

Inferring SHACL Constraints for Results of Composable Graph Queries (Extended Abstract)

Philipp Seifer¹, Daniel Hernández², Ralf Lämmel¹ and Steffen Staab^{2,3}

¹University of Koblenz, Koblenz, Germany

²University of Stuttgart, Stuttgart, Germany

³University of Southampton, Southampton, UK

Abstract


SPARQL CONSTRUCT queries allow for the specification of data processing pipelines that transform given input graphs into new output graphs. Input graphs are now commonly constrained through SHACL shapes allowing for both their validation and aiding users (as well as tools) in understanding their structure. However, it becomes challenging to understand what graph data can be expected at the end of a data processing pipeline without knowing the particular input data: Shape constraints on the input graph may affect the output graph, but may no longer apply literally, and new shapes may be imposed by the query itself. In our recent work, *From Shapes to Shapes: Inferring SHACL Shapes for Results of SPARQL CONSTRUCT Queries*, we studied the derivation of shape constraints that hold on all possible output graphs of a given SPARQL CONSTRUCT query by axiomatizing the query and the shapes with the *ALC^HOI* description logic. This extended abstract summarizes our previous work.

Keywords


Semantic Queries, Data Pipelines, SHACL, SPARQL CONSTRUCT


1. Introduction


Some graph query languages are composable (i. e., they construct new graphs as results) and thereby allow for the fruitful composition of queries into data processing pipelines. Examples for such languages are SPARQL (in particular, its CONSTRUCT [1] queries) for RDF graphs and more recently G-CORE [2] for property graphs. When graphs existing in the context of such composable query languages are validated using constraints specified in a shape description language such as SHACL [3] or ProGS [4], an interesting problem arises: Even though the input to a query is well-defined with SHACL or ProGS shapes, it becomes unclear which shapes apply after executing the query. Constraints that applied to the input graph may be invalidated by the query, e. g., because some required edges were removed, and new shapes may arise from the queries' construction template as well. Indeed, both downstream applications (e. g., programming languages [5]) and software developers working with graph data must understand what a query may output.

 DL 2024: 37th International Workshop on Description Logics, June 18–21, 2024, Bergen, Norway

 pseifer@uni-koblenz.de (P. Seifer); daniel.hernandez@ki.uni-stuttgart.de (D. Hernández); laemmel@uni-koblenz.de (R. Lämmel); steffen.staab@ki.uni-stuttgart.de (S. Staab)

 0000-0002-7421-2060 (P. Seifer); 0000-0002-7896-0875 (D. Hernández); 0000-0001-9946-4363 (R. Lämmel); 0000-0002-0780-4154 (S. Staab)

 © 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

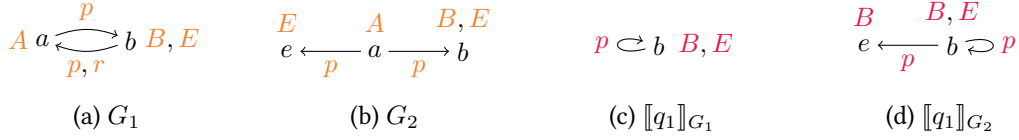


Figure 1: (a,b) Input graphs for q_1 valid with respect to S_1 . (c,d) Results of q_1 on G_1 and G_2 .

Problem Description In our recent paper [6], we define the problem of computing a set of SHACL shapes that characterises the possible output graphs of a SPARQL CONSTRUCT query. We represent queries as rules $H \leftarrow P$ where the *template* H and *pattern* P are sets of assertions with variables. Bindings for variables in P are used to construct a new graph according to H . We express SHACL shapes as *ALC*HOI axioms (cf. [7]).

Formally, we define the problem OUTPUTSHAPES with signature $(\mathcal{S}_{\text{in}}, q) \mapsto \mathcal{S}_{\text{out-max}}$, where q is a query and \mathcal{S}_{in} and $\mathcal{S}_{\text{out-max}}$ are two sets of shapes, called the *input* and *output* shapes. The set $\mathcal{S}_{\text{out-max}}$ is the maximum set of shapes such that for every graph G_{in} that satisfies the input shapes – we write $\text{valid}(G_{\text{in}}, \mathcal{S}_{\text{in}})$ – the graph resulting from evaluating q on G_{in} , denoted $\llbracket q \rrbracket_{G_{\text{in}}}$, satisfies the output shapes, i. e., $\text{valid}(\llbracket q \rrbracket_{G_{\text{in}}}, \mathcal{S}_{\text{out-max}})$.

