Project 22703: SYRF

Deliverable 4/5.1.3 and 4/5.2.3

Integrating synchrony & asynchrony : fundamentals and implementations

---

1 Progress report

1.1 The problems considered

1.1.1 The Essentials of the Synchronous Paradigm

The synchronous programming paradigm is essentially characterized as follows :

1. Programs progress via an infinite sequence of *reactions*: $P = R^\omega$, where R denotes the family of possible reactions[1].

2. Within a reaction, decisions can be taken on the basis of the *absence* of some events, as exemplified by the following typical statements, taken from Esterel, Lustre, and Signal respectively:

   ```
   present S else ‘stat’
   y = current x
   y := u default v
   ```

   The first statement is self-explanatory. The "`current`" operator delivers the most recent value of `x` at the clock of the considered node, it thus has to test for absence of `x` before producing `y`. The "`default`" operator delivers its first argument when it is present, and otherwise its second argument. While we are not going to give any formal presentation, we find it convenient refer to the status "absent" by using the distinguished value $\perp$ for a variable.

3. Parallel composition is given by taking the pairwise conjunction of associated reactions, whenever they are composable: $P_1 \| P_2 = (R_1 \wedge R_2)^\omega$. Typically, if specifying is the intention, then the above formula is a perfect definition of parallel composition. In contrast, if programming is the intention, then the need for this definition to be compatible with an operational semantics complicates very much the "when it is defined" prerequisite.

[1] In fact, "reaction" is a slightly restrictive term, as we shall see in the sequel that "reacting to the environment" is not the only possible kind of interaction a synchronous system may have with its environment.

Keeping in mind the essentials of the synchronous paradigm, we are now ready to discuss informally how asynchrony relates to synchrony. Referring to points 1, 2, and 3 above, the following can be stated about asynchrony:

1. Reactions cannot be observed any more: since no global clock exists, global synchronization barriers which indicate the transition from one reaction to the next one are no more observable. Instead, a reliable communication medium is assumed, in which messages are not lost, and for each individual channel, messages are sent and received in the same order. We call a *flow* the totally ordered sequence of values sent or received on a given communication channel.

2. Absence cannot be detected, and thus cannot be used to exercise control.

3. Composition occurs by means of unifying each individual flow shared between two processes. This models in particular the communications via asynchronous unbounded FIFOs, such as those in Kahn networks. Rendez-vous type of communication can also be abstracted in this way.

1.1.2 Desynchronizing synchronous programs

From our previous discussion, we can say that a run is a sequence of tuples of values (one for each variable) in domains extended with the extra symbol $\perp$. Each successive tuple represents one reaction. Desynchronizing a run amounts to discarding the synchronization barriers defining the successive reactions. Hence, for each individual variable, we only know the ordered sequence of *present* values. Thus desynchronizing a run amounts to mapping a *sequence of tuples* of values in domains extended with the extra symbol $\perp$, into a *tuple of sequences* of present values, one sequence per variable. This is formalized in [4, 3].

Denoting generically synchronous systems by Φ, and by Φ^a their corresponding desynchronized versions, and remembering our discussion in subsection 2.1.1, we consider

$$\begin{array}{rcl} \Phi_1 \parallel \Phi_2 & : & \text{the synchronous parallel composition} \\ \Phi_1^a \parallel_a \Phi_2^a & : & \text{the } a\text{synchronous parallel composition} \end{array}$$

At this point two natural questions arise, namely:

Question 1 (desynchronizing a single system) Is resynchronization feasible and uniquely defined? *More precisely, is it possible to reconstruct uniquely a synchronous run σ of our system from a desynchronized run σ^a?*

Question 2 (desynchronizing a communication) Does communication behave equivalently for both the synchronous and asynchronous compositions? *More precisely, does the following property hold?*

$$\Phi_1^a \parallel_a \Phi_2^a \quad = \quad (\Phi_1 \parallel \Phi_2)^a \tag{1}$$

If question 1 had a positive answer, then we could desynchronize a run of the considered system, and then still recover the original synchronous run. Thus a positive answer to question 1 would guarantee that the synchronous semantics is preserved when desynchronization is performed on a single system.

