Examples 1 and 2, but also many to many assignment problems such as conference paper assignment problems. The multiagent problems discussed in this chapter concern the case of centralized information. We assume that a central authority is responsible of computations and assignment of items. This is the case in various auctions problems and in conference paper assignment problems. It would also be interesting to study similar problems in decentralized contexts where the final assignment emerges from a sequence of local decisions of uncoordinated agents having only a partial view on the problem [22, 34]. Such problems are beyond the scope of our work.

This chapter is organized as follows: in Section 7.2 we recall some basic concepts used in Social Choice theory for the measurement of inequalities. In order to minimize agents' dissatisfaction while preserving fairness in assignment, we introduce the notion of infinite order Lorenz dominance as a refinement of Pareto and Lorenz dominance concepts. Then we establish a representation result for infinite order Lorenz dominance in Section 7.3, and we present some axiomatic properties of this model. The use of this model in multiagent assignment problems in presented in Section 7.4. In particular we formulate such problems as nonlinear 0-1 optimization problems, we study the problem complexity and present an approach to solve it using mixed integer linear programming. Finally numerical results showing the efficiency of our approach on randomly generated instances are presented, including a model of the paper assignment problem solved for realistic sizes.

7.2 Inequality measurement with Lorenz dominance relations

7.2.1 Notations and definitions

Considering a finite set of agents $N = \{1, \dots, n\}$, any solution of a multiagent combinatorial problem can be characterized by a cost vector $x = (x_1, \dots, x_n)$ in $\mathbb{R}^n_+$ whose i^{th} component represents the cost of solution x with respect to agent i. Hence, the comparison of solutions reduces to the comparison of their cost vectors. In this framework, the following definitions are useful:

Definition 18 *The Weak-Pareto dominance relation on cost vectors of* $\mathbb{R}^n_+$ *is defined, for all* $x, y \in \mathbb{R}^n_+$ *by:*

$$x \precsim_P y \iff [\forall i \in N, x_i \leq y_i)]$$

The Pareto dominance relation (P-dominance for short) on cost vectors of $\mathbb{R}^n_+$ *is defined as the asymmetric part of* $\precsim_P$*:*

$$x \prec_P y \iff [x \precsim_P y \text{ and } \text{not}(y \precsim_P x)]$$

Remark that $x \prec_P y$ means that x is preferred to y (x is less costly than y) since x and y are cost vectors representing individuals' dissatisfactions. Within a set X we say that x is *P-dominated* when $y \prec_P x$ for some y in X, and *P-nondominated* when there is no y in X such that $y \prec_P x$.

In order to decide whether a solution is better than another, we have to define a transitive preference relation $\precsim$ on cost vectors such that $x \precsim y$ when cost vector x is preferred to cost vector y. Let us introduce now the minimal requirements that such a relation $\precsim$ should satisfy to be seen as a reasonable synthesis of agents' opinions, favoring both efficiency and equity in comparisons. Firstly, we assume that all agents have the same importance. Hence, the following axiom formalizes the fact that all agents are treated equivalently:

Symmetry. For all $x \in \mathbb{R}^n_+$, for any permutation π of $\{1, \ldots, n\}$: $(x_{\pi(1)}, \ldots, x_{\pi(n)}) \sim (x_1, \ldots, x_n)$, where $\sim$ is the indifference relation defined as the symmetric part of $\precsim$.

In relation $\precsim$ we both want to capture the ideas of fairness and efficiency in cost-minimization. For this reason, $\precsim$ is expected to satisfy the following axioms:

P-Monotonicity. For all $x, y \in \mathbb{R}^n_+$, $x \precsim_P y \Rightarrow x \precsim y$ and $x \prec_P y \Rightarrow x \prec y$,

where $\succ$ is the strict preference relation defined as the asymmetric part of $\precsim$. P-monotonicity is a natural unanimity principle enforcing consistency with P-dominance.

Now the idea of fairness in comparisons is based on the following transfer principle:

Transfer Principle. Let $x \in \mathbb{R}^n_+$ such that $x_i > x_j$ for some i, j. Then for all ε such that $0 < \varepsilon < x_i - x_j$, $\;x - \varepsilon e_i + \varepsilon e_j \prec x$ where e_i (resp. e_j) is the vector whose i^{th} (resp. j^{th}) component equals 1, all others being null.

This axiom captures the idea of fairness as follows: if $x_i > x_j$ for some cost vector $x \in \mathbb{R}^n_+$, slightly improving (here decreasing) component x_i to the detriment of x_j while preserving the mean of the costs would produce a better distribution of costs and consequently improve the overall cost of the solution for the collection of agents. For example if $y = (9, 10, 9, 10)$ and $x = (11, 10, 7, 10)$ then the transfer principle implies $y \prec x$. Vector y is preferred because there exists a transfer of size $\epsilon = 2$ to pass from x to y. Note that using a similar transfer of size greater than 11 - 7 = 4 would increase inequality in terms of costs. This explains why the transfers must have a size $\varepsilon < x_i - x_j$. Such transfers are said to be *admissible* in the sequel. They are known as *Pigou-Dalton transfers* in Social Choice Theory, where they are used to reduce inequality of income distributions over a population (see [58] for a survey).

Note that the transfer principle possibly provides arguments to discriminate between vectors having the same average cost but does not apply in the comparison of vectors having different average costs. Hopefully, the possibility of discriminating is improved when combining the Transfer Principle with P-monotonicity. For example, to compare $w = (8, 10, 9, 10)$ and $z = (11, 10, 7, 12)$ we can use vectors x and y introduced above and observe that $w \prec y$ (P-Monotonicity), $y \prec x$ (Tranfer Principle explained above) and $x \prec z$ (P-Monotonicity). Hence $w \prec z$ by transitivity. In order to better characterize those vectors that can be compared using combinations of P-monotonicity and Transfer Principle we recall the definition of *Generalized* Lorenz vector and related concepts:

Definition 19 *For all $x \in \mathbb{R}^n_+$, the* Generalized Lorenz Vector *associated to x is the vector:*

$$L(x) = (x_{(1)}, x_{(1)} + x_{(2)}, \ldots, x_{(1)} + x_{(2)} + \ldots + x_{(n)})$$

where $x_{(1)} \geq x_{(2)} \geq \ldots \geq x_{(n)}$ represents the components of x sorted by decreasing order. The j^{th} component of $L(x)$ is $L_j(x) = \sum_{i=1}^{j} x_{(i)}$.

Definition 20 *The Generalized Lorenz dominance relation (L-dominance for short) on $\mathbb{R}^n_+$ is defined by:*

$$\forall x, y \in \mathbb{R}^n_+,\ x \precsim_L y \iff L(x) \precsim_P L(y)$$

Within a set X, element x is said to be *L-dominated* when $y \prec_L x$ for some y in X, and *L-nondominated* when there is no y in X such that $y \prec_L x$.

The notion of Lorenz dominance was initially introduced to compare vectors with the same average cost and its link to the transfer principle was established by Hardy, LittleHood and Polya [44]. The generalized version of L-dominance considered here is a classical extension allowing vectors with different averages to be compared (see [59]). In order to establish the link between Generalized Lorenz dominance and preferences satisfying combination of P-Monotonicity, Symmetry and Transfer Principle we recall a result of Chong [23] (see also [44] and [59]):

Theorem 1 *For any pair of distinct vectors $x, y \in \mathbb{R}^n_+$, if $x \prec_P y$, or if x obtains from y by a Pigou-Dalton transfer, then $x \prec_L y$. Conversely, if $x \prec_L y$, then there exists a sequence of admissible transfers and/or Pareto-improvements to transform y into x.*

For example we have: $L(w) = (10, 20, 29, 37) \prec_P L(z) = (12, 23, 33, 40)$ which directly proves the existence of a sequence of Pareto improvements and/or admissible transfers passing from z to w. This theorem establishes L-dominance as the minimal transitive relation (with respect to set inclusion) satisfying simultaneously P-Monotonicity, Symmetry and the Transfer Principle. Hence, the subset of L-nondominated elements defines the best candidates to optimality in fair optimization problems.

