Transactions on Graph Data and Knowledge

Volume 4 | Issue 1 | September, 2026

ISSN 2942-7517

ACM Classification 2012
Information systems → Resource Description Framework (RDF); Computing methodologies → Semantic networks; Hardware → Impact on the environment; Information systems → Information integration; Computing methodologies → Knowledge representation and reasoning; Applied computing → Arts and humanities; Software and its engineering → Object oriented languages; Computing methodologies → Modeling and simulation; Information systems → Data provenance

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.
Online available at
`https://www.dagstuhl.de/dagpub/2942-7517`.

Publication date
September, 2026

Digital Object Identifier
10.4230/TGDK.4.1.0

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://dnb.d-nb.de.

Aims and Scope
Transactions on Graph Data and Knowledge (TGDK) is an Open Access journal that publishes original research articles and survey articles on graph-based abstractions for data and knowledge, and the techniques that such abstractions enable with respect to integration, querying, reasoning and learning. The scope of the journal thus intersects with areas such as Graph Algorithms, Graph Databases, Graph Representation Learning, Knowledge Graphs, Knowledge Representation, Linked Data and the Semantic Web. Also in-scope for the journal is research investigating graph-based abstractions of data and knowledge in the context of Data Integration, Data Science, Information Extraction, Information Retrieval, Machine Learning, Natural Language Processing, and the Web.

The journal is Open Access without fees for readers or for authors (also known as Diamond Open Access).

Editorial Office
Schloss Dagstuhl – Leibniz-Zentrum für Informatik
TGDK, Editorial Office
Oktavie-Allee, 66687 Wadern, Germany
tgdk@dagstuhl.de
`https://www.dagstuhl.de/tgdk`

Contents

List of Authors

Zubaria Asma (4)
FORTH-ICS, Heraklion, Crete, Greece; University of Crete, Heraklion, Crete, Greece

Oleksandra Bruns (2)
FIZ Karlsruhe - Leibniz Institute for Information Infrastructure, Eggenstein-Leopoldshafen, Germany; Karlsruhe Institute of Technology (AIFB), Germany

Victor Charpenay (1)
Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, Saint-Étienne, France

Giorgos Flouris (4)
FORTH-ICS, Heraklion, Crete, Greece

Irini Fundulaki (4)
FORTH-ICS, Heraklion, Crete, Greece

Luis Galárraga (4)
Inria, Rennes, France

Martin Giese (3)
University of Oslo, Norway

Daniel Hernández (4)
University of Stuttgart, Stuttgart, Germany

Katja Hose (4)
TU Wien, Wien, Austria

Einar Broch Johnsen (3)
University of Oslo, Norway

Eduard Kamburjan (3)
IT University of Copenhagen, Denmark, University of Oslo, Norway

Vidar Norstein Klungre (3)
University of Oslo, Norway

Egor V. Kostylev (3)
University of Oslo, Norway

Jeff Z. Pan (2)
The University of Edinburgh, Scotland

Yuanwei Qu (3)
University of Oslo, Norway

Harald Sack (2)
FIZ Karlsruhe – Leibniz Institute for Information Infrastructure, Germany; Karlsruhe Institute of Technology (AIFB), Germany

Rudolf Schlatte (3)
University of Oslo, Norway

Jörg Waitelonis (2)
FIZ Karlsruhe - Leibniz Institute for Information Infrastructure, Germany; Karlsruhe Institute of Technology (AIFB), Germany

Antoine Zimmermann (1)
Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, Saint-Étienne, France

Mansour Zoubeirou A Mayaki (1)
Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, Saint-Étienne, France

On the Computational Cost of Knowledge Graph Embeddings

Victor Charpenay
Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, Saint-Étienne, France

Mansour Zoubeirou A Mayaki
Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, Saint-Étienne, France

Antoine Zimmermann
Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, Saint-Étienne, France

Abstract

Over a decade, numerous Knowledge Graph Embedding (KGE) models have been designed and evaluated on reference datasets, always with increasing performance. In this paper, we re-evaluate these models with respect to their computational *efficiency* during training, by estimating the computational cost of the procedure expressed in floating-point operations. We design a cost model based on analytical expressions and apply it on a collection of 20 KGE models, representative of the state-of-the-art.

We show that dimensionality or parameter efficiency, used in the literature to compare models with each other, are not suitable to evaluate the true cost of models. Through fixed-budget experiments, a novel approach to evaluate KGE models based on cost estimates, we re-assess the relative performance of model families compared to the state-of-the-art. Bilinear models such as ComplEx underperform with a low computational budget while hyperbolic linear models appear to offer no particular benefit compared to simpler Euclidian models, especially the MuRE model. Neural models, such as ConvE or CompGCN, achieve reasonable performance in the literature but their high computational cost appears unnecessary when compared with other models. The trade-off between efficiency and expressivity of both linear and neural models is to be further explored.

2012 ACM Subject Classification Information systems → Resource Description Framework (RDF); Computing methodologies → Semantic networks; Hardware → Impact on the environment

Keywords and phrases Knowledge Graph Embedding, Parameter Efficiency, Computational Budget, Green AI

Digital Object Identifier 10.4230/TGDK.4.1.1

Category Research

Supplementary Material

Software: `https://gitlab.emse.fr/victor.charpenay1/kge-cost-experiments/` [14]
archived at `swh:1:dir:29d727e535031eca0583c489c67c47d0b9165d30`

Funding This work was supported by the European Lighthouse to Manifest Trustworthy and Green AI (ENFIELD) project funded by the European Union's HORIZON Research and Innovation Programme. Grant agreement No: 101120657.

Acknowledgements Computations have been performed on the supercomputer facilities of the Mésocentre Clermont-Auvergne of the Université Clermont Auvergne.

Received 2024-10-07 **Accepted** 2026-01-05 **Published** 2026-03-31

1 Introduction

In 2018, OpenAI researchers released a meta-analysis suggesting that the amount of compute to train state-of-the-art deep learning models had grown exponentially since 2012 [4]. This trend was confirmed a few years later by the coming of *large* language models such as GPT-3 [17], BLOOM [36] or LLaMa [43], which achieved unprecedented performance in natural language processing tasks.

This exponential trend has raised concerns about the sustainability of Artificial Intelligence (AI) research. It was estimated that training GPT-3 emitted as much CO_2 as 100 humans for their food, material goods, travel, etc. over a year (502 tCO_2e vs. 5 tCO_2e/capita/year[1]) [27]. Given that humanity's high carbon emissions are themselves not sustainable, the carbon intensity of GPT-3's training is clearly too high. BLOOM, an open-source model of comparable size, is 20 times less carbon intensive according to the same estimate (25 tCO_2e). LLaMa, following the same design principles as GPT-3 and BLOOM, has been shown to perform better with less trainable parameters. However, training LLaMa (65B parameters) required as much energy as training BLOOM (176B parameters), partly because of additional normalization layers and more complex activation functions, and emitted significantly more CO_2 (173 tCO_2e).

In the emerging field of Green AI, researchers aim to study the non-trivial relationship between the performance, parameter efficiency, computational cost and environmental impact of AI models [38]. Green AI models must not only be accurate at the task at hand, they must also be efficient: models must minimize compute to lower their environmental impact. As illustrated with BLOOM and LLaMa, parameter efficiency is not a good proxy to estimate the environmental impact of training a model. On the other hand, CO_2 emissions may greatly vary for the same training procedure depending on external conditions: for example, if BLOOM had been trained in the USA instead of France (whose energy mix includes 70% of nuclear energy), it would have emitted as much CO_2 as LLaMa. As a consequence, Green AI advocates encourage to report computational costs measured in floating-point operations (FLOPs[2]) instead. The objective of this paper is to study the computational cost of a specific class of models: Knowledge Graph Embedding (KGE) models.

KGE models are to Knowledge Graphs (KGs) what language models are to text corpora. They consist in vector representations of KG entities and relations that are meant to capture latent structures of an input KG (such as soft rules or implicit classes of entities). Through operations on vectors, models can generalize beyond what is explicitly stated in the KG and detect missing or erroneous information.

KGE models generally have a significantly lower cost than language models, even for large KGs. However, they seem to follow the same pattern of achieving better performance through more parameters and higher computational costs. TransE, one of the first KGE models published in 2013, was evaluated with 20 to 50 latent dimensions [11]. Six years later, the RotatE model was compared to TransE with 1,000 dimensions – with substantial absolute improvements for both models [40]. As we will see later in the paper, we estimate that to train RotatE in 20 times more dimensions required 3,000 times more compute than the original TransE model.

Morever, non-parametric models such as AnyBURL [28] and its successor, SAFRAN [32], meet the performance of embedding-based models with far less compute. Their approach consists in mining composition rules from the KG instead of learning latent embeddings. The authors of AnyBURL give the following comparison: after 1,000 s of mining on a single CPU, AnyBURL yields good results while the best performing KGE model on the same dataset, ComplEx-N3 [22],

[1] The estimate for human emissions is taken from Strubell *et al.* [39].
[2] Note that 1 FLOP is not the same as 1 FLOPS, which measures floating-point operations *per second*.

requires between 10,000 s and 50,000 s of training on a GPU. The major drawback of AnyBURL and SAFRAN, however, is that a considerable amount of compute is necessary in the inference phase, while operations in the TransE, RotatE or ComplEx-N3 latent spaces are relatively cheap once trained.

More recent publications question the need for high dimensions to train KGE models [6, 13]. The AttH model has been shown to perform well with as few as 32 dimensions [13]. However, if one goes beyond dimensionality, as encouraged in Green AI, it can be shown that AttH requires significantly more FLOPs than RotatE, itself requiring more FLOPs than TransE with the same number of dimensions. In this paper, we review and compare the efficiency of representative KGE models in order to identify those able to improve performance with minimal computational overhead and to guide future work towards KGE training procedures with the lowest possible computational cost. Several large-scale studies have already compared the efficiency of KGE models under various constraints, such as dimensionality [35, 2] and execution time [34], but to the best of our knowledge, no comparison with respect to computational cost, measured in FLOPs, has ever been done. We aim to fill this gap in the paper.

We start our study with an overview of state-of-the-art KGE models in Section 2. We then compare models with respect to their size (number of trainable parameters) in Section 3, to highlight the limitations of this measure. To evaluate the computational cost of KGE models in FLOPs, the preferred Green AI metric, we proceed in two steps: we provide estimates for inference cost in Section 4.2 and build up on these results to provide estimates for full training cost in Section 4.3. Training is indeed roughly equivalent to a sequence of inferences on positive and negative examples of KG statements. Finally, in Section 5, we consider fixed-budget performance obtained by setting an upper bound to the number of FLOPs allowed for training. Fixed-budget performance is not only relevant in contexts where the computing platform has low power supply, they also provide a fairer comparison point as we will see towards the end of the paper.

2 State-of-the-Art Models

KGE models have been designed for various inductive learning tasks: link prediction, entity classification, triple classification, query answering, entity similarity measure, etc [21, Sec. 5.2]. In this paper, we restrict ourselves to the most common task, link prediction in a transductive setting. Given a KG defined as a set of $\langle h, r, t\rangle$ triples, where $h, t \in E$ are referred to as (head and tail) entities and $r \in R$ as a relation, link prediction consists in evaluating the plausibility of an unknown triple $\langle h, r, t'\rangle$ or $\langle h', r, t\rangle$, in which h', t' are known to be in E. What a model must learn is then a scoring function $f : E \times R \times E \to \mathbb{R}$ such that $f(h, r, t)$ encodes the plausibility of triple $\langle h, r, t\rangle$. Note that $f(h, r, t)$ is not necessarily a probability, as it is not restricted to the interval $[0, 1]$. In practice, scoring is only used to rank triples.

Early KGE models were being evaluated on two datasets, FB15k and WN18, respectively derived from Freebase and WordNet [11]. FB15k includes about 15,000 entities and more than 1,000 relations. WN18 has more entites (∼40,000) but less relations (18). It has later been found that naive models, that look for pairs of inverse relations $(r, r^-$ such that $\langle h, r, t\rangle$ and $\langle t, r^-, h\rangle$ are in the dataset), perform well on both datasets [42, 18]. For a more challenging evaluation, the FB15k-237 and WN18RR variants were introduced, in which one of r or r^- was filtered out. A leaderboard reports results for at least 68 KGE models on FB15k-237[3] and on WN18RR[4]. Some of these models were also evaluated on larger datasets such as YAGO3-10 but to be able to compare all models with each other, we choose to focus on FB15k-237 and WN18RR only in this paper.

[3] https://paperswithcode.com/sota/link-prediction-on-fb15k-237
[4] https://paperswithcode.com/sota/link-prediction-on-wn18rr

Table 1 Knowledge Graph Embedding models selected for this study.

Linear		Bilinear	Neural
Euclidian	**Hyperbolic**		
TransE [11]	MuRP [8]	DistMult [49]	R-GCN [37]
RotatE [40]	AttH [13]	ComplEx [44]	ConvE [18]
BoxE [1]	ConE [6]	ComplEx-N3 [22]	ConvKB [29]
ExpressivE [33]		TuckER [9]	CompGCN [45]
MuRE [8]		QuatE [50]	NBFNet [52]
AttE [13]		DualE [12]	

Most recent models, including LP-BERT [23], MoCoSA [20], Relphormer [10] and KERMIT [24], rely on pre-trained language models for learning. The computational cost of training language models is an order of magnitude higher than that of KGE models. The base version of BERT, for instance, has 110M trainable parameters [19] whereas the KGE models we consider here have up to 40M parameters. Its FLOP count is also significantly higher. For inference, BERT requires 131M FLOPs to output a single token (see calculation details in Appendix A). This estimate is a lower bound, given that a triple is represented at least with three tokens. In comparison, the two most expensive models in our study require 83M and 16M FLOPs to score the entire triple. In this paper, we choose to focus on KGE models that take a KG as the only input to training. Such models can only learn structural features of the KG (as opposed to textual features, in the case of pre-trained language models).

Out of the numerous KGE models available in the literature, we made a selection of 20 models[5]. These models are listed in Table 1. KGE models have traditionally been classified in three main categories: linear or distance-based models, bilinear or tensor factorization models and neural models [30, 47, 21]. Linear models can be subdivided into Euclidian models and hyperbolic models. TransE and RotatE are Euclidian models while AttH (which performs well in low dimensions but requires more FLOPs than RotatE) is a hyperbolic model, for instance. As the distinction between the two may be important in a computational cost analysis, we selected models such that both Euclidian and hyperbolic categories, as well as bilinear and neural categories, are evenly represented. We selected up to 5 models in each category: 4 Euclidian models, 3 hyperbolic models, 5 bilinear models and 5 neural models. Two out of the 3 hyperbolic models have Euclidian variants that have also been evaluated on FB15k-237 and WN18RR. One of the bilinear models, ComplEx [44], also has a variant that shows better performance. For comparison purposes, we included these 4 additional models, leading to a total of 20 KGE models. We describe each category of models in the remainder of the section.

2.1 Euclidian Models

Linear (Euclidian and hyperbolic) models have in common that their scoring function is defined in terms of a distance between head and tail entity embeddings. Most linear scoring functions have the following generic form:

$$f(h, r, t) = -\|g_r(\mathbf{e}_h) - \mathbf{e}_t\| \tag{1}$$

[5] These models were published between 2013 and 2021. Best-performing models published after 2021, from KG-BERT to KERMIT, were excluded due to the fact they rely on pre-trained language models.

where $\mathbf{e}_x$ is a vector embedding of term x in $\mathbb{R}^d$, $\mathbb{C}^d$ and $\|\cdot\|$ is the L1-norm or L2-norm. Embedding $\mathbf{e}_h$ is first transformed by a function specific to relation r, then compared to $\mathbf{e}_t$. The closer the two embeddings are, the more likely it is that triple $\langle h, r, t\rangle$ belongs to the KG.

The simplest model, TransE, represents relations as a single vector $\mathbf{w}_r \in \mathbb{R}^d$ used as a translation offset to transform $\mathbf{e}_h$. Scoring is done by calculating $f_{\text{TransE}}(h, r, t) = -\|(\mathbf{e}_h + \mathbf{w}_r) - \mathbf{e}_t\|$. RotatE and AttE replace translation with rotation, defined by the following block-diagonal matrix:

$$\mathbf{W}_r = \begin{bmatrix} \mathbf{R}_{\theta_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathbf{R}_{\theta_d} \end{bmatrix} \tag{2}$$

where $\mathbf{R}_{\theta_i}$ is a 2D rotation matrix:

$$\mathbf{R}_{\theta_i} = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) \\ sin(\theta_i) & cos(\theta_i) \end{bmatrix} \tag{3}$$

Relation embeddings can be concisely represented as a rotation vector $\theta \in \mathbb{R}^d$ but they can also be represented as a complex vector $\mathbf{w}_r \in \mathbb{C}^d$. If every member $w_{r,i}$ of $\mathbf{w}_r$ is normalized such that $w_{r,i} = \cos(\theta_{r,i}) + i\sin(\theta_{r,i})$, the two representations are equivalent. RotatE uses the complex representation and defines scoring as $f_{\text{RotatE}}(h, r, t) = -\|\mathbf{e}_h \odot \mathbf{w}_r - \mathbf{e}_t\|$, where $\odot$ is the Hadamard product. AttE uses the former representation.

Along with translation and/or rotation, some models introduce scaling. For instance, MuRE combines scaling and translation, giving the scoring function $f_{\text{MuRE}}(h, r, t) = -\|(\mathbf{e}_h \odot \mathbf{w}_r + \mathbf{w}'_r) - \mathbf{e}_t\|$. Contrary to RotatE, MuRE embeddings are defined over $\mathbb{R}^d$. AttE distinguishes itself from other linear models by introducing a further geometric transformation: reflection. However, as rotation and reflection are not entirely independent, AttE also introduces an attention mechanism across rotation and reflection with more parameters to train, i.e. with a higher computational cost. BoxE and ExpressivE are both translational but they have another distinguishing feature: they introduce a width parameter in each dimension in order to view relation embeddings as regions instead of points in the latent space. This also comes at the price of a higher computational cost.

A number of other linear models, such as TransR [26] and HAKE [51], also apply geometric transformations to the head entity before scoring. These models are not included in our study.

2.2 Hyperbolic Models

Hyperbolic linear models – MuRP, AttH and ConE in our study – redefine rotation, translation, scaling and reflection. Contrary to classical Euclidian space, distance in a hyperbolic space of curvature c is given by the following formula:

$$d_c(\mathbf{x}, \mathbf{y}) = \frac{2}{\sqrt{c}} \text{arctanh}(\sqrt{c}\|\mathbf{x} \oplus^c \mathbf{y}\|) \tag{4}$$

where $\oplus^c$ is the Möbius addition (the hyperbolic translation operator), defined as follows:

$$\mathbf{x} \oplus^c \mathbf{y} = \frac{(1 + 2c\mathbf{x}^T\mathbf{y} + c\|\mathbf{x}\|^2)\mathbf{x} + (1 - c\|\mathbf{x}\|^2)\mathbf{y}}{1 + 2c\mathbf{x}^T\mathbf{y} + c^2\|\mathbf{x}\|^2\|\mathbf{y}\|^2} \tag{5}$$

Scoring in hyperbolic KGE models is performed by calculating $-d_c(g_r(\mathbf{e}_h), \mathbf{e}_t)$ instead of $-\|g_r(\mathbf{e}_h) - \mathbf{e}_t\|$.

MuRP and AttH are equivalent to MuRE and AttE, up to a redefinition of operators in hyperbolic space. MuRP is a combination of translation and scaling while AttH uses rotation and reflection (with some attention weight). ConE combines rotation and scaling but, to the best of our knowledge, no Euclidian counterpart to ConE exists in the literature.

Among other properties, geometries in hyperbolic spaces are known to efficiently capture hierarchies of entities: a tree of entities can be embedded in a 2D hyperbolic space such that all entities at level i are equally distant to their parent at level $i-1$, without constraing the respective distance between same-level entities. The distance between embeddings of entities e_1 and e_2 can then represent the depth of e_2 in the hierarchy relative to e_1 while the distance between e_1 and e_3, at the same level, may represent their similarity for some non-hierarchical relation.

Other non-Euclidian, non-linear models also adapt the idea of defining a distance between head and tail entity representations. Bayesian models, for instance, measure a distance between probability distributions [46]. Such models are not included in our study.

2.3 Bilinear Models

Bilinear models define scoring in terms of a dot product instead of a distance, as follows:

$$f(h,r,t) = Re(g_r(\mathbf{e}_h) \cdot \overline{\mathbf{e}_t}) \tag{6}$$

where $Re(x)$ is the real part of complex number x and $\overline{\mathbf{x}}$ is the conjungate of complex vector $\mathbf{x}$. If $\mathbf{x} \in \mathbb{R}^d$, then $\mathbf{x}$ and $\overline{\mathbf{x}}$ are identical.

DistMult and ComplEx both define g_r as the Hadamard product with some relation embedding $\mathbf{w}_r$ but ComplEx gains expressivity compared to DistMult by simply going from real-valued to complex-valued embedding. The scoring function of ComplEx is $f_{\text{ComplEx}}(h,r,t) = \text{Re}((\mathbf{e}_h \odot \mathbf{w}_r) \cdot \overline{\mathbf{e}_t})$. QuatE and DualE look for more generality by extending further to quaternion-valued and dual quaternion-valued embeddings. Members of quaternion space $\mathbb{H}$ are generalizations of complex numbers of the form $a + ib + jc + kd$, where i, j and k are imaginary numbers. Members of the dual quaternion space have twice as many components. Both spaces redefine addition and multiplication, requiring more FLOPs than their scalar counterpart.

TuckER is an attempt to redefine and generalize ComplEx in $\mathbb{R}^d$. It is based on the idea that KGs, if seen as tensors of order 3, can be factorized into entity- and relation-specific embeddings, respectively in $\mathbb{R}^{d_e}$ and $\mathbb{R}^{d_r}$. To recover the full tensor, these factors can be multiplied to a core tensor defined in $\mathbb{R}^{d_e \times d_r \times d_e}$. This generalization involves significantly more computation than ComplEx, which might explain why ComplEx and ComplEx-N3 have remained the most popular KGE models so far. MetaSD [25], for instance, derives from ComplEx. It performs distillation on a pre-trained ComplEx model, intending both to reduce the size of the original model and to improve its performance. Model distillation goes beyond the scope of our study, though.

Other bilinear models include RESCAL [31], the very first wide-spread KGE model, and HypER [7], which was inspired by ConvE (see next section) and led to the design of TuckER.

2.4 Neural Models

The last category is slightly more heterogeneous. It includes various neural models whose architecture features 1D convolution (ConvKB), 2D convolution (ConvE) or graph convolution (R-GCN) and other message passing operations. A distinctive feature of these models is the introduction of non-linearities, such as the ReLU function, to latent vectors before calculating a score.

R-GCN, CompGCN and NBFNet follow the general architecture of Graph Neural Networks (GNNs), which consists in a sequence of combine-aggregate operations that follow the structure of the KG: the embedding of an input entity is calculated from the embeddings of its neighbors, in 2 steps. In the combine step, the embeddings of all neighbors are transformed by a relation-specific function. In the aggregate step, those intermediary embeddings are aggregated into a single vector

that can be used as a representation for the input entity. Several combine-aggregate operations can be computed on top of each other, forming GNN layers. The formulation for R-GCN is the following:

$$\mathbf{h}_t^{(l+1)} = \mathrm{ReLU}\left(\sum_{r\in R}\sum_{h\in B_{t,r}} \frac{1}{|B_{t,r}|}\mathbf{W}_r^{(l)}\mathbf{h}_h^{(l)}\right) \tag{7}$$

where $B_{t,r}$ is the set of neighbors of t ($h \in B_{t,r}$ if $\langle h, r, t\rangle$ is in the KG), $\mathbf{h}_e^{(l)}$ is the embedding of entity e at layer l and $\mathbf{W}_r^{(l)}$ is a trainable matrix for relation r at layer l. To reduce the number of trainable parameters per relation, $\mathbf{W}_r^{(l)}$ is defined as a block-diagonal matrix, which makes R-GCN close to bilinear models in spirit. CompGCN uses cross-correlation instead of matrix multiplication while NBFNet uses scaling and translation operators, and extends the calculation to all entities in the KG (as opposed to neighbors only).

Note that GNNs only produce embeddings. When training GNNs for link prediction, the obtained embeddings must still be compared to calculate a final score. R-GCN applies DistMult (with an additional relation embedding), CompGCN uses a pre-trained ConvE model and NBFNet appends a dedicated multilayer perceptron to the network to reduce embeddings to a scalar. GNNs thus necessarily require more FLOPs than the simpler models they may rely on.

Neural models also include models based on the Transformer architecture, which has recently found many applications. LP-BERT, MoCoSA, Relphormer and KERMIT all fall into this category because they rely on BERT, a bidirectional Transformer encoder. Other models, including HittER [16], adapt the Transformer architecture to KG structures instead of turning triples into text. However, at the time of writing, such Transformer-based models are still outperformed by NBFNet on FB15k-237 and WN18RR, despite significant FLOP counts. We decided to leave them for future work.

2.5 Training Procedure

All scoring functions mentioned above are to be trained by stochastic gradient descent, an iterative process. At each step, the score of each KG triple is compared to the score of one or more triples not stated in the KG, yielding a loss value that is to be minimized. Various loss functions have been introduced to optimize scoring functions. The most common loss is binary cross-entropy, treating the task of link prediction as binary classification. However, several studies suggest that loss functions are not tightly coupled with scoring functions [2, 35]: several models have been shown to perform better if trained with loss functions not originally designed for these models.

Several of the losses found in the KGE literature, including cross-entropy loss, are subsumed by the following formula:

$$\mathcal{L}(\tau) = -\log(\sigma(\lambda + f(\tau))) - \sum_{\tau'\in N_\tau} w_{\tau'}\log(1-\sigma(\lambda + f(\tau'))) \tag{8}$$

where $\tau = \langle h, r, t\rangle, \tau' = \langle h', r', t'\rangle$ are triples, σ is the sigmoid function and N_τ is a set of "negative" triples, assumed to be false statements. The standard practice for choosing N_τ is to construct it from $\langle h, r, t\rangle$, setting $r = r'$ and either $h = h'$ or $t = t'$ but not both.

The formula for $\mathcal{L}$ has two hyperparameters: λ and $w_{\tau'}$ (for each $\tau' \in N_\tau$). The λ value is a margin that intuitively captures that above a certain score (the margin), triples may be considered true and false otherwise. This margin hyperparameter is especially important to train linear models: since they can only output negative scores, the sigmoid function would only output values below 0.5 without a margin, with small gradients. Bilinear models more commonly use standard cross-entropy and set $\lambda = 0$. The other hyperparameter, $w_{\tau'}$, is either set to $w_{\tau'} = 1/|N_\tau|$ or to

some attention weight that depends on $f(\tau')$. It was indeed observed that applying a variable attention weight to negative triples, to focus on non-trivial triples with high classification error, yielded significantly better results on linear models [40]. In this setting, weight $w_{\tau'}$ is defined as the i-the element of vector $\text{softmax}([f(\tau'_1); \ldots; f(\tau'_{|N_\tau|})])$, given $\tau'_i = \tau'$.

The loss formula we give here is defined for a single positive triple. Dettmers *et al.* first noted that the loss may also be defined for several positive triples at once, e.g. all triples of the form $\langle e, r, t_1 \rangle, \ldots, \langle e, r, t_n \rangle$ [18]. We do not consider this case in our study. Some KGE models also add regularization terms to $\mathcal{L}$. The most common example is L2 regularization, which encourages embeddings to be small by adding a squared norm to the loss, as follows:

$$\mathcal{L}'(\tau) = \mathcal{L}(\tau) + \alpha\|\theta\|^2 \tag{9}$$

where α is a hyperparameter and θ is the concatenation of all trainable parameters involved in the computation of $\mathcal{L}(\tau)$. N3 regularization is a variant specifically designed for KGEs, involving cubic norms. ComplEx and ComplEx-N3 only differ in the regularization they apply during training: ComplEx applies L2 regularization by default whereas ComplEx-N3 applies N3 regularization.

3 Parameter Efficiency

It is difficult to uniformly calculate the computational cost of KGE models. As our state-of-the-art review shows, scoring functions rely on different embedding spaces ($\mathbb{R}^d$, $\mathbb{C}^d$, or $\mathbb{H}^d$, the quaternion vector space), different geometric operators (translation, rotation, scaling, reflection or more general matrix operations) and different distance functions (Euclidian or hyperbolic). Atomic operations underlying these components have no agreed-upon FLOP estimates. The only measure that is consistently reported across publications is d, the number of dimensions per entity or relation embedding. Yet, this number hides important details of the computation of plausibility scores. For instance, for the same hyperparameter d, RotatE and ComplEx would have twice as many parameters as TransE or DistMult, because they are defined in a complex embedding space, and thus require at least twice as many multiply-add operations. Because dimensionality is a poor proxy for computational cost overall, some publications also report *parameter efficiency*, defined as the minimum number of trainable parameters required for a model to reach a given performance. ConvE was shown to be 17 times more parameter-efficient than R-GCN and 8 times more parameter-efficient than DistMult for a given performance [18]. Before trying to express computational cost estimates with respect to FLOPs, we first study the parameter efficiency of KGE models.

3.1 Approach

Our objective in this preliminary study is to compare the performance of models with respect to their size. For each model, we expressed their size as an analytical expression counting their total number of parameters, as shown in Table 2. This expression depends on variables such as d (number of latent dimensions of the model) and $|E|$, $|R|$ (number of entities and relations in the KG). For instance, a TransE or DistMult model has d parameters for each entity and for each relation, giving a size of $d(|E| + |R|)$. Most non-neural KGE models only depend on these variables. The only exception is TuckER, which can have distinct embedding sizes for entities and relations: d becomes d_e for entities and d_r for relations. Neural models have more variability. Convolutional architectures (ConvKB, ConvE) also have a variable number of convolution filters denoted c and a filter size s_k. GNNs architectures typically have more than one layer. Their total number of layers is denoted L while their number of hidden layers (often with a different structure than input and output layers) is denoted m. Hidden layers have d_h dimensions. R-GCN additionally depends on b, the size of each block in block-diagonal aggregation matrices.

Table 2 Size of Knowledge Graph Embedding models ($|E|$: nb. of entities, $|R|$: nb. of relations, d, d_e, d_r: size of (entity/relation) embeddings, b: block size, d_h: size of hidden layer, c: nb. of convolution filters, s_k: size of convolutional filter, L: nb. of layers, θ_x: parameters of model x).

Model	Nb. parameters
TransE	$d(\|E\| + \|R\|)$
RotatE	$2d(\|E\| + \|R\|)$
BoxE	$d(2\|E\| + 4\|R\|)$
ExpressivE	$d(\|E\| + 6\|R\|)$
MuRE	$d(\|E\| + 2\|R\|) + \|E\|$
AttE	$d(\|E\| + 3\|R\|) + \|E\|$
MuRP	$d(\|E\| + 2\|R\|) + \|E\|$
AttH	$d(\|E\| + 3\|R\|) + \|E\| + \|R\|$
ConE	$2(d+1)\|E\| + 3d\|R\|$
DistMult	$d(\|E\| + \|R\|)$
ComplEx	$2d(\|E\| + \|R\|)$
ComplEx-N3	$2d(\|E\| + \|R\|)$
TuckER	$d_e^2 d_r + d_e\|E\| + d_r\|R\|$
QuatE	$4d(\|E\| + \|R\|)$
DualE	$8d(\|E\| + \|R\|)$
R-GCN	$d(\|E\| + bL(\|R\| + 1)) + d\|R\|$
ConvE	$d(\|E\| + \|R\|) + 2cd(d+3) + cs_k + 7d$
ConvKB	$d(\|E\| + \|R\|) + (4+d)c$
CompGCN	$d(\|E\| + \|R\|) + 4dd_h + \|\theta_{\text{ConvE}}\|$
NBFNet	$d\|R\|(Ld + L + 1) + Ld(13d+3) + d_h(2d+1) + d_h + 1$

Only few KGE models have several possible formalizations, hence possibly ambiguous parameter counts: R-GCN, RotatE, DualE and ConE. R-GCN has two variants, one that has been tested on entity classification, the other on link prediction. We only consider the link prediction variant (the one involving block-diagonal matrices) in this paper. In addition, for brevity, the R-GCN formula covers in fact a special case in which all layers have the same size, which is not mandatory in GNN architectures. The parameter counts we give here intend to mirror concrete model implementations, which are not necessarily optimal counts. RotatE and DualE respectively have $2d$ and $8d$ parameters per relation embedding but these parameters are in fact constrained, such that there are only d and $6d$ independent parameters per embedding. In the case of RotatE, the real and imaginary parts of each complex number could be reduce to a single rotation angle. The concrete implementation of RotatE still uses $2d$ parameters for each relation because rotation defined as a Hadamard product is more numerically stable (and faster) than trigonometric calculation. The same applies to DualE, in dual quaternion space. Finally, ConE may also have less parameters if KG relations were labeled as hierarchical or non-hierarchical (by an external source). Scoring indeed differs for the two types of relations. For fair comparison with other models, which are supposed to capture properties of relations through training, we chose the most general form for ConE involving a weighted sum of the two scoring functions.

Previous work already inspected the parameter efficiency of some of the models. Pavlović and Sallinger [33], and Zhu *et al.* [52] provide formulas for some of the models, which we report without changes. Other authors sometimes provide space complexity formulas [1, 13, 45]. These formulas are not accurate estimates, as they ignore constant factors. BoxE, for instance, has the

same space complexity as TransE or ExpressivE but it actually has twice as many parameters for entity embeddings. Some other authors also report parameter counts for given d, d_e and d_r values on FB15k-237 and WN18RR [18, 50, 9]. We made sure that our formulas are consistent with these counts for the two datasets.

Analytical expressions given in Table 2 are only the first step towards an evaluation of parameter efficiency. In a second step, we collected performance evalations for all models in the literature. That is, we collected hyperparameter values for d, b, c and other variables that authors chose in their experiments, together with the reported performance for their models. With this information, we can compare performance and size of a model for each experiment, where the size of the model is calculated from the corresponding analytical expression. For instance, Nguyen *et al.* evaluated TransE on FB15k-237 with $d = 100$ and reported a performance of hits@10 = 0.465 [29]. For FB15k-237, we have $|E| = 14,541$ and $|R| = 237$. The size of the TransE model in this case is $|\theta_{\text{TransE}}| = d(|E| + |R|) = 100 \times (14,541 + 237) = 1,477,800$. In another experiment, Sun *et al.* evaluated RotatE with $d = 500$ on WN18RR and reported a performance of hits@10 = 0.571 [40]. For WN18RR, we have $|E| = 40,943$ and $|R| = 11$, such that $|\theta_{\text{RotatE}}| = 2d(|E| + |R|) = 2 \times 500 \times (40,943 + 11) = 40,954,000$.

