Towards Formally Verified Rule Language Compilers

Antonio Hentschke

June 5, 2024

1 Introduction

TODO:
- insert an explanation what reasoning engines are and what they are used for

Reasoning engines (e.g. Nemo, Vlog) are used to process queries. Queries are formulated in a query language (e.g. SQL) and need to be compiled to a rule language (Datalog, in this case). The query language is based on a formalism, like relational algebra or first-order logic. Both the compilation to the rule language to the query language and the processing by the reasoning engine need to be verified. We want a mechanical way for this verification, which would otherwise be very tedious and error prone to conduct manually. Since verification is already used in the field of compilers, the central question to be answered is:

How can we adapt proof techniques from the field of formally verified compilers to the verification of Datalog reasoning engines?

We also want to answer the following questions:

- What is the notion of a program (to be verified) in a Datalog rule engine like Nemo?
  - so how does the input language / query language look like?
- What is a feasable specification for Datalog reasoners, formalized in a proof language (Coq or Lean)?
- How shall the semantics of the rule language be formalized?
  - Datalog
- What are (high-level to low-level) transformations to be addressed by the formalization?
  - as there exists a similarity to transformations of compilers (front end - middle end - back end)

- Can the analysed proof techniques help in the re-interpretation task (from low-level to high-level reasoning)?

1.1 Organisation

We will This article is an analysis that shall give an overview of the formalisms that already exist describing relational algebra. We will conclude what kind of formalisms there are and what are examples, as well as their respective advantages over the others. What questions came up when researching it? We also want to distinguish between formalisms in Coq and Lean. Based on those, we shall then discuss what theorem prover is more suitable for further implementation. At the end, we will state the next steps that are necessary to realise the formal verification of for example Nemo - a Datalog reasoner.

2 Background

2.1 What are the proof techniques in verifying compilers?

2.2 What formalisations of Datalog semantics are there already?

2.3 Are there already formalisations of rule engines?

2.4 Formalisations of query languages and their formalisations

2.4.1 Relational Algebra

3 How it all works together

4 How to verify Datalog reasoners

5 Can it be used for the re-interpretation task?