Due to P-monotonicity, the set of L-nondominated elements is included in the set of Pareto optimal vectors. Unfortunately, in multi-objective combinatorial optimization problems, the set of L-nondominated solutions can be huge (see [51]). This problems occurs also in multiagent assignment problems. As we will see later in Example 5, there exists family of instances where the number of L-nondominated cost vectors grows exponentially with the size of the problem. This is the reason why we introduce in the next section more discriminating dominance concepts that extend L-dominance to richer preference structures.

Other attempts in this direction have been proposed in Social Choice Theory. The most common way is resorting to a Schur-convex function ψ to construct a weak-order defined by $x \precsim y \Leftrightarrow \psi(x) \leq \psi(y)$. A Schur-convex function (also known as order-preserving function) is a function $\psi : \mathbb{R}^n \to \mathbb{R}$ such that

$\forall x, y \in \mathbb{R}^n$, $x \precsim_L y \Longrightarrow \psi(x) \leq \psi(y)$. For example, every function that is convex and symmetric is also Schur-convex. Well known examples of such functions are S-Gini indices and more generally instances of Yaari's model [64] of the following form:

Example 3 *The Yaari's Social Welfare Functions of the following form are Schur-convex:*

$$W_f(x) = \sum_{i=1}^{n} \left[f\left(\frac{n-i+1}{n}\right) - f\left(\frac{n-i}{n}\right) \right] x_{(i)} \tag{7.1}$$

where f is a strictly increasing continuous function such that $f(0) = 0$ and $f(1) = 1$. S-Gini indices are particular instances obtained for $f(z) = z^\delta$, $\delta > 1$, see [28]. Note that when $f(z) = z^2$ the absolute inequality index $G(x) = 1 - W_f(x)\ \bar{x}$ is nothing else but the so-called Gini index.

There are other ways of refining Lorenz dominance. We can import some ideas from the literature on Decision Making under risk, where people are interested in comparing probability distributions in terms of risk. In this context, the counterpart of Lorenz dominance is the second-order stochastic dominance (SSD for short) that defines a partial order on probability distributions. The SSD model does not permit to compare any pair of distributions, but it can be refined by stochastic dominances of higher orders, each of them refining the previous one. The ultimate result of this process is named infinite order stochastic dominance (see [40]). The next subsection proposes the construction of progressive refinements of L-dominance using similar mechanisms.

7.2.2 Infinite order Lorenz dominance

Refinement of Lorenz dominance can be obtained by iterating $L(.)$ transformation so as to define higher order L-dominance relations. Observing indeed that P-monotonicity holds for L-dominance (see Theorem 1), L-dominance appears as a refinement of Pareto dominance. Whenever x and y cannot be compared in terms of P-dominance we compare instead $L(x)$ and $L(y)$. If no Pareto dominance holds, the indetermination might be solved by comparing $L^2(x) = L(L(x))$ and $L^2(y) = L(L(y))$. This process can be iterated mechanically to higher levels with the aim of reducing incomparability. This leads to consider k^{th} order Lorenz vector $L^k(x)$ defined by:

$$L^k(x) = \begin{cases} x & \text{if } \ k = 0 \\ L(L^{k-1}(x)) & \text{if } \ k > 1 \end{cases}$$

and the k^{th} order Lorenz dominance defined by:

$$\forall x, y \in \mathbb{R}^n_+,\ x \precsim^k_L y \iff L^k(x) \precsim_P L^k(y)$$

Then we define strict *infinite order dominance* (strict L^∞-dominance for short) as follows[1]:

$$\prec_L^\infty = \bigcup_{k\geq 1} \prec_L^k$$

Note that, $\prec_L^0$ and $\prec_L^1$ correspond to P-dominance and L-dominance respectively. Then, due to P-monotonicity, $x \prec_L^k y \Rightarrow x \prec_L^{k+1} y$ for any k and relations $\prec_L^k$ form a *nested* sequence of strict partial orders. This suggests that $\prec_L^\infty$ might be computed, for any pair $x, y \in \mathbb{R}_+^n$ by Algorithm 1 given below:

Algorithm 1: Testing strict L^∞-dominance

```
$u \leftarrow x$;
$v \leftarrow y$;
while [$not(u \prec_P v$ or $v \prec_P u)$] do
    $u \leftarrow L(u)$;
    $v \leftarrow L(v)$;
end
if $(u \prec_P v)$ then $x \prec_L^\infty y$;
if $(v \prec_P u)$ then $y \prec_L^\infty x$
```

For example, consider a 4 agents problem with 3 Pareto optimal feasible vectors $x = (3, 2, 3, 2), y = (3, 3, 3, 0)$ and $z = (1, 3, 2, 4)$. We have $L(x) = (3, 6, 8, 10), L(y) = (3, 6, 9, 9)$ and $L(z) = (4, 7, 9, 10)$. Hence we get $x \prec_L^\infty z$ and $y \prec_L^\infty z$. We need to go one step ahead to compare x and y. We get $L^2(x) = (10, 18, 24, 27)$ and $L^2(y) = (9, 18, 24, 27)$, therefore $y \prec_L^\infty x$.

Note that our definition of infinite order Lorenz dominance assumes that the vectors to be compared are cost vectors. It does not fit for utility vectors. A simple way of adapting our approach to compare two utility vectors $(u_1, \ldots, u_n)$ and $(v_1, \ldots, v_n)$ according to infinite order dominance is to check whether $(M - u_1, \ldots, M - u_n) \prec_L^\infty (M - v_1, \ldots, M - v_n)$ for an arbitrary M chosen greater than all u_i and v_i, $i = 1, \ldots, n$. This adaptation is consistent with the definition of $\prec_L^\infty$ for cost vectors and does not depend on the choice of M.

Algorithm 1 tries to discriminate between vectors that were not discriminated by Lorenz dominance. However, nothing proves that the algorithm terminates for all pairs of vectors. Moreover the mechanical iteration of Lorenz dominance used to introduce the model is not easy to manipulate when we study the properties of the model. We need another characterization of $\prec_L^\infty$ to be able to propose a fully operational decision procedure and to be able to better understand the role of each agent in the decision process. For this reason, in the following section, we characterize the vectors that can be discriminated by Algorithm 1. We also provide a direct mathematical definition of strict L^∞-dominance making it possible to compare any pair of vectors in $O(n \log(n))$.

[1]Although L-dominance can be seen as a particular instance of second order stochastic dominance (assuming a uniform probability distribution on agents), the notion of infinite order Lorenz dominance we introduce here must not be confused with infinite order stochastic dominance that results from a different construction.

7.3 Properties of infinite order Lorenz dominance

7.3.1 A representation theorem

In this section, we establish a representation result for strict L^∞-dominance. We present an algebraic reformulation of Lorenz vectors and establish technical lemmas; the main result will follow immediately.

