An Executable Semantics as a Tool and Artifact of Language Standardization

Filip Murlak, University of Warsaw; Jan Posiadała & Paweł Susicki, Nodes and Edges; Poland

In 2017, members of ISO SC32/WG3 raised an idea to a create a standard of a property graph query language. Since then, the GQL Manifesto was published, and the standardization is gaining momentum. According to the GQL Manifesto, the new standard should draw from 5 different query languages for property graphs: openCypher by Neo4j, PGQL by Oracle, GSQL by TigerGraph, property graph extensions for SQL envisioned by ISO/IEC, and G-Core by LDBC. Combining these experiences in a uniform way, trying out choices, and comparing solutions would be easier if assisted by an executable (and readable) semantics developed in parallel with the standardization process. A completed executable semantics could be a valuable artefact of the process.

WHY FORMAL? As proven by many examples, including Cypher, one can go quite far with just an informal semantics: a team of smart people can design and implement a query language without ever writing a single formula; a newcomer would likely refer to an informal introduction to the language; a programmer solving a task will often rely on modifying existing working examples. But at a certain stage one cannot do without a formal semantics: one can hardly imagine developing critical code in an unformalized language, or providing a reliable independent implementation for such a language. Then, an accurate (and comprehensible) specification of the semantics is indispensable.

WHY EXECUTABLE? An executable semantics is a semantics that can be tested. While in principle the formal semantics is the golden standard, in practice it is developed to reflect an existing informal semantics. This process is inevitably error-prone. Such errors can be detected by comparing the actual behaviour of the semantics with the intended one on handcrafted examples, but it is a tedious task. An executable semantics allows testing non-trivial examples with considerably less effort. The other way around, if the executable semantics is accompanied by a comprehensive benchmark, one can actually test if a given implementation complies with the semantics. This creates a feedback loop, which can lead to improving both the semantics and the implementation.

WHY READABLE? To be really useful, an executable semantics should be readable and easy to maintain. Then it becomes invaluable to vendors seeking to develop implementations, and to language designers wishing to test out new features. It is particularly useful if developed in parallel with the language, allowing the designers to immediately see and share the results of their decisions. It supports easy prototyping, versioning, branching, and integrating. An additional bonus is that it can act as a light-weight reference implementation available together with the standard of the language, speeding up the adoption of the standard by supporting the development of industry grade implementations, and training end-users.

PROOF OF CONCEPT. Cypher.PL is a formal, readable, and executable semantics of Cypher. Currently, it supports the language to the extent sufficient to pass all the test scenarios from the Technology Compatibility Kit, which makes its scope much wider than that of the partial denotational semantics made available recently. At the moment, Cypher.PL is the most accurate semantics of Cypher. Cypher.PL is written in Prolog. Being an extremely declarative, logic-based language, Prolog has been often advocated as a convenient formalism for executable semantics. It natively supports parsing with Definite Clause Grammars (DCGs). Its term-based data structures are ideal for explicitly representing and accessing syntactic objects, like queries, but they also provide a direct yet abstract representation of the input graph and intermediate evaluation artifacts, like the driving table. Term unification, which is one of Prolog's fundamental build-in concepts, offers a powerful extension of pattern matching, which lies at the core of Cypher. Prolog's native support for collecting and processing multiple unification solutions not only naturally captures multiple matches, but can also be exploited to reveal ambiguities in query parsing and evaluation. Meta-programming allows to manipulate easily the way predicates are evaluated. Finally, at least for some intended users, Prolog may be more accessible than the mathematical formalism of a denotational semantics. Cypher.PL makes full use of these advantages. Last but not least, the semantics of Prolog has been fully formalized and Prolog Core is an ISO standard, which is an additional advantage in the context of the potential application in the standardization process. While GQL is not aimed to be based directly on the Cypher language, Cypher.PL could serve as an advanced starting point for an executable semantics of GQL.