Embeddings and Contexts for Link Graphs

Robin Milner

The Computer Laboratory, University of Cambridge,
Cambridge, UK
Robin.Milner@cl.cam.ac.uk

1 Introduction

Graph-rewriting has been a growing discipline for over three decades. It grew out of the study of graph grammars, in which – analogously to string and tree grammars – a principal interest was to describe the families of graphs that could be generated from a given set of productions. A fundamental contribution was, of course, the *double-pushout* construction of Ehrig and his colleagues [4]; it made precise how the left-hand side of a production, or rewriting rule, could be found to *occur* in a host graph, and how it should then be replaced by the right-hand side. This break-through led to many theoretical developments and many applications. It relies firmly upon the treatment of graphs as *objects* in a category whose arrows are embedding maps.

A simultaneous development was Petri nets [13], with a quite different motivation; it was the first substantial mathematical model of concurrent processes, and gives strong emphasis to the causality relation among events. Although Petri nets are graphical, their study has been largely independent of graph-rewriting; after all, a Petri net does not change its shape – only the tokens placed upon the net actually move.

A little later came the development algebraic calculi such as CSP [1] and CCS [12] to represent interactive concurrent processes. The key concept distinguishing them from (Petri) net theory was the emphasis upon modularity. Initially at least, net theory focussed upon complete systems, developing powerful techniques such as linear algebra to analyse them. In contrast, process calculi focussed upon assembling larger systems from smaller ones using a variety of combinators, and upon defining the behaviour of the whole in terms of abstract entities that can be constructed from the behaviours of the parts by algebraic operations corresponding to the combinators. This approach was inspired by the modularity present in all good programming languages, and by the categorical formulation of algebraic theories by Lawvere [9]; in contrast with graph-rewriting methodology, here the graphs are the *arrows* in a category whose objects are interfaces.

A recent development in process calculi by Leifer and Milner [10] is the demonstration that labelled transition systems can be derived uniformly for a wide variety of calculi, using the notion of *relative pushout* (RPO), in a category where the arrows are processes. In the particular case of *graphical* process calculi such as mobile ambients [2] or bigraphs [8], where graph-rewriting is used to

H.-J. Kreowski et al. (Eds.): Formal Methods (Ehrig Festschrift), LNCS 3393, pp. 343–351, 2005.

model various kinds of mobility among processes, this naturally leads to the need for a rapprochement between *graphs-as-objects* and *graphs-as-arrows*.

Connections between these two approaches have recently been surveyed by Ehrig [5]. In one of these connections, previously explored for example by Gadducci et al [7], graphs-as-arrows are obtained as cospans $I \to G \leftarrow J$ of graphs-as-objects, where the interfaces I and J are graphs of some simple form (e.g. discrete). A second connection goes the other way; graphs-are-objects arise in a coslice category of a category, or s-category, of graphs-as-arrows. This connection was first proposed by Cattani, and was exploited technically in the theory of action calculi [3].

The purpose of the present paper is to examine this latter connection more closely, in the context of *link graphs* [11], which are a constituent of bigraphs. It is shown that, for link graphs, the coslice category is isomorphic to the natural category of embeddings (as arrows) between so-called *ground* link graphs. The connection almost certainly extends to full bigraphs. More generally, it is an open challenge to characterise the classes of graphs (or other entities) and interfaces for which this elegant isomorphism exists. I suggest that the existence of the isomorphism is a valuable test of probity for any proposed class of graphs.

Preliminaries. Id_X will denote the identity function on a set X, and $\emptyset_X$ the empty function from $\emptyset$ to X. We shall use $X \uplus Y$ for union of sets X and Y known or assumed to be disjoint, and $f \uplus g$ for union of functions whose domains are known or assumed to be disjoint. This use of $\uplus$ on sets should not be confused with the disjoint sum '+', which disjoins sets *before* taking their union. We assume a fixed representation of disjoint sums; for example, $X + Y$ means $(\{0\} \times X) \cup (\{1\} \times Y)$, and $\sum_{v \in V} P_v$ means $\bigcup_{v \in V} (\{v\} \times P_v)$.

If $f\colon X \to Y$ is an arrow in a category or s-category $\mathbf{C}$, we denote its *domain* X and *codomain* Y by $\mathsf{dom}(f)$ and $\mathsf{cod}(f)$. The set of arrows from X to Y, called a *homset*, will be denoted by $\mathbf{C}(X, Y)$.

An *s-category* is like category except that every arrow f has an associated a finite set $|f|$, its *support*; the composition $gf\colon X \to Z$ of $f\colon X \to Y$ and $g\colon Y \to Z$ exists iff $|f| \cap |g| = \emptyset$, and then $|gf| = |f| \uplus |g|$. Furthermore, for $f\colon X \to Y$ and an injection ρ whose domain includes $|f|$, there is an arrow $\rho \cdot f\colon X \to Y$ called a *support translation* of f, with support $\rho(|f|)$. Support translation preserves all structure. A general treatment of s-categories can be found in Leifer and Milner [10], but we shall only be concerned with the special case of link graphs where the details are obvious.

2 Link Graphs

A link graph is essentially an ordinary graph, but it carries a little more information and each edge may link any number of nodes. A family of link graphs is determined by the kinds of nodes it has, and these are specified as follows:

Definition 1. (signature) A *signature* $\mathcal{K}$ provides a set whose elements are called *controls*. For each control K the signature also provides a finite ordinal $ar(K)$, its *arity*. We write $K\colon n$ for a control K with arity n. ∎