For $x \in \mathbb{R}^n_+$, we define $x^\uparrow$ as the vector resulting from sorting the components of x in increasing order, that is: $x^\uparrow = (x_{(n+1-i)})_{i=1\dots n}$ since we have defined $(.)$ as the permutation sorting the components of x in decreasing order. As the definition of $L(.)$ respects the Symmetry Axiom, we have: $L(x^\uparrow) = L(x)$. We now introduce the $n \times n$ matrix:

$$\mathcal{L} = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ \vdots & \iddots & \iddots & \vdots \\ 0 & \iddots & & 1 \\ 1 & \cdots & \cdots & 1 \end{pmatrix}$$

defined by: $l_{ij} = 1$ if $i + j > n$, 0 otherwise.

Proposition 1 *For $x \in \mathbb{R}^n_+$, $\forall k, L^k(x) = \mathcal{L}^k.x^\uparrow$*

Schetch of the proof. For a vector y whose components are sorted in increasing order, it is immediate to verify that $L(y)$ (the Lorenz vector of y) is equal to $\mathcal{L}.y$ (product of the matrix $\mathcal{L}$ and the vector y). Therefore, for any vector x, we have: $L(x) = L(x^\uparrow) = \mathcal{L}.x^\uparrow$. As $L(x)$ is a vector whose components are sorted in increasing order, the equality holds at any order: $\forall k, L^k(x) = \mathcal{L}^k.x^\uparrow$ $\square$

$\mathcal{L}$ being a symmetric real matrix, the finite-dimensional spectral theorem applies and we can find P, an orthogonal matrix (such that ${}^tP = P^{-1}$) and n real eigenvalues $\lambda_1 \dots \lambda_n$ (duplicated according to their multiplicity, with $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|$) such that: $\mathcal{L} = {}^tP\ Diag(\lambda_1 \dots \lambda_n)\ P$ and therefore $\mathcal{L}^k = {}^tP\ Diag(\lambda_1^k \dots \lambda_n^k)\ P$, $Diag(a_1 \dots a_n)$ being the diagonal matrix of elements $a_1, a_2, \dots, a_n$.

Let $w = \frac{\pi}{2n+1}$:

Lemma 1 *The n eigenvalues of $\mathcal{L}$ are $\lambda_k = \frac{(-1)^{k+1}}{2\sin\left(\frac{(2k-1)w}{2}\right)}$ for $k \in [1; n]$, with eigenvectors $V_k = \left(\sin\left(i(2k-1)w\right)\right)_{i=1\dots n}$*

Schetch of the proof. We introduce $A = 2\ \mathrm{Id} - \mathcal{L}^{-2}$, which is classical to diagonalize ($\mathcal{L}^{-2}$ is the square of the inverse matrix of $\mathcal{L}$). The n eigenvalues of A are $2\cos((2k-1)w)$, $k = 1 \dots n$, with eigenvectors $\left(\sin\left(i(2k-1)w\right)\right)_{i=1\dots n}$. These n eigenvalues of A are in $(0, 2)$ and have different absolute values. Using the relation between A and $\mathcal{L}$, we are able to determine the eigenvalues of $\mathcal{L}$ and its eigenvectors. $\square$

Decomposing $\mathcal{L}^k$ into n lines $\mathcal{L}_1^k \dots \mathcal{L}_n^k$, we have:

$$x \precsim_L^k y \quad \Leftrightarrow \quad \mathcal{L}^k.x^\uparrow \precsim_P \mathcal{L}^k.y^\uparrow \quad \Leftrightarrow \quad \begin{cases} \mathcal{L}_1^k.x^\uparrow \leq \mathcal{L}_1^k.y^\uparrow \\ \qquad \vdots \\ \mathcal{L}_n^k.x^\uparrow \leq \mathcal{L}_n^k.y^\uparrow \end{cases}$$

Thus, the k^{th} order Lorenz dominance can be rewritten as the intersection of n orders. To determinate $\prec_L^\infty$, we will express these n orders and prove that they are equivalent when $k \to \infty$, and that this equivalent admits a limit when $k \to \infty$, limit that can be rewritten as the strict order induced by an OWA function.

Definition 21 *OWA means* ordered weighted average. *It is a family of aggregators introduced by Yager [65] characterized by $W(x) = \sum_{k=1}^n w_k x_{(k)}$. W is a symmetric function of its arguments. The weights w_k do not represent the importance of agents but the attention we pay to agents depending on their rank in the satisfaction order.*

Let E_i be the square matrix of dimension n with all values to 0, excepted a 1 in position (i,i). Let P_i be the i^{th} line of matrix P and $A_i = {}^tP_iP_i$. We have:

$$\mathcal{L}^k = {}^tP\left(\sum_{i=1}^n \lambda_i^k E_i\right)P = \sum_{i=1}^n \lambda_i^k ({}^tP_iP_i) = \sum_{i=1}^n \lambda_i^k A_i$$

Lemma 2 *For $x \in \mathbb{R}_+^n$, $L^k(x) \sim \lambda_1^k\ {}^tP_1 \times \mathcal{W}(x)$ where $\mathcal{W}$ is the OWA function whose weights are the components of P_1 in reverse order and $\sim$ is the relation of asymptotical equivalence.*

Proof By Lemma 1 giving the eigenvalues of $\mathcal{L}$, we have $|\lambda_1| > |\lambda_2| > ... > |\lambda_n|$. Moreover, the eigenvector associated to the greatest eigenvalue λ_1 being $V_1 = {}^tP_1 = \left(\sin(iw)\right)_{i=1...n} > 0$, we have ${}^tP_1P_1 > 0$ and therefore, when $k \to +\infty$, we have (since for $1 < i \leq n$, $\lambda_i^k = o(\lambda_1^k)$ when $k \to +\infty$): $\mathcal{L}^k \sim \lambda_1^k\ {}^tP_1P_1$ so that we can write: $\mathcal{L}^k(x) \sim \lambda_1^k\ {}^tP_1(P_1.x^\uparrow)$. $P_1.x^\uparrow$ is a weighted average of the components of the increasing vector $x^\uparrow$. It can be rewritten as an OWA on x, since if we define $\mathcal{W}$ as the OWA criterion whose weights are the components of P_1 in reverse order, we have: $\mathcal{W}(x) = \sum_{i=1}^n (P_1)_{n+1-i}\ x_{(i)} = P_1.x^\uparrow$. We finally obtain the following equivalent of the iteration of Lorenz dominance when $k \to \infty$: $L^k(x) \sim \lambda_1^k\ {}^tP_1 \times \mathcal{W}(x)$ □

Lemma 3 *For $x, y \in \mathbb{R}_+^n$, $\mathcal{W}(x) < \mathcal{W}(y) \Rightarrow x \prec_L^\infty y$.*

Proof If $\mathcal{W}(x) < \mathcal{W}(y)$, then, as $L^k(x) \sim \lambda_1^k\ {}^tP_1 \times \mathcal{W}(x)$ (by Lemma 2), we have for k large enough: $L^k(x) \succ_P L^k(y)$, so $x \succ_L^k y$, and therefore $x \prec_L^\infty y$. □

Lemma 4 *For $x, y \in \mathbb{R}_+^n$, if $\mathcal{W}(x) = \mathcal{W}(y)$, then x and y are incomparable by $\prec_L^\infty$.*

Proof The equivalent of $L^k(x)$ of Lemma 2 cannot discriminate between two vectors x and y when $\mathcal{W}(x) = \mathcal{W}(y)$. We need to use the other eigenvalues of $\mathcal{L}$ to try to discriminate between these two vectors. As $\mathcal{W}(x) = \mathcal{W}(y)$, we can write, according to the result of Proposition 1:

$$L^k(y) - L^k(x) = \sum_{i=2}^{n} \lambda_i^k \; {}^tP_i P_i (y^\uparrow - x^\uparrow)$$

If for all $j, P_j(y^\uparrow - x^\uparrow) = 0$, then $P(y^\uparrow - x^\uparrow) = 0$ and $x^\uparrow = y^\uparrow$ (P being an invertible matrix), and therefore there is no strict $\prec_L^\infty$-dominance between x and y, since $L(x) = L(y)$, and they are incomparable by $\prec_L^\infty$.