Running Example Consider a problem OUTPUTSHAPES with a query q_1 and a set of shapes S_1 as input. Let $q_1 = \{y:E, z:B, (y, z):p\} \leftarrow \{(w, y):p, y:B, (x, z):p, z:E\}$. Given a graph G_{in} , query q_1 finds bindings for variable y that are B , and bindings for variable z that are E , both with incoming edges with role name p in G_{in} . Then, the query constructs the output graph, $G_{\text{out}} = \llbracket q_1 \rrbracket_{G_{\text{in}}}$, consisting only of these bindings for z and y , with inverted concept annotations and p -edges between them. Note, that we color-code the namespace of the input versus output graphs, since the extensions of the involved concept and role names are not the same.

Let $S_1 = \{s_1, s_2, s_3\}$ where $s_1 = A \sqsubseteq \exists p.B$, $s_2 = \exists r.\top \sqsubseteq B$ and $s_3 = B \sqsubseteq E$. The graphs G_1 (a) and G_2 (b) in Figure 1 are valid with respect to S_1 . Graphs $\llbracket q_1 \rrbracket_{G_1}$ (c) and $\llbracket q_1 \rrbracket_{G_2}$ (d) in Figure 1 are the respective output graphs for q_1 over G_1 and G_2 , respectively.

Some expected output shapes are $E \sqsubseteq \exists p.B$ and $E \sqsubseteq B$. The first shape follows directly from the query template. Each node labelled E has an outgoing edge to a node labelled B . The second shape $E \sqsubseteq B$ is valid because $B \sqsubseteq E$ holds on all input graphs: As a result, we can infer that all bindings for y are also bindings for z , such that $E \sqsubseteq B$ follows from the query template.

Proposed Algorithm In this paper, we summarize the algorithm we presented in [6] (an extended version with proofs is available in [8]), which constructs a *sound* (but not *complete*) approximation $\mathcal{S}_{\text{out}} \subseteq \mathcal{S}_{\text{out-max}}$.

To this end, we split the problem into two subproblems: First, the problem of deciding whether any given shape must be included in the output, and secondly the problem of generating a set of candidate shapes. The second problem turns out to be rather trivial, as the set of candidates is finite: We consider a subset of SHACL that is (syntactically) finite for a finite vocabulary, and show that all relevant candidates are indeed from the finite vocabulary of the query. In our extended version, we show that this also holds for a much larger subset of SHACL.

Thus, we focus on the first problem in Section 2 and Section 3.

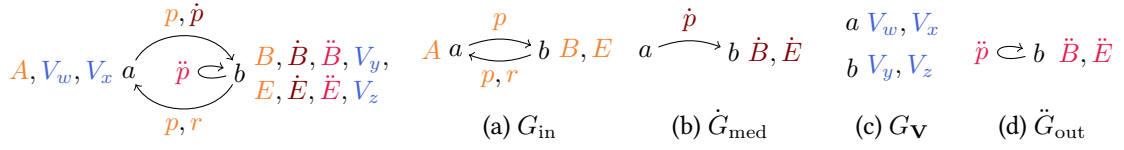


Figure 2: On the left the graph G_{ext} as the union of G_{in} (a), \dot{G}_{med} (b), $G_{\mathbf{V}}$ (c), and \ddot{G}_{out} (d).

2. Axiomatizations Over Query Executions

As hinted earlier, different occurrences of the same concept name do not have the same extensions: A query matches on input graphs, determines valuations (as subsets of the input), and constructs new graphs. We distinguish between the inputs, intermediate bindings, as well as the constructed output by rewriting input symbols A, p into *fresh* symbols \dot{A}, \dot{p} after the first step, and into \ddot{A}, \ddot{p} after the second step. These rewritten symbols allow us to encode assertions that are valid for only specific states of query execution. Variable bindings, on the other hand, hold throughout: We codify a variable binding $\mu(x) = a$ as a concept assertion $a:V_x$, where V_x is a fresh concept name.