We found 26 distinct link prediction experiments for FB15k-237 and 23 experiments for WN18RR. There were two different settings for TransE and ComplEx on FB15k-237 and none for R-GCN on WN18RR. Hyperbolic models had two different settings on both datasets except ConE. All other models have a single experimental setup for each dataset. ConvKB is an special case: initial evaluation results were invalidated by later third-party experiments that uncovered an issue in the link prediction experimental protocol [41]. We consider the latter only, with lower performance for the same number of parameters. The complete list of experiments we selected, with performance and hyperparameter values, is available in Appendix B.

Across experiments, the reported performance metrics are not always identical. Some papers report Mean Reciprocal Rank (MRR) while others report Mean Rank (MR), rarely both. Hits@k, which gives the ratio of test triples whose score rank is lower than k, are commonly used as a metric as well but k may vary across papers. We found hits@10 to be the most consistently used metric for link prediction. We limit our study, as well as subsequent cost studies, to this metric. It is also worth noting that $|R|$ is not always the same across experiments. In some papers, results are given for models with $2|R|$ relation embeddings, as per a common data augmentation technique [18, 22]: for every embedding capturing a latent representation of some relation r, there is a second embedding capturing a representation of its inverse relation r^-, artificially added to the input KG.

3.2 Performance-Size Comparison

Parameter counts vs. hits@10 of all found experiments are synthesized in Figure 1. At first sight, it appears that numerous models outperformed baselines at the expense of significantly more parameters. More than half of the models have 2 to 8 times more parameters than DistMult, ComplEx, ConvKB or TransE, for both FB15k-237 and WN18RR. The only exception is NBFNet, which performs significantly better than any other model with the same number of parameters. MuRE and MuRP improve over TransE but on WN18RR, both models have 3 times more parameters. These numbers are not necessarily optimal[6] but we assume that authors have put effort in finding reasonably good hyperparameters, which allows us to compare models with a given performance. For equal performance, AttE has 1.5 times less parameters than TuckER,

[6] Authors of MuRE/MuRP report results for 200-dimensional embeddings but they do not guarantee that smaller embeddings cannot reach the same performance (with longer training cycles or different hyperparameters).

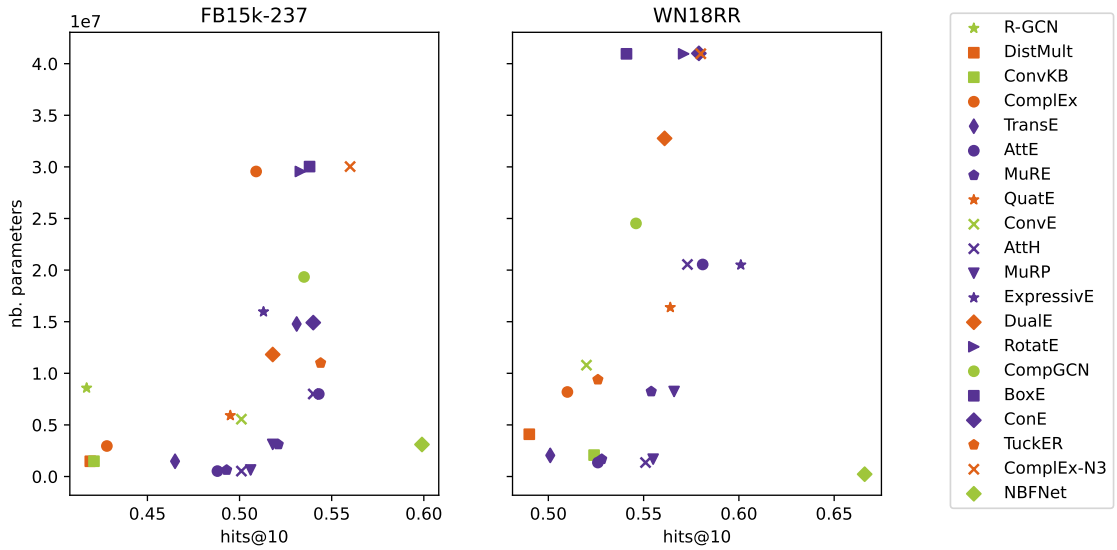


Figure 1 Model size with respect to hits@10 score (purple: linear models, orange: bilinear models, green: neural models).

2 times less than ConE and 4 times less than BoxE, for example. All these models indeed reach hits@10 close to 0.54 on FB15k-237 – the models rank the correct target entity among top 10 candidates for 54% of test triples.

This also allows us to hypothesize that there exist more parameter-efficient, non-neural models than ComplEx-N3, which performs well but is among the largest models for both datasets. It is however unlikely that any linear or bilinear can reach the performance of NBFNet with a comparable number of parameters. Both TransE and ComplEx were evaluated twice on FB15k-237, with 10 times more dimensions in the second experiment than in the first but with an absolute increase in hits@10 of only 6.6% and 8.1%. NBFNet outperforms the largest TransE and ComplEx models by another margin of 6.8% and 9.0%. On WN18RR, ExpressivE obtains good results but requires more parameters than half of the models.

By adding model size as another comparison point, Figure 1 highlights the high heterogeneity of state-of-the-art KGE models. For comparable hits@10, differences in parameter counts can be as high as 1 to 4 (between AttE and BoxE). What the figure hides, however, is that under heterogeneous model architectures, spatial complexity is a poor proxy for computational cost. GPT-3, BLOOM and LLaMa could be more legitimately compared against their size because they all rely on the Transformer architecture. The size of latent embeddings and the number of attention heads in a Transformer, which yield a given parameter count, strongly influence the number of FLOPs necessary for inference with these models. This is not true for KGE models, especially when comparing Euclidian and hyperbolic models. MuRE and MuRP, on the one hand, and AttE and AttH, on the other hand, have the same number of parameters for a similar performance but hyperbolic operators (Möbius addition, hyperbolic distance) are computationally more intensive than their Euclidian counterpart. Euclidian models could therefore have more parameters and still be more efficient than hyperbolic models. In 32 dimensions, AttH was shown to outperform AttE (by a 2.5%-5% margin on FB15k-237 and WN18RR) but no comparison was made with a higher-dimensional AttE model, with equal computational cost. The same holds for MuRE and MuRP. In the following, we go beyond parameter efficiency and compare models with respect to their computational cost.

4 Computational Cost

As already mentioned, FLOPs are a relevant intermediary between easy-to-calculate but inaccurate model sizes and environmental impact criteria such as CO_2 emissions, which typically depend on external conditions. Estimating a number of FLOPs from a mathematical expression such as $f_{\text{TransE}}(h, r, t)$ should be independent of the underlying computing platform (CPU, GPU) and concrete implementations of higher-level operations (PyTorch, TensorFlow, etc.), in such a way that the resulting number can be compared with $f_{\text{ComplEx}}(h, r, t)$ and other expressions for the same KG. This requirement imposes some constraints on what 1 FLOP is but it does not alleviate all ambiguities.

Software utilities such as `thop`[7] or `ptflops`[8] can automatically count FLOPs from model implementations in PyTorch. These utilities mostly provide coarse-grained estimates for widely used Neural Network layers (convolution, ReLU, max pooling, etc.) but not for finer-grained linear or bilinear operations in use in KGE models. They also ignore transcendental functions such as square root or exponential, as did OpenAI researchers in their 2018 study of deep learning models [4]. These functions often occur in the formalization of KGE models. Furthermore, their estimates may depend on implementation details. As an example, the following PyTorch statement, often encountered:

```
(a - b).norm(dim=-1)**2
```

induces more computation but is mathematically equivalent to:

```
((a - b)**2).sum(dim=-1)
```

where the first statement computes the squared L2-norm $(\|\mathbf{a} - \mathbf{b}\|)^2$ while the second statement directly computes $\sum_d (a_i - b_i)^2$. Ideally, the same number of FLOPs should be derived from the 2 statements – a requirement that existing utilities do not meet. In our study, we define analytical expressions in a semi-manual fashion instead. Our methodology is explained next.

4.1 Approach

In their 2018 study, OpenAI researchers restrict the definition of a FLOP to add and multiply operations – a common practice. Expressions $a + b$ and $a \times b$, where $a, b \in \mathbb{R}$ both have a cost of 1 FLOP. On that basis, they apply the following formula to calculate the cost of training neural model x [4]:

$$\begin{aligned} \kappa_x = \quad & \text{number of add-multiplies per forward pass} \\ & \times \text{ 2 FLOPs per add-multiply} \\ & \times \text{ 3 (for forward and backward pass)} \\ & \times \text{ number of examples in dataset} \\ & \times \text{ number of epochs} \end{aligned} \tag{10}$$

Add and multiply operations are commonly implemented in a single sequence of low-level instructions, referred to as a Multiply-Accumulate (MAC) operation, to limit rounding errors. However, a single MAC counts as 2 FLOPs. The FLOP count for a single inference step (or forward pass) is then multiplied by the number of examples on which the model is trained and a number of epochs. In our notation, the number of training examples is given by $|T|(1 + |N_\tau|)$, where T is the set of training triples in the KG.

[7] `https://github.com/Lyken17/pytorch-OpCounter/`
[8] `https://github.com/sovrasov/flops-counter.pytorch`

We take Definition 10 as a starting point, though ignoring the factor 3 accounting for back-propagation. This factor is indeed important when correlating FLOP count with training time on a CPU or GPU platform but in our case, we only intend to compare FLOP counts with each other. We then develop design principles to overcome the limitations of existing FLOP counting utilities: count of operations beyond MAC operations, especially transcendental function evaluation, on the one hand and algebraic reduction of equivalent operations on the other hand.

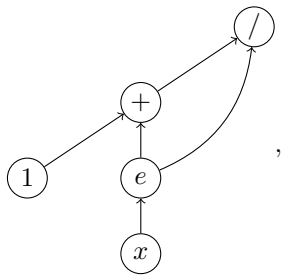
We adopt a compositional approach to FLOP counting, according to the following principles:

1. every mathematical operation on real values, including transcendental function application, costs 1 FLOP;
2. the cost of non-transcendental functions and functions on non-real values, is evaluated on their simplest analytical expression (the one with the lowest cost);
3. if a sub-expression appears in several places in the evaluated formula, it is counted only once.

From these principles, we define a FLOP count for basic expressions on scalars, vectors and matrices. The FLOP count of a given scoring function is then obtained by summing the count of all sub-expressions. It is important to keep in mind that resulting FLOP counts should not be taken as a basis to estimate execution times, which heavily depend on implementation details and on the available hardware. For instance, the function `arctanh`, rarely used in practice, has a slow PyTorch implementation on CPU. On the contrary, recurrent expressions such as $\log(1+x)$ have a dedicated, faster function in PyTorch (`log1p`). The correspondance between FLOP count and execution time is further explored in Appendix C.

We illustrate our approach with the sigmoid function. Its analytical expression, $\sigma(x) = e^x/(e^x+1)$, can be represented as a graph $G_{\sigma(x)}$ whose vertices are sub-expressions labeled either with an operator or an atomic operand. If a sub-expression appears n times in the expression, it has n outgoing edges to other vertices. Each vertex is given a FLOP count, depending on its operator and the dimension of its operands. The total cost $\kappa_{\sigma(x)}$ is the sum of FLOP counts of all vertices in $G_{\sigma(x)}$, as illustrated below:

$$G_{\sigma(x)} = \quad , \quad V_{\sigma(x)} = \{\textcircled{1}, \textcircled{x}, \textcircled{e}, \textcircled{+}, \textcircled{/}\}, \quad \kappa_{\sigma(x)} = \sum_{v \in V_{\sigma(x)}} \kappa_v \quad (11)$$



The cost of each operation $(e, +, /)$ is 1 FLOP and the cost of operands $(1, x)$ is 0 FLOP, which gives $\kappa_{\sigma(x)} = 3$. The sigmoid function can equivalently be expressed as $\sigma(x) = 1/(1+e^{-x})$, which however would give a cost of 4. In this work, we aim to provide a lower bound for FLOPs and thus keep the minimal value $\kappa_{\sigma(x)} = 3$. In addition, if the input is a d-dimensional vector instead of a scalar, the sigmoid function is mapped to each dimension, giving $\kappa_{\sigma(\mathbf{x})} = 3d$.

Table 3 summarizes FLOP counts of common KGE scoring expressions. The cost of vector operations is derived from costs on real values, accounting for the fact that sum reduction is fused with multiplication into MAC operations. For instance, the dot product of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ requires d multiplications ($x_i y_i$ for $1 \leq i \leq d$) and $d-1$ additions but we count instead d MAC operations, i.e. $2d$ FLOPs. Instead of making MACs explicit in cost calculation, we define that sum reduction of vector $\mathbf{x} \in \mathbb{R}^d$ costs d FLOPs. As a consequence, calculating the L1-norm of vector $\mathbf{x}$ has the same cost as a dot product: it requires d absolute value calculations and a sum reduction of d FLOPs. The L2-norm requires one more operation (square root). If a mathematical

Table 3 Computational cost of common operations.

	Operation	Cost
$x, y \in \mathbb{R}$	$x+y, xy, x-y, x/y, \lvert x\rvert, \max(x,y)$	1
	$\sqrt{x}, \log(x), e^x, \tanh(x), \text{arc}^*(x)$	1
	$\sigma(x) = e^x/(e^x+1)$	3
$x, y \in \mathbb{C}$	$\overline{x}$	1
	$x+y$	2
	$\lvert x\rvert$	4
	xy	6
$\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$	$\lVert\mathbf{x}\rVert_1, \mathbf{xy}$	$2d$
	$\lVert\mathbf{x}\rVert_2$	$2d+1$
	$\mathbf{x} \oplus^1 \mathbf{y}$	$16d+11$
	$\mathbf{x} \oplus^c \mathbf{y}, c \neq 1$	$16d+15$
	$d_c(\mathbf{x}, \mathbf{y})$	$19d+22$
$\mathbf{x} \in \mathbb{C}^d$	$\lVert\mathbf{x}\rVert_2$	$4d+1$
	$\lVert\mathbf{x}\rVert_1$	$5d$
$\mathbf{A} \in \mathbb{R}^{d_1 \times d_2}, x \in \mathbb{R}^{d_1}, y \in \mathbb{R}^{d_2}$	$\mathbf{xA}, \mathbf{Ay}$	$2d_1d_2$

expression includes $\|\mathbf{x}\|_2^2$, instead of giving it a cost of $2d+2$ FLOPs (cost of L2-norm plus cost of multiplication), we first reduce it to $\sum_d x_i^2$ and obtain a cost of $2d$ FLOPs. Note that in our cost model, comparison operators also cost 1 FLOP. $|x|$ and $\max(x,y)$ both cost 1 FLOP and since they are equivalent to the ternary operations $x > 0\ ?\ x : -x$ and $x > y\ ?\ x : y$, we consider the comparison to also have a cost of 1 FLOP for consistency. This further extends the usual definition of FLOP (which only considers add and multiply operations).

Operations on complex numbers are derived from operations on their real and imaginary parts. The addition of $x = a + ib$ and $y = a' + ib'$ is the complex number $x + y = (a + a') + i(b + b')$, requiring 2 additions ($a + a'$ and $b + b'$). Their multiplication is $xy = (aa' - bb') + i(ab' + ba')$, requiring 4 multiplications (aa', bb', ab' and ba'), 1 addition ($ab'+ba'$) and 1 substraction ($aa'-bb'$), for a total cost of 6 FLOPs. Similarly, addition and multiplication in $\mathbb{H}$ have a cost in FLOPs of 4 (4 additions) and 28 (4×4 multiplications and 3×4 additions). In complex and quaternion spaces, the absolute value – or norm – of a single number is defined as the Euclidian norm of their components, with a cost of 4 and 8 FLOPs respectively. When computing the Euclidian norm of a complex vector $\mathbf{x} \in \mathbb{C}^d$, it is not necessary to compute the full absolute value of its elements x_i ($1 \leq i \leq d$). The entire computation reduces to summing the square of the real and imaginary components of all elements ($2d$ FLOPs for squares, $2d$ FLOPs for sum reduction), followed by computing a square root, for a total cost of $4d + 1$ FLOPs.

Following the principles detailed above, we are able to obtain an analytical formula for the total cost of scoring for each KGE model. The result is in Table 4. In the scoring function of ComplEx, we considered the lowest possible amount of compute: since only the real part of the bilinear product is considered, scoring requires to compute the sum of four operands of the form $\mathbf{x} \odot \mathbf{y} \cdot \mathbf{z}$, where $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^{d/2}$. The total cost of ComplEx scoring is then $4 \times 3d$ per operand plus 3 FLOPs for the final sum. The scoring function of BoxE and ExpressivE includes a choice operator $\mathbf{b}\ ?\ \mathbf{x} : \mathbf{y}$, such that the output vector at index i is x_i if boolean value b_i is true or y_i otherwise. We consider that evaluating this operator costs as much as computing the values of $\mathbf{b}$, $\mathbf{x}$ and $\mathbf{y}$, which matches GPU implementations (based on vectorization), though typical CPU

Table 4 Computational cost of Knowledge Graph Embedding models (d, d_e, d_r: size of (entity/relation) embeddings, b: block size, d_h: size of hidden layer, c: nb. of convolution filters, s_k: size of convolutional filter, L: nb. of layers, B: avg. nb. of neighbors, S: batch size, κ_x: computational cost of model x).

Model	Nb. operations (score)
TransE	$4d + 1$
RotatE	$10d + 1$
BoxE	$44d + 2$
ExpressivE	$34d + 1$
MuRE	$5d + 3$
AttE	$29d + 13$
MuRP	$46d + 42$
AttH	$59d + 42$
ConE	$188d + 113$
DistMult	$3d$
ComplEx	$12d + 3$
ComplEx-N3	$12d + 3$
TuckER	$2(d_e^2 d_r + d_e d_r + d_e)$
QuatE	$48d + 3$
DualE	$128d$
R-GCN	$d(2bL(B + 1) + 2L + 3)$
ConvE	$2cd(2s_k + 2d + 4S + 7) + d(12S + 22)$
ConvKB	$7dc$
CompGCN	$d_h(2d(B + 2) + 4S + 7) + d(d + 1) + \kappa_{\text{ConvE}}$
NBFNet	$L(Bd(2d + 7) + d(26d + 4S + 19) + 4) + d_h(4d + 3) + 2$

implementations would require less FLOPs. Neural models have ReLU operators, which have a cost of 1 per dimension, given that $\text{ReLU}(x) = \max(0, x)$. To finish, it is worth noting that variable B, in Table 4, is not consistent across GNNs: for R-GCN and CompGCN, it represents the average number of neighbors, regardless of link direction whereas for NBFNet, only neighbors with *incoming* links are counted.

It is easy to see in the table that dimensionality alone is not a good proxy for the computational cost of KGE models. If one fixes d across models as is done in other large-scale studies [35, 2], there still is a high disparity in terms of FLOPs. In particular, the FLOP count of TuckER and all neural models grows quadratically with d.

4.2 Cost of Inference

Following the same procedure as for model size, we assign hyperparameter values to variables of Table 4 for all 49 experiments found in the literature, in order to compare performance and FLOP count estimates. Our overall objective is to compare models on the entire training procedure, which does not only depend on the computational cost of scoring but also on the size of the input KG and a number of epochs. However, we first study the cost of scoring alone. As mentioned in introduction, some models may be relatively cheap to train but expensive to use at inference time, i.e. when scoring arbitrary triples. Once an AI model is deployed in real-world systems, its inference cost is likely to outweigh training cost, even if the model is retrained on a regular basis [48]. This observation motivates a separate evaluation of KGE models on the cost of scoring.

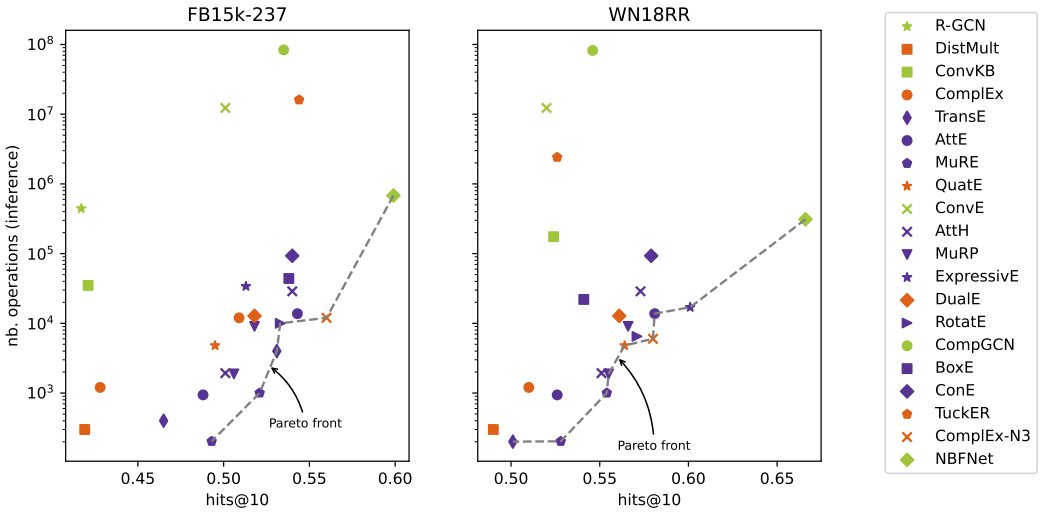


Figure 2 Computational cost of inference with respect to hits@10 score (purple: linear models, orange: bilinear models, green: neural models).

FLOP counts for scoring are shown in Figure 2, for FB15k-237 and WN18RR. These results allow to draw conclusions on the relative cost of neural models, on the efficiency of simple models (TransE, ComplEx) and on the cost of hyperbolic models. We comment each aspect in turn.

The figure highlights again the heterogeneity of KGE models, as does Figure 1 on parameter counts. However, we see significant differences across model families. All neural models are in the upper half of the figure (above 50k FLOPs), for both datasets and conversely, all linear or bilinear models but one are in the lower half. The high inference cost of neural models seems not to bring particular benefits: besides NBFNet (which outperforms all other models), neural models have competitors that are both more accurate and more efficient. The same applies to TuckER, the only non-neural model having an inference cost above 50k FLOPs. The main difference between neural models and TuckER is that the high cost of TuckER is highly correlated with its number of parameters, having a spatial complexity in $O(d_e^2 d_r)$, whereas neural models generate many FLOPs from few parameters. The computational overhead of GNNs in particular may be the price to pay for higher inductive abilities, as proven empirically on the inductive link prediction task [52]. The case of NBFNet is also interesting as it improves performance at the expense of a significantly higher inference cost compared to ComplEx-N3 and ExpressivE, the second best performing models on FB15k-237 and WN18RR. We come back to this observation in the next section about training cost.

TransE and MuRE, the simplest linear models, appear to be near-optimal: they reach a slightly lower performance than other linear models but with a much lower inference cost. Interestingly, TransE remains efficient in high dimensions, reaching a hits@10 of 0.53 for a cost of 4k FLOPs on FB15k-237[9]. MuRE appears even closer to optimality, although available data is too sparse to come to conclusions. A similar comment can be made with the bilinear ComplEx model, which also remains efficient in high dimensions, though to a lesser extent. What is worth noting about ComplEx is that its N3 variant, with the same number of parameters and the same inference cost, is able to outperform ComplEx by a large margin (again on FB15k-237). In the next section, we show that it is at the expense of a higher training cost.

[9] The lower-dimensional TransE model for FB15k-237 was not trained with self-adversarial sampling, which could have further improved performance.

Once we add the computation criterion to link prediction, hyperbolic models do not seem to stand out. ConE has a clearly higher inference cost than any other linear model, despite good results. Other hyperbolic models are also an order of magniture more costly than their Euclidian couterpart. The inference cost of MuRP, in particular, is 9 times higher than the inference cost of MuRE. What the figure also shows is the domination of MuRE over hyperbolic models: the high-dimensional MuRE (200 dimensions) beats the lower-dimensional AttH and MuRP (32 dimensions) both in performance and efficiency. Balažević *et al.* and Chami *et al.* independently give evidence that beyond 100 dimensions, the superiority of hyperbolic operators vanishes [8, 13]. That is, for a given computational budget, it may generally hold that Euclidian models are more accurate than hyperbolic models. We further evaluate KGE models in fixed-budget experiments in Section 5, to answer this question.

4.3 Cost of Training

The observations we have made so far are on the computational cost of inference. They may not necessarily hold for training. For instance, a high-dimensional Euclidian model may be harder to train than low-dimensional hyperbolic models. That is why we now look at training cost, by considering a further variable in Formula 10: the number of examples in the training dataset. Across experiments, this value varies greatly because of the N_τ hyperparameter, in the definition of $\mathcal{L}(\tau)$ (Formula 8). Early experiments set $|N_\tau| = 1$, others set it to 10, 50 or even 1,024 for RotatE [40]. Moreover, some models are trained with a slightly different loss formulation that takes all entities into account in a single pass, starting with Dettmers *et al.* [18, 22, 9, 45]. Instead of calculating the loss term from $|N_\tau| + 1$ triples ($|N_\tau|$ negative triples for one positive triple), the alternative is to calculate it from $|E|$ triples and consider the link prediction problem as multi-class classification. We approximate this configuration by setting $|N_\tau| = |E| - B$. Knowing that $|E| = 14,541$ for FB15k-237 and $|E| = 40,943$ for WN18RR, the way negative triples are generated has a significant impact on the total number of FLOPs for training.

To have a cost for the entire training procedure, the number of epochs must also be taken into account (Formula 10, p. 12). In this paper, we consider the cost of a single epoch. Most papers do not report an exact number of epochs for their experiments. Some papers report a maximum number of epochs but it is common to train with early stopping: every x epochs, the model is evaluated on a validation set and if some metric reaches a predefined threshold, training stops immediately. The actual number of training epochs could be much lower than the reported maximum. ComplEx-N3, for instance, was "trained for 100 epochs to ensure convergence" but "reported performances were reached within the first 25 epochs" [22]. In the absence of any more precise data, we calculate the cost of computing $\mathcal{L}(\tau)$ for a *single* triple τ. When we compare KGE models on such a cost, we implicitly assume that stochastic gradient descent has the same convergence rate across models, which is often not true in practice. However, if the single-triple training cost of two models differs by several orders of magnitude, we can safely consider that their true training cost also differs significantly.

Moreover, our estimates ignore entirely any fine-tuning that has taken place to ensure optimality. In the domain of language models for instance, systematic architecture search on a Transformer model required more than 3,000 repeated training phases to improve results [39]. We exclude these considerations here.

The cost of computing $\mathcal{L}(\tau)$, as defined on p. 7 (Formula 8), is $(|N_\tau| + 1)(\kappa_{f(\tau)} + 5)$ in the case $w_{\tau'}$ is a constant, where $\kappa_{f(\tau)}$ is the cost of scoring a single triple. Computing softmax in the self-adversarial setting costs $3|N_\tau|$ FLOPs. We can see in Figure 2 that $\kappa_{f(\tau)} \gg 3$ (TransE and MuRE have the lowest inference cost at 200 FLOPs), which makes the computation of softmax rather cheap compared to standard cross-entropy loss. We choose to ignore softmax in

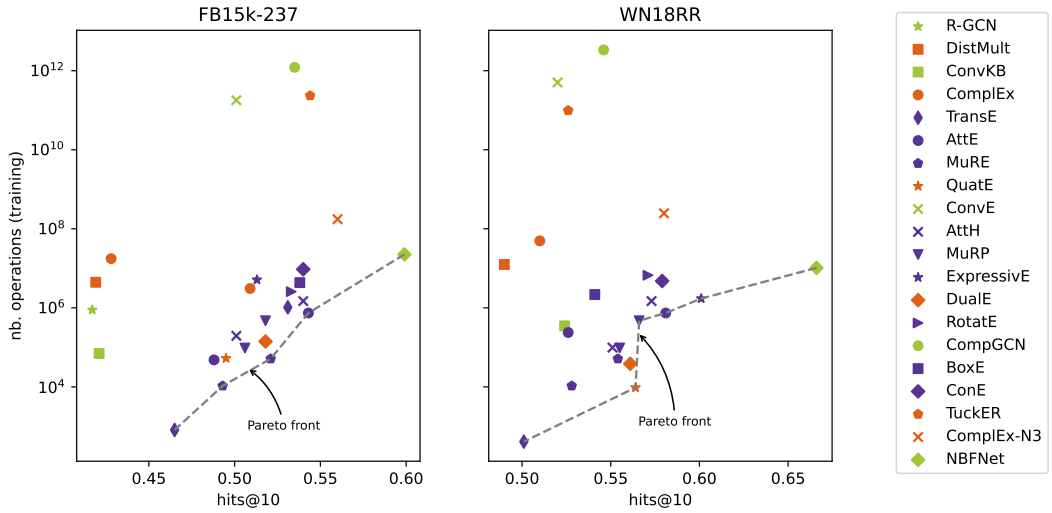


Figure 3 Computational cost of training (per triple) with respect to hits@10 score (purple: linear models, orange: bilinear models, green: neural models).

our calculation. The L2 and N3 regularizers of ComplEx, introduced on p. 8 (Formula 9), can be ignored with a similar argument. Their cost is respectively of $12d$ and $15d$ FLOPs, of the same order of magnitude as $\kappa_{\text{ComplEx}} = 12d + 3$, but ComplEx has only been trained in configurations generating a high number of negative triples. These two observations allow us to reduce the calculation of training cost to $\kappa_{\mathcal{L}(\tau)}$, with marginal estimation error.

Once we define the per-experiment hyperparameter $|N_\tau|$, we have an estimate of the total number of FLOPs for training, which we can then put in perspective with the performance of a model. Results are given in Figure 3, which we can compare to the previous figure on inference cost. As an example, it seems that R-GCN and NBFNet can be efficiently trained despite high inference cost. R-GCN is at the level of AttE (700-800k FLOPs) and NBFNet requires no more than twice as many FLOPs as ConE, the closest linear model. However, no experiment in the literature indicates whether NBFNet can beat linear models at a given training cost. The cost reported here is already for low 32-dimensional embeddings, with few negative triples ($|N_\tau| = 32$). Overall, the training cost of GNN models (R-GCN, NBFNet and CompGCN, in our study) seem not to follow any clear pattern. R-GCN and NBFNet have respectively the lowest and highest hits@10 for a moderate increase in computational cost ($\times 22$ increase for FB15k-237, to compare with the $\times 1,000$ increase between the two TransE experiments on this dataset) while CompGCN has the highest training cost but a rather low performance. GNNs deserve further evaluations on the link prediction task.

The general behavior of bilinear models is also unclear. DualE, for instance, seems relatively efficient when looking at training cost: it requires more FLOPs for scoring than most linear models but its overall training procedure was more effective than those same linear models. QuatE, on WN18RR, is already more efficient at inference time but the gap with linear models similarly increases for training. In contrast to these two models, most experiments on ComplEx and ComplEx-N3 give high training costs for a relatively low inference cost. One exception, published by Sun *et al.* [40], gives good results for ComplEx with 3M FLOPs for training, comparable to the cost of RotatE and ExpressivE. At first sight, it might suggest that other ComplEx and ComplEx-N3 experiments sampled an unnecessarily large number of negative triples. However, comparing the two models at equal dimensions gives a more contrasted picture. On FB15k-237,

ComplEx and ComplEx-N3 were both evaluated with 1,000 dimensions. Given our formalization and simplifications, the only difference between their cost estimates is the number of negative triples they use. For ComplEx, $|N_\tau| = 256$ while for ComplEx-N3, which uses the multi-class classification setting, $|N_\tau| = 14540$. If performance gains come solely from more negative triples, these are not superfluous and ComplEx necessarily comes with a high computational cost. If instead, performance gains come from N3 regularization, further experiments with self-adversarial sampling should be performed to quantify the optimal cost of ComplEx-N3.

5 Fixed-Budget Performance

Evaluations of KGE models found in the literature leave many questions open, including: can the training cost of NBFNet be reduced, is MuRE on the Pareto front of performance/cost criteria, are hyperbolic model intrinsically more expressive than higher-dimensional Euclidian model and what is the true impact of N3 regularization? Some models have already been compared with equal dimensions or equal parameter count but to answer this type of questions, they should be compared with equal *computational budget*. We now provide experiments where model hyperparameters are being set in such a way that all models have the same training cost (as per our approach).

5.1 Approach

According to Figure 3, only two models have a cost below 10k FLOPs (TransE, QuatE). We take this value as the maximum computational budget in our experiments, which gives an upper bound to d, $|N_\tau|$ and other variables of Table 4. Let us first set $|N_\tau| = 1$ and take the example of TransE. We have $\kappa_{\mathcal{L}(\tau)} = (|N_\tau| + 1)(\kappa_{f_{\text{TransE}}(\tau)} + 5) = 2((4d + 1) + 5) = 8d + 12$. Setting $\kappa_{\mathcal{L}(\tau)} = 10,000$, the maximum number of dimensions for TransE is $d = \lfloor (10,000 - 12)/8 \rfloor = 1248$. If instead, we take ComplEx, we have $\kappa_{\mathcal{L}(\tau)} = 2((12d + 3) + 5) = 24d + 16$ and $d = \lfloor (10,000 - 16)/24 \rfloor = 416$. Of course, instead of limiting the number of negative triples, we can also set an arbitrary number of dimensions, e.g. $d = 100$, and adjust $|N_\tau|$ so that a smaller model is trained on more data. For a 100-dimensional TransE, the maximum number of negative triples would be $|N_\tau| = \lfloor (10,000/406) - 1 \rfloor = 23$. For ComplEx, it would be $|N_\tau| = \lfloor (10,000/1208) - 1 \rfloor = 7$. We consider both configurations in our experiments, as they play different roles. When setting the same number of negative triples across models, we can compare models with each other and assess their efficiency – up to inherent limitations in expressivity of each model. When setting a fixed, arbitrary number of dimensions, we can compare a model with itself, showing how the dimensionality of the KG (through negative sampling) also influences training.