On the other hand, if question 2 had a positive answer, then we could interpret our system composition equivalently as synchronous or asynchronous.

Unfortunately, neither 1 nor 2 have positive answers in general, due to the possibility of exercising control by the way of absence in synchronous composition $\parallel$. In the following section, we show that questions 1 and 2 have positive answers under certain sufficient conditions, in which the two notions of *endochrony* (for point 1) and *isochrony* (for point 2) play a central role.

1.2 A theory of desynchronisation : endochrony, isochrony

1.2.1 Endochrony and resynchronization

For a synchronous system Φ with V as set of variables, and and $W' \subseteq W \subseteq V$, we say that W' is a *clock inference of* W, written $W' \hookrightarrow W$, if knowing the presence/absence and actual value carried by each variable belonging to W', allows us to determine exactly the presence/absence of each variable belonging to W. Thus we get a unique maximal increasing sequence of subsets of V, $\emptyset = V(0) \hookrightarrow V(1) \hookrightarrow V(2) \hookrightarrow \ldots \hookrightarrow V(k) \hookrightarrow \ldots$, and this sequence must become stationary from some index $k_{\max}$. In general, we only know that $V(k_{\max}) \subseteq V$. A Φ is said to be ***endochronous*** if $V(k_{\max}) = V$.

Theorem 1 *Conditions (a) and (b) given below are equivalent:*

(a) Φ *is endochronous.*

(b) For each run of Φ^a*, we can reconstruct the corresponding synchronous run in a unique way up to silent reactions.*

1.2.2 Isochrony, and synchronous/asynchronous compositions

Isochrony is slightly more subtle, as it addresses system composition. We provide only an informal presentation here, and we refer the reader to [4, 3] for details. For Φ a synchronous system, denote by ρ its associated transition relation.

The synchronous parallel composition of two systems $\Phi_1 \parallel \Phi_2$ is obtained by taking the conjunction of their transition relations $\rho_1 \wedge \rho_2$. This conjunction takes into account both presence/absence for common variables, and the values carried when present. Now we can weaken this operation by just taking into account the values carried by common variables, *provided* that they are present in the reactions of both components, this (informally) defines a “desynchronized” conjunction which we denote by $\rho_1 \wedge_a \rho_2$. A pair (Φ_1, Φ_2) of systems is said ***isochronous*** if identity

$$\rho_1 \wedge \rho_2 \quad \equiv \quad \rho_1 \wedge_a \rho_2 \qquad (2)$$

holds. Then

Theorem 2 *If pair* (Φ_1, Φ_2) *is isochronous, then (1) holds true.*

This theorem is not at all trivial, as condition (2) involves transition relations, while (1) involves infinite runs. In [4, 3] we also show that isochrony is in some sense minimal in order for (1) to hold.

1.2.3 Issues of compositionality

We show in [4, 3] that endochrony is not compositional. On the other hand, the following results are collected from [3], which show compositionality and locality of isochrony.

1. If pairs (Ψ, Φ_1) and (Ψ, Φ_2) are isochronous, then so is pair $(\Psi, \Phi_1 \parallel \Phi_2)$.
2. We are given a finite family $(\Phi_k)_{k=1,\ldots,K}$. Assume that each pair $(\Phi_k, \Phi_{k'})$ is isochronous. Then
   (a) For each disjoints subsets I and J of set $\{1,\ldots,K\}$, the pair
   $$\left(\parallel_{k\in I} \Phi_k \quad , \quad \parallel_{k'\in J} \Phi_{k'} \right) \tag{3}$$
   is isochronous. Thus *isochrony is compositional.*
   (b) Also, desynchronization extends to the network:
   $$(\Phi_1 \parallel \ldots \parallel \Phi_K)^a \quad = \quad \Phi_1^a \parallel_a \ldots \parallel_a \Phi_K^a \ . \tag{4}$$
3. Assume pair (Φ_1, Φ_2) is isochronous, and pair (Ψ_1, Ψ_2) is such that Ψ_1 has no common variable with $\Phi_2 \parallel \Psi_2$ and Ψ_2 has no common variable with $\Phi_1 \parallel \Psi_1$. Then pair $(\Psi_1 \parallel \Phi_1 \, , \, \Phi_2 \parallel \Psi_2)$ is also isochronous. Thus *isochrony is a local property.*