In order to axiomatize how (all possible) input graphs are connected to (all possible) output graphs, we define a (virtual) *extended graph* G_{ext} , that unifies the different steps, and therefore allows us to reason about them: $G_{\text{ext}} := G_{\text{in}} \cup \dot{G}_{\text{med}} \cup G_{\mathbf{V}} \cup \ddot{G}_{\text{out}}$, where G_{med} is $\bigcup_{\mu(P) \subseteq G_{\text{in}}} \mu(P)$ (i.e., the union of all graphs $\mu(P)$ resulting from replacing every variable x in P with $\mu(x)$), G_{out} is $\llbracket q \rrbracket_{G_{\text{in}}}$, and $G_{\mathbf{V}}$ is the graph containing an assertion $a:V_x$ if and only if there exists a valuation μ such that $\mu(P) \subseteq G_{\text{in}}$ and $\mu(x) = a$. Figure 2 shows the extended graph and its components for the running example query q_1 , and the graph G_1 in Figure 1.

Proposition 1. *Given a graph G_{in} and a query q , let the graphs G_{ext} , G_{med} , and G_{out} be the extended graph and its components. For every axiom φ that does not include names with dots (e. g., names \dot{A} , \ddot{A} , \dot{p} , or \ddot{p}), the following equivalences hold: $\text{valid}(G_{\text{in}}, \{\varphi\})$ if and only if $\text{valid}(G_{\text{ext}}, \{\varphi\})$, $\text{valid}(G_{\text{med}}, \{\varphi\})$ if and only if $\text{valid}(G_{\text{ext}}, \{\dot{\varphi}\})$, and $\text{valid}(G_{\text{out}}, \{\varphi\})$ if and only if $\text{valid}(G_{\text{ext}}, \{\ddot{\varphi}\})$.*

Given the notion of extended graphs, we can prove Proposition 1 (see [8] for the proof), which is essential to our method. Utilizing this proposition, we can show as a corollary that given a set of axioms Σ such that $\text{valid}(G_{\text{ext}}, \Sigma)$ for all extended graphs of a query q , if $\Sigma \models \ddot{s}$ then $\text{valid}(G_{\text{out}}, \{s\})$ for every output graph G_{out} of q . Thus, what remains is to show how to construct such a set of axioms Σ .

3. Axioms Valid on Extended Graphs

Let us now consider what axioms can be inferred, by inspecting the running example. First, we can notice that the input shapes \mathcal{S}_{in} are valid on all input graphs, by definition. Thus, we include them in our knowledge base Σ .

Some axioms of the validation knowledge base (unique name, closed world and domain closure assumptions) can be approximated by investigating the query q . The unique name

assumption is limited to individual names that occur in the query; in the running example there are none. Since a query does not determine the set of individual names, no axioms related to the domain closure assumption can be inferred. On the other hand, a query does restrict concept names that appear in the query with respect to the closed world assumption (CWA).

For the running example, we can thus infer $\{\dot{B} \equiv B \sqcap V_y, \dot{E} \equiv E \sqcap V_z\}$, because e. g., concept \dot{B} in the extended graph is defined by filtering B with variable V_y , based on the query pattern $y:B$ in q_1 . We can also infer axioms $\{V_w \equiv \exists p.V_y, V_x \equiv \exists p.V_z, V_y \equiv \exists p.V_w \sqcap B, V_z \equiv \exists p.V_x \sqcap E\}$ since variable concepts are defined by constraints to the variable in the query pattern. For example, V_y is constrained by patterns $(w, y):p$ and $y:B$ in q_1 , and thus bound by $\exists p.V_w \sqcap B$. This is a crucial step, since concept and role names in the extended graph are defined in terms of these variable concepts: $\{\ddot{B} \equiv V_z, \ddot{E} \equiv V_y\}$ since e. g., concept \ddot{B} in the extended graph is defined by V_z , as it only occurs in the single construct pattern $z:B$. With similar reasoning, $\{\exists \dot{p}.V_y \equiv V_w, \exists \dot{p}.V_z \equiv V_x, \exists \dot{p}.\top \equiv (V_w \sqcap \exists \dot{p}.V_y) \sqcup (V_x \sqcap \exists \dot{p}.V_z)\}$ as well as $\{\exists \ddot{p}.V_z \equiv V_y, \exists \ddot{p}.\top \equiv V_y \sqcap \exists \ddot{p}.V_z\}$ (and their inverse counterparts) can be constructed with respect to role names.