The cost estimate of neural models involves other free variables: b, d_h, c, s_k, L and S. We aim to assign them to the lowest possible values, such as $b = 2$, $L = 2$, $c = 2$ or $c = 4$, $S = 128$. It happens that CompGCN, NBFNet and ConvE have no satisfactory variable assignment that keeps the cost of training under 10k FLOPs. CompGCN and NBFNet have hidden layers with dimensions d_h but in practice, all experiments set $d_h = 2d$ for these models. The convolutional filter of ConvE must have a minimal size $s_k = 3 \times 3$. With these two values, even with a single negative triple, scoring is too costly. The only neural models that can be evaluated with a budget of 10k FLOPs are R-GCN and ConvKB. ConvKB embeddings depend on a number of convolution filters, c. If $c = 1$, the formalization of the model is analogous to a TransE-like linear model, which encourages to find instead a value above 1. We set $c = 4$, which is the largest value such that $d > 100$ in the configuration where $|N_\tau| > 1$. In the original experiments, ConvKB models have 50 to 500 filters [29]. R-GCN experiments, on the other hand, have set $b = 5$ and $L = 2$. With such a configuration, R-GCN would also be over the 10k computational budget. We decided to reduce c and let $L \in \{1, 2\}$. In fact, already if $c = 1$, R-GCN must have low-dimensional embeddings to remain under 10k FLOPs ($d < 32$). We therefore take $c = 1$ in all R-GCN experiments.

For each model, we have at least two configurations: one has $|N_\tau| = 1$, the others have $|N_\tau| > 1$. R-GCN has 4 configurations per dataset, as we also evaluate the influence of L. MuRP and AttH have 3 configurations, to better evaluate the impact of negative triples on hyperbolic models. All configurations are summarized in Table 5. For configurations with multiple negative triples, d has a maximum value of 100. If there can be no configuration, for a given model, such that $d = 100$ and $|N_\tau| > 1$, we set $|N_\tau| = 2$ or $|N_\tau| = 4$ and find the highest possible value for $d \in [32, 100]$. If no assignment for d exists in this interval, we discard the model (which applies to ConE, TuckER and DualE).

In total, we obtain 64 distinct fixed-budget experiments. We set $S = 128$, a common value across experiments from the literature (43% of experiments listed in Tables 6 and 7). Linear models were trained with $\lambda = 3$, also common for small dimensions; other models with $\lambda = 0$. Any other hyperparameter was set manually. In particular, we set a learning rate of 10^{-3} and run training for 50 epochs. Note that despite our effort to find suitable values for model-specific hyperparameters, we do not strictly guarantee that the reported performance is the best possible performance with a 10k computational budget.

For our evaluation, we used different KGE libraries, all based on PyTorch. The PyKEEN library[10], the most up-to-date KGE library [3], implements 11 of the 20 models we review in this paper (8 included in this section's experiments). To evaluate AttE, AttH and ComplEx-N3, we used the KGEmb library[11], which specifically targets hyperbolic KGE models [13]. For the remaining models, ExpressivE and MuRP, we used original implementations provided respectively by Pavlović[12] and Balažević[13]. We have made configuration scripts available to install these libraries and reproduce our fixed-budget experiments[14].

5.2 Model Performance

Table 5 gives the performance of KGE models along two metrics: hits@10 and Mean Rank (MR, lower is better). On both metrics, results show a clear advantage for linear models. With few exceptions, all linear models outperform any of the bilinear or neural models, by a large margin. Among linear models, the only exceptions are TransE, which performs poorly with a single negative triple, and RotatE, which is beaten by R-GCN on a single metric (hits@10 on FB15k-237). Conversely, QuatE performs relatively well on WN18RR, even if its MR is worse than the average MR among linear models. ComplEx-N3 and DistMult are the worst performing models, regardless of the configuration. Our results suggest that the N3 regularization scheme imposes a high computational cost, contrary to the classical L2 regularization used on ComplEx. It might however also be that better-suited hyperparameters yield good results for ComplEx-N3.

If we focus on linear models only, we can make two observations. First, despite a certain diversity in their parameterization, linear models have relatively close results. Hits@10 has a standard deviation below 0.05 on both datasets, if we exclude the first TransE configuration. Notably, MuRE and AttE have very close results despite 6 times more parameters for MuRE (with $|N_\tau| = 1$). Second, hyperbolic models (MuRP, AttH) are consistently outperformed by their Euclidian counterpart, even though the margin reduces as the training set includes more negative triples. In other words, simple linear models, such as MuRE or RotatE (on WN18RR only), appear to be able to compensate the richer geometric operations of other models simply with more dimensions – up to a certain number of dimensions.

[10] https://github.com/pykeen/pykeen
[11] https://github.com/HazyResearch/KGEmb/
[12] https://github.com/AleksVap/ExpressivE
[13] https://github.com/ibalazevic/multirelational-poincare
[14] https://gitlab.emse.fr/victor.charpenay1/kge-cost-experiments

Chami *et al.* report that MuRP and AttH achieve 0.544 and 0.551 hits@10 on WN18RR in their 32-dimensional version (ConE achieves 0.537 hits@10 under the same conditions). These results are well above those we find with more dimensions but they hold for $|N_\tau| = 50$. The computational cost of training the 2 models in this setting is 77,469 and 98,685 FLOPs – well beyond the 10k budget. We may find comparable results if we further reduce d to increase $|N_\tau|$, though. Going from 1 negative triple to 2 and 4 negative triples does indeed improve the performance of MuRP and AttH on FB15k-237 but not on WN18RR. Only BoxE follows the same pattern. ConvKB has a lower performance on both datasets; R-GCN has contrasted results if we compare $|N_\tau| = 1$ and $|N_\tau| = 2$. All other models improve if they are trained with more negative triples. MuRE, for instance, improves its hits@10 by 5.8% with more than 10 negative triples. This configuration is in fact the best overall: it is only beaten by RotatE with respect to hits@10 on WN18RR. A recent re-evaluation of MuRE, although with a different objective, arrives at a similar conclusion: with equal training conditions, MuRE are RotatE are among the best-performing models [15].

In the light of fixed-budget experiments, we can identify strengths and weaknesses of each of the KGE model families:

- Euclidian models are the most efficient models: they achieve good results with a low computational budget. MuRE appears to be the most efficient model overall. It deserves further experiments in high dimensions to better understand its potential limitations in terms of expressivity.
- Hyperbolic models display no performance gain for a fixed low budget. As the literature points that the hyperbolic operators offer no advantages in high dimensions, these models are likely to have a limited impact in downstream tasks. In contrast though, there is no theoretical work on how high-dimensional Euclidian models can efficiently capture hierarchies (as in WN18RR). Our results encourage further work in this area.
- Bilinear models require more computation than linear models, with no obvious gain in expressivity. This result questions the relevance of using ComplEx in other KGE learning tasks, such as query answering [5]; high-dimensional linear models might prove better suited. The performance of QuatE on FB15k-237 could still be further studied, in case hyperparameter tuning alone would improve its performance. The same applies to N3 regularization.
- Non-linearities, as introduced in ConvKB, R-GCN and other neural models, appear to be ineffective for low computational budgets. It is however still unclear whether they are necessary to increase the expressivity of KGEs. Interestingly, to match the (high) inference cost of NBFNet, MuRE could have more than 50,000 dimensions, i.e. more than the number of entities in the input KG. With such a large embedding space, it would most likely be more efficient to consider non-parametric approaches instead, along the same line as AnyBURL.

6 Conclusion

The main motivation of this paper was to quantify the computational cost of the numerous KGE models already published and to identify research directions that would take into account the *efficiency* of learning on KGs. To this end, we designed a simple cost model that estimates the number of FLOPs each KGE model would require for scoring and training, and compared the resulting estimates over 49 experiments. The first conclusion to draw from this analysis is that the efficiency of KGE models greatly varies across experiments. Most bilinear and neural models meet the performance of linear models only at the expense of a higher computational cost. In particular, state-of-the-art results for ComplEx and ConvE have only been given in a setting exposing much more data to the model than linear models – because of negative sampling. Fixed-budget experiments suggest that, when given an equal number of negative triples, bilinear

and neural models do not compete with linear models. Future research should either focus on expanding linear models or finding an equivalent formulation of ComplEx and QuatE scoring as a linear fonction.

On the other hand, linear models (TransE, RotatE, BoxE, etc.) have known limitations with respect to their ability to embed ontological knowledge and rules, regardless of their dimensionality. The exceptional performance of NBFNet suggest that GNN architectures may have richer inductive abilities, despite higher computational costs. It is however a still open question whether message passing and non-linearities are necessary. For instance, MuRE, which stands out as both accurate and efficient for transductive link prediction, has not been tested on other KG learning tasks such as inductive link prediction, complex query answering or ontology embedding. Interestingly, MuRE itself was not the main subject of its original publication (MuRP was) but fixed-budget experiments highlight its superior efficiency for low computational budgets. A higher-dimensional MuRE model (beyond 200 dimensions) might as well surpass GNNs if properly trained.

Analyzing the computational cost of KGE models, as we do in this paper, is an important research objective as we anticipate that large KGs will be commonly used through pre-trained embeddings. Training embeddings for the 100M+ entities of Wikidata, for example, would have a significant cost; minimizing this cost then becomes crucial. It is worth noting, still, that our cost model and experiments have several limitations. First, FLOP estimates are only loosely correlated with execution time and energy consumption. Typically, this means that ComplEx and MuRE can be trained on similar computing platforms, despite significant differences in FLOPs. If the two models were compared on a more elaborate cost model, also taking spatial complexity into account, results might be more contrasted than what we expose in the paper. In addition, the fixed-budget experiments we conducted only explore sparse configurations. A more thorough study to find scaling laws, relating dataset size and cost or cost and hits@10, would provide more insights as to what models are most efficient. We aim to cover such scaling experiments in future work.

References

1 Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. BoxE: A Box Embedding Model for Knowledge Base Completion. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/6dbbe6abe5f14af882ff977fc3f35501-Abstract.html.

2 Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. Bringing Light Into the Dark: A Large-Scale Evaluation of Knowledge Graph Embedding Models Under a Unified Framework. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(12):8825–8845, 2022. doi:10.1109/TPAMI.2021.3124805.

3 Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021. URL: http://jmlr.org/papers/v22/20-825.html.

4 Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. AI and Compute, 2018. URL: https://openai.com/research/ai-and-compute.

5 Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. Complex query answering with neural link predictors. In *International Conference on Learning Representations*, 2021. URL: https://openreview.net/forum?id=Mos9F9kDwkz.

6 Yushi Bai, Zhitao Ying, Hongyu Ren, and Jure Leskovec. Modeling Heterogeneous Hierarchies with Relation-specific Hyperbolic Cones. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 12316–12327, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/662a2e96162905620397b19c9d249781-Abstract.html.

7 Ivana Balažević, Carl Allen, and Timothy M. Hospedales. Hypernetwork knowledge graph em-

beddings. In Igor V. Tetko, Vera Kurková, Pavel Karpov, and Fabian J. Theis, editors, *Artificial Neural Networks and Machine Learning - ICANN 2019 - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings - Workshop and Special Sessions*, volume 11731 of *Lecture Notes in Computer Science*, pages 553–565. Springer, 2019. doi:10.1007/978-3-030-30493-5_52.

8 Ivana Balažević, Carl Allen, and Timothy M. Hospedales. Multi-relational Poincaré Graph Embeddings. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4465–4475, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/f8b932c70d0b2e6bf071729a4fa68dfc-Abstract.html.

9 Ivana Balažević, Carl Allen, and Timothy M. Hospedales. TuckER: Tensor Factorization for Knowledge Graph Completion. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 5184–5193. Association for Computational Linguistics, 2019. doi:10.18653/v1/D19-1522.

10 Zhen Bi, Siyuan Cheng, Jing Chen, Xiaozhuan Liang, Feiyu Xiong, and Ningyu Zhang. Relphormer: Relational graph transformer for knowledge graph representations. *Neurocomputing*, 566:127044, 2024. doi:10.1016/J.NEUCOM.2023.127044.

11 Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-relational Data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2787–2795, 2013. URL: https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html.

12 Zongsheng Cao, Qianqian Xu, Zhiyong Yang, Xiaochun Cao, and Qingming Huang. Dual Quaternion Knowledge Graph Embeddings. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 6894–6902. AAAI Press, 2021. doi:10.1609/AAAI.V35I8.16850.

13 Ines Chami, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. Low-Dimensional Hyperbolic Knowledge Graph Embeddings. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6901–6914. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.acl-main.617.

14 Victor Charpenay. KGE Cost Experiments. Software, swhId: swh:1:dir:29d727e535031eca0583c489c67c47d0b9165d30 (visited on 2026-03-27), URL: https://gitlab.emse.fr/victor.charpenay1/kge-cost-experiments/. doi:10.4230/artifacts.25692.

15 Victor Charpenay and Steven Schockaert. Less is MuRE: Revisiting shallow knowledge graph embeddings. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, November 2025. doi:10.18653/V1/2025.EMNLP-MAIN.779.

16 Sanxing Chen, Xiaodong Liu, Jianfeng Gao, Jian Jiao, Ruofei Zhang, and Yangfeng Ji. HittER: Hierarchical transformers for knowledge graph embeddings. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10395–10407, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.emnlp-main.812.

17 Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024. URL: http://jmlr.org/papers/v25/23-0870.html.

18 Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2D Knowledge Graph Embeddings. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1811–1818. AAAI Press, 2018. doi:10.1609/AAAI.V32I1.11573.

19 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019,*

Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4171–4186. Association for Computational Linguistics, 2019. doi:10.18653/V1/N19-1423.

20 Jiabang He, Liu Jia, Lei Wang, Xiyao Li, and Xing Xu. MoCoSA: Momentum Contrast for Knowledge Graph Completion with Structure-Augmented Pre-trained Language Models. *CoRR*, abs/2308.08204, 2023. arXiv: 2308.08204. doi:10.48550/arXiv.2308.08204.

21 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. *Knowledge Graphs*. Number 22 in Synthesis Lectures on Data, Semantics, and Knowledge. Springer, 2021. doi:10.2200/S01125ED1V01Y202109DSK022.

22 Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical Tensor Decomposition for Knowledge Base Completion. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2869–2878. PMLR, 2018. URL: http://proceedings.mlr.press/v80/lacroix18a.html.

23 Da Li, Ming Yi, and Yukai He. LP-BERT: Multi-task Pre-training Knowledge Graph BERT for Link Prediction. *CoRR*, abs/2201.04843, 2022. arXiv: 2201.04843. arXiv:2201.04843.

24 Haotian Li, Bin Yu, Yuliang Wei, Kai Wang, Richard Yi Da Xu, and Bailing Wang. Kermit: Knowledge graph completion of enhanced relation modeling with inverse transformation. *Knowledge-Based Systems*, 324:113500, 2025. doi:10.1016/j.knosys.2025.113500.

25 Yunshui Li, Junhao Liu, Min Yang, and Chengming Li. Self-distillation with meta learning for knowledge graph completion. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 2048–2054. Association for Computational Linguistics, 2022. doi:10.18653/V1/2022.FINDINGS-EMNLP.149.

26 Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2181–2187. AAAI Press, 2015. doi:10.1609/AAAI.V29I1.9491.

27 Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of BLOOM, a 176B parameter language model. *Journal of Machine Learning Research*, 24(253):1–15, 2023. URL: http://jmlr.org/papers/v24/23-0069.html.

28 Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 3137–3143. ijcai.org, 2019. doi:10.24963/ijcai.2019/435.

29 Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Q. Phung. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 327–333. Association for Computational Linguistics, 2018. doi:10.18653/v1/n18-2053.

30 Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proc. IEEE*, 104(1):11–33, 2016. doi:10.1109/JPROC.2015.2483592.

31 Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 809–816. Omnipress, 2011. URL: https://icml.cc/2011/papers/438_icmlpaper.pdf.

32 Simon Ott, Christian Meilicke, and Matthias Samwald. SAFRAN: An interpretable, rule-based link prediction method outperforming embedding models. In Danqi Chen, Jonathan Berant, Andrew McCallum, and Sameer Singh, editors, *3rd Conference on Automated Knowledge Base Construction, AKBC 2021, Virtual, October 4-8, 2021*, 2021. doi:10.24432/C5MK57.

33 Aleksandar Pavlovic and Emanuel Sallinger. ExpressivE: A Spatio-Functional Embedding For Knowledge Graph Completion. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL: https://openreview.net/pdf?id=xkev3_np08z.

34 Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Trans. Knowl. Discov. Data*, 15(2):14:1–14:49, 2021. doi:10.1145/3424672.

35 Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You CAN teach an old dog new tricks! on training knowledge graph embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=BkxSmlBFvr.

36 Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang,

Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. BLOOM: A 176B-parameter open-access multilingual language model. *CoRR*, abs/2211.05100, 2022. doi:10.48550/arXiv.2211.05100.

37 Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018. doi:10.1007/978-3-319-93417-4_38.

38 Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020. doi:10.1145/3381831.

39 Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and Policy Considerations for Deep Learning in NLP. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3645–3650. Association for Computational Linguistics, 2019. doi:10.18653/v1/p19-1355.

40 Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HkgEQnRqYQ.

41 Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha P. Talukdar, and Yiming Yang. A Re-evaluation of Knowledge Graph Completion Methods. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 5516–5522. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.acl-main.489.

42 Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In Alexandre Allauzen, Edward Grefenstette, Karl Moritz Hermann, Hugo Larochelle, and Scott Wen-tau Yih, editors, *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality, CVSC 2015, Beijing, China, July 26-31, 2015*, pages 57–66. Association for Computational Linguistics, 2015. doi:10.18653/v1/W15-4007.

43 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023. doi:10.48550/arXiv.2302.13971.

44 Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex Embeddings for Simple Link Prediction. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2071–2080. JMLR.org, 2016. URL: http://proceedings.mlr.press/v48/trouillon16.html.

45 Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=BylA_C4tPr.

46 Feiyang Wang, Zhongbao Zhang, Li Sun, Junda Ye, and Yang Yan. DiriE: Knowledge graph embedding with Dirichlet distribution. In Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 3082–3091. ACM, 2022. doi:10.1145/3485447.3512028.

47 Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Trans. Knowl. Data Eng.*, 29(12):2724–2743, 2017. doi:10.1109/TKDE.2017.2754499.

48 Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga Behram, Jinshi Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin S. Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim M. Hazelwood. Sustainable AI: environmental implications, challenges and opportunities. In Diana Marculescu, Yuejie Chi, and Carole-Jean Wu, editors, *Proceedings of the Fifth Conference on Machine Learning and Systems, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022. URL: https://proceedings.mlsys.org/paper_files/paper/2022/hash/462211f67c7d858f663355eff93b745e-Abstract.html.

49 Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May*

7-9, 2015, Conference Track Proceedings, 2015. URL: http://arxiv.org/abs/1412.6575.

50 Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion Knowledge Graph Embeddings. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2731–2741, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/d961e9f236177d65d21100592edb0769-Abstract.html.

51 Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3065–3072. AAAI Press, 2020. doi:10.1609/AAAI.V34I03.5701.

52 Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural Bellman-Ford networks: A general graph neural network framework for link prediction. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 29476–29490. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/f6a673f09493afcd8b129a0bcf1cd5bc-Paper.pdf.

A Cost of Pre-Trained Language Models

We use the calculation method first elaborated by OpenAI researchers and then applied on the Transformer architecture by EpochAI researchers[15]. A Transformer layer is mainly composed of two elements: a multi-head attention network (MHA) and a feed-forward network (FFN). The number of FLOPs of MHA, for a single token, is given by EpochAI as:

$$\kappa_{MHA} = 2H(d_i(2d_k + d_h) + (d_k + d_h) + d_h d_o) \tag{12}$$

where H is the number of heads and d_i, d_k, d_h, d_o are respectively the dimension of the input token embedding, of the key embedding, of the hidden vector and of the output vector of the MHA unit.

The number of FLOPs of a FFN with two layers, as is typical in Transformers, is:

$$\kappa_{FFN} = 2d_o d'_h + 2d'_h d_o \tag{13}$$

where d'_h is the dimension of the network's hidden vector.

The hyperparameters given by Devlin *et al.* for BERT base are the following [19]: $H = 12, d_o = 768, d'_h = 4 \times 768 = 3,072$. Values for other dimensions are not explicitly given but we assume $d_i = d_k = d_h = 64$, as in the original Transformer. As a result, we have $\kappa_{MHA} = 1.5 \times 10^6$ and $\kappa_{MHA} = 9.4 \times 10^6$ for a single BERT layer. BERT base having 12 layers, we have $\kappa_{BERT} = 12 \times (1.5 + 9.4) \times 10^6 = 1.3 \times 10^8$.

B Experiment Hyperparameters

We list all experiments with known hits@10 and hyperparameters in Tables 6 (for FB15k-237) and 7 (for WN18RR). We used these values to produce Figures 1, 2 and 3.

C Execution Time of Atomic Operations

We use execution time as a proxy for the number of FLOPs of atomic operations to evaluate the practical relevance of our estimates (Section 4.2). We report values for computation on a CPU and on a GPU, respectively on Figure 4 and Figure 5. Times have been measured for 100 iterations over 1,000-dimensional random vectors in both cases.

[15] https://epoch.ai/blog/estimating-training-compute#example-transformer

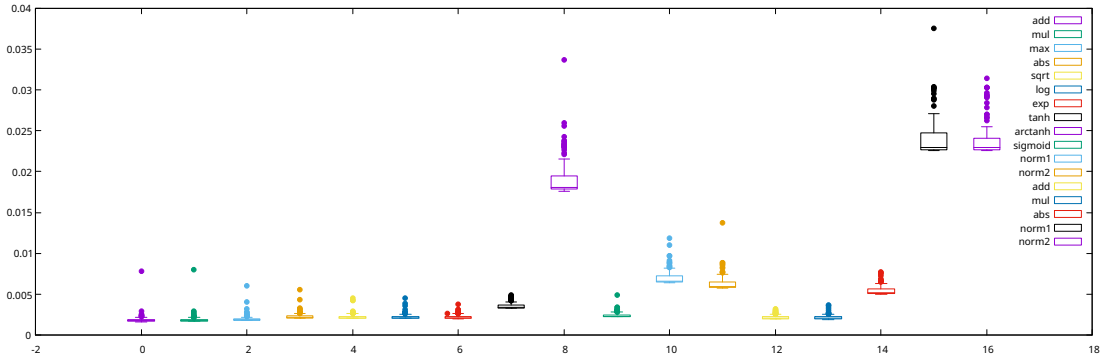


Figure 4 Execution time of atomic operations on random PyTorch tensors, on a CPU.

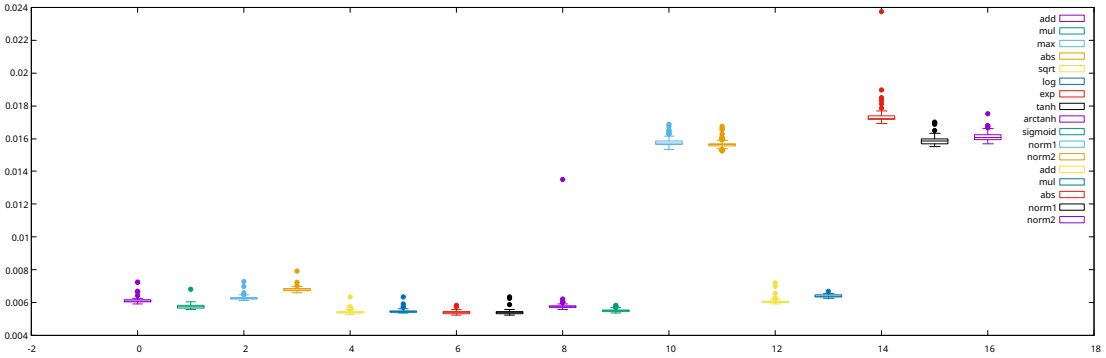


Figure 5 Execution time of atomic operations on random PyTorch tensors, on a GPU.

Table 5 Model performance for a 10k computational budget (d: dimension of embeddings, $|N_\tau|$: nb. of negative triples per KG triple, L: nb. of GNN layers).

				FB15k-237		WN18RR	
Model	**d**	**$\|N_\tau\|$**	**L**	**MR**	**hits@10**	**MR**	**hits@10**
TransE	1248	1	–	2402	0.173	11379	0.044
	100	23	–	247	0.371	1289	0.477
RotatE	499	1	–	301	0.311	1566	0.564
	100	8	–	219	0.409	3244	0.577
BoxE	113	1	–	219	0.349	2905	0.545
	75	2	–	199	0.384	3164	0.530
ExpressivE	146	1	–	272	0.347	3653	0.469
	97	2	–	208	0.381	4935	0.472
MuRE	998	1	–	189	0.409	2059	0.539
	100	18	–	216	0.467	2768	0.562
MuRP	107	1	–	209	0.400	2377	0.477
	71	2	–	190	0.429	2235	0.495
	42	4	–	185	0.441	2350	0.489
AttE	170	1	–	204	0.417	3061	0.523
	100	2	–	190	0.435	3132	0.509
AttH	82	1	–	212	0.413	3211	0.516
	54	2	–	199	0.430	3368	0.505
	33	4	–	198	0.436	3693	0.469
DistMult	1665	1	–	1171	0.188	15825	0.073
	100	31	–	1108	0.179	18251	0.029
ComplEx	416	1	–	1358	0.181	11359	0.323
	100	7	–	838	0.252	7631	0.432
ComplEx-N3	416	1	–	6979	0.001	13828	0.003
	100	7	–	3305	0.042	14541	0.013
QuatE	104	1	–	723	0.267	5290	0.524
	69	2	–	640	0.272	5526	0.527
R-GCN	53	1	1	307	0.318	–	–
	27	1	2	306	0.290	–	–
	35	2	1	276	0.321	–	–
	18	2	2	309	0.293	–	–
	293	1	1	–	–	5954	0.434
	161	1	2	–	–	5511	0.377
	100	4	1	–	–	4851	0.429
	100	2	2	–	–	5848	0.378
ConvKB	178	1	–	397	0.308	11226	0.135
	100	2	–	472	0.251	10635	0.112

Table 6 Hyperparameters found in the literature for experiments on FB15k-237 (d, d_e, d_r: size of (entity/relation) embeddings, b: block size, d_h: size of hidden layer, c: nb. of convolution filters, s_k: size of convolutional filter, L: nb. of layers, B: avg. nb. of neighbors, $|N_\tau|$: nb. of negative triples per KG triple, S: batch size) with indication whether inverse relations are explicitly modeled.

	Model	hits@10	d	$\|N_\tau\|$	S	Inverse	Other HPs
FB15k-237	R-GCN [37]	0.417	500	1	30000	?	$b = 5, L = 2, B = 43$
	DistMult [18]	0.419	100	14540	128	0	
	ConvKB [41]	0.421	100	1	256	0	$c = 50$
	ComplEx [18]	0.428	100	14540	128	0	
	TransE [29]	0.465	100	1	256	0	
	AttE [13]	0.488	32	50	500	1	
	MuRE [8]	0.493	40	50	128	1	
	QuatE [50]	0.495	100	10	27211	0	
	ConvE [18]	0.501	200	14540	128	0	$c = 32, s_k = 3 \times 3$
	AttH [13]	0.501	32	100	500	1	
	MuRP [8]	0.506	40	50	128	1	
	ComplEx [40]	0.509	1000	256	1024	0	
	ExpressivE [33]	0.513	1000	150	1024	0	
	DualE [12]	0.518	100	10	27211	0	
	MuRP [8]	0.518	200	50	128	1	
	MuRE [8]	0.521	200	50	128	1	
	TransE [40]	0.531	1000	256	1024	0	
	RotatE [40]	0.533	1000	256	1024	0	
	CompGCN [45]	0.535	100	14540	128	1	$d_h = 200, c = 200, s_k = 7 \times 7, B = 43$
	BoxE [1]	0.538	1000	100	1024	0	
	ConE [6]	0.54	500	100	1024	0	
	AttH [13]	0.54	500	50	500	1	
	AttE [13]	0.543	500	50	500	1	
	TuckER [9]	0.544	200	1	128	1	$d_e = d_r = d$
	ComplEx-N3 [22]	0.56	1000	1	1000	1	
	NBFNet [52]	0.599	32	32	256	1	$L = 6, d_h = 64, B = 23$

Table 7 Hyperparameters found in the literature for experiments on WN18RR (d, d_e, d_r: size of (entity/relation) embeddings, b: block size, d_h: size of hidden layer, c: nb. of convolution filters, s_k: size of convolutional filter, L: nb. of layers, B: avg. nb. of neighbors, $|N_\tau|$: nb. of negative triples per KG triple, S: batch size) with indication whether inverse relations are explicitly modeled.

	Model	hits@10	d	$\|N_\tau\|$	S	Inverse	Other HPs
WN18RR	DistMult [18]	0.49	100	40942	128	0	
	TransE [29]	0.501	50	1	256	0	
	ComplEx [18]	0.51	100	40942	128	0	
	ConvE [18]	0.52	200	40942	128	0	$c = 32, s_k = 3 \times 3$
	ConvKB [41]	0.524	50	1	256	0	$c = 500$
	TuckER [9]	0.526	200	1	128	1	$d_e = d, d_r = 30$
	AttE [13]	0.526	32	250	100	1	
	MuRE [8]	0.528	40	50	128	1	
	BoxE [1]	0.541	500	100	512	0	
	CompGCN [45]	0.546	100	40942	128	1	$d_h = 200, c = 200, s_k = 7 \times 7, B = 5$
	AttH [13]	0.551	32	50	500	1	
	MuRE [8]	0.554	200	50	128	1	
	MuRP [8]	0.555	40	50	128	1	
	DualE [12]	0.561	100	2	8683	0	
	QuatE [50]	0.564	100	1	8683	0	
	MuRP [8]	0.566	200	50	128	1	
	RotatE [40]	0.571	500	1024	512	0	
	AttH [13]	0.573	500	50	1000	1	
	ConE [6]	0.579	500	50	1024	?	
	ComplEx-N3 [22]	0.58	500	1	100	1	
	AttE [13]	0.581	500	50	1000	1	
	ExpressivE [33]	0.601	500	100	512	0	
	NBFNet [52]	0.666	32	32	128	1	$L = 6, d_h = 64, B = 3$

Temporal Modelling in Cultural Heritage Knowledge Graphs: Use Cases, Requirements, Evaluation, and Decision Support

Oleksandra Bruns[1]
FIZ Karlsruhe – Leibniz Institute for Information Infrastructure, Eggenstein-Leopoldshafen, Germany
Karlsruhe Institute of Technology (AIFB), Germany

Jörg Waitelonis
FIZ Karlsruhe – Leibniz Institute for Information Infrastructure, Germany
Karlsruhe Institute of Technology (AIFB), Germany

Jeff Z. Pan
The University of Edinburgh, Scotland

Harald Sack
FIZ Karlsruhe – Leibniz Institute for Information Infrastructure, Germany
Karlsruhe Institute of Technology (AIFB), Germany

Abstract

Our culture, history and world are in constant motion, continuously shaped by the flow of time, evolving narratives, and shifting relationships. Capturing this temporal complexity within cultural heritage (CH) knowledge graphs is essential for preserving the dynamic nature of human heritage. However, standard RDF predicates fail to effectively model the temporal aspects of cultural data, such as changing facts, evolving relationships, and temporal concepts. Over the past two decades, a variety of RDF-based approaches have been proposed to address this limitation, yet guidance is missing on which method best suits specific CH contexts. This paper presents a systematic evaluation of temporal RDF modelling approaches from a CH perspective. Based on an analysis of real-world CH use cases, core temporal requirements are identified that reflect both modelling expressivity and practical concerns. Six prominent approaches – RDF*, tRDF, Named Graphs, Singleton Property, N-ary Relations, and 4D Fluents – are assessed across these requirements. Our findings reveal that no single solution fits all scenarios, but suitable approaches can be selected based on project-specific priorities. To support practitioners, a decision-support tool is introduced to guide them in selecting the most suitable extension for their specific needs. This work provides practical guidance for CH modelling and contributes to the broader development of temporally aware Linked Data.

2012 ACM Subject Classification Information systems → Information integration; Computing methodologies → Knowledge representation and reasoning; Applied computing → Arts and humanities

Keywords and phrases Temporal Data Representation, RDF Extensions, Cultural Heritage, Knowledge Graphs

Digital Object Identifier 10.4230/TGDK.4.1.2

Category Survey

Supplementary Material
DataPaper: `https://github.com/sashabruns/RDF-Extensions-for-Cultural-Heritage`

Funding *Oleksandra Bruns*: acknowledges support from the Leibniz Association under project number SAW-2020- FIZ KA-4-Transraz.

Received 2024-10-31 **Accepted** 2026-03-09 **Published** 2026-03-31

[1] corresponding author

1 Introduction

Our past is not simply a collection of static facts and objects; it is a dynamic, evolving story where the context, relationships, and interpretations change over time. Historical events unfold over specific periods, social networks among persons evolve, cultural objects and terms gain new meanings, and jurisdictional practices and societal norms also change, continually reshaping how humanity understands past events. Preserving and analysing this web of changes provides deeper insight into our culture, society, and the world, revealing a more nuanced understanding of history.

Understanding time has been a central theme in philosophical discussions for centuries and continues to influence practical domains today, including the representation of time in knowledge graphs (KGs). In this context, KGs are graph-based structures designed to represent real-world knowledge, where nodes denote entities of interest and edges define various types of relationships between them [36]. How time is understood and contextualised impacts the methods and approaches used to model temporal data, affecting how evolving information within KGs is captured, reasoned about, and interpreted. For example, in [44], the distinction between the A- and B-theories of time has been introduced. The A-Theory of Time suggests that only the present is real, with the past and future existing merely as conceptual extensions. In contrast, the B-Theory posits that all points in time, past, present, and future, are equally real and exist simultaneously. Thus, when representing temporality within KGs, if aligned with A-Theory, models focus on versioning mechanisms, where the current state of entities and their relationships is actively tracked and updated. While models influenced by B-Theory adopt a snapshot-based representation, where models aim to provide a comprehensive view of all time points, enabling users to query and analyse data across different periods. Complementing this philosophical grounding, recent work on the evolution of open KGs [53] explores how temporal change is modelled and managed in real datasets. The study investigates practical perspectives of different modelling patterns, e.g. versioning and snapshotting, and offers a valuable technical interpretation of these philosophical distinctions in the context of KG evolution.

Similarly, the debate between Endurantism and Perdurantism evolves around the existence of objects over time [33]. Endurantism argues that objects are wholly present at each moment in time. In KGs, such entities are represented as static over time with evolving attributes. On the contrary, Perdurantism suggests that objects are spread across time, existing as a series of temporal parts. This has been formalised in foundational ontologies such as DOLCE [24], which explicitly classifies entities into endurants (e.g., physical objects or people) and perdurants (e.g., events or processes) to model the dynamic nature of reality.

