**From:** **Jan Posiadała** jan.posiadala@gmail.com
**Subject:** [Moderator Action] Position Statement: Executable semantics as a tool and artifact of the query language standardization process (resent due to Postmaster rejecting autoreplay)
**Date:** 11 January 2019 at 17:59
**To:** group-data-ws-pc@w3.org

Dear Members of The Graph Data Workshop Program Committee,

I am an active member of the openCypher community. I am most interested in the semantics of graph query and update languages: what it should be and how it should be delivered to the users. I strongly believe in formal semantics that are both human and machine readable. I am the architect and main developer of Cypher.PL, an executable semantics of Cypher written in Prolog. I was leading a research project on parsing of Polish language in graph database environment with crucial application of Cypher query language. In early 2017 I initiated a joined effort with the University of Warsaw, aiming at contributing to the development of graph data models.

I would like to lead discussion on executable semantics as a tool and artifact of the language standardization process. An executable semantics combines the advantages of a traditional formal semantics (explicitness, precision, comprehensiveness) with the advantages of a reference implementation (executability).

During the standardization process, a dynamic executable semantics provides, with very little delay, means to effortlessly test and compare language design choices on non-trivial examples. This could be particularly useful in the remote collaboration scenario, where explaining ideas informally may be time-consuming and inefficient. It offers natural ways of comparing, branching, and versioning. An executable semantics is a semantics you can test - by comparing the actual behaviour with the intended one. In my experience, developing an executable semantics promotes deeper understanding of the matter, faster identification of semantic gaps, and cleaner description of the issues. This can be illustrated by several examples related to aggregation and grouping, ambiguous semantics for reading/writing clauses, and the confusion around optional matching.

As a standardization artifact, an executable semantics is of tremendous use for vendors and language users, offering not only a reference implementation, but also a formal semantics that can be easily examined and modified. It allows training future language users before a production implementation can be developed.

Importantly, the effort needed to provide an executable semantics is comparable to that of providing any formal semantics, if the formalism is chosen wisely. For declarative languages, like Cypher, G-Core, or the intended GQL standard, such semantics can be naturally expressed in the logic programming paradigm, for instance in Prolog, as witnessed by Cypher.PL.

Additional resources:

R. A. Kowalski. The relation between logic programming and logic specification.
http://www.doc.ic.ac.uk/~rak/papers/logic%20programming%20and%20specification.pdf

Presentation of Cypher.PL at Fourth openCypher Implementers Meeting.
https://s3.amazonaws.com/artifacts.opencypher.org/website/ocim4/oCIM4_Insights+into+Cypher.PL.pdf

Research paper on the semantics of implicit group-by in Cypher (submitted to ICDT 2019).
https://drive.google.com/open?id=1NA1rJknAs0FJai-3g4dzg5Ah4GEABDiG

I am also interested in discussing:

- specific semantics choices (mixed expressions in implicit group-by, updates, optional matching),
- graph schema languages.

--
Best Regards,

Jan Posiadała