Query q_1 has two components $P_1 = \{(w, y):p, y:B\}$ and $P_2 = \{(x, z):p, z:E\}$ not sharing variables. The CWA encoding does not entail $V_y \sqsubseteq V_z$, even though this axiom is both valid in all extended graphs, and required for inferring, e. g., the result shape $E \sqsubseteq B$. In another step of the algorithm, we infer these additional subsumptions by constructing variable mappings h between query components, that are potentially extended with input shapes. In this case, we know based on input shape S_1 that $B \sqsubseteq E$. We can utilize this knowledge to extend component $(w, y):p, y:B$, adding the pattern $y:E$ which does not alter the queries results. Then, we can find the mapping $h(x) = w, h(z) = y$ such that $h(P_2) \subseteq P_1$, which implies $V_w \sqsubseteq V_x$ and $V_y \sqsubseteq V_z$.

4. Conclusion

We presented an algorithm for inferring a set of shapes that validate the possible output graphs of a CONSTRUCT query, where input graphs of this query can be constrained by a set of shapes as well. This enables the inference of shapes over result graphs of data processing pipelines (i. e., compositions of CONSTRUCT queries), which can be used both for validation purposes when working with these result graphs, and informatively, aiding developers directly.

Our approach differs from related work (e.g., [9, 10, 11, 12, 13, 14]) in that we infer shapes from statically known information (query and input shapes) and not from instance data. Thus, it is similar to approaches for inferring constraints over views on relational databases (e.g., [15, 16, 17, 18]), but utilizing a modelling approach that is feasible and supports crucial constraints for typing knowledge graphs. Some approaches construct SHACL from static information [19, 20, 21], such as RML rules or direct mappings, though they are limited in either a less expressive mapping language, or lack support for constraints on the input data. An implementation [22] for the algorithms presented in our work is available on GitHub¹.

Acknowledgments *This work was partially funded by Deutsche Forschungsgemeinschaft (DFG) under SPP 1921 – 318363223, and the DFG Germany’s Excellence Strategy – EXC 2120/1 – 390831618.*