Cultural heritage (CH) KGs are powerful tools for capturing, integrating and uncovering new insights from complex historical data [42, 40, 68, 13, 18]. By incorporating temporal information into these KGs, they can track changes over time, help users understand historical contexts, and uncover trends, leading to more sophisticated reasoning, better decision making, and deeper analysis. For example, understanding when an event took place and its relations to other events can be used to predict future outcomes or to reveal previously unknown details of the past.

Representing temporal context in CH KGs is crucial for several reasons, for example [67, 15, 65]:

- **Better understanding of temporal relationships**: In CH use cases, modelling temporal relationships enables more precise and nuanced historical interpretation. Accurately capturing when events occurred, how they overlap or follow one another, and how relationships evolve (e.g., changes in artistic movements) is essential for understanding narratives, context, and influence.
- **Gaining new insights**: Knowledge stored in CH KGs often comes from sources that are incomplete, fragmented, or disputed. Temporal modelling allows for reasoning over such data to infer possible sequences, detect contradictions, or identify missing links. For instance, when

reasoned over CH data, new connections between events or individuals not explicitly recorded can be detected. This aligns with the needs of historians, archivists, and scholars who work with uncertain or conflicting temporal accounts – for instance, when different historical sources assign different dates or periods to the same event.

- **Enhanced data integration**: CH data comes from diverse and distributed sources – archives, museums, libraries, galleries – and spans centuries. Temporal annotation supports the integration of such heterogeneous data by aligning information across different time periods and evolving terminologies. For instance, an entity may undergo e.g., changes in name, function, or classification over time; or different narratives may emerge around the same historical event with new sources uncovered with the time. Temporal modelling enables consistent representation of such information, ensuring that historical perspectives are (accurately) captured and understood.
- **More accurate provenance**: In CH, temporal (meta)data enables KGs to represent competing or evolving claims, e.g., conflicting dates for a historical event described in various sources. It also supports tracking how interpretations and representations change over time. For example, a stage performance of a classical play such as Antigone might be portrayed differently in the 1930s and the 1950s: under authoritarian regimes, the performance was likely shaped by censorship or propaganda. Temporal modelling captures these shifts, documenting not just what was performed, but how, when, and under which historical circumstances, preserving the meanings that evolve over time.

The Resource Description Framework (RDF) [22] is a widely adopted representation language and exchange format used to encode data on the Web and is a fundamental building block for creating KGs. However, due to its binary triple structure, the standard RDF model cannot natively express context in which a statement holds. Temporality is one type of such contextual information (other examples include location, provenance, certainty, source attribution, perspective etc.). Yet, temporal context comes with specific modelling challenges: it often involves complex structures such as intervals, durations, temporal relations (e.g., before, during), vague or conflicting time references. These aspects are especially prominent in CH applications, where representing temporal knowledge is not just a matter of attaching timestamps, but of capturing historical change, overlapping narratives, evolving interpretations, etc. As a result, various extensions to RDF have been proposed to enable temporal representation. Each of these extensions has its own advantages and limitations, and the choice of the approach depends on the specific requirements of a use case, balancing query simplicity, semantic expressivity, compatibility with existing RDF tools, and scalability. However, despite this variety of available temporal extensions, there remains a critical gap in understanding how well these models address the unique needs of specific application domains – particularly CH, which is characterised by highly heterogeneous and often ambiguous historical data, where facts are uncertain, timelines are incomplete, the meaning of entities evolves over time etc.

While prior efforts have introduced and compared various temporal extensions to RDF [57, 73, 78, 76], they often remain disconnected from concrete applications. By answering these research questions:

- **RQ1.** What RDF-based approaches exist for temporal representation?
- **RQ2.** What temporal requirements arise from CH use cases?
- **RQ3.** How intuitive and usable are different temporal modelling approaches for CH domain experts?
- **RQ4.** How can temporal modelling approaches be evaluated based on CH-specific requirements and technical criteria?
- **RQ5.** How can CH practitioners be supported in selecting a suitable temporal modelling approach?

this work aims to provide a systematic analysis that translates the abstract capabilities of RDF extensions into practical implications for representing time in CH KGs. In particular, it presents a comprehensive research study focused on identifying, structuring, and evaluating RDF-based temporal modelling approaches from a CH-specific perspective. The analysis is grounded in real-world case studies, deriving domain-specific temporal requirements, and developing a decision-support system to guide practitioners. The results aim to inform not only CH projects but also broader domains facing similar challenges in managing temporality in knowledge representation.

The remainder of the paper is structured as follows. Section 2 introduces representative CH use cases and derives a set of temporal modelling requirements. Section 3 reviews related work on temporal modelling in RDF and CH. Section 3.3 presents the RDF-based approaches evaluated in this study. Section 4 details the comparative assessment of these approaches against the identified requirements. Section 5 introduces the decision-support tool for selecting suitable temporal modelling strategies in CH contexts. Finally, Section 6 summarises the findings and outlines directions for future research.

2 Use Cases and Requirements

The modelling of time in CH KGs presents distinctive challenges coming from the nature of historical data and its interpretation. To understand these challenges and the corresponding modelling needs, this section introduces a set of representative CH use cases. Through their analysis, core temporal characteristics and complexities are identified, and a set of domain-specific requirements is derived. These requirements provide a foundation for the subsequent evaluation of temporal modelling for CH KGs.

In [15, 13, 71, 66, 70, 67], initial efforts towards defining challenges and requirements for representing temporal information in CH use cases are conducted[2]. However, these studies examined the use cases independently, each focusing on specific aspects within their respective projects — TRANSRAZ[3], Wiedergutmachung[4], "Subject Related Points of Access within Archivportal-D on Example of the Subject Area Weimar Republic"[5], and Linked Stage Graph[6]. While these studies provide valuable insights, they approach the temporal challenges from isolated perspectives, tailored to project-specific objectives. In this work, these prior findings are systematically consolidated and extended, offering a harmonised view across use cases.

2.1 Case Study: Cultural Heritage

It is acknowledged that the use cases discussed here represent only a fraction of the CH domain, and other subdomains, such as musicology, may pose additional challenges and require different adaptations in the context of temporality. Nonetheless, this structured analysis is a necessary step towards understanding temporal aspects of CH data and providing means for evaluating the existing approaches.

---

[2] The use cases presented in this section were selected based on their origin in distinct, real-world CH research projects that the authors were actively involved in.
[3] https://www.fiz-karlsruhe.de/en/forschung/transraz
[4] https://www.fiz-karlsruhe.de/en/forschung/wiedergutmachung
[5] https://www.fiz-karlsruhe.de/en/forschung/archivportal-d-sachthematische-zugaenge
[6] https://slod.fiz-karlsruhe.de/

Use Case 1: Connecting Time Layers of Nuremberg's History

The TRANSRAZ project aims to reconstruct the city of Nuremberg by integrating and linking historical information from different time periods. The project seeks to create a comprehensive and dynamic 3D model of the city, enriched with semantically connected historical data that spans centuries, enabling historians, researchers, and the general public to explore Nuremberg's history interactively, with the ability to navigate through different time periods and understand the city's evolution.

Data and Challenges. The dataset includes a wide variety of historical resources such as address books, biographical articles, research papers, historical maps, city plans, and others. One major challenge lies in modelling how the roles and functions of places or people change over time. For instance, a building served as a bakery in one century and was later converted into a school, even while retaining the same location or name. Another frequent issue is the renaming of streets or places due to political/administrative decisions. Capturing these changes while maintaining continuity in identity requires labelling associated entities with time validity.

Similarly, historical records describe people or institutions in evolving roles or under different names, complicating how such entities are consistently modelled across time. Precise dating is not always available. Sources provide full dates (e.g., "15 July 1892") or vague references (e.g., "during his studies" or "until 1923"), requiring the ability to represent and distinguish between exact timestamps, intervals, and intervals with missing boundaries. Furthermore, the data includes temporally overlapping or sequential events, such as changes in residency, profession, or ownership. This requires capturing not only when things occurred, but also how different events relate to each other chronologically – whether they overlap, follow one another, etc. Finally, much of the data is uncertain, approximate, or even conflicting across sources. For example, different documents provide inconsistent information about a person's occupation or address at a given time. A temporal model must support the representation of multiple interpretations, including metadata about the source and temporal validity of each claim.

Use Case 2: Temporal Alignment in Archival Records

Within the "Wiedergutmachung" project, the focus lies on reconstructing the processes of reparation and social transformation in post-war Germany, particularly after the fall of the Nazi regime. The project examines how restitution and compensation were handled from 1945 to the 1970s, based on a vast array of archival records that document these complex processes. This project is especially crucial as it aims to document the experiences of the victims of Nazi persecution, capturing their personal and family histories as narrated in their reparation claims.

Data and Challenges. The dataset includes extensive archival records spanning from 1945 to the 1970s, with millions of individual case files detailing the experiences and claims of those who suffered under the Nazi regime. These records are spread across various German archives and consist of personal records, organisational information, and detailed accounts of the reparation processes. The primary goal of this project is to reconstruct the timelines of people's lives, significant events, and legal processes by capturing and accurately integrating temporal information from these diverse records. This involves aligning data to reflect changes in personal details, organisational structures, and the progression of reparation processes over time.

Such records contain a large amount of temporal information that needs to be modelled. The primary challenge lies in the need to represent each fact along with its temporal validity and provenance. While resolving conflicting or ambiguous information is outside the scope of the

representation model itself, the model must still support the accurate and consistent encoding of changes over time. For example, if an individual is referred to as *"Löwental"* in an archival record from 1948 and as *"Friedrich"* in a document from 1950, a temporal model must be able to express both name variants, associate them with the correct time intervals, and link them to their respective sources. In this way, the temporal evolution of facts, as well as their provenance, can be tracked and maintained within the knowledge graph.
Moreover, reparation processes are inherently temporal and evolve over time. Claims often span multiple legal stages, involve different institutions, and result in varying outcomes depending on case-specific conditions. A claim investigated in 1951 under one administration led to a denial, while the same claim reevaluated in 1954 – after a policy reform – was accepted.

Capturing such processes requires the ability to model sequences of events (e.g., investigation followed by decision and payment), overlaps (e.g., parallel claims by family members), and dependencies between events (e.g., the decision phase relying on the completion of a medical evaluation). In addition, the historical records often contain temporal uncertainty and conflict. Some documents specify exact dates (e.g., the filing of a claim), while others reference vague or conflicting time frames (e.g., "shortly after the war").

Use Case 3: Dynamics in Archival Classification Scheme

The project "Subject Related Points of Access within Archivportal-D on Example of the Subject Area Weimar Republic" focuses on the digitisation and classification of archival records from the Weimar Republic. Unlike the previous use cases, which aim to extract and represent content data, this project is designed to assist archivists in the digitisation and classification process. Temporal aspects are managed through a complex dynamic classification scheme that evolves as new records are digitised and integrated into the archive. For archivists, it is crucial to track the digitisation process and how the classification scheme has developed over time. This allows them to understand the evolution of archival categorisation, ensure consistency, and make informed decisions about organising and accessing historical records as new data is added.

Data and Challenges. The dataset includes over 20,000 digitised archival records, with the digitisation process still ongoing. These records encompass government documents, legal texts, personal correspondences, and other materials from the Weimar Republic era. To collect, store long-term, and make historical records accessible to all, archival resources must be categorised for structured, intuitive search and exploration.

The main challenge is that the classification scheme, which comprises around 900 specialised keywords, 20 categories, and 130 subcategories, is not static. It must be continuously adapted to accommodate new records and additional resources from other archives. This dynamic nature of the classification system requires careful modelling to ensure that it can evolve over time without losing relevance or accuracy. For instance, as new records are added, previously used keywords might be eliminated, or certain documents might be reclassified, reflecting changes in understanding or context. Additionally, it is important for archivists to be able to keep track of all versions of the classification scheme, and allow researchers to analyse the evolution of document classification. This could lead to insights such as which keywords were most often eliminated or which documents were the most reclassified.

While the challenges of managing evolving taxonomies are well-known in archival and library sciences [34], this use case presents a distinct modelling task: capturing classification scheme dynamics as structured, queryable knowledge. Rather than maintaining a single live classification scheme, the aim is to construct a KG that reflects the validity of terms over time, supports time-based retrieval, and enables the analysis of concept evolution within the taxonomy itself. This extends traditional taxonomy management into the temporal KG domain, where the classification system is not only curated but semantically modelled as a historical and dynamic object.

Use Case 4: Linked Stage Graph for Performing Arts

The Baden-Württemberg State Archives[7] in Germany opened up their data to create a Linked Stage Graph, facilitating the exploration and analysis of historical performance data. The initiative aims to provide insights into theatrical performances during a time marked by significant social and political events in Germany, highlighting the context and evolution of stage design, actors, and performances.

Data and Challenges. The dataset includes 7,000 historical black and white photographs and EAD-XML metadata covering the period from the 1890s to the 1940s. The goal is to reveal political and social context of performing arts in Germany, e.g. to research which theatre performances were allowed during challenging times and how certain characters were depicted.

A central challenge in this use case lies in representing how elements of the performing arts evolve over time. Theatrical works are temporal: performances occur on specific dates, while actors, stage designs, and interpretations change across decades. For example, Bertolt Brecht's plays were interpreted very differently in the 1920s compared to their postwar productions. A particular character, like Antigone, was often portrayed as a tragic figure of resistance in one period and as a passive symbol of duty in another, shaped by political censorship and cultural movements. Capturing such evolution requires modelling not just entities like actors or plays, but also their changing roles and meanings over time. Moreover, descriptive terms used in archival metadata often shift. Stage roles are described using different terminology ("lead actor" versus "main character"), and theatre venues are renamed or repurposed. These terminological and functional changes must be represented. Additionally, performances are documented with varying degrees of temporal precision. Some are precisely dated, while others are loosely described as occurring "in the early 1930s" or "around 1940". Capturing temporal relationships between (performance-)related events is essential for situating theatrical life in its full socio-political context. For example, modelling that one actor's portrayal of Antigone ended in 1933 and was succeeded by another can reflect not only the change itself but also historical circumstances such as institutional changes or political pressure. Similarly, a premiere coinciding with the rise of National Socialism can significantly affect how a play was staged and received.

2.2 Requirements

The temporal modelling needs in CH projects are shaped by the way knowledge is produced, interpreted, and preserved over time. Unlike many domains where temporal data can be precise and well structured, CH data are often incomplete, uncertain, and deeply contextual. It must capture not only when entities or events occurred but also how they evolved, how they relate across time, and how they are interpreted differently in varying historical narratives. The complexity of this temporal dimension is further compounded by the diversity of technical infrastructures and data practices across CH institutions.

To identify the core requirements for temporal modelling in CH, the data and needs of real-world use cases presented in Section 2.1 are analysed. These requirements reflect both i) how well a model can express temporal phenomena and ii) how feasible and usable a model is in practice. While inspired in part by broader literature on knowledge graph evolution (e.g. [53]), our focus remains on the specific constraints and characteristics of CH datasets. The result is a set of requirements that guide the evaluation of temporal modelling approaches for CH use cases.

[7] https://www.landesarchiv-bw.de/de/en/68809

- **REQ1: Temporal Expressivity** is concerned with the granularity and richness with which temporal semantics can be described. It serves as the foundation for assessing the semantic adequacy of temporal modelling approaches in the CH domain and include:
  - **REQ1.1: Contextual Change** captures the need to model how entities and their descriptions evolve over time. Such changes may involve the functional roles of entities (e.g., a person becomes a director), their classification (e.g., a building's function changes from residential to commercial), naming conventions, or how they are interpreted in different historical contexts. Crucially, the identity of the described entity remains stable, while its representation is adapted across multiple temporal snapshots. The model must therefore support multiple temporally-scoped descriptions of the same entity and preserve the validity period of each[8].
  - **REQ1.2: Temporal statements.** Statements about the world, such as a person's role, a building's function, or an institutional relationship, are rarely static in CH data. This requirement involves the ability to associate such statements with explicit temporal scopes. The modelling approach must allow for the attachment of time intervals (or instants) to individual assertions in a way that distinguishes which statements held true at which times. This enables accurate reconstruction of timelines and state changes over historical periods.
  - **REQ1.3: Temporal Uncertainty and Incompleteness.** Historical records frequently contain vague, approximate, or partially missing temporal information. Dates may be imprecise (e.g., "circa 1900"), incomplete (e.g., "before 1935"), or based on subjective interpretations. A temporal model must provide vocabulary that can represent such uncertainty, without reducing it to artificially precise or misleading constructs.
  - **REQ1.4: Interval Relations.** Beyond isolated timestamps or intervals, CH research often requires understanding the sequence, overlap, or causality of events. This requirement calls for supporting qualitative temporal relations between time intervals or events (e.g., before, meets, overlaps) as well as reasoning over those relations. It enables the reconstruction of event timelines and dependencies, such as legal or historical developments, and supports the integration of events into broader historical narratives.

However, semantic expressivity alone is not sufficient to ensure practical applicability. In real-world CH projects, the deployment of temporal knowledge graphs must also satisfy technical, operational and usability requirements. These additional dimensions reflect recurring concerns raised during the implementation and integration phases of CH projects in which the authors have been involved, especially in discussions with domain partners on performance constraints, tool support, and user interaction. Furthermore, these structural aspects align with established concerns in the Semantic Web literature (see Section 3.4).

- **REQ2: Storage and Scalability**: In CH, where datasets often grow to substantial sizes due to the rich historical details they contain, scalability is essential. Efficient representation of temporal information plays a key role in enabling systems to handle large volumes of data without sacrificing performance. Approaches that produce excessive structural overhead may pose challenges for storage, querying, and reasoning at scale.
- **REQ3: RDF Syntax Extension**: Many temporal models extend the standard RDF syntax to incorporate a temporal dimension, which can influence compatibility with existing RDF frameworks and, e.g. reasoning. This requirement aims to provide an overview to the

[8] Contextual change, as used here, is different from semantic shift in ontological terms: it does not imply changes in the meaning of classes or properties themselves, but rather changes in the description of instances under evolving circumstances.

extension, considering that it may directly influence the choice of triple store, reasoner, etc. Approaches that introduce minimal but effective extensions are often preferred, as they maintain compatibility with standard RDF tools and infrastructures, leading to easier integration into existing systems.

- **REQ4: Model Semantics**: This requirement assesses the level of semantic expressiveness required by a modelling approach to effectively represent the information. Specifically, it examines whether the approach relies on basic RDF(S) semantics or if it requires more advanced semantic frameworks, rules or even custom logic to capture additional constraints or specialised concepts that are often introduced in temporal approaches.
- **REQ5: Query Language**: In many CH projects, querying is not just a technical detail but a practical constraint shaped by the institutional tooling in place. CH institutions often rely on existing SPARQL endpoints for accessing and maintaining their data, and these endpoints may not support advanced SPARQL extensions or custom query languages. As a result, the ability to use standard SPARQL becomes not only a matter of usability but also a strict requirement imposed by the technical infrastructure.
- **REQ6: Use of RDFS/OWL Reasoning**: This requirement investigates whether the modelling approach preserves compatibility with OWL/RDFS reasoning, and to what extent temporal constructs can participate in standard inference processes. Evaluation considers whether RDFS- and OWL-based entailment, e.g., class membership, property inheritance, inverse properties, or domain/range inference, can still be performed over temporally enriched data.
- **REQ7: Ease-of-Use**: In the CH domain, users are often domain experts with limited technical expertise, particularly in RDF and querying. Therefore, it is crucial to evaluate the ease-of-use of each approach, ensuring that domain experts can easily understand and use the models without extensive training. While specialised tools and user interfaces can enhance usability by hiding the complexities of direct modelling, this study focuses specifically on the intrinsic ease-of-use of the modelling approaches themselves. We argue that understanding how data is modelled and ensuring its accuracy is crucial for CH domain experts, even if they are not directly responsible for data modelling and querying. Their feedback is essential for validating the effectiveness and correctness of the ontologies and KGs used in the CH domain. Consequently, evaluating ease-of-use helps ensure that the models not only meet technical requirements but also reflect the practical needs and expectations of end-users.

The seven identified requirements highlight the distinct temporal and technical demands of CH projects. Together, they provide a foundation for evaluating existing temporal data models and inform decision support for the CH community in selecting appropriate approaches for temporal data representation. While these requirements are grounded in domain-driven design and real-world constraints derived from iterative collaboration with CH experts, they also align with broader ontology evaluation principles such as clarity, usability, extensibility, and scalability, as outlined in established frameworks [23]. These principles are reflected, for instance, in the attention to ease-of-use, RDF compatibility, and the support for temporal semantics. However, the focus of this work lies in the pragmatic application of temporal modelling within CH projects, where practical solutions often outweigh formal ontology design ideals. Thus, the presented requirements are not intended to exhaustively reflect theoretical completeness but rather to guide realistic decision-making based on specific needs of a CH project. The following section reviews existing work in relation to these requirements.

3 Related Work

To address the challenges and requirements of temporal representation in CHKGs, this section surveys a range of relevant modelling approaches and comparative studies. It begins by reviewing how temporality is currently modelled in existing CHKGs, followed by an overview of how foundational ontologies conceptualise time. The section then explores established temporal data modelling frameworks, with a particular focus on RDF-based extensions. In addition, to lay the groundwork for answering RQ4 (*"How can temporal modelling approaches be evaluated based on CH-specific requirements and technical criteria?"*) Section 3.4 examines comparative studies and surveys.

3.1 Temporal Description Logics

Temporal Description Logics (TDLs) extend classical Description Logics (DLs) by incorporating temporal constructs to represent and reason over knowledge that changes over time. While DLs are widely used for modelling structured knowledge with formal semantics and decidable reasoning, they typically assume that all statements are eternally valid. However, this assumption does not hold for many domains, including CH, where facts are often temporally situated.

Initial work in TDLs, such as combination of DL $\mathcal{FLEN}\mathcal{R}^-$ with the modal logic $\mathcal{HS}$ [58, 31], laid the foundation for representing temporal relations such as *before*, *meets*, or *overlaps* [2]. Although expressive, such early combinations proved undecidable, prompting later efforts to design fragments that balance expressiveness with decidability and practical reasoning. Surveys such as [6, 7] categorise these into three primary types: point-based, interval-based, and metric TDLs.

Point-based TDLs define time as a sequence of discrete instants, using temporal operators from temporal logics like LTL [52] and CTL [20]. Interval-based TDLs model time through bounded intervals and often leverage decidable subsets of $\mathcal{HS}$. Recent developments also introduce metric TDLs [30, 8], which allow for quantitative temporal constraints (e.g., events occurring within a specific duration). Some TDLs, such as those used in ontology-based systems like *BALOoNSE* [55], demonstrate promising applications in areas like patient care and industry.

Despite their high expressiveness and robust reasoning capabilities, TDLs introduce additional modelling and computational complexity. Reasoning typically requires specialised inference engines beyond standard OWL reasoners and may involve logic programming, automata-theoretic methods, or model checking. Moreover, many fragments are not supported natively in common triple stores or SPARQL-based workflows, requiring alternative tools or custom implementations.

Within the scope of this research, TDLs are not evaluated in depth for two reasons. First, while CH applications certainly benefit from temporal reasoning, most CH infrastructures are based on RDF(S) and OWL and rely on OWL inference provided by common tools. In practice, domain partners in CH projects often lack access to specialised reasoning environments and require solutions that integrate easily with SPARQL endpoints and Linked Data publication pipelines. Second, the requirements analysed in this study originate from concrete use cases (Section 2.2), which prioritise intuitive modelling (REQ7), standard query ability (REQ5), manageable complexity (REQ2), etc. TDL-based systems of today struggle to meet this without significant adaptation. While TDLs provide important theoretical foundations for formalising temporal reasoning, they are not included in the main comparative evaluation of this work due to their limited adoption and practical integration in current CH infrastructures. Nonetheless, they offer valuable insights that may inform future efforts to enhance temporal expressivity and reasoning in RDF-systems.

3.2 Event-based Models for Temporal Representation

One of the most widely adopted paradigms for modelling temporality in CH data is the use of event-based representations. The event-based approach treats events as the main entities that serve to contextualise changes in time. Thus, people, places, objects, and time are linked through event entities, which enables capturing complex temporal relationships via direct attachment of temporal information to the event. This has become a foundational strategy in many upper ontologies as well as CH ontologies and KGs. The following subsections explore how this approach is grounded in foundational ontologies and how it is practically applied in existing CH knowledge graphs.

Time in Foundational Ontologies

Foundational ontologies typically approach temporality through core ontological distinctions that categorise entities based on their persistence and change over time. The *Basic Formal Ontology* (BFO) [5, 51] introduces a fundamental separation between *continuants*, which exist wholly at any time they are present, and *occurrents*, which unfold over time and have temporal parts. Temporality is modelled through participation in occurrents that are explicitly linked to temporal intervals. Similarly, *DOLCE* [24] distinguishes between *endurants* and *perdurants*, where perdurants include events and processes that exist in time. Endurants are then "participate" in perdurants (events), providing a way to model temporality. *GFO* (General Formal Ontology) [35] represents time as a dimension parallel to space, with processes being temporal and possessing temporal parts, and *presentials* defined as entities that exist at specific time boundaries. Concrete individuals thus exist in time, but cannot be described as having temporal parts. The *Unified Foundational Ontology* (UFO) [27] incorporates temporality through three interlinked sub-ontologies: UFO-A for endurants, UFO-B for perdurants (events and situations), and UFO-C for social entities. Temporality is handled most explicitly in UFO-B, where temporality is, similarly to other ontologies, is modelled through structured event participation. UFO-B, however, is completely contextualised in First-Order Logic. Lighter-weight ontologies such as *PROTON* [64] distinguish between *happenings*, *objects*, and *abstractions*, introducing categories for events and situations. More comprehensive upper-level ontologies like *SUMO* [47] and *Cyc* [26] provide extensive temporal vocabularies and formalisations, including support for time intervals, recurring events, and duration constraints. However, they are often defined in higher-order logic or proprietary languages, making integration into RDF-based CH applications more challenging.

Overall, while foundational ontologies provide theoretical tools for conceptualising temporality through entity classification (so-called "objects" vs events) and temporal relations, they lack direct support for the kind of temporal annotation and querying required in RDF-based knowledge graphs. As such, their use in CH often requires additional layers of abstraction.

Time in Existing CH KGs

In context of CH, CIDOC-CRM [16], for instance, is a well-established event-based ontology used to model complex historical and cultural events. Via introducing events and activities, it captures complex temporal relationships between entities, actions, and time periods, and is widely adopted in domains like archaeology and museum documentation. Unlike direct RDF relationships in entity-centric representations, which might struggle to incorporate temporal annotations, event-based models provide a framework where temporal aspects are managed through intermediary events. For instance, in the scenario, `<:play :director :WalterFriedrichPeters>`, an extension to RDF is required to annotate the fact with specific temporal information, such as the year 1938. In contrast, event-based representations allow for representing this information within an event:

```
:play :subjectOf :DirectingEvent .
:DirectingEvent :performedBy :WalterFriedrichPeters .
:DirectingEvent :validIn "1938" .
```

A number of CHKGs build on CIDOC-CRM or similar event-centric paradigms to address temporal modelling needs. For example, the ArCo Knowledge Graph [18], developed for Italian CH data, leverages CIDOC-CRM and introduces specialised modules to support temporal modelling, such as dates of artifact creation, transformations, and changes in ownership. ArCo integrates multiple temporal ontologies and supports both time intervals and approximate dates (as literals), offering a rich but complex modelling strategy that inherits the layered nature of event-based design. Several domain-narrower data models extend CIDOC-CRM to represent the knowledge. For example, the TRANSRAZ data model [14], designed to support historical research on urban transformation in Nuremberg, builds directly on CIDOC-CRM and emphasises the temporal evolution of entities such as buildings and street networks. The Sampo model family [38], adopted across several Finnish CH projects (e.g., BiographySampo, WarSampo), follows a similar pattern by using event-based structures to represent life events, affiliations, and temporal relationships. Alternative to CIDOC-CRM is the Europeana Data Model (EDM) [25] that supports only a lightweight form of temporal annotation using properties such as `dcterms:created` and `edm:hasMet`, and represents events as contextual nodes. EDM provides less semantic depth in comparison to CIDOC-CRM, but prioritises interoperability and integration across a wide variety of heritage institutions. Thus, temporal modelling is often simplified, reflecting the need to harmonise data from heterogeneous sources.

While event-based models are effective and widely used for representing temporality in CH, they introduce additional modelling complexity by requiring events as intermediary entities. This added abstraction can complicate both data creation and querying, especially when only simple temporal annotations are needed. In many CH applications, temporal data primarily concerns approximate periods or dates associated directly with entities, such as persons, roles, or facts, without necessitating the introduction of separate event constructs. Moreover, CH datasets often originate from raw, unstructured sources, e.g. archival documents or spreadsheets, with no preexisting URIs or ontology alignment. In such contexts, restructuring the data to fit an event-based paradigm becomes especially cumbersome. It requires rethinking what counts as a temporal entity (e.g., transforming a fact into an event) and introducing additional layers of modelling just to attach time information. This not only increases the modelling effort, but also risks obscuring the original data structure in ways that are confusing or inaccessible to domain experts.

Given these challenges, this study explores whether alternative approaches, RDF-based extensions, can offer more intuitive ways to represent temporality. By enabling time to be attached directly to facts or relationships without the need for intermediate events, RDF extensions may better support the pragmatic needs of CH practitioners. Consequently, the remainder of this work focuses on RDF extensions as a potentially more lightweight and accessible solution for temporal modelling in CH knowledge graphs.

3.3 Temporal Extensions to RDF

As an alternative to event-based models, RDF reification has been proposed as a way to annotate RDF triples with contextual information[9]. Using the RDF vocabulary (`rdf:Statement`, `rdf:subject`, etc.), a triple can be reified and metadata, e.g. start and end dates, attached. For

[9] `https://www.w3.org/TR/rdf-schema/#ch_reificationvocab`

example, to express that theatre performance of *Minna von Barnhelm* was directed by *Hans Tessmer* and by *Wilhelm Speidel* in different time periods, the corresponding triples are reified and annotated with their temporal context, for this:

- two blank nodes of type `rdf:Statement` are introduced.
- the blank nodes are then linked to their subjects (*performances of Minna von Barnhelm*), predicate, e.g. `:directedBy`, and objects (*Hans Tessmer* and *Wilhelm Speidel*) via corresponding properties `rdf:subject, rdf:predicate, rdf:object`.
- the blank nodes are linked to their corresponding time validity (e.g. *1933* or *1939*)

While reification enables annotation of triples with metadata, it introduces significant verbosity and structural overhead, since for every triple `<s, p, o>`, blank nodes and additional triples are introduced (`<_:x, rdf:subject, s>, <_:x, rdf:predicate, p>, <_:x, rdf:object, o>`). Furthermore, RDF reification is a general-purpose mechanism and does not provide built-in semantics for interpreting the metadata it attaches – including temporality.

A variety of approaches have been proposed that extend RDF in different ways to support temporal representation more explicitly and efficiently. The goal of this section is to provide a high-level overview of the most prominent RDF-based temporal modelling approaches. A more detailed analysis of each approach and how it performs with respect to CH requirements is presented in Section 4. The RDF extensions are grouped according to the level at which they introduce temporality – component, triple, or graph level – to illustrate the different strategies available for temporal modelling in RDF.

Component Level Extensions

In component level approaches, temporality is introduced by modifying individual components of RDF triples, such as properties or entities.

- **Singleton Property** is proposed in [45]. The authors argue that in every temporal context, the relationship between two entities is unique. They introduce the concept of "generic property" to represent the general characteristics of relationships and "singleton property" to capture the specific relationship within a particular time context. Thus, a unique property is modelled as an instantiation of a generic property via `rdf:singletonPropertyOf`. The temporal fact has a form of triple `<s, p@t, o>`, where `p` is a generic property and `p@t` is a singleton property that defines a unique temporal context of `p` at time `t`.
- **N-ary Relations** approach introduced in [48] addresses the problem of binary representation of relations in RDF/OWL. The inability to relate more than two individuals or to describe a relation itself is proposed to be solved via objectifying a relation. An objectified relation is a relation converted into an entity that may serve as domain or range of a property. Thus, a temporal fact has a form of a triple `<s, p_1, o>`, where `o` is an objectified relation that can be further described with its time validity `t` – `<o, p_2, t>`.
- **4D Fluents** [75] are based on the perdurantist view, which argues that on a certain scale every entity appears to be different in a different period of time. According to this perspective, each entity is made up of temporal parts that reflect its state at various time points. To convert regular entities into 4D Fluents, class `fl:TemporalPart` and properties `fl:temporalPartOf` and `fl:temporal` are introduced. Consequently, a temporal fact is represented via triple `<s@t, p, o@t>`, where `s@t` and `o@t` are temporal parts of `s` and `o` that are valid at time `t`. In [11] the 4D Fluents ontology is extended via introducing the new class `fl:TimeInterval` with property `fl:isTimeInterval`. Additionally, this extension incorporates Allen's 13 interval relations, such as *before*, *met by*, and *equals*, thereby providing additional semantic depth to the model.

- **Valid Time RDF** (*VTRDF*) [72] extends the ideas of 4D fluents, defining valid time resources and relations. Thus, every fact in VTRDF is temporal and represented by a VTRDF triple <s^t, p^t, o^t>, where every entity s and o is entailed with its existence in time t, while p establishes a valid time relationship between s and o. To achieve such representation, the standard RDF vocabulary as well as the standard IRI are extended. A valid time IRI consists of a resource and its valid time that is identified by a delimiter •, e.g. http://example.org/Henry•[01-01-1990-now]. Furthermore, literals, data types, blank nodes of standard RDF are all temporalised to their valid time variants. Additionally, the Valid Time SPARQL language for querying VTRDF graphs is designed.

Triple Level Extensions

Triple level extensions incorporate temporality directly into RDF triples, either by modifying them or by attaching additional temporal labels.