Else, let j be the first index such that $P_j(y^\uparrow - x^\uparrow) \neq 0$. As ${}^tPP = Id$, for $i > 1$, ${}^tP_1P_i = 0$. But since $P_i \neq 0$ (P being an invertible matrix) and $P_1 > 0$, P_i must have a component i_1 strictly positive and another i_2 strictly negative to verify ${}^tP_1P_i = 0$. But we have, for any component i:

$$L_i^k(y) - L_i^k(x) \sim \lambda_j^k \; {}^t(P_j)_i (P_j(y^\uparrow - x^\uparrow))$$

so for k sufficiently large, the component i_1 of $L^k(y) - L^k(x)$ is strictly positive and i_2 stricly negative; thus $\succ_L^k$-dominance cannot hold and x and y are incomparable by $\prec_L^\infty$. □

We are now in position to formulate our main result:

Theorem 2 *The strict L^∞-dominance has a direct numerical representation using the following ordered weighted average:*

$$\mathcal{W}(x) = \sum_{k=1}^{n} \sin\left(\frac{(n+1-k)\pi}{2n+1}\right) x_{(k)}$$

This representation is given by the following property:

$$\forall x, y \in \mathbb{R}_+^n,\; x \prec_L^\infty y \iff \mathcal{W}(x) < \mathcal{W}(y)$$

Proof $\Rightarrow$: if $x \prec_L^\infty y$, then x and y are not incomparable by $\prec_L^\infty$. Therefore, the contrapositive of Lemma 4 ensures that $\mathcal{W}(x) \neq \mathcal{W}(y)$. But then, as $x \prec_L^\infty y$, we cannot have $\mathcal{W}(x) > \mathcal{W}(y)$ (since Lemma 3 would apply and imply that $y \prec_L^\infty x$). We finally must have $\mathcal{W}(x) < \mathcal{W}(y)$.

$\Leftarrow$: the proof is straightforward by Lemma 3. □

Actually, to compare two vectors x and y according to strict L^∞-dominance, we do not need to run Algorithm 1. We compute instead $\mathcal{W}(x)$ and $\mathcal{W}(y)$ in $O(n\log(n))$. If $\mathcal{W}(x) \neq \mathcal{W}(y)$, the vector having the smallest score by $\mathcal{W}$ strictly L^∞-dominates the other. Hence, Algorithm 1 would stop after a sufficiently large number of iterations. Whenever $\mathcal{W}(x) = \mathcal{W}(y)$, no strict dominance holds at any order of the iteration of Lorenz dominance. This shows that Algorithm 1 would never terminate in this case. This illustrates the meaning and utility of our representation result.

The weights of $\mathcal{W}$ are strictly decreasing. This is a typical feature of OWA operators compatible with the transfer Principle [49]. Attaching more importance to less satisfied agents seems natural to model fairness. Another by-product of Theorem 2 is to explain how we could choose the weights of the OWA operator to model fairness, instead of using arbitrary strictly decreasing weights.

The last remark deals with vectors incomparable by strict L^∞-dominance: the relation "is incomparable with" is a relation of equivalence (since it means having the same value by function $\mathcal{W}$). Therefore, it is natural to extend the strict L^∞-dominance to a weak order as follows:

Definition 22 *The L^∞-dominance is a weak order extending strict L^∞-dominance as follows:*

$$\forall x, y \in \mathbb{R}_+^n,\ x \precsim_L^\infty y \iff \mathcal{W}(x) \leq \mathcal{W}(y)$$

This numerical representation of L^∞-dominance is very helpful to analyze the axiomatic properties of the model. The fact that $\mathcal{W}(x)$ is an ordered weighted average (OWA) with positive and *strictly decreasing* weights w_k as k increases makes sense. It means that all agents play a role in the evaluation of solutions but, when evaluating a given solution, we attach more importance to least satisfied agents. This is in accordance with the intuitive idea of fairness presented in the introduction. In particular we have $(4,3,3,3,3) \prec_L^\infty (7,1,2,3,1)$ and $(10,1,2,2,1) \prec_L^\infty (9,4,4,2,4)$ as desired in Examples 1 and 2, which outperforms the possibilities of min-sum and min-max criteria.

7.3.2 Main properties of L^∞-dominance

We first exhibit two important propositions concerning $\mathcal{W}(x)$, the main properties satisfied by $\precsim_L^\infty$ will then derive immediately.

Proposition 2 *$\mathcal{W}(x)$ can be expressed as a linear combination of the components of $L(x)$ using only strictly positive coefficients. We have:*
$\mathcal{W}(x) = \sum_{k=1}^{n-1}(w_k - w_{k+1})L_k(x) + w_n L_n(x) = w'.L(x)$
with $w' = (w_1 - w_2, w_2 - w_3, \ldots, w_{n-1} - w_n, w_n)$.

Proof Remarking that $x_{(1)} = L_1(x)$ and $x_{(k)} = L_k(x) - L_{k-1}(x)$ for $k = 2, ..., n.$, it is sufficient to make the substitution to get the desired linear combination with weights $w'_k = w_k - w_{k+1} > 0$ for $k < n$ and $w'_n = w_n > 0$. $\square$

Proposition 3 *$\mathcal{W}$ is a Schur convex function.*

Proof We have to prove that $x \precsim_L y \Rightarrow \mathcal{W}(x) \leq \mathcal{W}(y)$. If $x \precsim_L y$ then by definition we have $L(x) \precsim_P L(y)$. Hence, considering the positive weighting vector w' used in the proof of Proposition 2, we have $w'_k.L_k(x) \leq w'_k.L_k(y)$ for $k = 1, ..., n$. After summing these n equalities, we get the result using proposition 2. $\square$

This shows that $\precsim_L^\infty$ is based on a Schur-convex function, like S-Gini indices and Yaari's model introduced in Example 3. As recalled before, Schur convex functions are known as convenient tools to measure inequalities in majorization theory (see [44]). We present now five properties satisfied by $\precsim_L^\infty$ which are consequences of Proposition 3. Property P1 shows that all vectors having the same Lorenz vector are treated equivalently:

P1: Neutrality. For all x, y in X, $L(x) = L(y) \Rightarrow x \sim_L^\infty y$.

Property P2 makes explicit the fact that $\prec_L^\infty$ is a refinement of Lorenz-dominance.

P2: Strict L-Monotonicity. $x \prec_L y \Rightarrow x \prec_L^\infty y$.

Then we introduce 3 axioms that better explain how $\precsim_L^\infty$ works with Lorenz vectors.

P3: Complete weak-order. $\precsim_L^\infty$ is reflexive, transitive and complete.