1.2.4 GALS architectures

Globally Synchronous Locally Asynchronous (GALS) architectures are commonly considered, in particular for hardware or HW/SW systems. Based on the results lised in subsection 2.2.3, our framework provides an elegant solution, namely:

GALS networks are networks of endochronous, pairwise isochronous, components.

1.3 Applicability & variations

So far the following assumptions have been made about the supporting communication layer for our theory to apply:

Assumption 1 *The communication layer behaves as a point-to-point network of channels. The channels do not loose messages, and deliver them in correct order.*

Thus we do not assume broadcast, and do not need any global notion of time or state. In turn we use high level communication primitives of the following kind: `await, get, put`. This means that we may resource to blocking communications (only for receivers but not for senders). This is acceptable in many cases, in particular for object oriented design of reactive systems, we come back to this point in the perspectives.

However, blocking is not always acceptable. In particular, distributed control and sensoring systems do not accept blocking: data are sent by the sensors at their own local pace to the bus, and the controller would read the bus at its own pace. This is the very basis, for instance, of the TTP (Time Triggered Protocol) proposed by Kopetz & al. for automotive industries, and the OSEK standard is also close to this philosophy. In draft note [5] *we show that our theory can nevertheless be deployed on such type of communication layer by resourcing to a certain degree of oversampling.* This idea has not been experienced, however.

1.4 Data structures : hierarchies

Endochrony and isochrony can be model checked using standard tools (and provided that suitable abstractions have been made). However data structures used by general purpose model checkers are not dedicated to endo/isochrony. This was first noticed (at least for endochrony) by Paul Le Guernic in the early 90's, and specific data structures, called *clock hierarchies,* were early used in the SIGNAL compiler [1, 2].

This year we have significantly extended these results [8]. We have provided a more systematic and elegant presentation of so-called Hierarchic Normal Forms (HNF). HNF are new decision diagrams associated to synchronous transition systems. They enjoy the following features :

- They are canonical and do not need any prior variable ordering for this.
- They reveal clock hierarchies as well as nested states such as exhibited in Statecharts.
- HNF provide endochrony checking for free, and isochrony checking for very cheap.
- Compositional algorithms exist for computing HNF.
- HNF can also be seen as a systematic calculus of don't care in hardware design. They also play a central role in optimizations for code generation.

2 Conclusions and new perspectives

2.1 What is implemented ? Industrial transfer

Distributed code generation for SIGNAL, the DC_+ common format, and the SILDEX tool by TNI is implemented using simplified versions of endochrony and isochrony. In particular only the current state is considered, not the previous state. Therefore nested states are not captured today, only clock hierarchies are captured — probably the most important, however.

Based on this, the following methodology for system design can be followed :

1. Specify the functional architecture. Resulting specification can be complete or alternatively partial (partial designs are currently not handled by SILDEX, however). Predefined components can be imported.
2. Deploy the functional architecture on an abstraction of the distributed execution architecture. To this end, additional schedulers and protocols are automatically generated to enforce endo/isochrony at wish. This makes the components to become robust against asynchrony of the communication medium.
3. Generate code for each properly enhanced component, by using calls to communication resources from the communication layer.
4. To analyse the whole system, model the communication medium using multiclock, nondeterministic synchronous systems. This emulates asynchrony at will, provided that only bounded memory is used by the communication channels — in particular unbounded fifos are not handled. Then the whole model {components + communication media} can be given to analysis tools, for property checking or timing analysis.

In talks given at Femsys'99 (Munich) and FM'99 (Toulouse), a Statecharts example was discussed.