- **Temporal RDF** (*tRDF*) [28] and its variants [29, 37, 54] are the extension of standard RDF graphs that allow to label triples with time by means of reification. The standard RDF vocabulary, rdf:Statement, rdf:subject, rdf:object and rdf:predicate, is used for reification purposes, while temporal semantics is incorporated by introducing temporal vocabulary: properties trdf:temporal, trdf:initial, trdf:final, trdf:instant, trdf:interval and literal NOW. Thus, a temporal RDF triple has form <s, p, o>[t], where t represents the time interval or the time point the triple <s, p, o> is valid in. In [29] the problem of uncertain temporality is addressed by allowing anonymous timestamps to label temporal triples. Thus, the temporal label t may state the temporality of a fact implicitly without mentioning the exact time point/interval. Additionally, temporal constraints are included, and reasoning over temporal graphs with intervals is entailed and practically tested [37]. Such enhancement of the model contributes to defining temporal relations between triples even if the exact time boundaries of their validity are uncertain or unknown.
- **Annotated RDF** (*aRDF*) [69] is a generic extension to the RDF model that is proposed to support various annotations of RDF triples, such as their provenance or temporal validity. Annotations of an aRDF triple are defined as algebraic structures of domain values, e.g. for temporal domain – time intervals or time points, and employs operators *meet* and *join* from annotated logic [39] to combine the values. The aRDF formalism is represented as <s, p, o>[a], where a represents an annotation from a partially ordered set of annotations A. aRDF employs annotated logic, which differs from the standard RDF reasoning mechanisms. This divergence requires specialised frameworks for data integration and querying, as the underlying logic for annotations may not directly align with traditional RDF processes.
- **RDF*** [32] extends standard RDF by introducing nesting triples. More specifically, an RDF* fact has a form of a triple <<s>, p, <o>>, where s and o may be directly represented by other triples. The depth of nesting is denoted by k, where if k=0, the triple is a standard RDF triple; and if k>0, the triple is a nested RDF* triple. Turtle* and SPARQL* are the extensions of Turtle and SPARQL that enable writing and querying RDF* graphs. RDF* triples remain backwards compatible and may be transformed into standard RDF triples via unfold, black node assignment and reification functions.

Graph Level Extensions

Graph level extensions introduce temporality by grouping RDF triples into separate graphs based on their temporal dimensions.

- **Named Graphs** are used to group all RDF triples that share the same meta knowledge into separate graphs. This is achieved by adding additional dimension(s) to RDF triples. In [17], the quad format of Named Graph model – `<s, p, o, NamedGraph>` is adopted by W3C. In temporal context, such representation is useful for modelling and querying data from multiple time intervals or time stamps, where events and facts for each time interval can be managed as separate graphs associated with time `t`: `<NamedGraph, p_2, t>`. In contrast to reification, Named Graphs allow for modelling and querying without transformation of original triples.
- **RDF+** [59], the extension of RDF for querying metadata knowledge, proposes a quintuple format by extending the named graph quad with a statement identifier – `<s, p, o, NamedGraph, statementID>`. The identifier is used as the subject of the meta-knowledge assertion, which is an RDF triple.

Most of the mentioned extensions to RDF were not initially designed specifically for temporal contexts but aimed to incorporate meta-knowledge into RDF, thus lacking temporal vocabulary. However, there are several external time ontologies that offer comprehensive temporal semantics and can be used to address various needs for expressive temporal representation [21, 10, 4, 50, 56, 12, 19]. These ontologies, discussed and evaluated in Section 4.1, provide specialised vocabularies and frameworks to incorporate temporal information into RDF.

In summary, various approaches have been developed to extend RDF with temporal capabilities, each offering a unique perspective on integrating temporality into RDF data. However, given this diversity, it becomes increasingly important to understand how these approaches compare in terms of their practical applicability, expressivity, and usability, particularly in the context of specific domain needs such as those in CH. This leads to a crucial research question: **RQ4: How can temporal modelling approaches be systematically evaluated based on CH-specific requirements and technical criteria?**. The following section reviews the work related to this RQ, highlighting existing comparative efforts, their focus areas, and methodological approaches.

3.4 Surveys on Temporal Extensions to RDF

To the best of our knowledge, the first comprehensive survey on the representation of temporal data in RDF is conducted by Rula et al. [57]. This study categorises various approaches to temporal metadata description into three main types: document-centric, sentence-centric, and relationship-centric. It focuses on evaluating the adoption of these approaches within a substantial dataset of RDF quads – quadruples of the form `<s, p, o, c>`, where `c` specifies the contextual information of an RDF triple `<s, p, o>`. The survey employs quantitative methods, including random sampling and manual verification, to assess the prevalence and accuracy of temporal metadata representations. The evaluation is primarily data-centric, concentrating on how frequently and effectively different approaches are implemented.

In [73, 72], a comprehensive analysis is conducted, organising temporal models into explicit and implicit reification-based approaches. For this, a detailed taxonomy based on technical criteria such as RDF, RDFS, or OWL extension availability, requirement of additional objects, and support for instant or interval time values is developed. This evaluation framework provides a nuanced view of how different models handle temporal data.

Later, Zhang et al. [78] address the need for a more detailed review of storage techniques and categorise temporal approaches based on syntax. Their work focuses on models that extend RDF triples to new forms and those that add timestamp information to standard triples. They emphasise the importance of storage efficiency and practical implementation details. A brief discussion of the contextual annotation in RDF is provided in [36]. Their taxonomy includes direct representation, reification approaches, higher-arity models, and annotation models that define context.

Another study discussed in [60] focuses on the integration of meta-knowledge in the Semantic Web, particularly with regard to time, trust, fuzzy logic, and provenance dimensions. It provides a systematic review and experimental evaluation of various modelling approaches, including their performance in terms of graph size, number of statements, redundancy, storage, query length, and response time.

The work by Polleres et al. [53] provides a conceptual framework for analysing the evolution of open knowledge graphs, identifying dimensions such as terminological drift, structural transformation, and data-level changes. This research offers a typology of KG evolution that is valuable for understanding dynamics. Our work complements this research by grounding the evaluation in specific temporal modelling approaches at the RDF level, focusing on how such models can be applied to represent temporality in CH data. The emphasis is placed on evaluating concrete modelling approaches in terms of their applicability to CH case study, rather than categorising types of change abstractly. In this way, the two lines of research address complementary aspects of temporality in KGs: one through high-level evolution frameworks, the other through practical modelling and evaluation of RDF extensions.

To summarise, the current work provides the following contributions that differentiate it from the related works:

- **Cultural Heritage Focus**: Unlike previous works that focus on generic temporal RDF models, this research specifically addresses the representation of temporal data in CH context, which introduces unique requirements, such as evolving historical facts and concepts, timelines, and complex relationships that change over time.
- **Real-World Use Cases**: The research is application-based and integrates insights from four real-world CH use cases, which inform the identification of critical temporal requirements. This practical perspective is missing from other works, which tend to focus on theoretical models or broad taxonomies without engaging with real-world data needs.
- **Modelling Requirements obtained from CH Use Cases**: To evaluate the models, a set of evaluation requirement is developed to address the specific needs of the CH domain.
- **Comprehensive RDF Extension Analysis**: The work goes beyond merely focusing on technical metrics like storage size or binary criteria such as whether OWL reasoning is supported or not when evaluating the extensions. Instead, an in-depth analysis and discussion of RDF extensions is provided, thoroughly evaluating their capability to handle temporal data in CH settings. This includes exploring the potential consequences of adopting each extension, assessing their suitability for different use cases, and evaluating the extent to which these models address the complexities.
- **Decision-support Tool for Practitioners**: The work proposes a practical, scenario-based decision-support system to help practitioners prioritise and select the most suitable RDF extensions based on specific project needs. Other works offer taxonomies and classifications but lack this actionable, decision-making framework, making it harder for users to apply the findings in real-world (CH) projects.

4 Comparative Evaluation and Discussion

In this section, a qualitative evaluation of six widely used temporal modelling approaches is conducted: N-ary Relations, Named Graphs, Singleton Properties, tRDF, RDF*, and 4D Fluents. These approaches are chosen because they have seen the most practical use in real-world applications. The aim is to provide empirical insights and discussions that deepen the understanding of how each approach meets the requirements, the implications of selecting a particular approach, and the inherent strengths and weaknesses associated with each. This analysis lays a comprehensive foundation for developing a decision-support tool (Section 5), offering context for interpreting its design and outcomes.

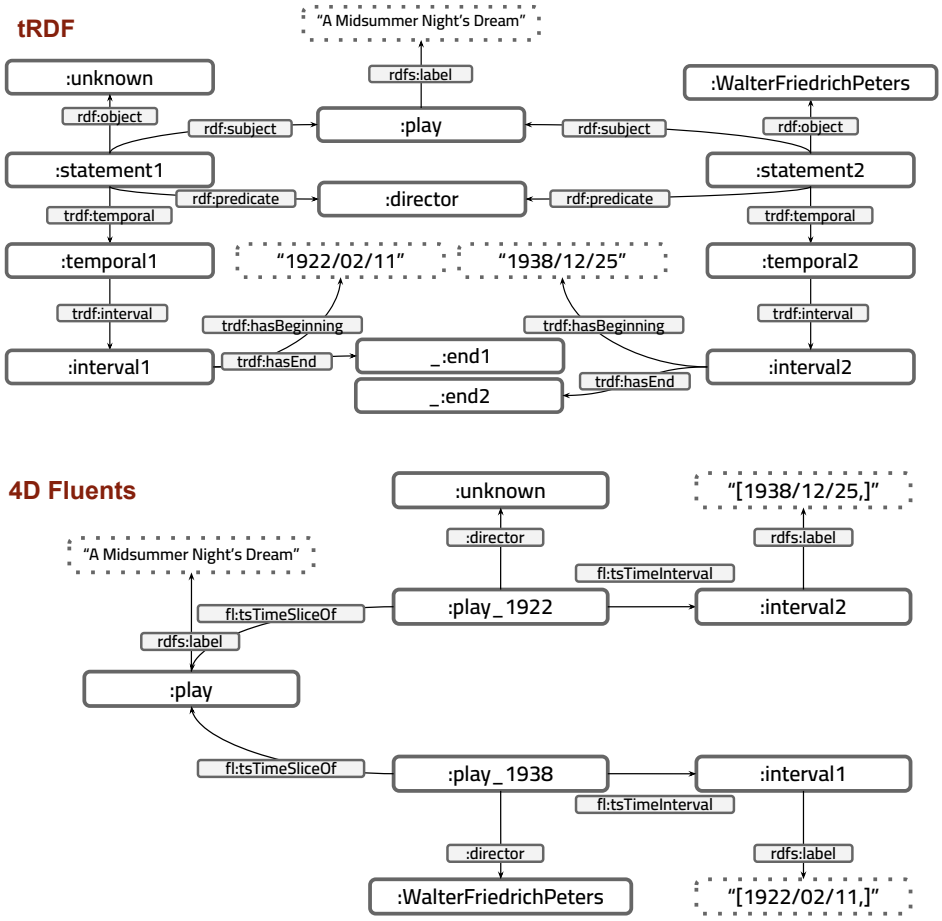


Figure 1 Modelling semantic drift with tRDF and 4D Fluents.

4.1 REQ1: Temporal Expressivity

Temporal expressivity is a core modelling requirement in CH contexts. As discussed in Section 2.2, this requirement is structured into four sub-dimensions – contextual change, temporal statements, temporal uncertainty and incompleteness), and interval relations.

Contextual Change (REQ1.1) is foundational to temporal modelling in CH and is implicitly addressed by all evaluated approaches. It refers to the need to capture how entities and their descriptions evolve over time, e.g. changes in roles, names, functions, or classifications, while maintaining a stable identity. Each model handles this requirement through different structural mechanisms: some reify statements and annotate them with temporal information (e.g., tRDF, RDF*), some transform properties into temporal relations (e.g., N-ary Relations), and some explicitly model temporal slices of entities (e.g., 4D Fluents). These structural strategies enable the representation of contextual change, allowing multiple descriptions of the same entity at different points in time (see Section 3.3 for technical modelling details).

REQ1.1 reflects the need to represent contextual change – that is, the ability to track how the description of an individual entity evolves across time without changing its identity. These temporally-scoped descriptions are crucial in CH, where narratives are rarely static and must

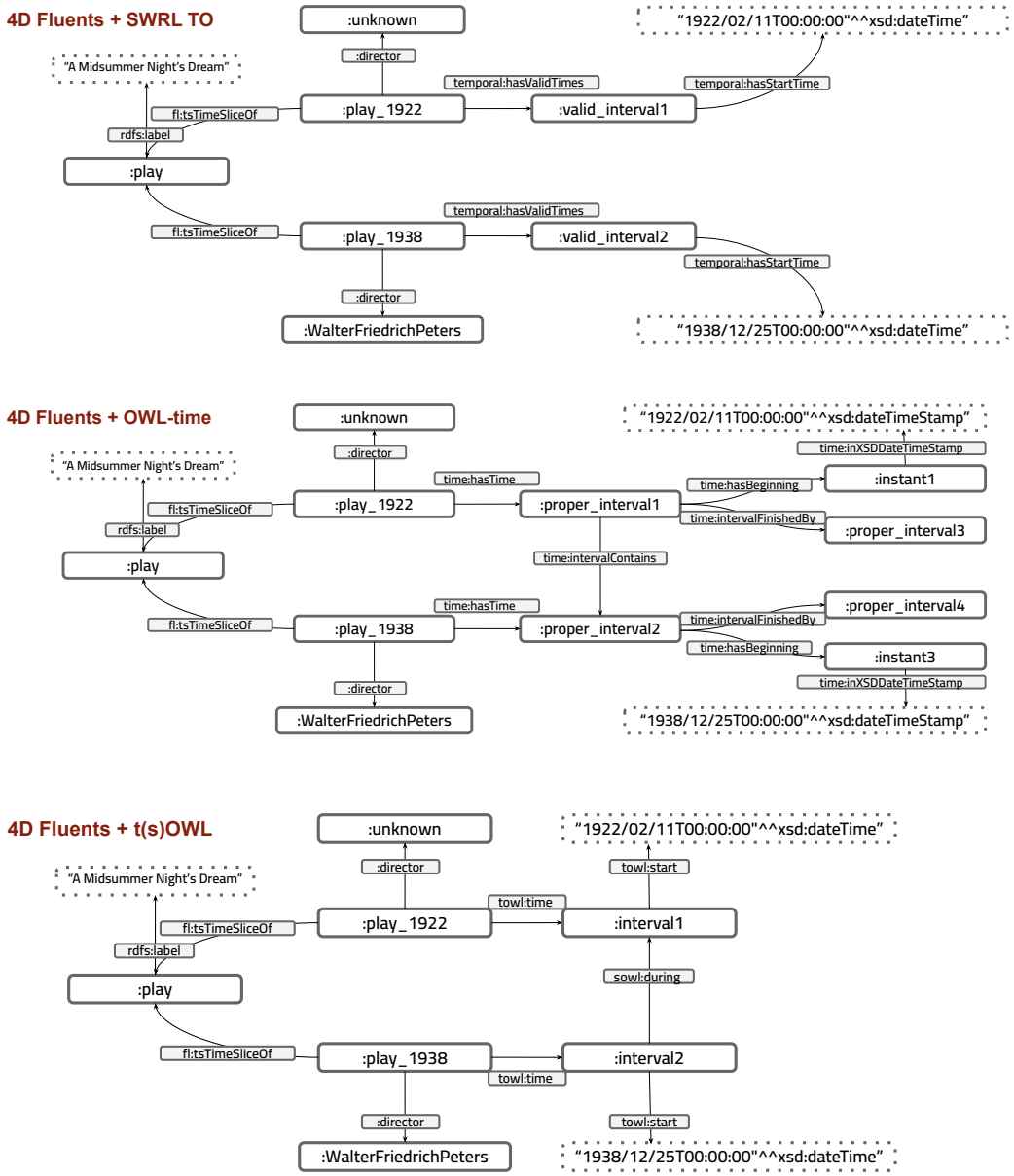


Figure 2 Modelling semantic drift with temporal ontologies – SWRL Temporal Ontology, OWL-Time, and tOWL.

accommodate contested or evolving facts. While REQ1.1 forms the conceptual foundation, the approaches diverge significantly in how they support more specific temporal capabilities, e.g. representing temporal scopes, modelling uncertainty in temporal data, or supporting relations between time intervals. The following discussion addresses each of these dimensions in turn, examining how different models fulfill these CH requirements. To support the discussion, the modelling of a contextual change scenario is visualised:

> *A play titled "A Midsummer Night's Dream" was performed starting from 1922/02/11. On 1938/12/25, a new director was hired, leading to a conceptual change in the storyline of the play, resulting in significant differences during a certain period.*

Figure 1 depicts this scenario using tRDF and 4D Fluents, while Figure 2 shows representations using external temporal ontologies.

REQ1.2: Temporal Statements. Time instants and intervals are crucial temporal concepts in RDF-based temporal modelling. In tRDF, dedicated vocabulary like `trdf:Instant` and `trdf:Interval` is used, and intervals are defined by linking them to their start and end time points via properties like `trdf:hasBeginning` and `trdf:hasEnd`. The entailment of the temporality to a statement succeeds via property `trdf:temporal` which time borders are then further specified via `trdf:instant` or `trdf:interval`. In the 4D fluents model, temporal vocabulary involves introducing two classes: `fl:TimeSlice` to represent slices of entities in a specific time, and `fl:TimeInterval` to represent periods of time. Additionally, `fl:tsTimeSliceOf` property links a time slice to its generic entity, and the `fl:tsTimeInterval` property connects a time slice to a time interval, thus associating temporal details with each time slice.

The SWRL Temporal Ontology [50] is a lightweight approach that can be layered on existing OWL languages to add a temporal dimension to a standard model. The core class of the ontology is `temporal:Fact`, it models a temporal individual that is connected to time periods when it is held to be true via the property `temporal:hasValidTimes`. The range of the property is the class `temporal:ValidTime`, which has sub-classes `temporal:ValidInstant` and `temporal:ValidInterval`. The borders of the interval are modelled via connecting intervals to the time points of their beginnings and ends via properties `temporal:hasStartTime` and `temporal:hasFinishTime`. The instants are linked to temporal data using properties like `temporal:hasTime`, which connects an instant to a precise time value (e.g., a date or timestamp). Additionally, to address the granularities of intervals and points, e.g. days and minutes, class `temporal:Granularity` is introduced.

tOWL (Temporal OWL) [21] is an extension of OWL designed to incorporate temporal reasoning within ontologies. tOWL offers a specialised framework for temporal modelling with classes like `towl:Instant` and `towl:Interval` for representing time instants and intervals. Instants of tOWL are represented using XML Schema `dateTime`, capturing a specific point in time. Intervals are modelled with properties `towl:start` and `towl:end`, representing the beginning and end of a time span. These properties rely on the concrete domain based on rational numbers to ensure the start is always strictly before the end, enforcing proper interval definitions. Similarly to 4D Fluents, `towl:TimeSlice` describes temporal aspects of entities, associating them with a period using `towl:time`.

OWL-Time provides[10] the most expressive and granular ontology for time instants and intervals. It starts with the class `time:TemporalEntity` and its two primary sub-classes: `time:Interval` and `time:Instant`. The vocabulary includes properties `time:hasBeginning` and `time:hasEnd`

[10] `https://www.w3.org/TR/owl-time/`

to describe temporal limits of an interval and to link it to the entity of type `time:Instant`. Additionally, intervals can be linked to `time:Duration` via `time:hasTemporalDuration` to represent an extent of an interval, expressed as a scaled value or a nominal value. Instants can be specified with properties like `time:inXSDDateTimeStamp, time:inXSDgYear, time:inTimePosition, time:inDateTime` to provide various ways of indicating their temporal position.

REQ1.3: Temporal Uncertainty and Incompleteness. In [41], the SWRL Fuzzy Temporal Ontology (SWRL-FTO), extension of the SWRL temporal ontology is introduced to address modelling and reasoning with imprecise temporal expressions. Thus, the `fuzzytemporal:FuzzyTime` class represents imprecise or uncertain time values associated with fuzzy temporal propositions. It has two main sub-classes: `fuzzytemporal:FuzzyTimeInstant` and `FuzzyTimePeriod` that represent an event spanning an imprecise time instant and interval with uncertain start and end times accordingly. The `fuzzytemporal:FuzzyModifiers` class includes terms like "about" or "around" that describe the fuzziness of temporal information, helping to quantify this uncertainty. Although the approach introduces a promising concept for handling imprecise temporal expressions through the SWRL-FTO ontology, further evaluation and empirical evidence, such as a concrete OWL implementation or experimental results, are needed to fully demonstrate its practical effectiveness.

A recent contribution to temporal modelling is the Onto-mQoL framework [61], which offers a solution for representing both precise and uncertain temporal information. Onto-mQoL builds on the n-ary relation approach. To capture both precision and uncertainty, the framework introduces specialised temporal classes. For precise temporal values, it defines `PreciseMoment` and `PreciseInterval`. In contrast, for modelling vagueness and uncertainty in temporal expressions, it defines classes `UncertainMoment` and `UncertainInterval`. An uncertain moment includes a reference point, a degree of imprecision (`hasUncertainRange`, typed as a duration), and a functional modifier (`hasUncertainFunction`, e.g., "about", "before", "early"). Similarly, an uncertain interval allows both a precise moment and an uncertain moment at its temporal boundaries. Custom SWRL rules are defined to infer temporal relationships such as `almostSurelyBefore`, accounting for the fuzziness inherent in uncertain intervals. To support such expressive logic, Onto-mQoL relies on SQWRL [49]. Overall, Onto-mQoL presents a robust and OWL DL–compliant approach for modelling both precise and uncertain temporal aspects. However, due to its high expressivity and reliance on SWRL-based reasoning, formulating queries becomes complex and less accessible, particularly in comparison to standard SPARQL workflows. Future work may investigate its practical applicability in CH contexts, including the learning curve for non-technical users and opportunities for optimising rule management and querying efficiency.

Alternatively, temporal uncertainty can be represented in KGs either textually through plain literals, offering a straightforward but less formalised solution, or through translating them into qualitative intervals, which provide a more structured approach but still lack precise definitions. However, for more advanced representations (and reasoning) about uncertain dates, normalisation techniques are necessary. While this work does not cover these techniques, tools like HeidelTime [63, 62] specialise in detecting and normalising temporal expressions in text. HeidelTime employs pattern matching to identify various temporal references and can be extended to enhance its coverage and accuracy, thus offering a more robust solution for managing temporal uncertainty.

REQ1.4: Interval Relations. For many use cases, it is crucial not only to represent the temporal boundaries of facts, but also to express how time intervals relate to one another, e.g. whether one interval precedes, follows, overlaps another. Such temporal relations become even more important in the presence of temporal uncertainty or incompleteness (see REQ1.3). In [37] a formalism known as c-temporal graphs, which extends traditional temporal RDF by integrating

constraints and interval algebra is introduced to model relations between intervals with imprecise or unknown boundaries. sOWL [10] extends tOWL by supporting qualitative temporal relations between intervals, such as `sowl:during`. This property allows for the representation of how one interval can occur during another, without needing to define their concrete start and end points. SWRL Temporal Ontology also integrates functions based on Allen's interval algebra, providing operators like `temporal:before, temporal:overlaps, temporal:contains`. These functions enable reasoning about the relationships between intervals based on qualitative aspects, regardless of whether the exact start and end points are specified. OWL-Time is primarily designed for representing and reasoning about temporal entities with explicitly defined start and end points, and provides mechanisms for detailing temporal ordering and relationships with 15 interval relations [3] such as `time:intervalBefore, time:intervalAfter, and time:intervalMeets`. The authors acknowledge that temporally incomplete intervals and their relations can be modelled within OWL-Time, but restrict themselves from addressing reasoning about such intervals, stating that additional methods or extensions may be necessary beyond what is provided by OWL-Time.

Overall, the approaches reviewed in this section reflect a wide spectrum of solutions for achieving temporal expressivity in CHKGs. All models aim to support contextual change, but they differ in their structural strategies and in their expressive power and complexity. Native support for time instants and intervals is uneven: it is built-in in tRDF, 4D Fluents, tOWL, and OWL-Time, while RDF* and Named Graphs leave the temporal domain to be defined externally. Only a subset of models, e.g. SWRL Temporal Ontology, SWRL-FTO, and Onto-mQoL, provide mechanisms to address temporal uncertainty and incompleteness, though with varying degrees of maturity and practical implementability. Similarly, modelling qualitative interval relations is not natively supported by most models and typically requires external ontologies (e.g., sOWL, OWL-Time) or rule-based extensions based on Allen's interval algebra. More expressive frameworks such as Onto-mQoL introduce robust capabilities for handling vague or uncertain temporal expressions but rely on custom SWRL rules and SQWRL querying, which can be complex to implement and query in practice.

Although no single approach fully satisfies all four dimensions of REQ1, several offer strong partial support depending on the modelling priorities (e.g., clarity vs. expressiveness) and infrastructure requirements (e.g., tool compatibility, reasoning support) of the use case. This highlights the importance of individual use case-aware model selection.

4.2 REQ2: Storage and Scalability

To assess this requirement, the number of RDF triples required by each approach in a representative modelling scenario is compared. While not a standalone measure of scalability, triple count provides a practical and widely adopted proxy for estimating modelling overhead. To assess the number of triples utilised by each temporal modelling approach, an example from use case 1 (see Section 2) has been used to illustrate a terminological change of context over time. Specifically, the example models the renaming of a street:

Until 1809, the street was called "Roßmarkt," but after 1809, it was renamed to "Adlerstraße".

This example is depicted in Figure 3 and Figure 4, and serves as a basis for evaluating how different RDF temporal extensions handle this simple yet common scenario[11]. For the purpose of this evaluation, only the ABox is considered when counting triples, and the definition of classes and class memberships is excluded to simplify the analysis. The focus is placed on the number of triples required to model the temporal aspects of the change, thereby providing a clear comparison of how each approach handles temporal data in terms of scalability and efficiency.

[11] All examples are available at: `https://github.com/sashabruns/RDF-Extensions-for-Cultural-Heritage/blob/main/CR1_number_of_triples.md`

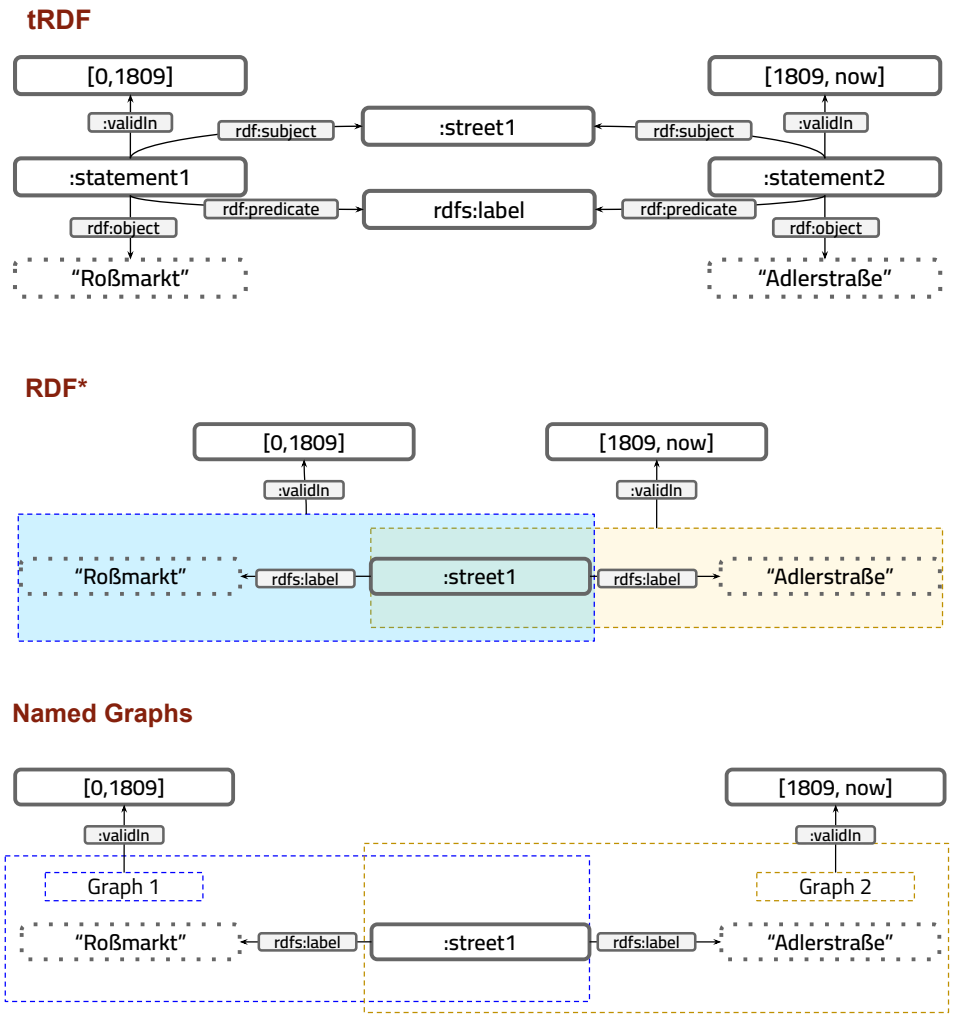


Figure 3 The running example of the triple and graph level extensions to RDF.

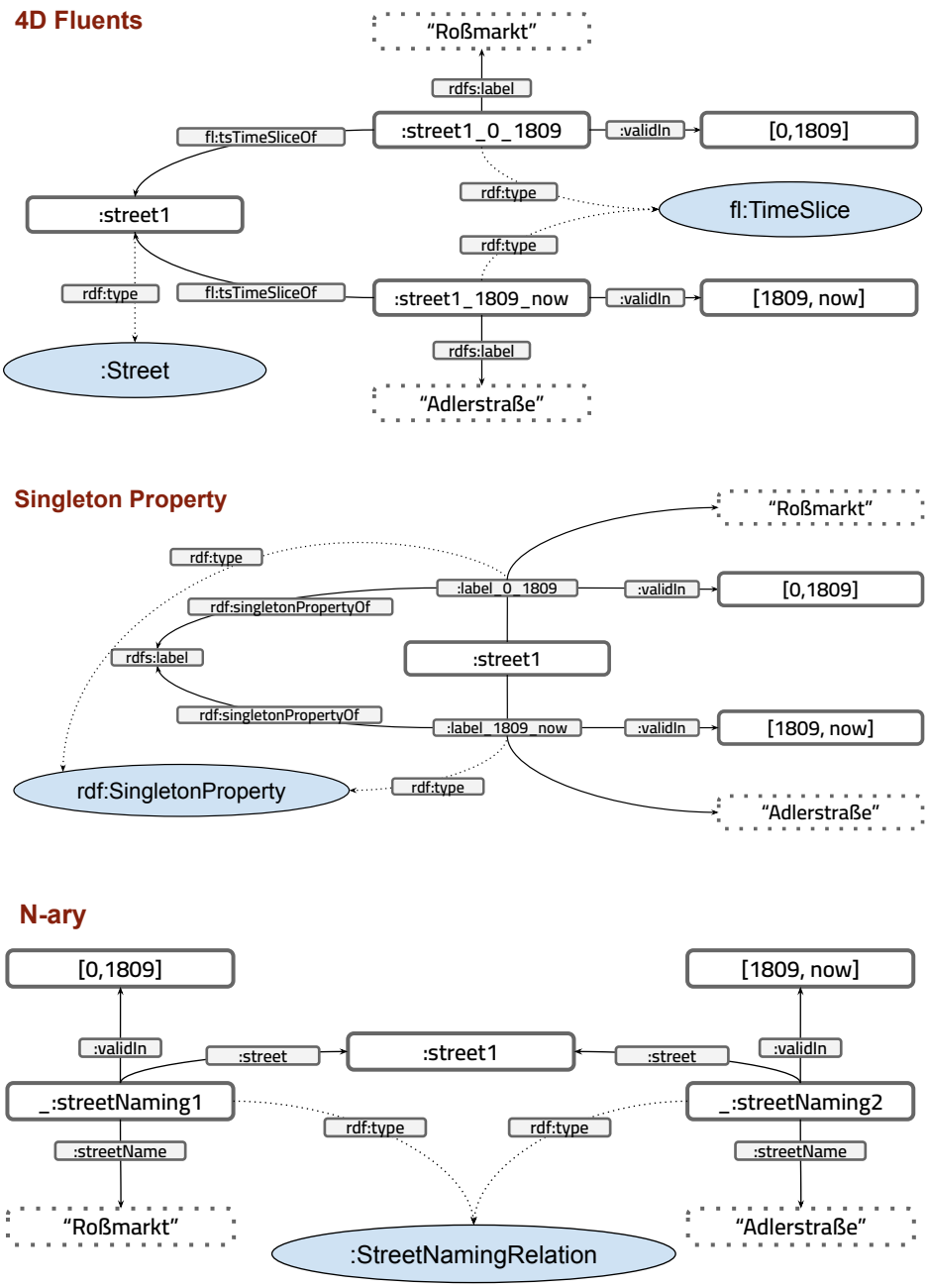


Figure 4 The running example of the component level extensions to RDF.

In the tRDF approach, each temporal statement is explicitly modelled using a combination of triples. Therefore, each statement about the street's label over time results in multiple triples, leading to a high count of triples – 8. RDF* uses reification to embed temporal validity directly within the triple structures. By integrating the temporal information with reified triples, and using RDF* syntax, RDF* achieves a more compact representation – 4 triples. Named Graphs represent temporal information by asserting RDF triples in different named graphs, where each named graph corresponds to a temporal snapshot. Triples that are valid within the same time interval are grouped under the same named graph, while additional RDF triples describe the temporal validity of these graphs. In the example, this results in two triples asserted in named graphs and two triples describing their temporal validity. The 4D Fluents approach involves creating separate entities for each time slice of the street, with associated labels and validity periods. Each time slice is modelled as an independent entity, resulting in multiple triples for both the temporal statement and the association with the original entity (6 triples). In the Singleton Properties approach, additional unique predicates are used to represent each temporal change (6 triples). The N-ary Relations approach models temporal changes by objectifying the relations between entities. In the example, separate entities to link the street with its names and validity periods are created that results in 6 triples.

Examining the representation of only one street provides a vision how each approach behaves but may not fully visualise the possible scalability challenges and efficiency differences between approaches when the amount of data increases. With a single street, the information is relatively unique and limited, leading to similar results across approaches. However, when additional information is introduced – such as name changes for four streets simultaneously – the differences between the approaches become more pronounced. Thus, in tRDF, each street and its associated changes require multiple triples, with each temporal statement needing its own set of triples. For the example involving four streets, it results in 32 triples in total. RDF* requires 16 triples (8 statements, each with associated temporal validity) to model four streets. Named Graphs group triples that are valid in the same time period under a single named graph, thereby consolidating the information. In this example, 8 triples are asserted in named graphs, and 2 additional triples describe the temporal validity of the two graphs. 4D Fluents, Singleton Properties and N-ary relations involve creating distinct entities for a temporal change. With four streets, the total number of triples is 24.