P4 Continuity. Let x, y, z be 3 cost-vectors such that $x \prec_L^\infty y \prec_L^\infty z$. There exists $\alpha, \beta \in]0,1[$ such that:

$$\alpha x + (1-\alpha)z \prec_L^\infty y \prec_L^\infty \beta x + (1-\beta)z.$$

Proof We have indeed $x \prec_L^\infty y \prec_L^\infty z \Rightarrow \mathcal{W}(x) < \mathcal{W}(y) < \mathcal{W}(z)$. Whenever $\alpha \to 1$ then the sequence of vectors of general term $\alpha x + (1-\alpha)z$ tends to x and, by continuity of $\mathcal{W}$, $\mathcal{W}(\alpha x + (1-\alpha)z) \to \mathcal{W}(x)$. Hence for α sufficiently close to 1, $\mathcal{W}(\alpha x + (1-\alpha)z)$ is sufficiently close to $\mathcal{W}(x)$ to be inferior to $\mathcal{W}(y)$. Hence $\alpha x + (1-\alpha)z \prec_L^\infty y$. We deliberately omit the other part of the proof that works similarly with $\beta \to 0$. □

The last property is a restriction to comonotonic vectors of the so-called independence axiom proposed by Von Neumann and Morgenstern [63] in the framework of utility theory. Comonotonicity of vectors is defined as follows:

Definition 23 *Two cost vectors x and y are said to be* comonotonic *if $x_i > x_j$ and $y_i < y_j$ for no $i, j \in \{1, \ldots, n\}$.*

Two solutions having comonotonic cost vectors satisfy the agents in the same order. It is useful to remark that, for any pair (x, y) of comonotonic vectors, there exists a permutation π of $\{1, \ldots, m\}$ such that $x_{\pi(1)} \geq x_{\pi(2)} \geq \ldots \geq x_{\pi(m)}$ and $y_{\pi(1)} \geq y_{\pi(2)} \geq \ldots \geq y_{\pi(m)}$. Consequently, $\mathcal{W}(\alpha x + (1-\alpha)y) = \alpha\mathcal{W}(x) + (1-\alpha)\mathcal{W}(y)$. We can now establish our last property:

P5 Comonotonic Independence. Let x, y, z 3 comonotonic cost vectors. Then, for all $\alpha \in]0,1[$:

$$x \prec_L^\infty y \Longrightarrow \alpha x + (1-\alpha)z \prec_L^\infty \alpha y + (1-\alpha)z.$$

Proof We have $x \prec_L^\infty y \Rightarrow \mathcal{W}(x) < \mathcal{W}(y)$. Hence $\mathcal{W}(\alpha x) < \mathcal{W}(\alpha y)$ and $\mathcal{W}(\alpha x) + \mathcal{W}((1-\alpha)z) < \mathcal{W}(\alpha y) + \mathcal{W}((1-\alpha)z)$. Since x and z are comonotonic we have $\mathcal{W}(\alpha x) + \mathcal{W}((1-\alpha)z) = \mathcal{W}(\alpha x + (1-\alpha)z)$. Moreover, y and z are comonotonic; hence we have $\mathcal{W}(\alpha y) + \mathcal{W}((1-\alpha)z) = \mathcal{W}(\alpha y + (1-\alpha)z)$. Finally we get $\mathcal{W}(\alpha x + (1-\alpha)z) < \mathcal{W}(\alpha y + (1-\alpha)z)$ and therefore $\alpha x + (1-\alpha)z \prec_L^\infty \alpha y + (1-\alpha)z$. □

Note that the restriction to comonotonic vectors is necessary within an independence axiom used for the measurement of inequalities [64]. If we forget it in the premisses of P5, we obtain a property which is incompatible with the Strict L-monotonicity axiom, as shown by the following:

Example 4 *Let us consider* $x = (24, 24)$*,* $y = (22, 26)$ *and* $z = (26, 22)$ *which are not comonotonic. Due to Strict L-monotonicity* $x \prec_L^\infty y$*. Hence, usual independence would imply* $(25, 23) = \frac{1}{2}x + \frac{1}{2}z \prec_L^\infty \frac{1}{2}y + \frac{1}{2}z = (24, 24)$ *which is in contradiction with* $(24, 24) \prec_L (25, 23)$*.*

The above properties exhibit nice features of L^∞-dominance and underline some relationships with Yaari's model introduced in Example 3. This is natural because all these models are based on OWA operators with decreasing weights.

We have shown that fair optimization in multiagent problems can reasonably be formulated as minimizing function $\mathcal{W}(x)$ over feasible cost vectors. However $\mathcal{W}(x)$ is not a linear function since, for non-comonotonic vectors x, y, $\mathcal{W}(x + y) \neq \mathcal{W}(x) + \mathcal{W}(y)$ in general. Hence minimizing $\mathcal{W}(x)$ requires non-linear optimization. The next section is devoted to this point in the context of many to many multiagent assignment problems.

7.4 Solving multiagent assignment problems

The general many to many multiagent assignment problem we are considering can be stated as follows: we want to assign m items to n agents. The number of items assigned to agent i is restricted to interval $[l_i, u_i]$, $i = 1, \dots, n$. Item j must be assigned to a number of agents restricted to the interval $[l'_j, u'_j]$, $j = 1, \dots, m$. A $n \times m$ matrix gives the cost c_{ij} of assigning item j to agent i.

This general problem occurs in many contexts such as paper assignment problems, social meeting on the web, resource allocation, transportation problems. The possible solutions are characterized by a $n \times m$ matrix of booleans z_{ij} representing the possibility of assigning item j to individual i. Hence the problem can be formalized as a multiobjective 0-1 linear optimization problem:

$$\text{Min } \; x_i = \sum_{j=1}^{m} c_{ij} z_{ij}, \quad i = 1, \dots, n$$

$$s.t. \begin{cases} l'_j \le \sum_{i=1}^{n} z_{ij} \le u'_j & j = 1, \dots, m \\ l_i \le \sum_{j=1}^{m} z_{ij} \le u_i & i = 1, \dots, n \\ z_{ij} \in \{0, 1\} & \forall i, \forall j \end{cases}$$

This general multiobjective program fits to many different situations involving multiple agents. For example, in fair allocation of indivisible goods, we set $l'_j = u'_j = 1$, $j = 1, \dots, m$. This is the case when we have to distribute presents to kids as in the Santa Claus problem [9], or in some auctions problems. Alternatively, the multiobjective problem can easily model a conference paper assignment problem. In this case $l'_j = u'_j = 3$, $j = 1, \dots, m$ (a paper must be reviewed by 3 PC members) and u_i (resp. l_i) represent the maximal (resp. minimal) number of papers we want to assign to reviewer i. In this case, x_i represents the overall charge or dissatisfaction of agent i.

In such multiagent combinatorial problems, the number of Pareto-optimal solutions and L-nondominated solutions can be large, as illustrated in the following example.

Example 5 *Consider a particular instance of the above problem with m items to be assigned to 2 agents ($n = 2$). Assume that $l_1 = l_2 = 0$, $u_1 = u_2 = n$, $l'_j = u'_j = 1$, $j = 1, \dots, m$ with costs $c_{1j} = 2^j$ and $c_{2j} = 2^{j-1}$, $j = 1 \dots m-1$, $c_{1m} = 4^m$, $c_{2m} = 2^m + 1$. This gives 2^m distinct feasible assignments. The half of them assigns item m to agent 1 which is prohibitive. All of them are L-dominated. The other half produces cost vectors $\{(2k, 3 \times 2^{m-1} - k), k \in \{0, \dots, 2^{m-1} - 1\}$. Note that $3 \times 2^{m-1} - k > 2k$. Consequently, the associate Lorenz vectors are $\{(3 \times 2^{m-1} - k, 3 \times 2^{m-1} + k),\ k \in \{0, \dots, 2^{m-1} - 1\}\}$. No P-dominance holds between these Lorenz vectors because their sum of components is constant. Hence we have 2^{m-1} L-nondominated solutions.*

In this family of instances, the number of L-nondominated feasible cost vectors grows exponentially with m. Even if we want only one feasible solution for each distinct cost vector, the size of the output set remains exponential in m. Clearly, L^∞-dominance can help to reduce the set of optimal solutions.

