Contextual Equivalence for Signal Flow Graphs

Filippo Bonchi[1], Robin Piedeleu[2⋆], Paweł Sobociński[3⋆⋆], and Fabio Zanasi[2⋆](✉)

[1] Università di Pisa, Italy
[2] University College London, UK, {r.piedeleu, f.zanasi}@ucl.ac.uk
[3] Tallinn University of Technology, Estonia

Abstract. We extend the signal flow calculus—a compositional account of the classical signal flow graph model of computation—to encompass affine behaviour, and furnish it with a novel operational semantics. The increased expressive power allows us to define a canonical notion of contextual equivalence, which we show to coincide with denotational equality. Finally, we characterise the realisable fragment of the calculus: those terms that express the computations of (affine) signal flow graphs.

Keywords: signal flow graphs · affine relations · full abstraction · contextual equivalence · string diagrams

1 Introduction

Compositional accounts of models of computation often lead one to consider *relational* models because a decomposition of an input-output system might consist of internal parts where flow and causality are not always easy to assign. These insights led Willems [33] to introduce a new current of control theory, called *behavioural* control: roughly speaking, behaviours and observations are of prime concern, notions such as state, inputs or outputs are secondary. Independently, programming language theory converged on similar ideas, with *contextual equivalence* [25,28] often considered as *the* equivalence: programs are judged to be different if we can find some context in which one behaves differently from the other, and what is observed about "behaviour" is often something quite canonical and simple, such as termination. Hoare [17] and Milner [23] discovered that these programming language theory innovations also bore fruit in the nondeterministic context of concurrency. Here again, research converged on studying simple and canonical contextual equivalences [24,18].

This paper brings together all of the above threads. The model of computation of interest for us is that of signal flow graphs [32,21], which are feedback systems well known in control theory [21] and widely used in the modelling of linear dynamical systems (in continuous time) and signal processing circuits (in

⋆ Supported by EPSRC grant EP/R020604/1.
⋆⋆ Supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001)

discrete time). The *signal flow calculus* [10,9] is a syntactic presentation with an underlying compositional denotational semantics in terms of linear relations. Armed with *string diagrams* [31] as a syntax, the tools and concepts of programming language theory and concurrency theory can be put to work and the calculus can be equipped with a structural operational semantics. However, while in previous work [9] a connection was made between operational equivalence (essentially trace equivalence) and denotational equality, the signal flow calculus was not quite expressive enough for contextual equivalence to be a useful notion.

The crucial step turns out to be moving from *linear* relations to *affine* relations, i.e. linear subspaces translated by a vector. In recent work [6], we showed that they can be used to study important physical phenomena, such as current and voltage sources in electrical engineering, as well as fundamental synchronisation primitives in concurrency, such as mutual exclusion. Here we show that, in addition to yielding compelling mathematical domains, affinity proves to be the magic ingredient that ties the different components of the story of signal flow graphs together: it provides us with a canonical and simple notion of observation to use for the *definition* of contextual equivalence, and gives us the expressive power to prove a bona fide full abstraction result that relates contextual equivalence with denotational equality.

To obtain the above result, we extend the signal flow calculus to handle affine behaviour. While the denotational semantics and axiomatic theory appeared in [6], the operational account appears here for the first time and requires some technical innovations: instead of traces, we consider *trajectories*, which are infinite traces that may start in the past. To record the time, states of our transition system have a runtime environment that keeps track of the global clock.

Because the affine signal flow calculus is oblivious to flow directionality, some terms exhibit pathological operational behaviour. We illustrate these phenomena with several examples. Nevertheless, for the linear sub-calculus, it is known [9] that every term is denotationally equal to an executable realisation: one that is in a form where a consistent flow can be identified, like the classical notion of signal flow graph. We show that the question has a more subtle answer in the affine extension: not all terms are realisable as (affine) signal flow graphs. However, we are able to characterise the class of diagrams for which this is true.

Related work. Several authors studied signal flow graphs by exploiting concepts and techniques of programming language semantics, see e.g. [4,22,29,2]. The most relevant for this paper is [2], which, independently from [10], proposed the same syntax and axiomatisation for the ordinary signal flow calculus and shares with our contribution the same methodology: the use of *string diagrams* as a mathematical playground for the compositional study of different sorts of systems. The idea is common to diverse, cross-disciplinary research programmes, including Categorical Quantum Mechanics [1,11,12], Categorical Network Theory [3], Monoidal Computer [26,27] and the analysis of (a)synchronous circuits [14,15].

Outline In Section 2 we recall the affine signal flow calculus. Section 3 introduces the operational semantics for the calculus. Section 4 defines contextual equivalence and proves full abstraction. Section 5 introduces a well-behaved class of

circuits, that denotes functional input-output systems, laying the groundwork for Section 6, in which the concept of realisability is introduced before a characterisation of which circuit diagrams are realisable. Missing proofs can be found in the extended version of this paper [7].

2 Background: the Affine Signal Flow Calculus

The *Affine Signal Flow Calculus* extends the signal flow calculus [9] with an extra generator $\vdash$ that allows to express affine relations. In this section, we first recall its syntax and denotational semantics from [6] and then we highlight two key properties for proving full abstraction that are enabled by the affine extension. The operational semantics is delayed to the next section.

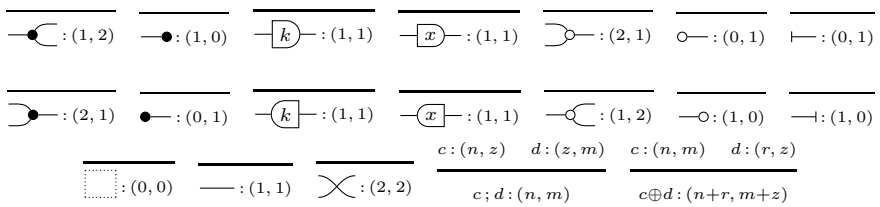

Fig. 1. Sort inference rules.

2.1 Syntax

$$c ::= \quad | \quad | \quad k \quad | \quad x \quad | \quad | \quad | \quad | \tag{1}$$

$$\quad | \quad | \quad k \quad | \quad x \quad | \quad | \quad | \quad | \tag{2}$$

$$| \quad | \quad | \quad c \oplus c \quad | \quad c\,;c \tag{3}$$

The syntax of the calculus, generated by the grammar above, is parametrised over a given field k, with k ranging over k. We refer to the constants in rows (1)-(2) as *generators*. Terms are constructed from generators, ⬚, —, ╳, and the two binary operations in (3). We will only consider those terms that are *sortable*, i.e. they can be associated with a pair $(n,\, m)$, with $n, m \in \mathbb{N}$. Sortable terms are called *circuits*: intuitively, a circuit with sort $(n,\, m)$ has n ports on the left and m on the right. The sorting discipline is given in Fig. 1. We delay discussion of computational intuitions to Section 3 but, for the time being, we observe that the generators of row (2) are those of row (1) "reflected about the y-axis".

2.2 String Diagrams

It is convenient to consider circuits as the arrows of a symmetric monoidal category ACirc (for Affine Circuits). Objects of ACirc are natural numbers (thus