Abstract Frequency Analysis of Synchronous Systems

Alexandre Chapoutot*[1],
Matthieu Martel*[1]

* *CEA Saclay, DRT/LIST/DTSI,*
91191 Gif-sur-Yvette Cedex, France

ABSTRACT

Embedded systems often are described with graphical data flow languages like Matlab/Simulink or Lustre/SCADE, which are used at the first stage of development cycle, that is at the specification level. Most of program verification activities currently are applied to the source code that is a low level description of systems. We propose to study a new approach based on frequency analysis to apply static analysis to data flow languages.

KEYWORDS: Abstract Interpretation, Synchronous Languages, Fourier Transforms.

1 Introduction

Embedded softwares continuously interact with their physical environment through sensors. A sensor is a hardware component which gathers values of a physical entity with respect to a fixed frequency and a given precision (i.e. values belong to a finite discrete set). The physical environment usually is defined by continuous time functions (i.e. *signals*) that the sensors sample and quantify in order to be processed by a computer. However, there is another description of signals which uses frequency functions. Every signal, under some hypotheses, is a sum of elementary periodic signals (sine and cosine) which describe fully the signal. Fourier transforms connect both definitions and this operation basically is the starting point of our work.

Abstract interpretation methods have been successfully applied to safety critical softwares [Blan03]. But, this method was applied to source code and is seldom applied to system specifications. Our work is to provide a new framework for using static analysis by abstract interpretation on program specifications written in data flow languages, like Matlab/Simulink or Lustre/SCADE [Casp87], using the Fourier transforms. Other transforms are of interest as well but this is out of the scope of this paper: Monniaux uses successfully Z-transforms to analyse (by abstract interpretation) numerical linear filters [Monn05].

2 Abstract interpretation: a very short view

A program is described with variables representing states and a program semantics is a mathematical model of all the execution paths (i.e. sequences of states) with all the initial

[1]E-mail: {Alexandre.Chapoutot, Matthieu.Martel}@cea.fr

states (the *concrete semantics* of Figure 1). A program property introduces invalid states and so forbidden zones (in red in Figure 1). In our case, we consider properties of divergence of control flow which can answer the question: "How does the discretization of input signals act upon the control process?". Verifying a program property is to prove that the intersection between forbidden zone and all the execution paths is empty. This is a mathematical proof which is *not computable* because of manipulating possibly infinite set of states, and Rice theorem says that verified a non trivial property is *undecidable*.

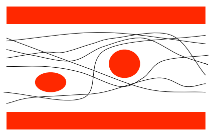

Figure 1: *Concrete semantics.*

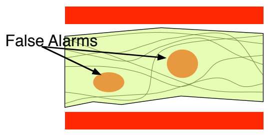

Figure 2: *Abstract semantics (1).*

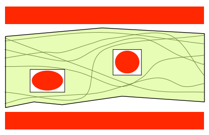

Figure 3: *Abstract semantics (2).*

Abstract interpretation [Cous92] avoids these two problems by introducing *abstract semantics* based on concrete one. Abstract semantics merge sets of concrete executions in one abstract execution. So, verifying a program property is to prove that the intersection between forbidden zones and abstract executions is empty (Figure 2). A coarse abstract semantics may lead to generate false alarms, that is the abstract execution intersects forbidden zones while concrete executions do not (Figure 2). In this case, a refinement must be found to avoid it, see Figure 3. The concrete and the abstract semantics are related thanks to *Galois connections* defined by an abstraction function α and an concretization function γ.

3 Frequency analysis

We focus our work, for the moment, on the Lustre language, a data flow synchronous language, which is simpler to analyze than Matlab/Simulink. On the one hand, the synchronous hypothesis associates to language statements zero time execution and avoids semantics problems due to asynchrony. On the other hand, the mathematical foundation of the Lustre language simplifies our approach of the problem of verifying safety properties on embedded systems specifications. Furthermore, Caspi and *al* [Casp03] showed that it is possible to translate a synchronous part of Simulink into Lustre and conversely. So, all methods we define and use in this paper may, in the future, be applied to a fragment of Matlab/Simulink and are a good starting point.