7.4.1 Fair multiagent optimization

Using the result established in Theorem 2, the search of an optimal many to many assignment problem with respect to L^∞-dominance can be formulated as the following 0-1 nonlinear optimization problem (Π):

$$\text{Min } \mathcal{W}(x) = \sum_{k=1}^{n} \sin\left(\frac{(n+1-k)\pi}{2n+1}\right) x_{(k)} \tag{7.2}$$

$$(\Pi) \qquad s.t. \begin{cases} x_i = \sum_{j=1}^{m} c_{ij} z_{ij} & i = 1, \dots, n \\ l'_j \le \sum_{i=1}^{n} z_{ij} \le u'_j & j = 1, \dots, m \\ l_i \le \sum_{j=1}^{m} z_{ij} \le u_i & i = 1, \dots, n \\ z_{ij} \in \{0, 1\} & \forall i, \forall j \end{cases} \tag{7.3}$$

Proposition 4 *The problem P_α consisting in deciding whether there exists an assignment with cost $\mathcal{W}(x) \le \alpha$ is an NP-complete decision problem for any fixed positive α.*

Proof P_α is clearly in NP. To establish NP-completeness, we reduce the NP-complete Partition Problem to our problem. The Partition Problem is stated as follows:
Instance: finite set $A = \{a_1, \dots, a_m\}$ of items and a size $s(a) \in \mathbb{N}$ for each $a \in A$.

Question: is it possible to partition A into two sets of objects of equal weights? From an instance of Partition Problem, we construct in polynomial time an instance of P_α with $n = 2$, $l_1 = l_2 = 0$, $u_1 = u_2 = m$, $l'_j = u'_j = 1$, and $c_{1j} = c_{2j} = s(a_j)$, $j = 1, \ldots, m$. Moreover we set $\alpha = (w_1 + w_2)\beta$ with $\beta = \sum_{a \in A} s(a)/2$. Hence, the answer to P_α is YES if and only if the answer to the partition problem is YES. Indeed, if there is a solution to the partition problem, then there exists an assignment with cost (β, β) and $\mathcal{W}(\beta, \beta) = \alpha$. Moreover, if the answer to the partition problem is NO, then any partition of A into two subsets is unfair and the corresponding assignment leads to a cost vector of type $(\beta - \varepsilon, \beta + \varepsilon)$, $\varepsilon \in (0, \beta]$. Since $(\beta, \beta) \prec_L (\beta - \varepsilon, \beta + \varepsilon)$ we have $\mathcal{W}(\beta - \varepsilon, \beta + \varepsilon) > \mathcal{W}(\beta, \beta) = \alpha$. So there is no assignment such that $\mathcal{W}(x) = \alpha$; the answer to P_α is NO. $\square$

7.4.2 Linearization of the problem

Thanks to Proposition 2, Π rewrites:

$$(\Pi') \qquad \text{Min } \mathcal{W}(x) = \sum_{k=1}^{n} w'_k L_k(x) \;\; s.t. \;\; (7.3)$$

with $w' = (w_1 - w_2, w_2 - w_3, \ldots, w_{n-1} - w_n, w_n)$. Following an idea introduced in [50], we express the k^{th} component $L_k(x)$ of the Lorenz vector $L(x)$ as the solution of the following linear program:

$$\text{Max } \Big(\sum_{i=1}^{n} \alpha_{ik} x_i\Big) \quad s.t. \left\{ \begin{array}{ll} \sum_{i=1}^{n} \alpha_{ik} = k & \\ 0 \le \alpha_{ik} \le 1 & i = 1 \ldots n \end{array} \right.$$

Its optimal value is clearly the sum of the k greatest components of x, that is $L_k(x)$. This is also the optimal value of the dual problem:

$$\text{Min } \Big(k\, r_k + \sum_{i=1}^{n} b_{ik}\Big) \quad s.t. \left\{ \begin{array}{ll} r_k + b_{ik} \ge x_i & i = 1 \ldots n \\ b_{ik} \ge 0 & i = 1 \ldots n \end{array} \right.$$

We can therefore combine the linear program above with Π' (since both are in minimization and $w' > 0$) and rewrite our problem Π as the following mixed integer linear program:

$$\text{Min } \sum_{k=1}^{n} w'_k \left(k \times r_k + \sum_{i=1}^{n} b_i^k \right)$$

$$(\Gamma) \qquad s.t. \left\{ \begin{array}{ll} l'_j \le \sum_{i=1}^{n} z_{ij} \le u'_j & j = 1, \ldots, m \\ l_i \le \sum_{j=1}^{m} z_{ij} \le u_i & i = 1, \ldots, n \\ r_k + b_{ik} \ge \sum_{j=1}^{m} c_{ij} z_{ij} & \forall i, k = 1, \ldots, n \\ b_{ik} \ge 0 & \forall i, \forall k \\ z_{ij} \in \{0, 1\} & \forall i, \forall j \end{array} \right.$$

Γ has $2(n^2 + m + n)$ constraints, nm 0-1 variables, and $n^2 + n$ continuous variables.

7.4.3 Numerical tests

We present here numerical tests[2] performed on random instances of one-to-one and many-to-many multiagent assignment problems. To solve the mixed integer linear program Γ, we used ILOG CPLEX 11.100 on a computer with 4 Go of memory and an Intel Core 2 Duo 2.66 GHz processor. Table 1 (resp. Table 2) gives the results obtained for the assignment of m objects to $n = m$ agents, with $l_i = l'_i = u_i = u'_i = 1$ and costs randomly generated in $[1, 1000]$ (resp. $[1, 20]$). Table 3 is the test on the paper assignment problem modeled as follows: $n = m/4$, each reviewer receives at most 9 papers ($l_i = 0$ and $u_i = 9$), a paper has to be reviewed by exactly 2 reviewers ($l'_j = u'_j = 2$), and a reviewer expresses his preferences for reviewing a paper with a number between 0 and 5 (i.e. costs are in $[0, 5]$). The computation times expressed in seconds represent average solution times over 20 random instances of the same size m (number of objects) with a timeout set to 1000 seconds.

m	t
10	.01
20	.09
30	.33
40	1.52
50	5.14
60	16.1
70	34.0
80	81.8
90	136
100	275

m	t
100	.93
200	3.65
300	17.4
400	52.8
500	104
600	161
700	390
800	482
900	843
1000	>1000

m	t
200	3.51
300	5.63
400	13.9
500	35.7
600	79.4
700	148
800	303
900	478
1000	904
1100	>1000

1. Costs in $[1, 1000]$ 2. Costs in $[1, 20]$ 3. Paper Assignment

So, it is possible to find a fair solution to the paper assignment problem with realistic parameters for a standard conference within a reasonable time. The approach presented here remains valid for finding fair assignments by optimization of S-Gini indices and other instances of the Yaari's model. Indeed, as ordered weighted averages, such indices can be linearized similarly as $\mathcal{W}$. We have performed tests showing that solution times using such criteria are in the same order of magnitude.