The analysis is visualised on the example where the object of the triples is a literal value (street names). To explore the impact of using entities instead of literals, further scenarios were considered, where the triples involve entities, such as when annotating temporal facts, e.g. `«:person1 :livesIn :street1> validIn '1908'>, «:person1 :livesIn :street2> validIn '1910'>`. The results for most approaches remained consistent with those observed for literals. However, in the case of 4D Fluents, using entities rather than literals results in an increased number of required triples. This is because both entities, in the subject and object positions, need to be associated with their respective time slices, further complicating the model and requiring additional triples. This extension underscores the added complexity and potential scalability issues that arise when modelling temporal changes with entities instead of literals. Singleton Properties would also see an increase in the number of triples needed if the object is an entity. Similar to 4D Fluents, each temporal change would require creating a new unique predicate, resulting in additional triples.

While counting number of RDF triples is a convenient and quantifiable proxy for modelling complexity, it should not be viewed in isolation or as a definitive measure of model performance. The evaluation in this section is based on small, illustrative examples, which do not capture the full variability of real-world CH datasets. However, triple count has been shown to correlate

with broader performance measures such as data loading time and repository size. For instance, the Ontotext study[12] benchmarks various RDF modelling approaches on a subset of Wikidata, showing that RDF*, which requires fewer triples, significantly outperforms traditional reification and N-ary relations in terms of both repository size and load time.

4.3 REQ3: RDF Syntax Extension

Introducing temporal context to standard RDF often requires extending or adapting its syntax. While this is a technical consideration, it remains relevant to CH contexts where RDF-based systems are often pre-defined and tool usage may be constrained. Many CH infrastructures rely on fixed tools, and deviations from standard RDF syntax can limit compatibility and practical applicability. REQ3 therefore examines whether a temporal model introduces such syntax extensions and what concrete syntaxes (e.g., TTL, TriG, TTL*) it requires.

tRDF does not alter the fundamental RDF triple structure and can be implemented using standard RDF constructs. This makes it compatible with existing RDF tools and serialisations such as Turtle or RDF/XML. RDF* extends the RDF data model by allowing triples to be embedded within other triples, which simplifies metadata annotations, including temporal ones. Although it leads to a more streamlined representation, RDF* requires specialised syntax (TTL*) and querying tools (e.g., SPARQL*), which may not be available or permitted in institutional CH settings. While RDF* can be translated into standard reification, its full usability depends on support for these extended tools. In Named Graphs, each graph represents a distinct context within a dataset. For this, triples asserted in named graphs are commonly represented together with a graph identifier, a representation often referred to as a quad. This mechanism is well supported in most RDF frameworks and standardised in RDF and SPARQL. However, to fully utilise named graphs, dataset serialisations such as TriG are typically required, which introduce additional syntactic complexity compared to simpler graph-level formats such as Turtle.

Singleton Property extends RDF by creating unique predicates to express context, including time. Though it adheres to the RDF triple model, this approach increases the number of distinct properties significantly. Without specialised support to generate, maintain, and interpret these properties, the model can become error-prone and hard to manage, especially in long-lived CH projects. 4D Fluents introduce time slices as additional entities, linking them to base entities and time intervals. This modelling strategy is compatible with RDF but structurally complex, requiring careful construction of time-related entities and relations to ensure consistency. N-ary Relations transform simple binary assertions into more complex structures involving intermediate nodes (reified events or states). This approach does not require syntax extensions and is fully RDF-compliant, but adds modelling overhead by increasing the graph's depth and indirection.

Overall, the models vary significantly in the extent to which they modify or extend the standard RDF syntax. tRDF and Named Graphs introduce no or only minor changes to the RDF structure and are considered highly interoperable and easily adoptable. RDF* requires extended syntaxes for full usability, however, it can be easily translated into standard RDF. Singleton Property maintains RDF compliance but introduces a large number of unique predicates, complicating maintenance and tool support. 4D Fluents requires explicit modelling of time slices and temporal linkage structures, adding a significant representational layer. N-ary Relations remain syntactically within RDF, but their reliance on intermediate reified nodes and objectified relations create deep graph structures.

[12] https://www.ontotext.com/blog/rdf-reification-wikidata/

4.4 REQ4: Model Semantics

This requirement considers whether an approach can be expressed using standard RDF(S) semantics or whether it relies on more advanced semantic frameworks. This is relevant in CH contexts, where the modelling of temporal change often involves complex identity, classification, and constraint structures. The degree of semantic support influences both the expressiveness and the practical feasibility of a given approach.

When incorporating temporal information into RDF, the approaches vary in how much semantic richness they require. Some approaches, e.g. tRDF and RDF* stay within RDF(S) and aim for lightweight integration. For instance, RDF* uses embedded triples to capture temporal metadata while preserving compatibility with the RDF(S) model. These approaches are advantageous in settings with limited semantic infrastructure or tool constraints. 4D Fluents, N-ary Relations, and Named Graphs can be implemented using RDF(S), but are typically used with OWL to capture more complex semantics. For example, 4D Fluents introduce temporal slices of entities, which require properties like fl:tsTimeSliceOf and class axioms to maintain coherence between temporal parts parts and the main static entity. Such semantics are difficult to enforce purely in RDF(S) without OWL constructs. Similarly, N-ary Relations rely on intermediate nodes to connect entities and their temporal qualifiers. RDF(S) can represent this structure syntactically, but cannot express constraints like functional or cardinality restrictions on these connections. Named Graphs offer a standardised RDF extension for contextualising triples, but RDF(S) provides no native means of reasoning across graphs or expressing constraints between them. Semantic interpretation of graph boundaries or relations between graphs must be handled externally or with custom logic. Singleton Property introduces a new predicate for each contextualised statement, and requires an explicit semantic mapping (via rdf:singletonPropertyOf) to relate each singleton property to its generic property. This mapping must be explicitly interpreted to preserve entailment.

For the purposes of evaluation, this requirement is assessed in isolation, focusing solely on the semantic expressiveness required to support temporal modelling. From this perspective, an approach that achieves temporal entailment with simpler semantics (e.g., RDF(S) rather than OWL or rule-based logic) is considered advantageous in terms of conceptual and technical overhead. However, this does not imply that more expressive approaches are weaker overall. In practice, the semantics requirement must be balanced with usability, performance, and institutional tooling constraints.

4.5 REQ5: Query Language

In this section, the aim is not to analyse how different SPARQL engines handle RDF data in practice (for this see, e.g. [1]) but to provide overview of how querying is achieved when using different RDF extensions and approaches. By understanding the querying mechanisms associated with each approach, their usability and compatibility with existing semantic web technologies can be better evaluated. While REQ5 and REQ3 are closely related, they capture two distinct dimensions of technical compatibility. REQ3 addresses whether an approach adheres to standard RDF syntax, while REQ5 focuses on whether it remains compatible with native SPARQL querying workflows.

Approaches such as N-ary Relations and tRDF replace standard relations by introducing new concepts – `rdf:Statement` for tRDF and `:Relation` for N-ary Relations – and associating these with temporal labels. This additional layer of abstraction complicates querying because it requires working with these new entities or statements that encapsulate triples, rather than querying the triples directly. As a result, the model's complexity increases, making the querying process less intuitive. Named Graphs, on the other hand, use standard static predicates but involve creating a

number of graphs, each associated with unique timestamps. Managing these numerous graphs can lead to graph overload, making the querying process cumbersome. The need to handle many individual graphs can result in performance issues and increased difficulty in formulating queries.

Despite these complexities, these approaches can be queried using native SPARQL without requiring extensions beyond those already supported for standard RDF. In these cases, temporal information is either embedded directly within triples using RDF reification (as in tRDF) or encapsulated within named graphs, which SPARQL naturally supports. Thus, these approaches score high in terms of compatibility with native SPARQL due to their straightforward representation of temporal data.

RDF* necessitates an extension of standard SPARQL, SPARQL*, to handle nested triples effectively. Although more and more query engines today provide support of SPARQL* (e.g. GraphDB[13], Apache Jena[14]), it is crucial for users choosing the RDF* approach for modelling their temporal data to be aware of the querying capabilities of the specific engine they are planning to use. This awareness ensures that the chosen engine can fully leverage RDF*'s enhanced expressiveness without compromising performance or functionality in querying.

Singleton Property, on the other hand, does not require a SPARQL extension. Instead, it can be integrated within the existing SPARQL framework, allowing for the direct association of (temporal) metadata with RDF triples without modifying the query language itself. However, the use of Singleton Property introduces its own complexities in querying. When dealing with datasets that incorporate Singleton Properties, several considerations come into play [46]. In particular, the nature of the dataset significantly influences the querying process. For datasets containing only Singleton Property triples, standard SPARQL queries may produce empty results due to the absence of traditional RDF triples. To address this, it is essential to ensure that 1) the query engine supports RDFS inference to compute the RDFS closure, and 2) that the schema includes the specific axiom `<rdf:singletonPropertyOf rdfs:subPropertyOf rdfs:subPropertyOf>`. On the other hand, when a dataset comprises both RDF triples and Singleton Property triples, querying becomes more straightforward, but can introduce redundancy. Including inferred RDF triples derived from Singleton Properties can enhance query performance; however, it may also lead to inconsistencies if the data triples are removed while the Singleton Property triples remain. This highlights a trade-off between achieving efficient query performance and maintaining dataset consistency. Understanding these dynamics is crucial for effectively managing and querying datasets that utilise Singleton Property approach, ensuring a balance between query efficiency and data integrity.

Similarly, 4D Fluents involve more intricate representations where entities are often associated with time slices or involve multiple relationships that go beyond simple triple patterns. In such cases, while native SPARQL can still be used, it often requires custom solutions or more complex query patterns to effectively retrieve the desired information. For instance, querying relationships like `<:John@1908 :wife :Mary@1908>` requires searching through instances of `:TemporalSlice` instead of `:Person`, complicating the query structure. Additionally, relationships must often be explicitly modelled to account for their temporal dimensions, introducing redundancy and further complicating queries.

In contrast, some approaches go even further, requiring entirely new query languages to manage the complexities introduced by their specific temporal representations. For example, Annotated RDF or Extended 4D Fluents may require dedicated languages like AnQL [43] or TOQL [9], respectively, because the standard SPARQL framework cannot easily express the nuanced semantic

[13] `https://graphdb.ontotext.com/documentation/10.7/rdf-sparql-star.html`
[14] `https://jena.apache.org/documentation/RDF*/`

relationships or nested temporal structures these models introduce. The need for these specialised languages arises from the fact that these approaches fundamentally change how data is structured and how temporal information is linked to RDF entities, making standard SPARQL insufficient for querying without significant extensions.

4.6 REQ6: Use of RDFS/OWL Reasoning

Incorporating temporal context into RDF models often requires extending the basic RDF framework, which can have significant implications for OWL and RDF(S) reasoning. Reasoning relies on well-defined logical constructs, such as class hierarchies, property relationships, and inverse properties, to infer new knowledge from existing data. However, when temporal dimensions or other complex structures are introduced, these logical constructs can become less effective or even incompatible.

4D Fluents maintain strong RDF(S)/OWL reasoning capabilities by integrating temporal dimensions within the OWL framework itself. This approach represents entities and their temporal parts as distinct classes preserving standard RDF triple structure and exploiting original properties. Thus, OWL property constructs like inverse properties and transitivity are fully supported. However, on the other hand, when changing the original class belonging, reasoning over class hierarchies becomes more complicated. For example, given the RDF data:

```
:Director rdfs:subClassOf :Person .
:director rdfs:range :Director .
:play@1938 :director :WalterFriedrichPeters@1938 .
:WalterFriedrichPeters@1938 :timeSliceOf :WalterFriedrichPeters .
```

A reasoner will infer that `:WalterFriedrichPeters@1938` is a `:Director` and, by subclass logic, a `:Person`. However, the reasoning engine does not automatically infer that `:WalterFriedrichPeters` (the generic entity) is a `:Director` or `:Person` just because its time slice `:WalterFriedrichPeters@1938` is. To enable these inferences to the generic entity, additional rules or custom reasoning mechanisms are needed beyond standard OWL.

Named Graphs do not change the structure of RDF triples but organise triples into separate graphs within an RDF dataset. This allows standard OWL reasoning to be applied within individual named graphs. However, practical support depends on the triple store's ability to perform reasoning and query processing over named graphs. In particular, querying patterns that extensively rely on the graph component may lead to performance degradation, as the efficiency of querying within named graphs varies significantly across implementations.

Due to reification in tRDF, the original triple's properties are encapsulated within the `rdf:Statement` instance, making them opaque to standard RDF(S)/OWL reasoning. For example, the triple `<:play :director :WalterFriedrichPeters>` is reified as `<:statement1 rdf:predicate :director .>`. This encapsulation turns the original properties into objects of new triples, which disrupts direct reasoning. As a result, defining `<:director owl:inverseOf :directed>` does not provide any inference results because OWL cannot infer across the boundary created by reification. Furthermore, reification itself does not imply the existence of the original triple, nor does the original triple imply the existence of its reified version, significantly limiting the ability to reason about the original triple's metadata or related relationships. Similarly, in the N-ary Relation approach, the property `:director` is transformed into an entity rather than a direct property, e.g. `:Directing`, and the related information is linked to the new entity, e.g. `<:Directing :directorEntity :WalterFriedrichPeters>`. This transformation results in the loss of the direct connection between `:play` and `:WalterFriedrichPeters`, as this relationship

is now mediated through the new entity `:Directing`. Consequently, enabling reasoning in this context would require more sophisticated reasoning rules and potentially custom constructs to interpret these indirect relationships effectively.

In the Singleton Property approach, a unique instance of the property is created for each specific relationship to represent temporal information. For example:

```
:Director rdfs:subClassOf :Person .
:director rdfs:range :Director ;
          owl:inverseOf :directed .
:director_1938 rdf:singletonPropertyOf :director ;
               :validIn "1938" .
:play :director_1938 :WalterFriedrichPeters .
```

The property `:director_1938` is a singleton property of `:director`, meaning it is treated as a unique instance associated with the year 1938. However, because it is not explicitly defined as a sub-property of `:director`, the `owl:inverseOf :directed` relationship does not automatically apply. This means that the triple `<:WalterFriedrichPeters :directed :play>` will not be inferred. Similarly, the range of `:director_1938` does not automatically lead to the inference that `:WalterFriedrichPeters` is a `:Director` or a `:Person` because the sub-class relationship depends on recognising the singleton as a sub-property of the generic property. To enable these kinds of standard OWL inferences requires several custom solutions to enable efficient reasoning, e.g. defining inverse properties for each singleton, explicitly defining each singleton property as a sub-property of the original property, defining custom reasoning rules, e.g. `<rdf:singletonPropertyOf rdfs:subPropertyOf rdfs:subPropertyOf .>`, etc. However, each additional rule or definition significantly increases the number of triples in the dataset. This not only complicates the RDF model but also leads to performance concerns, as reasoning engines must process a larger and more intricate set of rules and triples. As a result, reasoning with singleton properties in an RDF context becomes cumbersome and less efficient.

RDF* introduces a mechanism for annotating triples with (temporal) information, which results in a significant challenge in RDF(S)/OWL reasoning due to RDF*'s referential opacity. In RDF*, there is a distinction between a quoted triple and an unquoted triple, as shown in the following example:

```
:Director rdfs:subClassOf :Person .
:director rdfs:range :Director ;
          owl:inverseOf :directed .
«:play :director :WalterFriedrichPeters» :validIn "1938" .
```

Here, the quoted triple `«:play :director :WalterFriedrichPeters»` represents a specific statement and is associated with the year 1938. However, because of RDF*'s referential opacity, RDF(S)/OWL reasoning does not treat this quoted triple the same as the unquoted triple `<:play :director :WalterFriedrichPeters>`. This means that the inverse property relationship or a class belonging from the range definition of `:director` does not apply to the quoted triple, and the inference, e.g. `«:WalterFriedrichPeters :directed :play»` is not generated. This limitation arises because OWL reasoners do not inherently recognise a quoted triple as denoting the same relationship as its unquoted counterpart. This distinction exists for a reason: the quoted triple is only valid within its specified context, meaning the relationship it describes is context-dependent. As a result, it cannot be inferred that the inverse property necessarily applies to the unquoted version, as this relationship may not hold universally outside the given context. To address this for specific contexts, RDF* introduces the concept of Transparency Enabling Properties

(TEPs[15]), which can be used to selectively enable referential transparency for specific properties. For example, stating `<:validIn rdf:type RDF*:TransparencyEnablingProperty>` will ensure that a reasoner considers the quoted triple as equivalent to the unquoted triple, thus allowing the inverse relationship and other inferences to apply. However, this selective transparency only applies within the context of the RDF* graph where the TEP is defined, which can become cumbersome when working with multiple RDF* graphs. While TEPs provide a mechanism to address the limitations of RDF*'s referential opacity, their use requires careful consideration by users in determining which properties should be defined as transparent. Additionally, users must ensure that reasoning engines are correctly configured to interpret these properties, adding another layer of complexity to the reasoning process. This complexity is particularly seen in cases where RDF* data from various sources is integrated or queried together, as inconsistencies in TEP definitions across graphs could lead to undecidability in reasoning results.

Overall, evaluating various approaches against this criterion aims to reveal their effects on standard reasoning. The discussion shows that all approaches require some form of adaptation to achieve effective reasoning. While certain solutions, such as TEPs in RDF* or Named Graph reasoning mechanisms, are developed directly by the authors, others necessitate user-created custom rules. Therefore, when selecting an approach, it is essential to consider not only the built-in reasoning capabilities but also the potential need for user-defined modifications to ensure optimal performance.

4.7 REQ7: Ease-of-Use

In CH, knowledge graph users are typically domain experts rather than computer scientists or ontology engineers. These users may lack extensive technical training in data modelling but are nonetheless expected to integrate, interconnect, and retrieve complex historical data through semantic technologies. Therefore, evaluating the *ease of use* of temporal modelling approaches is crucial for ensuring broad adoption and meaningful participation in semantic knowledge engineering across CH communities.

To assess this requirement, a targeted usability survey is conducted. This survey is designed as a study to capture the intuitions and practical preferences of CH professionals. Its goal is to complement technical assessment with user-centred insights, particularly regarding which models are perceived as most intuitive and accessible. This aligns with the broader objective of our work, in particular to support informed, context-sensitive modelling decisions through decision-support mechanisms tailored to CH practice.

- **Survey Procedure.** The survey took approximately 10 minutes to complete. Participants first answered demographic questions about their educational and domain background, as well as their experience with data modelling or knowledge engineering. The core part of the survey involved a set of pairwise modelling comparisons. For each comparison, participants were asked to select which they found easier to understand, more intuitive, or generally better suited for their work. They were encouraged to briefly justify their preferences, providing qualitative insight into their choices.
- **Survey Setup.** To ensure consistency and comparability across participants, the survey used a fixed example: a simple natural language sentence describing a historical change "Until the year 1809, the street was called 'Roßmarkt'. In 1809, it was officially renamed to 'Adlerstraße'."[16]

[15] `https://www.w3.org/2021/12/RDF*.html#RDF*-vocabulary`

[16] This example is adapted from data used in the TRANSRAZ project. The primary historical sources in this project are the annual address books (*Adreßbücher*) of the city of Nuremberg. Changes in street names over time, such as this one, represent a typical temporal modelling scenario encountered in the data.

This sentence was then represented using different modelling approaches (e.g., 4D Fluents, RDF*, tRDF, etc.), which were shown two at a time for pairwise comparison, see Figure 5. Each representation was schematically visualised and participants were asked to evaluate which version they found easier to understand or more intuitive. The same example was used throughout the survey to isolate the modelling approach as the only varying factor and prevent differences in content from influencing the results. Participants compared each approach against every other, resulting in a full pairwise comparison across all methods. To reduce learning effects and avoid biased results due to repetition or familiarity, the ordering of the pairs was randomised as much as possible. It was specifically ensured that no representation was shown in two consecutive questions, and that each approach appeared roughly equally in both the top and bottom positions across different comparisons.

- **Participants.** The survey was completed by 22 participants with diverse backgrounds. This mix enabled us to explore how ease-of-use perceptions vary across CH professionals and technically trained participants. Key statistics are as follows[17]:
  - **Domain Background:** 58% cultural heritage experts (History, Digital Humanities, Archaeology, Numismatics, Musicology, Theology, etc.), 38% computer scientists working on CH data and technologies (Semantic Web, ontologies, data mining, etc.).
  - **Educational Background:** 54.5% hold a PhD, 41.4% a Master's degree.
  - **Modelling Experience:** 81.8% reported prior experience with data modelling or knowledge engineering.
- **Results.** As shown in Figure 6, 4D Fluents emerged as the overall favorite, praised for its intuitive and clear structure. RDF* also received strong support, particularly for its syntactic clarity and minimal complexity. Named Graphs were well received among cultural experts, likely due to their conceptual similarity to archival structures. Meanwhile, tRDF and N-ary Relations were appreciated by some participants for their explicitness and flexibility, though these approaches were generally seen as more complex. The direct comparison between 4D Fluents and RDF* yielded nearly even results, with RDF* slightly ahead (12 vs. 10 votes), suggesting that both are strong candidates from a usability standpoint.

Overall, the survey contributes valuable insights into how different temporal modelling approaches are perceived by CH experts and technical experts. While not intended as a formal empirical evaluation, the survey captures practitioner perspectives that are often overlooked in technical assessments. This user-centred viewpoint complements the project requirements and supports the broader goal of aligning technical choices with the practical needs and preferences of CH experts. However, several limitations must be acknowledged. Firstly, the survey focuses on simplified visualisations of RDF representations. The decision to show schematic visualisations rather than raw RDF syntax (e.g., Turtle or RDF/XML) was made to make the survey accessible to participants regardless of their technical background. While this lowers the barrier to participation, it also abstracts away some of the actual complexity of the models, possibly affecting how effort or clarity is perceived. Participants may judge models more on layout or visual simplicity than on how intuitive the representations are. Secondly, the representations are derived from a single example, which introduces a risk of learning effects over time. While pair order and positioning were randomised and balanced to reduce this, the possibility of biases remains. Thirdly, the survey measures perceived ease-of-use rather than actual modelling performance; it does not involve task-based validation such as modelling data or querying a knowledge graph.

[17] All survey statistics and results can be accessed here: `https://github.com/sashabruns/RDF-Extensions-for-Cultural-Heritage/blob/main/CR7_survey_statistics.ipynb`

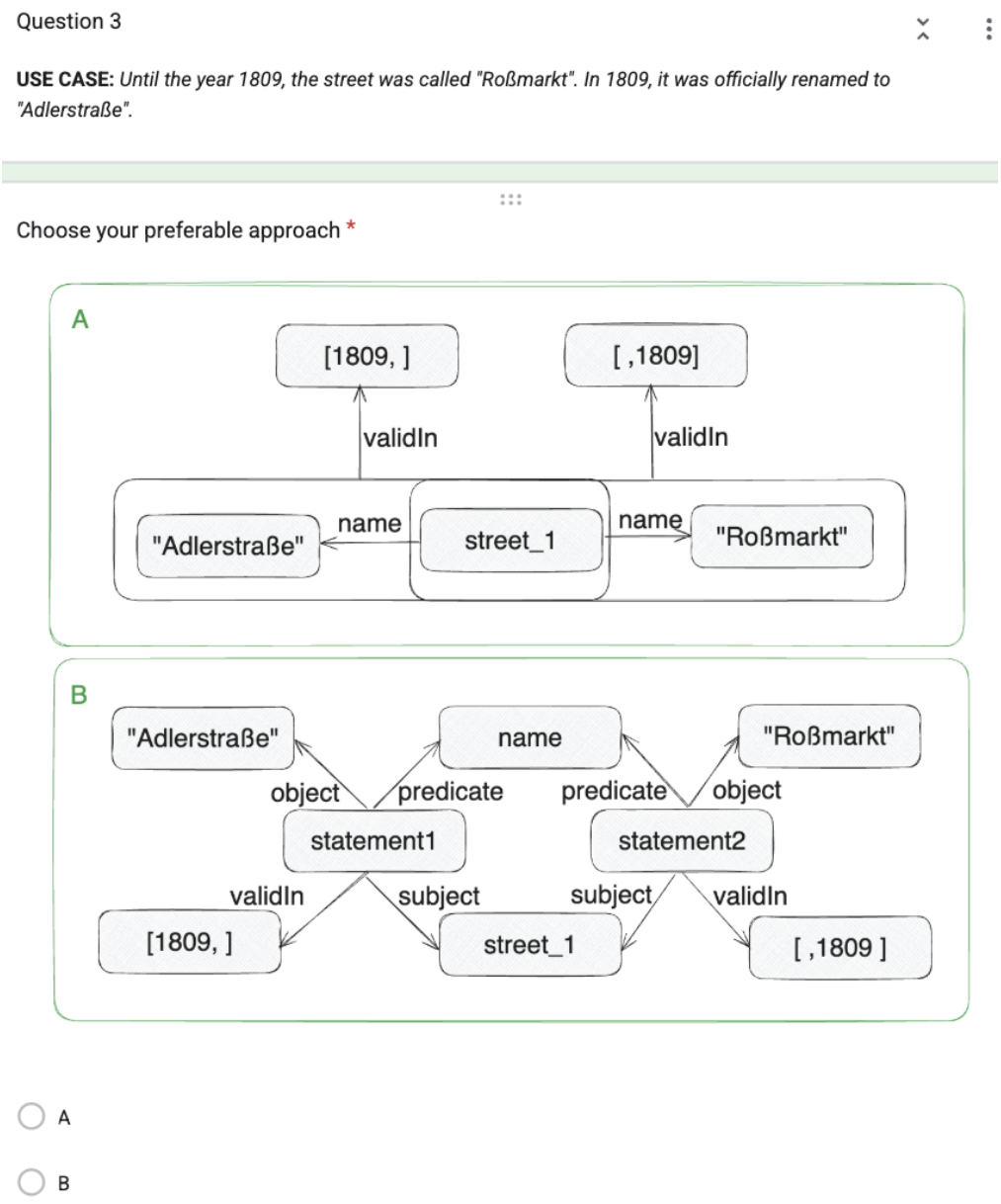
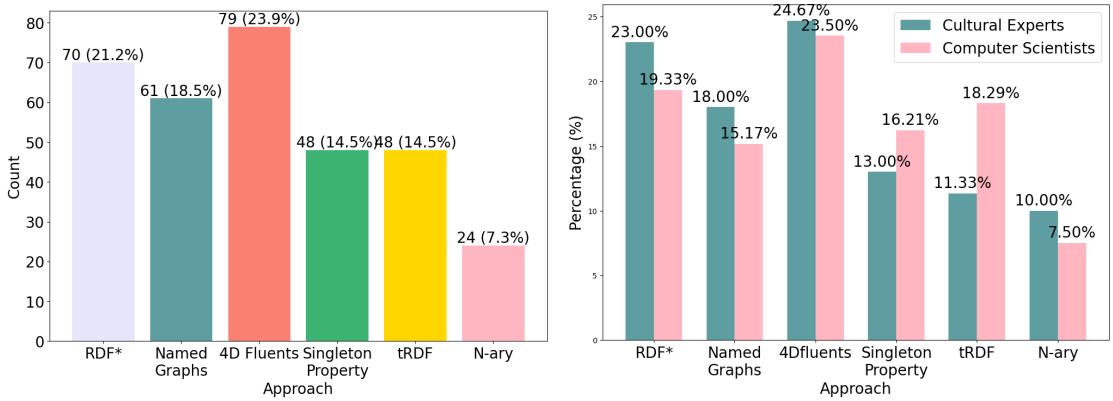


Figure 5 Example survey question used to evaluate ease-of-use.

Future studies could expand this preliminary analysis by incorporating task-based evaluations, larger and more diverse samples, and scenarios that engage with full modelling pipelines.

4.8 Limitations and Future Work

This section provides a structured analysis of temporal modelling approaches for RDF-based CH KGs, evaluated against concrete requirements derived from real-world CH use cases. By investigating and discussing requirements one by one, model comparison is achieved in a systematic and use-case-aware manner, moving beyond purely formal or purely empirical evaluation strategies.



(a) Overall preferences of the participants.

(b) Results based on the domain background.

Figure 6 Survey results evaluating the ease-of-use of each modelling approach.

There are several limitations of the study that must be acknowledged. First, while the evaluation captures a broad range of temporal modelling approaches, it is not exhaustive. The focus lies on representative RDF extensions, however, more event-based and foundational models may offer alternative perspectives that deserve future inclusion. Second, the evaluation is based on schematic examples than large-scale deployment or empirical benchmarking. While this enables comparison, it does not fully capture the implementation costs, scalability issues, or other impacts on modelling and querying tasks. Third, the ease-of-use survey with CH practitioners complements technical evaluations, however the survey scope is limited to pairwise comparisons on a single example. Broader user studies are beneficial to validate these findings at scale.

Future work will address these limitations by expanding the comparison to include other temporal data models. Additionally, in future work, the current analysis will be complemented with a more extensive empirical evaluation using representative datasets from CH domains. While the present study offers a structured comparison, modelling scenarios, and expert feedback, it focuses primarily on qualitative aspects. Future research will extend this by applying the evaluated approaches to larger and more diverse datasets, measuring practical outcomes such as scalability, query complexity and reasoning performance. Such an empirical layer will enhance the generalizability of the findings and offer deeper validation of how different temporal models perform under real-world conditions.

5 Decision Support for Selecting Temporal Modelling Approaches

The discussion of temporal modelling approaches, presented in Section 4, demonstrates how each approach responds to a specific requirement, but it still lacks clear guidelines on which approach to choose. Every extension has its own set of strengths and weaknesses, making it difficult to give a recommendation, as they cannot be easily reduced to a straightforward or binary answer when assessing their suitability for specific requirements. To provide a more nuanced comparison, this section introduces a structured decision support mechanism. This comprehensive tool aims at providing systematic guidelines for practitioners in evaluating approaches by prioritising the requirement based on their specific project needs and diverse scenarios, and navigating the decision-making process for temporal data modelling. The following subsections present: i) the scoring methodology used to quantify criteria, ii) three illustrative scenarios grounded in real-world CH project use cases, and iii) a simple ranking mechanism that helps identify the most fitting approach based on project-specific priorities.

5.1 Design and Methodology

Recognising that the choice of a temporal modelling technique is highly use case dependent, the decision-support aims to provide a nuanced analysis rather than a simplistic binary assessment. By defining specific metrics for each requirement, the tool enables a detailed and systematic comparison of how well different approaches meet project/data requirements. This methodology not only emphasises the relative advantages and limitations of each approach but also supports informed decision-making by aligning the evaluation process with the unique needs of each use case.

This methodology involves i) assigning raw scores to approaches per requirement, ii) applying normalisation to enable comparability, iii) incorporating weights to reflect project-specific priorities, and iv) computing similarity to an ideal solution.

- **Scoring and Scales.** Each modelling approach is assigned a raw score per requirement using requirement-specific metrics. These metrics are not uniform across all requirements: for example, *REQ2: Storage and Scalability* is based on empirical thresholds of triple counts, while *REQ1: Temporal Expressivity* and *REQ3: RDF Syntax Extension* use more nuanced qualitative scales. This variability reflects the fundamentally different natures of the requirements: some (e.g., scalability) are quantifiable and bounded, while others (e.g., expressivity) require expert judgment across a conceptual spectrum. The full set of scoring scales is presented in Table 1.
- **Normalisation.** To enable cross-requirement comparison, all raw scores (see Table 2) are min-max normalised to a common scale between 0 and 1, where 0 represents the best achievable performance. For example, a raw score of 1 is normalised to 0, and higher raw values are scaled proportionally based on the range defined for that requirement.
- **Weighting.** Users can assign weights to each requirement to reflect their relative importance for a given use case. The weights sum to 1.0 and allow the decision-support system to tailor recommendations to specific project contexts. The weighted normalised scores are then combined for each approach.
- **Distance to Ideal.** Finally, the Euclidean distance between each approach's weighted score vector and the ideal vector `[0.0, 0.0, ..., 0.0]` is calculated. The approach with the smallest distance is considered the best fit for that scenario.

This methodology yields a transparent and flexible mechanism for assessing temporal modelling approaches based on both domain-specific requirements and project-specific constraints.

This decision-support system is designed to support both technical implementers and CH practitioners in making informed, transparent, and requirement-driven decisions when modelling temporality in knowledge graphs. Its value lies not only in ranking options, but in making the trade-offs between different modelling priorities explicit.

5.2 Proof-of-Concept: Scenarios and Applications

This section presents a prototype decision-support tool developed to demonstrate how the proposed decision-support tool can aid in selecting a temporal RDF modelling approach based on project-specific priorities. The tool allows users to assign custom weights to the identified requirements (see Section 2.2) and visualises the alignment of each approach with a perfect score across those requirements. The goal of this prototype is not to prescribe definitive modelling choices, but to illustrate the potential of a structured, transparent, and comparative evaluation mechanism tailored to CH needs. The tool and the example scenarios are available online[18].

[18] `https://github.com/sashabruns/RDF-Extensions-for-Cultural-Heritage/blob/main/framework.ipynb`

Table 1 Requirements, scoring bases, and raw score scales used in the decision-support system.

Requirement	Scoring Basis	Raw Score Scale				
		1 (best)	**2**	**3**	**4**	**5 (worst)**
REQ1: Temporal Expressivity	Support for REQ1.1–REQ1.4	Full semantics	Intervals + relations	Intervals only	Limited support	No support
REQ2: Storage and Scalability	Triple count for representing facts	Minimal increase	Moderate increase	High increase	–	–
REQ3: RDF Syntax Extension	Degree of RDF syntax modification	None / minimal	Moderate (compatible)	Significant (custom tooling)	Complex (new layers)	RDF overhaul
REQ4: Model Semantics	Semantic expressiveness required	RDF(S)	OWL	Custom / rule-based	–	–
REQ5: Query Language	Compatibility with SPARQL	Native SPARQL	SPARQL + minor adaptation	Requires SPARQL extensions	Requires custom query	–
REQ6: RDFS/OWL Reasoning Support	OWL/RDFS reasoning capability	Full support	Moderate support	Limited support	No support	–
REQ7: Ease-of-Use	Expert judgement in CH context	Top-rated	2nd most usable	3rd in preference	4th in preference	Least usable

Table 2 Raw scores of temporal modelling approaches per requirement (lower is better).

Requirement	Temporal Modelling Approach					
	tRDF	**4D Fluents**	**RDF***	**Named Graphs**	**N-ary Relations**	**Singleton Property**
REQ1: Temporal Expressivity	2	1	4	4	4	4
REQ2: Storage and Scalability	3	3	1	1	2	2
REQ3: RDF Syntax Extension	1	5	3	2	5	4
REQ4: Model Semantics	1	2	1	2	2	3
REQ5: Query Language	1	2	3	1	1	2
REQ6: RDFS/OWL Reasoning Support	2	2	2	1	3	3
REQ7: Ease-of-Use	4	1	2	3	5	4

To demonstrate its functionality, three fictional scenarios inspired by recurring patterns and decision-making challenges are introduced. These scenarios are not directly extracted from specific project documentation but were shaped through iterative discussions with CH practitioners and technical partners during past collaborations. They reflect recurring tensions between expressivity, reasoning, technical constraints, and usability that shape real-world modelling decisions. While inspired by such discussions, these scenarios serve only as illustrative examples, designed to showcase the flexibility of the tool rather than document exact real-world processes. The assignment of weights in each scenario reflects typical trade-offs encountered in the field rather than normative recommendations.