Lustre programs manipulate (possibly infinite) sequences, that is discrete time functions. So, it seems natural to associate Lustre programs with systems coming from signal theory. However, we want to consider Lustre programs like ideal mathematical models, that is system descriptions using continuous time functions. We must therefore reformulate Lustre's semantics to manipulate continuous time functions. For now, we restrict signal space to periodic signals, so we will only use Fourier series. This limitation avoids the problems of convergence due to the occurrence of integral operations in Fourier transforms formula (used with aperiodic functions) which will be studied in a future work.

We represent the sensor activity, with two periodic functions with different ranges, in Figure 4. The first line is our ideal mathematical model using real continuous time functions. The second line describes the sampling effect and how the time discretization affects signal

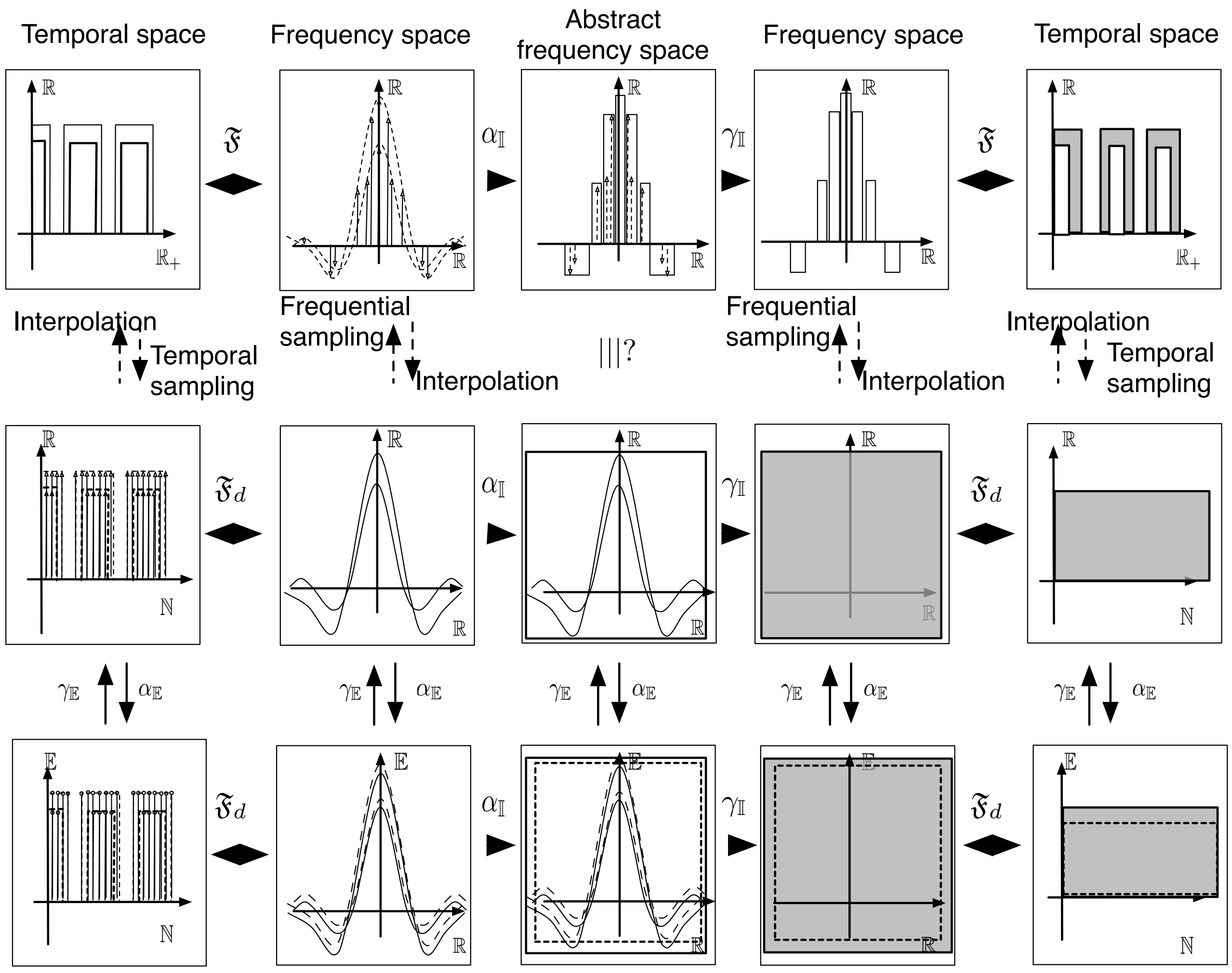

Figure 4: *Sensor activity and signal representations.*

spectra. Finally, the third line is the quantization operation which makes the range of the functions discrete.