Our results can be extended to the case of weighted agents (which occurs, for example, in resource allocation problems, where agents can have exogenous rights represented by individual weights). It is possible to show that the weighted extension of L-dominance converges by iteration towards a weighted extension of OWA. The associate optimization problem can be solved efficiently by slightly modifying the mixed integer linear program Γ.

[2]We wish to thank Julien Lesca (LIP6-UPMC) for his participation to numerical tests.

Chapter 8

Conclusion

We have defined a minimalist & unified semantics for complex industrial systems and their architecture. Our aim is to give a unified formal semantics to the concepts manipulated on a daily basis by engineers from various fields working together on the design of such systems. Indeed, they need formal tools to reason on & model those systems in a unified & consistent way, with a clear understanding of the underlying concepts.

We have first introduced a definition of time unifying both continuous and discrete times. We have also defined data and dataflows, which are the fundamental objects for computation and communication between systems. Systems have then been defined as algorithmic transformations of dataflows, in a way that is equivalent to transfer functions. Our definition of systems captures two very important properties of complex industrial systems: *heterogeneity* (being able to deal with various types of systems naturally modeled with heterogeneous time & data) and *recursive integration* (taking into account the integrative dimension of such systems that are build recursively with multiple levels of subsystems). Three operators on systems have allowed us to define the integration of such systems: product, feedback & abstraction.

To handle the underspecification of systems during the design process and to model the internal structure of a multiscale system resulting from the recursive application of integration operators on elementary systems, we have introduced a logic on systems to express requirements, and a formalism to define a minimalist systems architecture framework. We have finally presented an original fair optimization model & algorithm that can be relevant to make architecture choices when designing systems.

Still, the subject of complex systems architecture is tremendously complex. We thus want to outline some limits of our approach & formalism:

1. A strong assumption in our work is that each system has its own rhythm, and that this rhythm cannot be disrupted by an interaction with another

system. It could lead to modeling artefacts when integrating systems with different time scales (since in our model we synchronize the data flows through a projection).

2. Also, we assume that systems live according to a static time scale that cannot be modified during the execution. It would be meaningful to extend our definition of systems to dynamic time scales progressively defined during the execution of the system (e.g. according to the events occurring during the execution, but still avoiding Zeno's effect).

3. We do not directly model nondeterministic behavior. However, the more complex the systems are, the more important nondeterminism is[1]. Also, the very mechanical description of system behavior is of limited practical use when we intend to consider systems at a rather high level of abstraction (and not a a level at which their concrete behavior is meaningful). Extending our formalism to event-based approaches could be an elegant solution to both issues, to define systems in practice in a more simple & actionable way[2].

4. Parts I & II have not be proven consistent, and some key results of equivalence are missing between our formalisms. The Chapter 6 should especially be more closely linked to the previous ones.

5. We consider that the internal structure of a system does not vary in time, what may not be the case for distributed systems and systems of systems. Being able to model such phenomenas would improve the expressivity of our formal framework.

6. It is possible to express different kinds of systems in our formalism. However, this "expressivity" does not mean that, in practice, it is really possible to use our formalism to model the full range of such systems in a natural, non-artificial way (there is a substantial difference between expressivity in theory and in practice, as the ability to model naturally a system in a given language or formalism is key in practice).

7. We did not compare our dataflow-centric formalism with the underlying semantics of synchronous languages like Lustre or Simulink, which use similar concepts.

This work is the theoretical part of a broader project aiming at building a science for systems design & architecture, extending and generalizing the models & methods existing, for instance, for software design. Within the last years, we

[1]In practice, a limit of our descriptive, mechanistic approach is also the sensibility to initial conditions (especially for physical systems that exhibit chaotic behavior).

[2]it also means that to turn our formalism into a usable modeling language, time should be rather considered as more abstract than the logical time we have defined.

have applied our framework to several real industrial cases from various industries (aeronautics, defence, banking, nuclear engineering, automotive), allowing to strengthen our model and assess its relevancy & genericity.

We believe that the following topics are promising directions & subjects to explore to go beyond the present work and contribute to build this science:

- We are willing to provide a formal framework to describe a design process using our semantics, providing a formalization of design approaches mixing top-down and bottom-up approaches to explore the recursive structure of integrated systems being designed. As we have seen in Chapter 6, taking into account the design process has significant impacts on the models we define, so that they can be meaningful & useful in real world situations. This is a significant challenge.

- An architecture framework is very rich, with for instance several layers that we have not taken into account in the present work. It would be of great interest to extend our minimalist formalism to all aspects of an architecture framework.

- In-depth case studies are needed to show how our approach & formalism helps on real industrial projects.

- The most complicated integration operator, i.e. *abstraction*, should be refined by different operators performing specialized kind of abstractions on systems, consistently with the reality of the specialized and meaningful abstractions encountered in Systems Engineering. It would make it more actionable in practice.

- Computability is a key topic for computer scientists. The properties of the internal state are decisive to study computability, and our model should also be compared from the point of view of computability on dataflows with other models of systems[3].

Thank you for reading this manuscript! Hoping that you have enjoyed this formal riddle in the stunning world of complex systems modeling & architecture!

And this is just the beginning...

[3]The strength of our model is that it handles both heterogeneity and integration of systems. Still, it would be interesting to compare our approach to the one consisting in defining integration operators & an architecture framework upon standard models of hybrid systems (possibly slightly modified if necessary).

Bibliography

[1] Jean-Raymond Abrial. *The B-book - assigning programs to meanings.* Cambridge University Press, 2005.

[2] P. Aczel and N. Mendler. A final coalgebra theorem. In D.-H. Pitt, D.-E. Ryeheard, P. Dybjer, A.-M. Pitts, and A. Poigne, editors, *Proceedings category in computer science*, Lecture Notes in Computer Science, pages 357–365. Springer-Verlag, 1989.

[3] M. Aiguier, F. Boulanger, and B. Kanso. A formal abstract framework for modeling and testing complex software systems. *Theoretical Computer Science*, 455:66–97, 2011.

[4] M. Aiguier, B. Golden, and D. Krob. An adequate logic for heterogeneous systems. In *18th IEEE International Conference on Engineering of Complex Computer Systems*, 2013.

[5] E. Alaksen and R. Belcher. *Systems Engineering.* Prentice Hall, 1992.

[6] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comp. Sci.*, 138:3–34, 1 1995.

[7] André Arnold, Gérald Point, Alain Griffault, and Antoine Rauzy. The altarica formalism for describing concurrent systems. *Fundam. Inf.*, 40(2-3):109–124, November 1999.

[8] J. Bacon and J. Van Der Linden. *Concurrent Systems: An Integrated Approach to Operating Systems, Distributed Systems and Database.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[9] N. Bansal and M. Sviridenko. The santa claus problem. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 31–40, 2006.

[10] G. Berry. *The foundations of Esterel.* MIT Press, 2000.

[11] B.-S. Blanchard and W.-J. Fabrycky. *Systems engineering and analysis.* Prentice Hall, 1998.

[12] S. Bliudze and D. Krob. Modelling of complex systems: Systems as dataflow machines. *Fundamenta Informaticae*, 91:1–24, 2009.

[13] IEEE Standards Board. *IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-1993).* IEEE, June 1994.

[14] O. Bournez and M.-L. Campagnolo. *New Computational Paradigms. Changing Conceptions of What is Computable*, chapter A Survey on Continuous Time Computations, pages 383–423. Springer-Verlag, 2008.

[15] S. Bouveret and J. Lang. Efficiency and envy-freeness in fair division of indivisible goods. In *Proceedings of IJCAI*, 2005.

