Initial Algebra Semantics for Cyclic Sharing Structures

Makoto Hamana

Department of Computer Science, Gunma University, Japan
hamana@cs.gunma-u.ac.jp

Abstract. Terms are a concise representation of tree structures. Since they can be naturally defined by an inductive type, they offer data structures in functional programming and mechanised reasoning with useful principles such as structural induction and structural recursion. In the case of graphs or "tree-like" structures – trees involving cycles and sharing – however, it is not clear what kind of inductive structures exists and how we can faithfully assign a term representation of them. In this paper we propose a simple term syntax for cyclic sharing structures that admits structural induction and recursion principles. We show that the obtained syntax is directly usable in the functional language Haskell, as well as ordinary data structures such as lists and trees. To achieve this goal, we use categorical approach to initial algebra semantics in a presheaf category. That approach follows the line of Fiore, Plotkin and Turi's models of abstract syntax with variable binding.

1 Introduction

Terms are a convenient, concise and mathematically clean representation of tree structures used in logic and theoretical computer science. In the field of traditional algorithms or graph theory, one usually uses unstructured representations for trees, such as a pair (V, E) of vertices and edges sets, adjacency lists, adjacency matrices, pointer structures, etc, which are more complex and unreadable than terms. We know that term representation provides a well-structured, compact and more readable notation.

However, consider the case of "tree-like" structures such as that depicted in Fig. 1. This kind of structures – graphs, but almost trees involving (a few) exceptional edges – quite often appears in logic and computer science. Examples include internal representations of expressions in implementations of functional languages that share common sub-expressions for efficiency, control flow graphs of imperative programs used in static analysis and compiler optimizations [CFR+91], data models of XML such as trees with pointers [CGZ05], proof trees admitting cycles for cyclic proofs [Bro05], and term graphs in graph rewriting [BvEG+87, AK96].

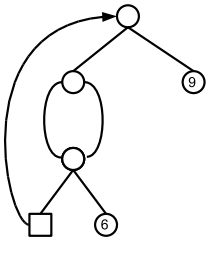

Fig. 1.

Suppose we want to treat such structures in a pure functional programming language such as Haskell, Clean, or a proof assistant such as Coq, Agda [Nor07]. In such a case, we would have to abandon the use of naive term representation, and would instead be compelled to use an

P.-L. Curien (Ed.): TLCA 2009, LNCS 5608, pp. 127–141, 2009.

unstructured representation such as (V, E), adjacency lists, etc. Furthermore, a serious problem is that we would have to abandon structural recursion/induction to decompose them because they look "tree-like" but are in fact graphs, so there is no obvious inductive structure in them. This means that in functional programming, we cannot use pattern matching to treat tree-like structures, which greatly decreases their convenience. This lack of structural induction means failure of being an inductive type. But, are there really no inductive structures in tree-like structures? As might be readily apparent, tree-like structures are almost trees and merely contain finite pieces of information. The only difference is the presence of "cycles" and "sharing".

In this paper, we give an initial algebra characterisation of cyclic sharing structures in the framework of categorical universal algebra. The aim of this paper is to derive the following practical goals *from the initial algebra characterisation*.

[I] To develop a simple term syntax for cyclic sharing structures that admits structural induction and structural recursion principles.
[II] To make the obtained syntax directly usable in the current functional languages and proof assistants, as well as ordinary data structures, such as lists and trees.

The goal [I] also intends that the term syntax *exactly* characterises cyclic sharing structures (i.e. no junk terms exist) to make structural induction possible. The goal [II] intends that the obtained syntax should be *lightweight as possible*, which means that e.g. well-formedness and equality tests on terms for cyclic sharing structures should be fast and easy, as are ordinary data structures such as lists and trees. We do not want many axioms to characterise the intended structures, because in programming situation, to check the validity of axioms every time is expensive and makes everything complicated. Therefore, ideally, formulating structures *without axioms* is best. The goal [II] is rephrased more specifically as:

[II'] To give an inductive type that represents cyclic sharing structures uniquely. We therefore rely on that a type checker automatically ensures the well-formedness of cyclic sharing structures.

We choose a functional programming language Haskell to concretely show it in this paper. To archive these goals, we use the category theoretic formulation of initial algebra semantics.

Recently, varying the base category other than **Set**, initial algebra semantics for functor-algebras has proved to be useful framework to characterise various mathematical/computational structures in a uniform setting. We list several: S-sorted abstract syntax is characterised as initial algebra in $\mathbf{Set}^S$ [Rob02], second-order abstract syntax as initial algebra in $\mathbf{Set}^{\mathbb{F}}$ [FPT99, Ham04, Fio08] (where $\mathbb{F}$ is the category of finite sets), explicit substitutions as initial algebras in the category $[\mathbf{Set}, \mathbf{Set}]_f$ of finitary functors [GUH06], recursive path ordering for term rewriting systems as algebras in the category **LO** of linear orders [Has02], nested datatypes [GJ07] and generalised algebraic datatypes (GADTs) [JG08] in functional programming as initial algebras in $[C, C]$ and $[|C|, C]$ respectively, where C is a ω-cocomplete category.

This paper adds a further example to the above list. We characterise cyclic sharing structures as an initial algebra in the category $(\mathbf{Set}^{\mathbb{T}^*})^{\mathbb{T}}$, where $\mathbb{T}$ is the set of all "shapes"