¹<https://github.com/softlang/s2s>

References

- [1] S. Harris, A. Seaborne, E. Prud'hommeaux, SPARQL 1.1 query language, 2013. URL: <https://www.w3.org/TR/sparql11-query/>.
- [2] R. Angles, M. Arenas, P. Barceló, P. A. Boncz, G. H. L. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. F. Sequeda, O. van Rest, H. Voigt, G-CORE: A core for future graph query languages, in: Proceedings of SIGMOD, ACM, 2018, pp. 1421–1432. doi:10.1145/3183713.3190654.
- [3] H. Knublauch, D. Kontokostas, Shapes Constraint Language (SHACL), 2017. URL: <https://www.w3.org/TR/shacl/>.
- [4] P. Seifer, R. Lämmel, S. Staab, ProGS: Property graph shapes language, in: Proceedings of ISWC, volume 12922 of LNCS, Springer, 2021, pp. 392–409. doi:10.1007/978-3-030-88361-4_23.
- [5] M. Leinberger, P. Seifer, C. Schon, R. Lämmel, S. Staab, Type checking program code using SHACL, in: Proceedings of ISWC, volume 11778 of LNCS, Springer, 2019, pp. 399–417. doi:10.1007/978-3-030-30793-6_23.
- [6] P. Seifer, D. Hernández, R. Lämmel, S. Staab, From shapes to shapes: Inferring SHACL shapes for results of SPARQL CONSTRUCT queries, in: Proceedings of the ACM Web Conference 2024, WWW 2024, ACM, 2024, pp. 2064–2074. doi:10.1145/3589334.3645550.
- [7] B. Bogaerts, M. Jakubowski, J. V. den Bussche, SHACL: A description logic in disguise, in: Proceedings of Logic Programming and Nonmonotonic Reasoning, volume 13416 of LNCS, Springer, 2022, pp. 75–88. doi:10.1007/978-3-031-15707-3_7.
- [8] P. Seifer, D. Hernández, R. Lämmel, S. Staab, From shapes to shapes: Inferring SHACL shapes for results of SPARQL CONSTRUCT queries (extended version), 2024. arXiv:2402.08509.
- [9] B. Spahiu, A. Maurino, M. Palmonari, Towards improving the quality of knowledge graphs with data-driven ontology patterns and SHACL, in: Proceedings of the Workshop on Ontology Design and Patterns (WOP 2018) co-located with ISWC 2018, volume 2195 of CEUR Workshop Proceedings, CEUR-WS.org, 2018, pp. 52–66. URL: https://ceur-ws.org/Vol-2195/research_paper_2.pdf.
- [10] D. Fernández-Álvarez, J. E. L. Gayo, D. Gayo-Avello, Automatic extraction of shapes using sheXer, Knowledge-Based Systems 238 (2022) 107975. doi:10.1016/J.KNOSYS.2021.107975.
- [11] K. Rabbani, M. Lissandrini, K. Hose, Extraction of validating shapes from very large knowledge graphs, Proceedings VLDB Endow. 16 (2023) 1023–1032. doi:10.14778/3579075.3579078.
- [12] N. Mihindukulasooriya, M. R. A. Rashid, G. Rizzo, R. García-Castro, Ó. Corcho, M. Torchiano, RDF shape induction using knowledge base profiling, in: Proc. of the Symposium on Applied Computing, ACM, 2018, pp. 1952–1959. doi:10.1145/3167132.3167341.
- [13] P. G. Omran, K. Taylor, S. J. R. Méndez, A. Haller, Learning SHACL shapes from knowledge graphs, Semantic Web 14 (2023) 101–121. doi:10.3233/SW-223063.
- [14] B. Groz, A. Lemay, S. Staworko, P. Wiecek, Inference of shape graphs for graph databases, in: ICDT, volume 220 of LIPICs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 14:1–14:20. doi:10.4230/LIPICs.ICDT.2022.14.

- [15] A. C. Klug, R. Price, Determining view dependencies using tableaux, *ACM Trans. Database Syst.* 7 (1982) 361–380. doi:10.1145/319732.319738.
- [16] W. Fan, S. Ma, Y. Hu, J. Liu, Y. Wu, Propagating functional dependencies with conditions, *Proceedings VLDB Endow.* 1 (2008) 391–407. doi:10.14778/1453856.1453901.
- [17] M. Stonebraker, Implementation of integrity constraints and views by query modification, in: *Proceedings of SIGMOD*, ACM, 1975, pp. 65–78. doi:10.1145/500080.500091.
- [18] B. E. Jacobs, A. R. Aronson, A. C. Klug, On interpretations of relational languages and solutions to the implied constraint problem, *ACM Trans. Database Syst.* 7 (1982) 291–315. doi:10.1145/319702.319730.
- [19] R. B. Thapa, M. Giese, Mapping relational database constraints to SHACL, in: *Proceedings of ISWC*, volume 13489 of *LNCS*, Springer, 2022, pp. 214–230. doi:10.1007/978-3-031-19433-7_13.
- [20] R. B. Thapa, M. Giese, A source-to-target constraint rewriting for direct mapping, in: *Proceedings of ISWC*, volume 12922 of *LNCS*, Springer, 2021, pp. 21–38. doi:10.1007/978-3-030-88361-4_2.
- [21] T. Delva, B. D. Smedt, S. M. Oo, D. V. Assche, S. Lieber, A. Dimou, RML2SHACL: RDF generation taking shape, in: *Proceedings of Knowledge Capture Conference*, ACM, 2021, pp. 153–160. doi:10.1145/3460210.3493562.
- [22] P. Seifer, D. Hernández, R. Lämmel, S. Staab, Code for from shapes to shapes, 2024. doi:10.18419/darus-3977.