[16] J. Bradfield and C. Stirling. Modal mu-calculi. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, pages 721–756. Elsevier, 2007.

[17] M. Broy. Refinement of time. *Theor. Comput. Sci.*, 253(1):3–26, February 2001.

[18] M. Broy and G. Stefănescu. The algebra of stream processing functions. *Theor. Comput. Sci.*, 258:99–129, May 2001.

[19] M. Broy and K. Stølen. *Specification and development of interactive systems: focus on streams, interfaces, and refinement.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[20] P. Caspi and M. Pouzet. Synchronous kahn networks. *SIGPLAN Not.*, 31(6):226–238, June 1996.

[21] D. Cha, J. Rosenberg, and C. Dym. *Fundamentals of Modeling and Analysing Engineering Systems.* Cambridge University Press, 2000.

[22] Y. Chevaleyre, U. Endriss, J. Lang, and N. Maudet. Negotiating over small bundles of resources. In *Proceedings of AAMAS*, pages 296–302, 2005.

[23] K. M. Chong. An induction theorem for rearrangements. *Canadian Journal of Mathematics*, 28:154–160, 1976.

[24] A. H. Clifford and G. B. Preston. *The Algebraic Theory of Semigroups.* Math. Surveys 7, Amer. Math. Soc., R.I., 1961.

[25] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM Symposium on Principles of Programming Languages*, 1977.

[26] J. B. Dennis. First version of a data flow procedure language. In *Programming Symposium, Proceedings Colloque sur la Programmation*, pages 362–376, London, UK, UK, 1974. Springer-Verlag.

[27] F. Diener and G. Reeb. *Analyse Non Standard*. Hermann, 1989.

[28] T. Gajdos. Single crossing lorenz curves and inequality comparisons. *Mathematical Social Sciences*, 47(3):21–36, 2004.

[29] I. P. Gent, R. W. Irving, D. F. Manlove, P. Prosser, and B. M. Smith. A constraint programming approach to the stable marriage problem. In *In CP01*, pages 225–239. Springer, 2001.

[30] B. Golden, M. Aiguier, and D. Krob. Modeling of complex systems ii: A minimalist and unified semantics for heterogeneous integrated systems. *Applied Mathematics and Computation*, 218(16):8039–8055, 2012.

[31] B. Golden and Y. Hourdel. A minimalist formal framework for systems architecting. In *3rd International Workshop on Model Based Safety Assessment*, 2013.

[32] B. Golden and P. Perny. Infinite order lorenz dominance for fair multiagent optimization. In *AAMAS*, pages 383–390, 2010.

[33] J. Goldsmith and R. Sloan. The conference paper assignment problem. In *Proc. AAAI Workshop on Preference Handling for Artificial Intelligence*, 2007.

[34] M. Guo and V. Conitzer. Undominated VCG redistribution mechanisms. In *Proceedings of AAMAS'08*, pages pp. 1039–1046, 2008.

[35] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language lustre. In *Proceedings of the IEEE*, volume 79, 1991.

[36] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 278–, Washington, DC, USA, 1996. IEEE Computer Society.

[37] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information processing*, pages 471–475, Stockholm, Sweden, Aug 1974. North Holland, Amsterdam.

[38] P. Kosiuczenko and M. Wirsing. Timed rewriting logic with an application to object-based specification. *Science of Computer Programming*, 28(2–3):225–246, 1997.

[39] D. Krob. Modelling of complex software systems: a reasoned overview. In *Proceedings of the 26th IFIP WG 6.1 international conference on Formal Techniques for Networked and Distributed Systems*, FORTE'06, pages 1–22, Berlin, Heidelberg, 2006. Springer-Verlag.

[40] H. Levy. Stochastic dominance and expected utility: Survey and analysis. *Management Science*, 38(4):555–593, 1992.

[41] J. Lygeros. Lecture notes on hybrid systems. In *Notes for an ENSIETA Workshop*, 2004.

[42] M.-W. Maier and E. Rechtin. *The art of system architecturing.* CRC Press, 2002.

[43] L. Mandel and M. Pouzet. ReactiveML, a Reactive Extension to ML. In *ACM International Conference on Principles and Practice of Declarative Programming (PPDP)*, Lisboa, July 2005.

[44] W. Marshall and I. Olkin. *Inequalities: Theory of Majorization and its Applications.* Academic Press, London, 1979.

[45] P. Marwedel. *Embedded System Design.* Kluwer, 2003.

[46] G. H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal 34*, 1955.

[47] H. Moulin. *Axioms of cooperative decision making.* Monograph of the Econometric Society. Cambridge University Press, 1988.

[48] E. Nelson. Internal set theory: a new approach to nonstandard analysis. *Bulletin of the American Mathematical Society*, 83:1165–1198, 1977.

[49] W. Ogryczak. Inequality measures and equitable approaches to location problems. *European Journal of Operational Research*, 122:374–391, 2000.

[50] W. Ogryczak and T. Sliwinski. On solving linear programs with the ordered weighted averaging objective. *European Journal of Operational Research*, 148(1):80–91, 2003.

[51] P. Perny and O. Spanjaard. An axiomatic approach to robustness in search problems with multiple scenarios. In *Proceedings of the 19th conference on Uncertainty in Artificial Intelligence*, pages 469–476, 2003.

[52] A. Rabinovitch. Automata over continuous time. *Theor. Comput. Sci.*, 300:331–363, 2003.

[53] F. Robert. Les systèmes dynamiques discrets. *Mathématiques et Applications*, 19, 1994.

[54] A. Robinson. *Non-standard analysis.* American Elsevier, 2nd. ed. edition, 1974.

[55] J.-M.-M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.

[56] J.-M.-M. Rutten. Algebraic specification and coalgebraic synthesis of mealy automata. In *International Workshop on Formal Aspects of Component Software (FACS 2005)*, volume 160 of *Electronic Notes in Computer Science*, pages 305–319. Elsevier, 2006.

[57] A.-P. Sage and J.-E. Amstrong. *Introduction to system engineering.* John Wiley, 2000.

[58] A. Sen. *On economic inequality.* Clarendon Press, expanded edition edition, 1997.

[59] A.F. Shorrocks. Ranking income distributions. *Economica*, 50:3–17, 1983.

[60] E. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, volume 6 of *Textbooks in Applied Mathematics.* Springer-Verlag, 1998.

[61] B.A. Trakhtenbrot. Understanding basic automata theory in the continuous time setting. *Fundam. Inform.*, 62, 2003.

[62] W.-C. Turner, J.-H. Mize, K.-E. Case, and J.-W. Nazemeth. *Introduction to industrial and systems engineering.* Prentice Hall, 1993.

[63] J. von Neumann and O. Morgenstern. *Theory of games and economic behavior.* 2nd Ed. Princeton University Press, 1947.

[64] M.E. Yaari. The dual theory of choice under risk. *Econometrica*, 55:95–115, 1987.

[65] R.R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. In *IEEE Trans. Systems, Man and Cybern.*, volume 18, pages 183–190, 1998.

[66] J. Zaytoun. *Systèmes dynamiques hybrides.* Hermes, 2001.

[67] B. P. Zeigler, H. Praehofer, and K. T. Gon. *Theory of Modeling and Simulation — Integrating Discrete Event and Continuous Complex Dynamic Systems.* Academic Press, 2000.

[68] Changyan Zhou and Ratnesh Kumar. Semantic translation of simulink diagrams to input/output extended finite automata. *Discrete Event Dynamic Systems*, 22(2):223–247, June 2012.