With a left to right reading of the first line, we start with real continuous time functions which are transformed into real functions from discrete frequency space. This first transformation is guarantee to be sound thanks to Fourier series. We denote the Fourier series relations by $\mathfrak{F}$ and $\mathfrak{F}_d$ for the discrete one. From this point, we use abstract interpretation to represent sets of frequency functions (i.e. exponential form). We use the Galois connection which abstracts, with α_I, a set of functions into an interval, that is with a box holding all the frequency functions. The concretization function γ_I transforms a box into set of functions (gray boxes) which represents all the frequency functions inside the box. And finally, the inverse Fourier transforms can be applied to come back to the temporal space. These different operations, applied to the continuous time space, can be used in the same way for the other spaces which are discrete time spaces with real ranges (second line) and discrete time spaces with discrete ranges (third line).

Going from the first to the second line, which models the sampling operation, is a reversible operation if the hypotheses of Shannon-Nyquist theorem are verified. As we only consider periodic functions, it is easy to find the maximal frequency from a signal, so we assume that the hypotheses of the theorem always are true. The difficult point is the relation between the abstract continuous frequency space and the abstract discrete frequency space, which is unclear for the moment. We suppose that an equivalence relation exists but further investigation is required to prove it.

The discrete Fourier transforms generates periodic continuous frequency functions, so the abstract representation and the concretization of the second line are less precise than the continuous one because we abstract the whole functions instead of the function's components. Finally, the third line of Figure 4 uses the space $\mathbb{E}$ to represent the set of all possible quantified values. This representation is a decomposition of the real value in a computer representation together with a distance between real and computer representations. We can use floating point numbers with errors [Mart02], for example, for which a Galois connection with intervals could be used, denoted $\alpha_{\mathbb{E}}$ and $\gamma_{\mathbb{E}}$. After applying Fourier transforms, we obtain quantization frequency functions and "real" frequency functions. So, the interval abstraction generates two boxes which represent these two types and the concretization generates two sets of possible continuous time functions.

4 Conclusion

We have presented a new way to depict data in real-time system specifications. Our sensor activity representation allows to gather information on ideal signals corresponding to Lustre sequences. The main idea of this paper, the use of frequency analysis, has to be more studied to be really usable. We want to use these methods to validate system specifications and hence to use formal methods as soon as possible in the development cycle. For example, we concentrate our efforts to define good abstract domains in order to verify properties like control flow divergence between mathematical and computer descriptions of signals.

References

[Blan03] B. BLANCHET, P. COUSOT, R. COUSOT, J. FERET, L. MAUBORGNE, A. MINÉ, D. MONNIAUX, AND X. RIVAL. A Static Analyzer for Large Safety-Critical Software. In *PLDI'03*. ACM, 2003.

[Casp03] P. CASPI, A. CURIC, A. MAIGNAN, C. SOFRONIS, AND S. TRIPAKIS. Translating Discrete-Time Simulink to Lustre. In *EMSOFT'03*, LNCS 2855. Springer, 2003.

[Casp87] P. CASPI, D. PILAUD, N. HALBWACHS, AND J. PLAICE. LUSTRE: a declarative language for real-time programming. In *POPL'87*, pages 178–188. ACM, 1987.

[Cous92] P. COUSOT AND R. COUSOT. Abstract Interpretation Frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[Mart02] M. MARTEL. Propagation of Roundoff Errors in Finite Precision Compuations: a Semantics Approach. In *ESOP'02*, LNCS 2305. Springer, 2002.

[Monn05] D. MONNIAUX. Compositional analysis of floating-point linear numerical filters. In *Computer-aided verification (CAV'05)*, LNCS. Springer, 2005.