2.2 Dynamicity and mobility

Thanks to the compositional and local nature of isochrony (subsection 2.2.3), a certain degree of dynamicity becomes easily implementable, based on the synchronous programming paradigm. This may be regarded as a small revolution, as it has always been said that dynamic instanciation and synchronous progrsamming are mutually exclusive concepts. Let us sketch how one can proceed (formalizing this is the subject of ongoing work).

1. Pick an endo/isochronous design, e.g., a GALS architecture.
2. Regard its components as *object classes* rather than programs.
3. Instanciating an object consists in renaming some prototypes by creating fresh names. Pick two classes Φ and Ψ with common set of variables W. Joint instanciation of these would consist in renamings $\Phi[V/V_{\text{new}}, \Psi[U/U_{\text{new}}]$ in such a way that $V \cap U = W$ is consistently mapped into $V_{\text{new}} \cap U_{\text{new}}$. As isochrony is compositional and local, it is preserved by such an instanciation.
4. Therefore an off-line preparation of the classes for endo/isochrony guarantees a correct on-line and asynchronous instanciation. It remains to resourcing to the available mechanisms for dynamic instanciation offered by existing object oriented languages for the execution (e.g., C_{++}, Java, Eiffel,...).

In the same vein, we are currently investigating mobility, by relating our approach to the Join Calculus of G. Gonthier [7].

2.3 UML

From the previous results and remarks, it is natural to investigate the use of our techniques to establish a behavioural semantic backbone for UML, targeted to reactive systems [2]. To this end we have proposed a new formalism called BDL (Behavioural Description Language) [6, 9]. As compared to existing synchronous languages, BDL possesses the following features:

- It is targeted to architectures and partial designs. It aims at providing behavioural mechanisms of inheritance (a recognized difficult subject). To this end it is equipped with two composition operators: the parallel composition $\|$ (to specialize behaviours by adding constraints), and the nondeterministic choice $\vee$ (to enrich a given behaviour by adding new behaviours).
- It has a dual synchronous/asynchronous semantic.
- Statecharts as well as Sequence Diagrams and Message Sequence Charts can be translated into BDL, thus offering a blending of these formalisms at a semantic level.
- A visual syntax is under study, as a mild variation of Statecharts.

[2] This ongoing work is the central subject of the ongoing REUTEL project, funded by Alcatel.

References

[1] T. P. Amagbegnon, L. Besnard, and P. Le Guernic. Arborescent canonical form of boolean expressions. Technical Report 2290, Inria Research Report, June 1994.

[2] T. P. Amagbegnon, L. Besnard, and P. Le Guernic. Implementation of the dataflow language SIGNAL. In *Proceedings of the ACM SIGPLAN Conference on Programming Languages Design and Implementation, PLDI'95*, pages 163–173, 1995.

[3] A. Benveniste, B. Caillaud, and P. Le Guernic. Compositionality in dataflow synchronous languages: specification & distributed code generation. *Information and Computation*, to appear.

[4] A. Benveniste, B. Caillaud, and P. Le Guernic. From synchrony to asynchrony. In J.C.M. Baeten and S. Mauw, editors, *CONCUR'99, Concurrency Theory, 10th International Conference*, volume 1664 of *Lecture Notes in Computer Science*, pages 162–177. Springer, August 1999.

[5] A. Benveniste and P. Caspi. Distributing synchronous programs on communication media of time triggered type. Draft paper, August 1999.

[6] C. Jard, J-M. Jézéquel, A. Le Guennec, and B. Caillaud. Protocol engineering using uml. Draft paper, submitted, October 1999.

[7] J-P. Talpin and A. Benveniste. Pre-order transition systems for calculi of mobile processes. Draft paper, submitted, October 1999.

[8] J-P. Talpin, A. Benveniste, B. Caillaud, and P. Le Guernic. Hierarchic normal forms for desynchronization. Draft paper, October 1999.

[9] J-P. Talpin, A. Benveniste, and P. Le Guernic. Asynchronous deployment of synchronous pre-order transition systems. Technical report, Inria Research Report, October 1999. in preparation.