Inductive Triple Graphs: A Purely Functional Approach to Represent RDF

Jose Emilio Labra Gayo\textsuperscript{1}, Johan Jeuring\textsuperscript{2}, and Jose María Álvarez Rodríguez\textsuperscript{3}

\textsuperscript{1} University of Oviedo, Spain
labra@uniovi.es
\textsuperscript{2} Utrecht University, Open University of the Netherlands, The Netherlands
j.t.jeuring@uu.nl
\textsuperscript{3} South East European Research Center, Greece
jmalvarez@seerc.org

\textbf{Abstract.} RDF is one of the cornerstones of the Semantic Web. It can be considered as a knowledge representation common language based on a graph model. In the functional programming community, inductive graphs have been proposed as a purely functional representation of graphs, which makes reasoning and concurrent programming simpler. In this paper, we propose a simplified representation of inductive graphs, called Inductive Triple Graphs, which can be used to represent RDF in a purely functional way. We show how to encode blank nodes using existential variables, and we describe two implementations of our approach in Haskell and Scala.

1 Introduction

RDF appears at the basis of the semantic web technologies stack as the common language for knowledge representation and exchange. It is based on a simple graph model where nodes are predominantly resources, identified by URIs, and edges are properties identified by URIs. Although this apparently simple model has some intricacies, such as the use of blank nodes, RDF has been employed in numerous domains and has been part of the successful linked open data movement.

The main strengths of RDF are the use of global URIs to represent nodes and properties and the composable nature of RDF graphs, which makes it possible to automatically integrate RDF datasets generated by different agents.

Most of the current implementations of RDF libraries are based on an imperative model, where a graph is represented as an adjacency list with pointers, or an incidence matrix. An algorithm traversing a graph usually maintains a state in which visited nodes are collected.

Purely functional programming offers several advantages over imperative programming \cite{13}. It is easier to reuse and compose functional programs, to test properties of a program or prove that a program is correct, to transform a program, or to construct a program that can be executed on multi-core architectures.
In this paper, we present a purely functional representation of RDF Graphs. We introduce popular combinators such as fold and map for RDF graphs. Our approach is based on Martin Erwig’s inductive functional graphs [10], which we have adapted to the intricacies of the RDF model. The main contributions of this paper are:

- a simplified representation of inductive graphs
- a purely functional representation of RDF graphs
- a description of Haskell and Scala implementations of an RDF library

This paper is structured as follows: Section 2 describes purely functional approaches to graphs. In particular, we present inductive graphs as introduced by Martin Erwig, and we propose a new approach called triple graphs, which is better suited to implement RDF graphs. Section 3 presents the RDF model. Section 4 describes how we can represent the RDF model in a functional programming setting. Section 5 describes two implementations of our approach: one in Haskell and another in Scala. Section 6 describes related work and Section 7 concludes and describes future work.

2 Inductive Graphs

2.1 General Inductive Graphs

In this section we review common graph concepts and the inductive definition of graphs proposed by Martin Erwig [10].

A directed graph is a pair $G = (V, E)$ where $V$ is a set of vertices and $E \subseteq V \times V$ is a set of edges. A labeled directed graph is a directed graph in which vertices and edges are labeled. A vertex is a pair $(v, l)$, where $v$ is a node index and $l$ is a label; an edge is a triple $(v_1, v_2, l)$ where $v_1$ and $v_2$ are the source and target vertices and $l$ is the label.

Example 21. Figure 1 depicts the labeled directed graph with $V = \{(1, a), (2, b), (3, c)\}$, and $E = \{(1, 2, p), (2, 1, q), (2, 3, r), (3, 1, s)\}$.

![Fig. 1. Simple labeled directed graph](image_url)