Figure 7 visualises the results. Each chart shows how well each approach performs against a normalised "perfect" score, with the Euclidean distance used to rank the most suitable approach for each set of requirements.

Scenario 1: Large-Scale Data Integration for Historical Archives

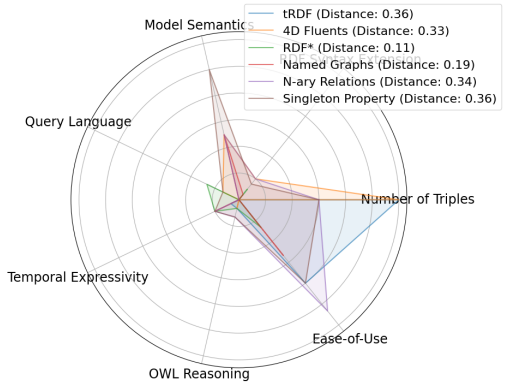
- **Context:** Integration of vast archival datasets, focusing on temporal annotation and federated querying of historical events.
- **Project Requirements:**
  - *Scalability:* Efficient handling of high triple volumes is critical.
  - *Model Semantics:* Lightweight semantics are preferred for performance reasons.
  - *Ease-of-Use:* Accessibility for a diverse set of researchers is key.
  - *Query Language:* SPARQL compatibility ensures interoperability.
  - *Temporal Expressivity:* Sufficient to support timeline construction.
- **Criteria Weighting:**
  - Storage and Scalability: 0.3
  - Model Semantics: 0.25
  - Ease-of-Use: 0.2
  - Query Language: 0.1
  - Temporal Expressivity: 0.05
  - OWL/RDF(S) Reasoning: 0.05
  - RDF Syntax Extension: 0.05
- **Preferred Approach:** RDF* (see Figure 7a).

Scenario 2: Integration and Reliability Checking for Historical Data of Nuremberg

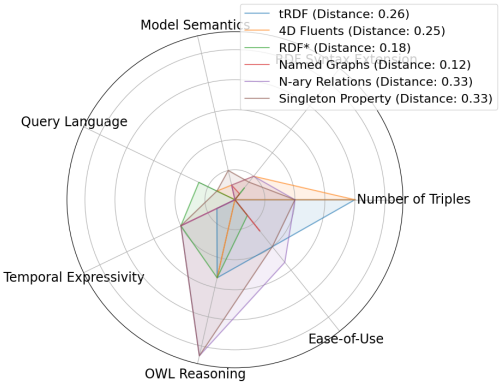
- **Context:** Aggregating and validating data from diverse CH sources related to the historical evolution of a city.
- **Project Requirements:**
  - *OWL/RDF(S) Reasoning:* Needed to support inference and consistency checks.
  - *Scalability:* Must handle large-scale data ingestion and reasoning.
  - *Temporal Expressivity:* Important for tracing entity changes over time.
- **Criteria Weighting:**
  - OWL/RDF(S) Reasoning: 0.4
  - Storage and Scalability: 0.2
  - Temporal Expressivity: 0.1
  - Query Language: 0.1
  - Ease-of-Use: 0.1
  - Model Semantics: 0.05
  - RDF Syntax Extension: 0.05
- **Preferred Approach:** Named Graphs (see Figure 7b).

Scenario 3: User-Friendly Temporal Data Management for Digital Humanities

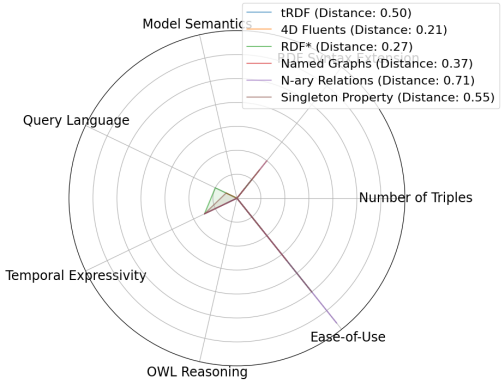
- **Context:** Supporting archivists in managing evolving classification schemes with minimal technical complexity.
- **Project Requirements:**
  - *Ease-of-Use:* Primary users are non-technical domain experts.
  - *RDF Syntax:* Adherence to familiar, standard RDF is important.
  - *Query Language:* Native SPARQL must be supported.
  - *Temporal Expressivity:* Needed to annotate changing labels and roles.
- **Criteria Weighting:**
  - Ease-of-Use: 0.5
  - RDF Syntax Extension: 0.2
  - Query Language: 0.15
  - Temporal Expressivity: 0.15
  - Model Semantics: 0.00
  - Storage and Scalability: 0.00
  - OWL/RDF(S) Reasoning: 0.00
- **Preferred Approach:** 4D Fluents (see Figure 7c).



(a) Scenario 1: Archival data integration.



(b) Scenario 2: City-wide integration and validation.



(c) Scenario 3: Archivist-friendly modelling.

Figure 7 Scenario-specific ranking of modelling approaches based on requirement weights and normalised distance from the ideal score.

Discussion

The comparative evaluation of temporal modelling approaches across three different cultural heritage scenarios provides a more nuanced understanding of their suitability under varying priorities. The decision-support prototype quantifies how closely each approach aligns with the requirements defined for a specific use case and offers a transparent rationale behind each recommendation.

In **Scenario 1**, where scalability, lightweight semantics, and ease-of-use are emphasised, RDF* is selected as the preferred solution. Its compact modelling and minimal syntax extension match the needs of high-volume archival integration. Named Graphs follow closely, with similar strengths but slightly higher semantic complexity. tRDF and Singleton Property scored the highest distances, mainly due to their larger triple count and more complex RDF syntax, which make them less suitable for large-scale deployments with non-technical users. In **Scenario 2**, which requires reasoning capabilities and temporal expressivity for validating integrated data, Named Graphs turned out to be the top choice. Its native compatibility with OWL reasoning and straightforward contextualisation align well with the scenario's focus on consistency and logical inference. Singleton Property and N-ary Relations rank lower, as their reasoning support is limited or dependent on custom semantics, which reduces their effectiveness in tasks involving inference over complex data relationships. In **Scenario 3**, which prioritises usability and accessibility for non-technical users such as archivists, 4D Fluents scores the lowest distance and is identified as the preferred approach. Despite its triple verbosity, it offers intuitive temporal modelling, straightforward RDF syntax, and high usability in describing classification schemes over time. RDF* performs nearly as well due to its compact and intuitive syntax. In contrast, N-ary Relations and Singleton Property exhibit higher distances, reflecting the cognitive overhead associated with their more complex or less standard modelling constructs, which may hinder adoption in scenarios requiring minimal technical knowledge.

This analysis shows how weighting domain-specific requirements enables tailored recommendations. While RDF* and Named Graphs often appear in top positions, their suitability is not universal but context-sensitive. Similarly, approaches that score lower in the discussed scenarios are not universally less valid but may impose trade-offs that are mismatched with the scenario-specific constraints. By making these trade-offs explicit, the tool supports transparent, criteria-driven decision-making for temporal representation in CH projects.

Overall, the prototype illustrates how decision-support tools can assist in navigating the complex landscape of temporal RDF modelling. The Euclidean distance from the ideal solution offers a quantifiable and interpretable signal about the relative suitability of different approaches under competing priorities. This approach does not replace human judgment or detailed project-specific modelling design, but it enables a more structured comparison of alternatives, which remains underexplored in CH-oriented RDF practice.

5.3 Limitations and Future Work

While the decision support mechanism presented in this paper offers a structured way to align temporal modelling approaches with project-specific requirements, several limitations remain and thus inform directions for future work.

First, although the current implementation is intentionally lightweight and easy to apply, it requires users to assign numerical weights to requirements based on their relevance to the project context. This step can be challenging, particularly for users without prior experience in such decision-making processes. To address this, alternative ways of prioritising requirements are to be explored. One direction is to replace or complement numerical weighting with a ranking-based

approach, where users simply rank requirements in order of importance. This method could reduce cognitive load and encourage more intuitive engagement, especially for non-technical stakeholders. Second, the evaluation currently relies on a fixed scoring scheme, which is derived from conceptual and practical analysis of the approaches but has not yet been empirically validated. In future work, we intend to refine this scoring through systematic benchmarking and testing, capturing performance characteristics such as reasoning behaviour, scalability, and actual ease-of-use across CH projects. Third, since some temporal approaches are closely scored, slight changes in requirement priorities can shift the final recommendation. We therefore plan to incorporate sensitivity analysis into the tool to highlight how robust a recommendation is to small variations in input priorities. This would help users better understand trade-offs and the comparative strengths of alternative approaches. Lastly, the prototype is currently implemented as a Jupyter notebook, which assumes some technical familiarity. We are planning to develop a more accessible graphical user interface and evaluate the tool in real-world CH infrastructure planning settings, with the goal of supporting broader adoption and iterative refinement based on user feedback.

6 Summary and Conclusions

This works presents a systematic evaluation of RDF-based approaches for representing temporal information in Cultural Heritage Knowledge Graphs. Temporal representation in CH is especially challenging due to the need to model evolving entities, uncertain or incomplete historical data, and richly contextualised timelines. To structure this investigation, six RQs were formulated, which guided both the theoretical foundation and evaluation of the study.

RQ1: What RDF-based approaches exist for temporal representation? was addressed by investigating the current practices for representing temporal information in RDF, while key RDF extensions such as RDF*, tRDF, Named Graphs, N-ary Relations, 4D Fluents, and Singleton Property were identified. In addition to the main focus of the research – RDF extensions – alternative approaches for representing time in RDF, such as event-based models and temporal description logics, were briefly explored. These methods differ from RDF extensions in their strategies for handling temporal information, whether through focusing on event-driven representations, or applying logic-based temporal reasoning. This allowed a more comprehensive view of the methodologies available for managing temporality.

Suitability of the approaches for CH KGs was a central concern throughout this study. While many temporal modelling approaches originate from general-purpose use cases, their application in CH remained open. For this, answering **RQ2: What temporal requirements arise from Cultural Heritage use cases?**, in Section 2, temporal modelling needs that are specific to CH contexts were identified. Based on four CH use cases, data and their project requirements, seven requirements were presented. These include: *REQ1:* Temporal Expressivity – to capture contextual changes over time and to represent time statements, uncertainty, and interval relations; *REQ2:* Storage and Scalability – to ensure that models can handle large CH datasets efficiently; *REQ3:* RDF syntax extension – to investigate which model and how they extend RDF syntax to entail temporality, and how this influences interoperability and data integration; *REQ4:* Model semantic – to define the level of logic and inference needed for the extension; *REQ5:* Query language – to evaluate support for expressive and practical querying, especially with SPARQL; *REQ6:* Use of OWL reasoning – to observe how extending RDF with temporal constraint influence the standard reasoning mechanisms; and *REQ7:* Ease-of-Use – to assess accessibility for domain experts with limited technical background. Together, these criteria provide a structured lens through which different modelling approaches can be assessed in terms of their applicability and usability in CH contexts.

To complement the technical evaluation and answer **RQ3: How intuitive and usable are different temporal modelling approaches for CH domain experts?**, a small-scale usability survey (Section 4.7) was conducted among CH and technical experts. The study focused on perceived intuitiveness using a controlled example and schematic visualisations. While 4D Fluents and RDF* emerged as the most preferred approaches overall, N-ary Relations were perceived as less intuitive and scored lower in terms of ease-of-use.

Section 4 was dedicated to answer **RQ4: How can temporal modelling approaches be evaluated based on CH-specific requirements and technical criteria?**. It presented a detailed comparison across seven identified REQs. The evaluation showed that no temporal modelling approach fully meets all CH requirements, but each offers specific strengths depending on project needs. RDF* and Named Graphs perform well in terms of scalability and integration into existing infrastructures (REQ2), with RDF* offering strong ease-of-use and readability (REQ7), while Named Graphs also provide solid support for query formulation and reasoning (REQ5, REQ6). 4D Fluents stands out for its rich temporal expressivity (REQ1) and was found to be highly intuitive by CH experts (REQ7), but it introduces structural overhead that negatively impacts RDF compatibility and infrastructure alignment (REQ3). The Nary Relations approach is particularly well suited for modelling complex events and states, the direct attachment of temporal information to objectified relations enables easy querying (REQ5), aligning with OWL reasoning (REQ4). It scales well (REQ2), yet falls short in representing temporal information (REQ1), extends RDF syntax (REQ3), and is not intuitive for users (REQ7). Singleton Property supports contextual metadata and integrates well with infrastructure (REQ5), but performs poorly in modelling semantic change (REQ1), introduces high overhead (REQ2), and lacks usability and readability for users (REQ7). Finally, tRDF scores well in RDF compatibility and OWL alignment (REQ3, REQ4), and moderately in supporting query formulation (REQ5), but it does not scale as well in large infrastructures (REQ2) and is less intuitive for domain experts (REQ7).

These results underscore that "suitability" in CH contexts is not only a question of expressivity but of trade-offs: between representational power and model simplicity, between formal reasoning and system compatibility, and between technical potential and practical implementation. Our findings suggest that CH projects must evaluate temporal models not just by their semantic capabilities, but also by their alignment with the project requirements and conditions, e.g. institutional tooling. In this sense, answering **RQ5: How can CH practitioners be supported in selecting a suitable temporal modelling approach?**, the decision-support tool was introduced in Section 5. Based on the outcomes of the requirement-based evaluation, the tool aims to support experts in choosing an appropriate approach based on specific project needs. It enables users to prioritise different requirements and select approaches accordingly. The decision matrix offers a structured path through complex trade-offs and is intended to serve as a practical tool for project planning.

While this work offers a structured, requirement-driven comparison of temporal RDF models, several limitations must be acknowledged. The evaluation was conducted on a limited set of example scenarios and does not include large-scale performance metrics or full CH datasets. Furthermore, the usability study was not a formal empirical validation but rather an exploratory survey intended to highlight practitioner perspectives. Future studies could extend this work by performing task-based evaluations (e.g., modelling, querying, or annotation tasks) across diverse domains. Another direction lies in expanding the range of modelling strategies considered. Event-centric models were not covered in depth here but are highly relevant for CH and deserve systematic inclusion in future evaluations. Likewise, the interplay between temporal and spatial dimensions in CH datasets presents a rich area for future exploration.

Additionally, the compatibility and interoperability of RDF extensions will be a focus of future work. The aim is to explore how effectively different RDF extensions can be aligned and evaluate the effort required to translate between various temporal modelling approaches. This is particularly important for use cases that rely on diverse extensions and require integration to enable simpler data access, discovery, and querying. Achieving such interoperability will ensure more cohesive, flexible, and efficient temporal representations within CH KGs, improving their usability across diverse applications.

Overall, this research has the potential to significantly improve how humanity preserves, accesses, and interprets the temporal dimensions of cultural heritage. It offers new means for temporal representations, and, thus, new means to engage with history and shared past, not merely as static moments but as a rich, unfolding story over time.

Resource Availability Statement

All supplementary resources are openly available on GitHub[19], including the visualisations presented in the paper, example queries supporting the discussion in Section 4, statistical analyses and insights from the user study in Section 4.7, and the implementation of the framework described in Section 5.

Declaration of AI Usage

AI-based tools by OpenAI GPT-4–class models[20] were used to support the writing process, in particular, for the linguistic refinement of the text, including spelling, grammar, punctuation, and stylistic refinement. All scientific content, research ideas and methodology are the work of the authors.

References

1 Waqas Ali, Muhammad Saleem, Bin Yao, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. A survey of RDF stores & SPARQL engines for querying knowledge graphs. *VLDB J.*, 31(3):1–26, 2022. doi:10.1007/s00778-021-00711-3.

2 James F. Allen. An interval-based representation of temporal knowledge. In Patrick J. Hayes, editor, *Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI '81, Vancouver, BC, Canada, August 24-28, 1981*, volume 81, pages 221–226. International Joint Conferences on Artificial Intelligence Organization (IJCAI), William Kaufmann, 1981. URL: http://ijcai.org/Proceedings/81-1/Papers/045.pdf.

3 James F. Allen and George Ferguson. Actions and events in interval temporal logic. *J. Log. Comput.*, 4(5):531–579, 1994. doi:10.1093/logcom/4.5.531.

4 Eleftherios Anagnostopoulos, Sotiris Batsakis, and Euripides G. M. Petrakis. CHRONOS: A reasoning engine for qualitative temporal information in OWL. In Junzo Watada, Lakhmi C. Jain, Robert J. Howlett, Naoto Mukai, and Koichi Asakura, editors, *17th International Conference in Knowledge Based and Intelligent Information and Engineering Systems, KES 2013, Kitakyushu, Japan, 9-11 September 2013*, volume 22 of *Procedia Computer Science*, pages 70–77. Elsevier, 2013. doi:10.1016/j.procs.2013.09.082.

5 Robert Arp, Barry Smith, and Andrew D Spear. *Building ontologies with Basic Formal Ontology*. MIT Press, 2015. doi:10.7551/mitpress/9780262527811.001.0001.

6 Alessandro Artale and Enrico Franconi. A survey of temporal extensions of description logics. *Ann. Math. Artif. Intell.*, 30(1-4):171–210, 2000. doi:10.1023/A:1016636131405.

7 Alessandro Artale and Enrico Franconi. Temporal description logics. In Michael Fisher, Dov M. Gabbay, and Lluís Vila, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Foundations of Artificial Intelligence*, pages 375–388. Elsevier, 2005. doi:10.1016/S1574-6526(05)80014-8.

8 Franz Baader, Stefan Borgwardt, Patrick Koopmann, Ana Ozaki, and Veronika Thost. Metric temporal description logics with interval-rigid

[19] https://github.com/sashabruns/RDF-Extensions-for-Cultural-Heritage
[20] OpenAI. (2024). ChatGPT-4o [Large language model]. https://chat.openai.com

names. *ACM Trans. Comput. Log.*, 21(4):30:1–30:46, 2020. doi:10.1145/3399443.

9 Evdoxios Baratis, Euripides G. M. Petrakis, Sotiris Batsakis, Nikolaos Maris, and Nikos Papadakis. TOQL: temporal ontology querying language. In Nikos Mamoulis, Thomas Seidl, Torben Bach Pedersen, Kristian Torp, and Ira Assent, editors, *Advances in Spatial and Temporal Databases, 11th International Symposium, SSTD 2009, Aalborg, Denmark, July 8-10, 2009, Proceedings*, volume 5644 of *Lecture Notes in Computer Science*, pages 338–354. Springer, Springer, 2009. doi:10.1007/978-3-642-02982-0_22.

10 Sotiris Batsakis and Euripides G. M. Petrakis. SOWL: A framework for handling spatio-temporal information in OWL 2.0. In Nick Bassiliades, Guido Governatori, and Adrian Paschke, editors, *Rule-Based Reasoning, Programming, and Applications - 5th International Symposium, RuleML 2011 - Europe, Barcelona, Spain, July 19-21, 2011. Proceedings*, volume 6826 of *Lecture Notes in Computer Science*, pages 242–249. Springer, Springer, 2011. doi:10.1007/978-3-642-22546-8_19.

11 Sotiris Batsakis and Euripides GM Petrakis. Representing temporal knowledge in the semantic web: The extended 4d fluents approach. In *Combinations of intelligent methods and applications*, pages 55–69. Springer, 2011. doi:10.1007/978-3-642-19618-8_4.

12 Elena Beisswanger, Stefan Schulz, Holger Stenzhorn, and Udo Hahn. Biotop: An upper domain ontology for the life sciencesa description of its current structure, contents and interfaces to OBO ontologies. *Appl. Ontology*, 3(4):205–212, 2008. doi:10.3233/AO-2008-0057.

13 Oleksandra Bruns, Tabea Tietz, Mehdi Ben Chaabane, Manuel Portz, Felix Xiong, and Harald Sack. The nuremberg address knowledge graph. In Ruben Verborgh, Anastasia Dimou, Aidan Hogan, Claudia d'Amato, Ilaria Tiddi, Arne Bröring, Simon Maier, Femke Ongenae, Riccardo Tommasini, and Mehwish Alam, editors, *The Semantic Web: ESWC 2021 Satellite Events - Virtual Event, June 6-10, 2021, Revised Selected Papers*, volume 12739 of *Lecture Notes in Computer Science*, pages 115–119. Springer, Springer, 2021. doi:10.1007/978-3-030-80418-3_21.

14 Oleksandra Bruns, Tabea Tietz, Sandra Göller, and Harald Sack. TRANSRAZ data model: Towards a geosocial representation of historical cities. In Maribel Acosta, Silvio Peroni, Sahar Vahdati, Anna Lisa Gentile, Tassilo Pellegrini, and Jan-Christoph Kalo, editors, *Knowledge Graphs: Semantics, Machine Learning, and Languages - Proceedings of the 19th International Conference on Semantic Systems, 20-22 September 2023, Leipzig, Germany*, volume 56 of *Studies on the Semantic Web*, pages 161–176. IOS Press, 2023. doi:10.3233/SSW230012.

15 Oleksandra Bruns, Tabea Tietz, Mahsa Vafaie, Danilo Dessì, Harald Sack, et al. Towards a representation of temporal data in archival records: Use cases and Requirements. In *CEUR Workshop Proc.*, volume 3019, pages 128–134. CEUR-WS, 2021. doi:10.5445/IR/1000141526.

16 George Bruseker, Nicola Carboni, and Anaïs Guillem. Cultural heritage data management: the role of formal ontology and cidoc crm. *Heritage and archaeology in the digital age: acquisition, curation, and dissemination of spatial cultural heritage data*, pages 93–131, 2017. doi:10.1007/978-3-319-65370-9_6.

17 Gavin Carothers and Lex Machina, Inc. RDF 1.1 N-Quads. A line-based syntax for RDF datasets. W3C Recommendation, 25 February 2014, 2014. Accessed: 2024-09-16. URL: https://www.w3.org/TR/n-quads/.

18 Valentina Anita Carriero, Aldo Gangemi, Maria Letizia Mancinelli, Ludovica Marinucci, Andrea Giovanni Nuzzolese, Valentina Presutti, and Chiara Veninata. Arco: The italian cultural heritage knowledge graph. In Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtech Svátek, Isabel F. Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon, editors, *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*, volume 11779 of *Lecture Notes in Computer Science*, pages 36–52. Springer, Springer, 2019. doi:10.1007/978-3-030-30796-7_3.

19 Simon Cox, Chris Little, Jerry R. Hobbs, and Feng Pan. Time ontology in owl. https://www.w3.org/TR/2022/CRD-owl-time-20221115/, 2022. Accessed: 2024-09-16. doi:10.62973/16-071r3.

20 E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982. doi:10.1016/0167-6423(83)90017-5.

21 Flavius Frasincar, Viorel Milea, and Uzay Kaymak. towl: Integrating time in OWL. In Roberto De Virgilio, Fausto Giunchiglia, and Letizia Tanca, editors, *Semantic Web Information Management - A Model-Based Perspective*, pages 225–246. Springer, 2009. doi:10.1007/978-3-642-04329-1_11.

22 Fabien Gandon, Guus Schreiber, and Dave Beckett. Rdf 1.1 xml syntax. W3C Recommendation, 2014. Accessed: 2024-10-31. URL: https://www.w3.org/TR/rdf-syntax-grammar/.

23 Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehmann. Modelling ontology evaluation and validation. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006, Proceedings*, volume 4011 of *Lecture Notes in Computer Science*, pages 140–154. Springer, Springer, 2006. doi:10.1007/11762256_13.

24 Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening ontologies with DOLCE. In Asunción Gómez-Pérez and V. Richard Benjamins, editors, *Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 13th International Conference, EKAW 2002, Siguenza, Spain, October 1-4, 2002, Proceedings*, volume 2473 of *Lecture Notes in Computer Science*, pages

166–181. Springer, Springer, 2002. doi:10.1007/3-540-45810-7_18.

25 Stefan Gradmann. Objects, process, context in time and space: what we can do with the europeana data model (edm)-and what we cannot do. In *Brown University, Date: 2012/03/15-2012/03/15, Location: Providence*, 2012. doi:10.1021/acsomega.7b00274.s002.

26 Ramanathan V. Guha and Douglas B. Lenat. CYC: A midterm report. *AI Mag.*, 11(3):32–59, 1990. doi:10.1609/aimag.v11i3.842.

27 Giancarlo Guizzardi, Gerd Wagner, João Paulo Andrade Almeida, and Renata S. S. Guizzardi. Towards ontological foundations for conceptual modeling: The unified foundational ontology (UFO) story. *Appl. Ontology*, 10(3-4):259–271, 2015. doi:10.3233/AO-150157.

28 Claudio Gutierrez, Carlos A. Hurtado, and Alejandro A. Vaisman. Temporal RDF. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*, volume 3532 of *Lecture Notes in Computer Science*, pages 93–107. Springer, Springer, 2005. doi:10.1007/11431053_7.

29 Claudio Gutierrez, Carlos A. Hurtado, and Alejandro A. Vaisman. Introducing time into RDF. *IEEE Trans. Knowl. Data Eng.*, 19(2):207–218, 2007. doi:10.1109/TKDE.2007.34.

30 Víctor Gutiérrez-Basulto, Jean Christoph Jung, and Ana Ozaki. On metric temporal description logics. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 837–845. IOS Press, 2016. doi:10.3233/978-1-61499-672-9-837.

31 Joseph Y. Halpern and Yoav Shoham. A propositional modal logic of time intervals. *J. ACM*, 38(4):935–962, 1991. doi:10.1145/115234.115351.

32 Olaf Hartig. Foundations of rdf⋆ and sparql⋆ (an alternative approach to statement-level metadata in RDF). In Juan L. Reutter and Divesh Srivastava, editors, *Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017*, volume 1912 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017. URL: https://ceur-ws.org/Vol-1912/paper12.pdf.

33 Sally Haslanger. Persistence through time. *The Oxford Handbook of Metaphysics*, pages 315–354, 2003. doi:10.1093/oxfordhb/9780199284221.003.0012.

34 Muhammad Rosyihan Hendrawan, Azman Mat Isa, and Ahmad Zam Hariro Samsudin. Evolving taxonomies: Fifty years of taxonomy development in library and information science. *Journal of Librarianship and Information Science*, page 09610006251334372, 2025. doi:10.1177/09610006251334372.

35 Heinrich Herre. *General Formal Ontology (GFO): A Foundational Ontology for Conceptual Modelling*, pages 297–345. Springer Netherlands, Dordrecht, 2010. doi:10.1007/978-90-481-8847-5_14.

36 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4):71:1–71:37, 2022. doi:10.1145/3447772.

37 Carlos A. Hurtado and Alejandro A. Vaisman. Reasoning with temporal constraints in RDF. In José Júlio Alferes, James Bailey, Wolfgang May, and Uta Schwertel, editors, *Principles and Practice of Semantic Web Reasoning, 4th International Workshop, PPSWR 2006, Budva, Montenegro, June 10-11, 2006, Revised Selected Papers*, volume 4187 of *Lecture Notes in Computer Science*, pages 164–178. Springer, Springer, 2006. doi:10.1007/11853107_12.

38 Eero Hyvönen. Digital humanities on the semantic web: Sampo model and portal series. *Semantic Web*, 14(4):729–744, 2023. doi:10.3233/SW-223034.

39 Michael Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. Log. Program.*, 12(3&4):335–367, 1992. doi:10.1016/0743-1066(92)90007-P.

40 Mikko Koho, Esko Ikkala, Petri Leskinen, Minna Tamper, Jouni Tuominen, and Eero Hyvönen. Warsampo knowledge graph: Finland in the second world war as linked open data. *Semantic Web*, 12(2):265–278, 2021. doi:10.3233/SW-200392.

41 Abba Lawan and Abdur Rakib. FT-SWRL: A fuzzy-temporal extension of semantic web rule language. *CoRR*, abs/1911.12399, 2019. doi:10.48550/arXiv.1911.12399.

42 Pasquale Lisena, Daniel Schwabe, Marieke van Erp, Raphaël Troncy, William Tullett, Inger Leemans, Lizzie Marx, and Sofia Colette Ehrich. Capturing the semantics of smell: The odeuropa data model for olfactory heritage information. In Paul Groth, Maria-Esther Vidal, Fabian M. Suchanek, Pedro A. Szekely, Pavan Kapanipathi, Catia Pesquita, Hala Skaf-Molli, and Minna Tamper, editors, *The Semantic Web - 19th International Conference, ESWC 2022, Hersonissos, Crete, Greece, May 29 - June 2, 2022, Proceedings*, volume 13261 of *Lecture Notes in Computer Science*, pages 387–405. Springer, Springer, 2022. doi:10.1007/978-3-031-06981-9_23.

43 Nuno Lopes, Axel Polleres, Umberto Straccia, and Antoine Zimmermann. Anql: Sparqling up annotated RDFS. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, volume 6496 of *Lecture Notes in Computer Science*, pages 518–533. Springer, Springer, 2010. doi:10.1007/978-3-642-17746-0_33.

44 J Ellis McTaggart. The unreality of time. *Mind*, pages 457–474, 1908. doi:10.1093/mind/xvii.4.457.

45 Vinh Nguyen, Olivier Bodenreider, and Amit P. Sheth. Don't like RDF reification?: making statements about statements using singleton property. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 759–770. ACM, 2014. doi:10.1145/2566486.2567973.

46 Vinh Nguyen, Olivier Bodenreider, Krishnaprasad Thirunarayan, Gang Fu, Evan Bolton, Núria Queralt Rosinach, Laura Inés Furlong, Michel Dumontier, and Amit P. Sheth. On reasoning with RDF statements about statements using singleton property triples. *CoRR*, abs/1509.04513, 2015. doi:10.48550/arXiv.1509.04513.

47 Ian Niles and Adam Pease. Towards a standard upper ontology. In *2nd International Conference on Formal Ontology in Information Systems, FOIS 2001, Ogunquit, Maine, USA, October 17-19, 2001, Proceedings*, pages 2–9. ACM, 2001. doi:10.1145/505168.505170.

48 Natasha Noy, Alan Rector, Pat Hayes, and Chris Welty. Defining n-ary relations on the semantic web. *W3C working group note*, 12(4), 2006. Accessed: 2024-10-31. URL: https://www.w3.org/TR/swbp-n-aryRelations/.

49 Martin J. O'Connor and Amar K. Das. SQWRL: A query language for OWL. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009*, volume 529 of *CEUR Workshop Proceedings*, pages 1–8. CEUR-WS.org, 2009. URL: https://ceur-ws.org/Vol-529/owled2009_submission_42.pdf.

50 Martin J. O'Connor and Amar K. Das. A method for representing and querying temporal information in OWL. In Ana L. N. Fred, Joaquim Filipe, and Hugo Gamboa, editors, *Biomedical Engineering Systems and Technologies - Third International Joint Conference, BIOSTEC 2010, Valencia, Spain, January 20-23, 2010, Revised Selected Papers*, volume 127 of *Communications in Computer and Information Science*, pages 97–110. Springer, Springer, 2010. doi:10.1007/978-3-642-18472-7_8.

51 J. Neil Otte, John Beverley, and Alan Ruttenberg. BFO: basic formal ontology. *Appl. Ontology*, 17(1):17–43, 2022. doi:10.3233/AO-220262.

52 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE, IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

53 Axel Polleres, Romana Pernisch, Angela Bonifati, Daniele Dell'Aglio, Daniil Dobriy, Stefania Dumbrava, Lorena Etcheverry, Nicolas Ferranti, Katja Hose, Ernesto Jiménez-Ruiz, Matteo Lissandrini, Ansgar Scherp, Riccardo Tommasini, and Johannes Wachs. How does knowledge evolve in open knowledge graphs? *TGDK*, 1(1):11:1–11:59, 2023. doi:10.4230/TGDK.1.1.11.

54 Andrea Pugliese, Octavian Udrea, and V.S. Subrahmanian. Scaling RDF with time. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 605–614. ACM, 2008. doi:10.1145/1367497.1367579.

55 Katerina El Raheb, Theofilos Mailis, Vladislav Ryzhikov, Nicolas Papapetrou, and Yannis E. Ioannidis. Balonse: Temporal aspects of dance movement and its ontological representation. In Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rinke Hoekstra, Pascal Hitzler, and Olaf Hartig, editors, *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part II*, volume 10250 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2017. doi:10.1007/978-3-319-58451-5_4.

56 Yves Raimond and Samer Abdallah. The timeline ontology. http://motools.sourceforge.net/timeline/timeline.html, 2007. Accessed: 2024-09-16.

57 Anisa Rula, Matteo Palmonari, Andreas Harth, Steffen Stadtmüller, and Andrea Maurino. On the diversity and availability of temporal information in linked open data. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, volume 7649 of *Lecture Notes in Computer Science*, pages 492–507. Springer, Springer, 2012. doi:10.1007/978-3-642-35176-1_31.

58 Albrecht Schmiedel. Temporal terminological logic. In Howard E. Shrobe, Thomas G. Dietterich, and William R. Swartout, editors, *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, USA, July 29 - August 3, 1990, 2 Volumes*, pages 640–645. AAAI Press/The MIT Press, AAAI Press / The MIT Press, 1990. URL: http://www.aaai.org/Library/AAAI/1990/aaai90-096.php.

59 Bernhard Schueler, Sergej Sizov, Steffen Staab, and Duc Thanh Tran. Querying for meta knowledge. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 625–634. ACM, 2008. doi:10.1145/1367497.1367582.

60 Sangeeta Sen, Mariana Curado Malta, Biswanath Dutta, and Animesh Dutta. State-of-the-art approaches for meta-knowledge assertion in the web of data. *IETE Technical Review*, 38(6):672–709, 2021. doi:10.1080/02564602.2020.1819891.

61 Clauirton A. Siebra and Katarzyna Wac. Engineering uncertain time for its practical integration in ontologies. *Knowl. Based Syst.*, 251:109152, 2022. doi:10.1016/j.knosys.2022.109152.

62 Jannik Strötgen, Thomas Bögel, Julian Zell, Ayser Armiti, Tran Van Canh, and Michael Gertz. Extending heideltime for temporal expressions referring to historic dates. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asunción Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014, Reykjavik, Iceland, May 26-31, 2014*, pages 2390–2397. European Language Resources Association (ELRA), 2014. URL: http://www.lrec-conf.org/proceedings/lrec2014/summaries/849.html.

63 Jannik Strötgen and Michael Gertz. Heideltime: High quality rule-based extraction and normalization of temporal expressions. In Katrin Erk and Carlo Strapparava, editors, *Proceedings of the 5th International Workshop on Semantic Evaluation, SemEval@ACL 2010, Uppsala University, Uppsala, Sweden, July 15-16, 2010*, pages 321–324. The Association for Computer Linguistics, 2010. URL: https://aclanthology.org/S10-1071/.

64 Ivan Terziev, Atanas Kiryakov, and Dimitar Manov. D. 1.8. 1 base upper-level ontology (bulo) guidance. *Deliverable of EU-IST Project IST*, 2005.

65 Tabea Tietz, Mehwish Alam, Harald Sack, and Marieke van Erp. Challenges of knowledge graph evolution from an NLP perspective. In Alessandro Adamou, Enrico Daga, and Albert Meroño-Peñuela, editors, *Proceedings of the Third Workshop on Humanities in the Semantic Web (WHiSe 2020) co-located with 15th Extended Semantic Web Conference (ESWC 2020), Heraklion, Greece, June 2, 2020 (online)*, volume 2695 of *CEUR Workshop Proceedings*, pages 71–76. CEUR-WS.org, 2020. URL: https://ceur-ws.org/Vol-2695/paper8.pdf.

66 Tabea Tietz, Oleksandra Bruns, Sandra Göller, Matthias Razum, Danilo Dessì, and Harald Sack. Knowledge graph enabled curation and exploration of nuremberg's city heritage. In Adrian Paschke, Georg Rehm, Jamal Al Qundus, Clemens Neudecker, and Lydia Pintscher, editors, *Proceedings of the Conference on Digital Curation Technologies (Qurator 2021), Berlin, Germany, February 8th - to - 12th, 2021*, volume 2836 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021. doi:10.5445/IR/1000131502.

67 Tabea Tietz, Oleksandra Bruns, and Harald Sack. A data model for linked stage graph and the historical performing arts domain (short paper). In Antonis Bikakis, Roberta Ferrario, Stéphane Jean, Béatrice Markhoff, Alessandro Mosca, and Marianna Nicolosi Asmundo, editors, *Proceedings of the International Workshop on Semantic Web and Ontology Design for Cultural Heritage co-located with the International Semantic Web Conference 2023 (ISWC 2023), Athens, Greece, November 7, 2023*, volume 3540 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. doi:10.5445/IR/1000169249.

68 Tabea Tietz, Jörg Waitelonis, Kanran Zhou, Paul Felgentreff, Nils Meyer, Andreas Weber, and Harald Sack. Linked stage graph. In Mehwish Alam, Ricardo Usbeck, Tassilo Pellegrini, Harald Sack, and York Sure-Vetter, editors, *Proceedings of the Posters and Demo Track of the 15th International Conference on Semantic Systems co-located with 15th International Conference on Semantic Systems (SEMANTiCS 2019), Karlsruhe, Germany, September 9th - to - 12th, 2019*, volume 2451 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019. doi:10.5445/IR/1000100183.

69 Octavian Udrea, Diego Reforgiato Recupero, and V. S. Subrahmanian. Annotated RDF. *ACM Trans. Comput. Log.*, 11(2):10:1–10:41, 2010. doi:10.1145/1656242.1656245.

70 Mahsa Vafaie, Oleksandra Bruns, Danilo Dessì, Nastasja Pilz, and Harald Sack. Modelling archival hierarchies in practice: Key aspects and lessons learned. In Yasunobu Sumikawa, Ryohei Ikejiri, Antoine Doucet, Eva Pfanzelter, Mohammed Hasanuzzaman, Gaël Dias, Ian Milligan, and Adam Jatowt, editors, *Proceedings of the 6th International Workshop on Computational History (HistoInformatics 2021) co-located with ACM/IEEE Joint Conference on Digital Libraries 2021 (JCDL 2021), Online event, September 30-October 1, 2021*, volume 2981 of *CEUR Workshop Proceedings*, page 6. Aachen, Germany: RWTH Aachen, CEUR-WS.org, 2021. doi:10.5445/IR/1000139652.

71 Oleksandra Vsesviatska, Tabea Tietz, Fabian Hoppe, Mirjam Sprau, Nils Meyer, Danilo Dessì, and Harald Sack. Ardo: an ontology to describe the dynamics of multimedia archival records. In Chih-Cheng Hung, Jiman Hong, Alessio Bechini, and Eunjee Song, editors, *SAC '21: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, March 22-26, 2021*, pages 1855–1863. ACM, 2021. doi:10.1145/3412841.3442057.

72 Hsien-Tseng Wang. *Valid Time RDF*. PhD thesis, City University of New York, USA, 2020. URL: https://academicworks.cuny.edu/gc_etds/4037.

73 Hsien-Tseng Wang and Abdullah Uz Tansel. Temporal extensions to RDF. *J. Web Eng.*, 18(1-3):125–168, 2019. doi:10.13052/jwe1540-9589.18134.

74 Robert Weida and Diane Litman. Subsumption and recognition of heterogeneous constraint networks. In *Proceedings of the Tenth Conference on Artificial Intelligence for Applications*, pages 381–388. IEEE, 1994. doi:10.1109/caia.1994.323650.

75 Christopher A. Welty and Richard Fikes. A reusable ontology for fluents in OWL. In Brandon Bennett and Christiane Fellbaum, editors, *Formal Ontology in Information Systems, Proceedings of the Fourth International Conference, FOIS 2006, Baltimore, Maryland, USA, November 9-11, 2006*, volume 150 of *Frontiers in Artificial Intelligence and Applications*, pages 226–236. ACM / IOS Press, 2006. doi:10.5555/1566079.1566106.

76 Di Wu, Hsien-Tseng Wang, and Abdullah Uz Tansel. A survey for managing temporal data in RDF. *Inf. Syst.*, 122:102368, 2024. doi:10.1016/j.is.2024.102368.

77 Fouad Zablith, Grigoris Antoniou, Mathieu d'Aquin, Giorgos Flouris, Haridimos Kondylakis,

Enrico Motta, Dimitris Plexousakis, and Marta Sabou. Ontology evolution: a process-centric survey. *Knowl. Eng. Rev.*, 30(1):45–75, 2015. doi:10.1017/S0269888913000349.

78 Fu Zhang, Zhiyin Li, Dunhong Peng, and Jingwei Cheng. RDF for temporal data management - a survey. *Earth Sci. Informatics*, 14(2):563–599, 2021. doi:10.1007/s12145-021-00574-w.

Semantically Reflected Programs

Eduard Kamburjan
IT University of Copenhagen, Denmark
University of Oslo, Norway

Vidar Norstein Klungre
University of Oslo, Norway

Yuanwei Qu
University of Oslo, Norway

Rudolf Schlatte
University of Oslo, Norway

Egor V. Kostylev
University of Oslo, Norway

Martin Giese
University of Oslo, Norway

Einar Broch Johnsen
University of Oslo, Norway

Abstract

This paper addresses the dichotomy between the formalization of structural and the formalization of *executable* behavioral knowledge by means of semantically lifted programs, which explore an intuitive connection between imperative programs and knowledge graphs. While knowledge graphs and ontologies are eminently useful to represent formal knowledge about a system's individuals and universals, programming languages are designed to describe the system's evolution. To address this dichotomy, we introduce a *semantic lifting* of the program states of an executing progam into a knowledge graph, for an object-oriented programming language. The resulting graph is exposed as a *semantic reflection* layer within the programming language, allowing programmers to leverage knowledge of the application domain in their programs during execution. In this paper, we formalize semantic lifting and semantic reflection for a small imperative programming language, SMOL, explain the operational aspects of the language, and consider type correctness and virtualization for runtime program queries through the semantic reflection layer. We illustrate semantic lifting and semantic reflection through a case study of geological modeling and discuss different applications of the technique. The language implementation is open source and available online.

2012 ACM Subject Classification Software and its engineering → Object oriented languages; Computing methodologies → Knowledge representation and reasoning; Computing methodologies → Modeling and simulation

Keywords and phrases Knowledge Graphs, Ontologies, Object-Oriented Modelling, Imperative Programming Languages, Reflection, Type Safety

Digital Object Identifier 10.4230/TGDK.4.1.3

Category Research

Supplementary Material *Software (Source Code)*: `https://github.com/smolang/SemanticObjects` archived at `swh:1:dir:2849ae81e3c7dfde440e3677f5285e35347abd7d`

Acknowledgements We thank the anonymous reviewers for their constructive feedback, in particular for the suggestion of the ABox-based algorithm for typing described in Sec. 5.2.2.

Received 2025-07-21 **Accepted** 2026-03-20 **Published** 2026-03-31

1 Introduction

There is a dichotomy between the formalization of structural and the formalization of behavioral knowledge, which can be expressed through knowledge graphs and programming languages, respectively. We address this dichotomy by introducing a semantic lifting from program states to description logic (DL) ontologies that enables imperative programs to exploit a semantic view of their own state during execution. This way, structural knowledge can be used from within behavioral knowledge.

Knowledge graphs and ontologies are eminently useful representations of formal knowledge about the individuals and universals of systems. Among others, they give us decidable reasoning, easy avenues for negotiating domain knowledge with non-technical stakeholders, "native" ways of integrating information sources, and, not least, a wealth of well established standards. However, they are less suitable for the representation of change, and in particular dynamic behavior. Although concepts of change have been investigated ontologically [75, 76], and time stamped sensor readings can be represented in RDF [31], the essence of state change remains external to description logic-based knowledge representation, and *how* states change is not readily expressed.

In contrast, programming languages are specifically designed to describe *behavior*, i.e., the evolution of systems. The most common use of programming languages is to specify programs to be executed, but the use of programming languages for behavioral modeling for simulation and analysis is also well established [2,37]. In fact, the object-oriented programming paradigm emerged from discrete event simulation languages as a more natural way of representing the interaction between different entities [14]. However, the systems specified by programming languages are rarely pure models, but contain additional implementation-driven structure that interferes with domain modeling and may even become the dominant view of a system, especially when independently developed models need to be integrated.

It is natural to ask for a formalism that combines the advantages of semantic technologies for the representation of states with the elegance and maturity of programming languages to describe the execution and evolution of states. Different approaches have been proposed that attempt such a combination. For example, one can try to express program behavior in terms of actions on a description logic interpretation [77] or a DL ontology [9]. A recent approach [19, 20] has combined a guarded command language with DL reasoning to enable probabilistic model checking over the combination. A combination of RDF and rewriting theories in Maude has also been investigated [16, 74]. These approaches are all quite far from current state-of-the-art programming paradigms, and come with their own set of technical challenges. Other approaches, such as Owlready [53] or libraries operating directly on the OWL metamodel such as the OWL-API [35], give up the separation of concerns between ontological modeling and data modeling, a problem known as the semantic gap or impedance mismatch [3, 13, 43]. The semantic gap leads to a lack of tool support and prohibits the use of patterns aiming at code reuse and behavior in the program directly.

While processes and other kinds of behavior can be described by ontologies [32], they follow different patterns and aims than programs, which describe *executable behavior* and are thus concerned with code reuse. For example, behavioral subtyping [58] is a crucial pattern that expresses that from the perspective of a caller, the contract of a method m in a class C has also to hold for m in all subclasses of C. This pattern is not decidable (as it compares two Turing complete methods), and is not useful in a purely conceptual setting where execution and code reuse are not present or are not presented.

We propose a connection between programs and knowledge graphs that integrates both kinds of knowledge: we develop a semantic lifting that maps from program states in an object-oriented programming language to an RDF graph, including the running program's objects, fields, and call stack. Abstraction is supported in the mapping by integrating computations in the lifting process, thereby allowing, e.g., implementation-specific structure to be ignored by the mapping. The RDF graph can be exposed within the programming language, which adds a *semantic reflection* layer to programs. This reflection layer enables *semantic programming* where the semantic view of the state can be exploited by the program; in particular, formalized knowledge of the application domain can be used within the program by querying for objects using domain knowledge.

In this paper, we focus on the essence of semantic lifting and semantic reflection: the paper formalizes semantic lifting of object-oriented program states and semantic reflection for a small programming language SMOL (short for Semantic Micro Object Language) and explains both the operational aspects of the language and the mapping between states and RDF graphs; further, we discuss type correctness and virtualization for queries on semantically lifted program states from within the programs. An important aspect of this work lies in the intricate relationship between object-oriented typing, and class membership and subclassing in RDF(s).

Contributions

This paper, which builds on work published at ESWC [44], reports on a strand of research on semantically lifted programs. Compared to the previous paper, this paper features a reworked presentation of SMOL based on our experiences with several case studies and applications – including the removal of features that proved to be less useful in practice.

This paper includes the following technical improvements to semantic lifting and semantic reflection, compared to the original publications on semantic lifting [44] and its type system [46]:

1. a new *semantic pointer mechanism* that explicitly connects the program knowledge graph with a domain knowledge graph;
2. The *ontology* of the lifting has been remodeled, compared to the previous work [44];
3. a *full formalization* of the type system, including a new result that shows that all reachable states are semantically lifted to *consistent* knowledge graphs; and
4. a discussion of the *virtualization* of semantically lifted program states.

We furthermore discuss several published case studies and applications of semantic lifting outside the SMOL language.

Paper Overview

Section 2 gives a general overview of semantic lifting and reflection by means of a motivating example. Section 3 introduces SMOL, a small object-oriented language and Section 4 details its semantic lifting mechanism. Section 5 explains semantic reflection in SMOL and type safety for queries through the semantic reflection layer. We discuss the implementation of SMOL and describe how our work with applications influenced the language design in Section 6. Related work is reviewed in Section 7 and Section 8 concludes the paper.

A Note on Notation

We assume a general familiarity with the standard Semantic Web stack of RDF, OWL, SPARQL and SHACL; for an introduction, see, e.g., Hitzler et al. [34] and the online documentation.[1]

Some notions, most prominently "class" and "object", denote different entities in program semantics and knowledge representation. In cases where the exact meaning is not clear from the immediate context, we use "concept", "individual" and "node" for the knowledge representation entities and "class", "instance" and "runtime object" for the program semantics entities.

In this paper, we use DL syntax for axioms in the program semantics and RDF Turtle syntax in examples. Given a SPARQL query Q, an entailment regime er and an knowledge graph $\mathcal{K}$, the function $\mathsf{Ans}_{\mathsf{er}}(\mathcal{K}, \mathsf{Q})$ returns the result set, $\mathsf{Sha}(\mathcal{K}, \mathsf{shacl})$ returns a Boolean depending on whether $\mathcal{K}$ conforms to the SHACL shape shacl, and $\mathsf{Mem}(\mathcal{K}, \mathsf{owl})$ returns all members of the OWL concept owl in $\mathcal{K}$. Query containment for queries Q_1 and Q_2 under an entailment regime er and a knowledge graph $\mathcal{K}$ is denoted $\mathsf{Q}_1 \subseteq^{\mathcal{K}}_{\mathsf{er}} \mathsf{Q}_2$.

[1] https://www.w3.org/TR/rdf11-primer/

2 Motivating Example

We introduce the techniques of semantic lifting and semantic reflection through a motivating example to illustrate how these techniques allow us to combine domain knowledge for static modeling and programming for dynamic modeling. We consider an example based on a simulator for geological processes, developed in `SMOL` by Yu et al. [70], to show how complex domain knowledge expressed in an ontology can be integrated into a program.

Let us implement a program that simulates geological processes in a system that captures the deposition and erosion of geological layers in petroleum geoscience, as well as the transformation of organic matter inside these layers to petroleum. The program needs to access domain knowledge about conditions that trigger such transformations in order to perform a meaningful simulation. Whereas Yu et al. [70] considered a realistic ontology for this domain, our ontology will be simplified to focus on the interactions between the program and the ontology.

A *petroleum system* in the energy industry describes the different entities that relate to hydrocarbon production and storage [66]. We focus on the physical-geological components and processes that are involved in the formation of hydrocarbon accumulation, which can be separated into three classes: *physical-geological components*, the different geological layers and their types of rocks and properties; *thermal transformations*, the processes describing transformation and accumulation of hydrocarbons within these layers; and *compaction*, the change of physical properties in the rock during its burial. We consider stacks of layers; i.e., the geological layers are layered upon each other.

We distinguish between *source rock*, that can generate petroleum, and *reservoir rock*, that can store it. Each layer has one type of homogeneous rock as material, where we model *shale*, *limestone*, and *sandstone*. In our model, each layer of rock has homogeneous rock properties such as grain size, porosity, and permeability.

Given a description of the state of a geological system, different geological processes can affect the layers. Let us consider *cooking*, which transforms *kerogen* in a source rock into petroleum. Kerogen refers to a collection of large and complex, insoluble molecules that are dehydrated from fresh organic matter after burial and compaction by overlying at least 100 m sediments [4].

Temperature plays a key role during kerogen's thermal transformation, although other factors such as pressure, time, and mineral type also play a role. We concentrate on the North Sea and the Norwegian Sea, where the general gradient is about 30 $°C$ increase in temperature for each kilometer depth [4, 61]. Cooking of oil starts at 60 $°C$ [4].

The static models, i.e., knowledge graphs and ontologies, are used to model the structure of the domain and the current state of the geological layers. The dynamic models, i.e., programs, are used to describe the processes that transfer the system between states. At their interaction, we must be able to interpret the program state in the static model and retrieve information from it to determine the triggering layers for the processes.

2.1 An Ontology for the Static Model

The concepts of layers, their properties and their relation to each other can be described in an ontology. The ontology does not describe processes, but rather describes *triggers*: A layer is a trigger if it fulfills the conditions to trigger some geological process. For example, a layer is a *cooking trigger*, if it *(a)* contains uncooked kerogen, *(b)* is below a certain minimal depth and *(c)* is above a certain maximal depth.

The basic geological notions that we need for our simulator are organic matter, rocks and layers. Organic matter is kerogen, oil or gas. These notions are represented as follows.

```OWL
domain:Oil SubClassOf domain:OrganicMatter
domain:Gas SubClassOf domain:OrganicMatter
domain:Kerogen SubClassOf domain:OrganicMatter
```

We here focus on two types of rocks, *shale* and *sandstone*, among the different rocks and layers. A layer consists of one kind of rock and may contain organic matter. We model stratigraphic layers that are stacked on each other.

```OWL
domain:SiliciclasticRock SubClassOf domain:Rock
domain:Shale SubClassOf domain:SiliciclasticRock
domain:Sandstone SubClassOf domain:SiliciclasticRock
domain:StratigraphicLayer SubClassOf domain:constitutedBy exactly 1 domain:Rock
domain:StratigraphicLayer SubClassOf
        domain:constitutedBy only domain:SiliciclasticRock
```

In addition to the geological notions, we model *triggers*. A trigger is a stratigraphic layer that enables some process. We focus on the trigger for the *cooking* process here; in general, any layer can be in a state that triggers a process. Thus, a trigger is a layer, expressed using the following axiom:

```OWL
domain:Trigger SubClassOf domain:StratigraphicLayer
```

A layer can trigger the cooking process if it contains kerogen and is below 2000 m depth but above 5000 m. We do not describe the cooking process itself, i.e., what happens to the kerogen during or after cooking, in the ontology.

```OWL
domain:CookingTrigger EquivalentTo domain:Trigger
  and (domain:constitutedBy some (domain:contains some domain:Kerogen))
  and (domain:depth some xsd:integer[≥ 2000, ≤ 5000])
```

2.2 A Program for the Dynamic Model

The `SMOL` program uses the ontology developed in Section 2.1 to simulate geological process. The program's input is a *geological scenario*, which is a sequence of deposition and erosion events, and its output is the final state of the system. The program's internal structure mirrors the structure of the domain, so its central data structure is a stack of geological layer objects.

Observe that these geological layers play a dual role as both computational and domain-specific artifacts [41]. On one hand, they implement behavior like migration of hydrocarbons or perform computations like their current depth. On the other hand, they relate to the domain knowledge encoded in the above axioms. Let us first examine the classes in Figure 1. They model generic geological layers as class `GeoLayer`, with a state that includes a given thickness, depth and neighboring layers, and methods to manipulate the state. The `Bedrock` class describes the lowest layer of rock that we consider in our scenarios. The `Shale` class specializes `GeoLayer` to a layer that contains only shale. This class has a field `kerogen` that contains the status of kerogen within the modeled layer. If this field has value 1 or 2, the layer contains kerogen, if the field has value 0, the layer has no kerogen, if the field has any other value, the layer contains overcooked kerogen.

SMOL

```
abstract class GeoLayer(domain Int thickness,  domain depth,
                        hidden GeoLayer above, hidden GeoLayer below)
  Unit update()
    Int res = 0;
    if(this.above != null) then
      res = this.above.depth + this.above.thickness;
    end
    this.depth = res;
  end
  Boolean canPropagate() return False; end
  Unit migrate() skip; end
end

class Bedrock extends GeoLayer() end

class Shale extends GeoLayer(hidden Int kerogen)
  links(this.kerogen == 1 || this.kerogen == 2)
    "a domain:Stratigraphic_Layer;
     domain:constitutedBy [a domain:Shale];
     domain:constitutedBy [domain:contains [a domain:Kerogen]].";
  links "a domain:Stratigraphic_Layer;
         domain:constitutedBy [a domain:Shale].";
  Unit cook() this.kerogen = this.kerogen +1; end
end
```

Figure 1 Geological layers in the simulator.

Let us now examine the semantic lifting of a `Shale` object, for the moment ignoring the **links** clause, and the **domain** and **hidden** modifiers of the class definition (see Figure 1). For this example, we consider an object created with the following statements.

SMOL

```
Bedrock bed =
  new Bedrock(/*thickness:*/100, /*depth:*/100, /*above:*/null, /*below:*/null);
Shale sh =
  new Shale(/*kerogen:*/1,/*thickness:*/100,/*depth:*/0,/*above:*/null,/*below:*/bed);
bed.above = sh;
```

Figure 2 shows an excerpt of the resulting semantic lifting (ignoring the modifiers and special clauses, and the class table). It is a serialization in RDF, outlined for a node `run:obj1` for the shale object and a node `run:obj2` for the bedrock object.

Observe that the semantic lifting of objects, without any connection to the domain ontology, is already useful. For example, we can use SHACL to formulate the restriction that (a) there is only one object acting as bedrock and (b) a bedrock object is the lowest one. In other words, semantic technologies can be used as a specification language for object-oriented programs, with built-in support for logical inference. Reflective features have been explored in object-oriented programming languages such as CLOS, Java and Smalltalk [8, 50], to support, e.g., introspection by implementing methods that reveal structural aspects of the program such as the class of a given object. Semantic lifting also enables introspection, but it happens outside of the runtime system: we can use SPARQL to retrieve objects that satisfy particular logical properties, without the need to manually traverse the state using a debugger. We refer to this way of using the lifted state, which is external to the program semantics, as *semantic state access*.

RDF

```
prog:GeoLayer a owl:Class;
prog:Shale owl:subClassOf prog:GeoLayer.
prog:Bedrock owl:subClassOf prog:GeoLayer.
run:obj1 a prog:Shale;
         prog:kerogen 1;  prog:thickness 100; prog:depth 0;
         prog:above smol:null; prog:below run:obj2.
run:obj2 a prog:Bedrock;
         prog:thickness 100; prog:depth 100;
         prog:above run:obj1; prog:below smol:null.
```

Figure 2 Excerpt of the lifting without modifiers and linking clause.

RDF

```
prog:GeoLayer a owl:Class;
prog:Shale owl:subClassOf prog:GeoLayer.
prog:Bedrock owl:subClassOf prog:GeoLayer.
run:obj1 a prog:Shale; smol:links run:l1.
run:l1 domain:thickness 100; domain:depth 0;
       a domain:Stratigraphic_Layer; domain:constitutedBy [a domain:Shale];
       domain:constitutedBy [domain:contains [a domain:Kerogen]].
```

Figure 3 Excerpt of the lifting with modifiers and linking clause.

The `Shale` object is lifted as a node of class `prog:Shale`. This class is *not* part of the domain ontology. In fact, this node is not part of the geological domain at all: If it were, the node would have the properties of the `domain:StratigraphicLayer` class and be restricted by the axioms governing the domain ontology. Such a design would be problematic because this would restrict the program with constraints not concerned with computational structures and merge the domain model with the computational model.

We want to preserve the separation of concerns between these two modeling paradigms, and instead *link* the lifted state to the domain. For each `SMOL` object, two nodes are generated: one representing the object itself (the above `run:obj1`) and one node representing an entity in the domain to which the object is linked. These two objects are connected using a special relation `smol:links`.

Semantic lifting serializes the program state, and provides a way to specify how domain objects link to the program state. In the `SMOL` code of Figure 1, these are the modifiers and the **links** clause. The **hidden** modifiers prohibit a field from being lifted. This allows us to control the knowledge graph: As parts of the program that are unrelated to the operations to the lifted state are removed, the resulting graph is (a) smaller and (b) more focused. In contrast, the **domain** modifier moves information from the computational object to the linked domain node. In the example above, this will attach the edge lifting field `depth` not to the object `run:obj1`, but to its linked node.

The **links** clause is a general way to annotate information to the linked object. The clause in the `Shale` class (see Figure 1) expresses that every node linked to the lifting of a `Shale` object is a stratigraphic layer constituted by shale. The class has two **links** clauses. The first is conditional – if the expression **this**`.kerogen == 1 || `**this**`.kerogen == 2` evaluates to true, then the linked object contains kerogen, otherwise the unconditional clause is used and the linked object does not contain kerogen. This way, the semantic lifting precisely captures the meaning of the `kerogen` field in terms of the domain ontology. The above `Shale` object is, when these features are considered, lifted in the graph in Figure 3. Here, the object `run:l1` is the linked object.

SMOL

```
List<Shale> layers = member("<smol:links> some <domain:CookingTrigger>");
while(layers != null) do
  Shale layer = layers.content;
  layer.cook();
  layers = layers.next;
end
```

Figure 4 Executing the cooking process.

Semantic state access can be used to exhibit the state. For example, the following query extracts all objects containing kerogen (more precisely, all `SMOL` objects that are linked to an OWL object that contains kerogen).

SPARQL

```
SELECT ?x { ?x [smol:links [domain:contains [a domain:Kerogen]]}
```

Queries can be executed from within the program to reflect on the state. We refer to such queries as *semantic reflection*, because the domain ontology and the semantically lifted program state are directly used in the program. Consider the code in Figure 4, which queries for all `Shale` objects that are linked to a layer triggering the cooking process. In our work, we use semantic reflection to facilitate the following:

- **A separation of concerns** between the modeling of structure, such as layers, their properties and relations to each other, and the modeling of behavior, i.e., changes in these structures.
- **A prevention of redundancy**: the properties of the layers must not be expressed in both the program and the ontology. Instead, the ontology is used directly.
- **A semantic view:** The queries are expressed in the terminology of the domain, using standard semantic technologies accessible to domain experts.

3 SMOL: An Object-Oriented Language with Semantic Lifting

This section introduces semantic lifting by defining a programming language and its runtime semantics, allowing us to formalize the mapping from program state to knowledge graph and detail the consequences of this mechanism for programming language design. As mainstream object-oriented languages, such as Java, are unnecessarily complex to present their complete and formal runtime semantics here, we do so by introducing `SMOL` (short for *Semantic Micro Object Language*), a small object-oriented language with an ALGOL-inspired syntax, enhanced with semantic lifting.

We introduce `SMOL`, emphasizing syntactic support for semantic lifting, and formally define `SMOL` in terms of *surface syntax* and *runtime syntax*. The surface syntax describes the program as written by the programmer, while the runtime syntax describes its internal representation during execution. The runtime semantics, i.e., the rules to execute a program, is defined as transitions between states described in the runtime syntax. To focus on semantic lifting, we elide many standard aspects of `SMOL`'s semantics; for completeness, the full language semantics is included in Appendix A. We will extend `SMOL` to investigate semantic reflection (i.e., the ability to access the knowledge graph generated by the semantic lifting at runtime from within a program) in Section 5.

Prog	::= $\overline{\text{Class}}$ **main** Stmt **end**	Programs
Class	::= **class** `C` [**extends** `C`] ($\overline{\text{Field}}$) [Linkage] $\overline{\text{Met}}$ **end**	Classes
Type	::= `t` | `C` | `List<C>`	Types
Field	::= [**hidden** | **domain**] Type `f`	Fields
Linkage	::= **links**`(`$\overline{\text{Expr}}$`) le;` **links** `le;`	Domain linkage
Met	::= Type `m(`$\overline{\text{Type v}}$`)` Stmt **end**	Methods
Stmt	::= Loc `=` RHS `;` | **if** Expr **then** Stmt **else** Stmt **end**	Statements
	| Expr`.m(`$\overline{\text{Expr}}$`);` | **skip**`;` | **while** Expr **do** Stmt **end**	
	| Type `v =` RHS`;` | Stmt Stmt | **return** Expr`;`	
RHS	::= **new** `C(`$\overline{\text{Expr}}$`)` [Linkage] | Expr`.m(`$\overline{\text{Expr}}$`)` | Expr	RHS expressions
Expr	::= **this** | **null** | Loc | `a` | Expr *op* Expr | Expr `==` Expr | Expr `!=` Expr	Expressions
Loc	::= Expr`.f` | `v`	Locations

Figure 5 Surface syntax of `SMOL`.

3.1 Surface Syntax

Assume the standard sets of literal values (i.e., constants), such as integers $\{1, 2, \ldots\}$, Booleans $\{$`true`, `false`$\}$ and the unit and null singletons $\{$`unit`$\}$ and $\{$**null**$\}$, respectively, given; we refer to the names `Int`, `Boolean`, `Unit`, `Null` of these sets as *basic type names*. For now, we consider basic type names as purely syntactic constructs; we return to the type system in Section 5.2. In the sequel, let $\overline{\cdot}$ denote comma-separated lists (i.e., zero or more repetitions), and $[\cdot]$ denote optional constructs.

▶ **Definition 1** (Surface Syntax). *The syntax of* `SMOL` *is given by the grammar in Figure 5, where* `C`*,* `f`*,* `m`*,* `v` *range over* class, type variable, field, method and variable names*, respectively, which are strings. We also let* `le` *range over lists complying* `predicateObjectList` *production in Turtle syntax,*[2] `b` *over string literals,* `t` *over basic type names,* `a` *over literal values (including string literals), and op over Boolean and arithmetic operators (such as* $+$ *and* $\leq$*).*

We use blue bold keywords to highlight syntax relevant for semantic lifting, and black bold keywords for all other syntax highlighting.

A program in `SMOL` consists of a set of classes and a **main** block with a statement. A class declaration Class defines fields and methods. Classes can extend other classes (using single inheritance). For simplicity, if a class extends another, then all fields and methods of the superclass are copied to the subclass. Inherited fields are placed before newly declared fields. Types are basic types, class names, or lists of class names. Thus, to avoid unnecessary complexity, we do not include generic types and restrict parametric types to lists of class names only.

Statements `s` and expressions `e` are standard, including a **null** reference and the self reference **this**. Right-hand sides RHS extend expressions with imperative constructs with side effect. These include object creation and method calls. For simplicity, these can only occur in assignments. Consequently, nested object creation and method calls inside expressions need to be encoded. Method calls can additionally occur as standalone statements (in which case the return value from the method call is ignored). Moreover, fields `f` in `SMOL` are publicly accessible, and field access is always prefixed by the target object (e.g., **this**`.f`).

[2] See `https://www.w3.org/TR/turtle/#grammar-production-predicateObjectList`; the missing subject of the `predicateObjectList` will be filled in at runtime by an individual for the object.

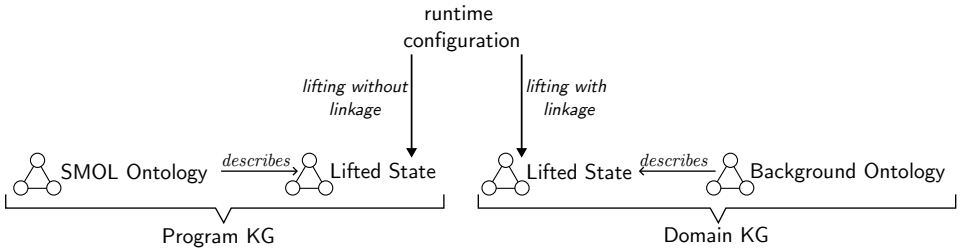


Figure 6 High-level overview over the relation between lifting and different part of the knowledge graph.

The constructs **hidden**, **domain** and **links** are specific to SMOL. These constructs enable a certain control of the semantic lifting. We here introduce these constructs informally, as their formal introduction requires the exact structure of the semantic lifting (see Section 4). The lifted knowledge graph consists of two parts: the *program knowledge graph* that describes the state itself and the *domain knowledge graph* that describes context knowledge provided by the user. This is shown in Figure 6. The difference between the two parts is how the lifted configuration is described, either in terms of the SMOL language (in the program knowledge graph) or in terms of the domain (in the domain knowledge graph).

Let us first introduce the optional field modifiers **hidden** and **domain**. The modifier **hidden** excludes the field from semantic lifting; i.e., the field will not have a counterpart in the lifted knowledge graph. The modifier **domain** treats the field not as part of the program knowledge graph, but as additional information in the domain knowledge graph. In addition, SMOL supports *domain* Linkage as a programming construct with the **links** keyword, which connects the program knowledge graph explicitly to the domain knowledge graph. Domain linkage can also be used with object creation.

▶ **Example 2** (A SMOL Program). We consider a program $\mathsf{Prog}_{\mathsf{street}}$ modelling urban infrastructure, shown in Figure 7. The program defines classes `Room`, `Building` and `Street` that include references to each other, as well as the size of a room and the accumulated size of a building. The **main** statement block of $\mathsf{Prog}_{\mathsf{street}}$ creates three rooms, which are in two buildings in a single street.

3.2 Runtime Syntax and Semantics of SMOL without Reflection

We briefly introduce the *runtime syntax* and *semantics* of SMOL programs, the formalisms used to define program execution, before semantic lifting is detailed in Section 4 and semantic reflection in Section 5. Runtime syntax describes *runtime configurations*, i.e., terms representing the states of a program at different steps of the program execution. The runtime semantics of SMOL formalizes program execution by defining an evaluation function on expressions and a transition system between configurations. This transition system itself is given in Appendix A, as the transitions without the concepts of semantic reflection are standard.

Compared to the surface syntax given in Section 3.1, the runtime configurations, which are specified by the runtime syntax, describe the statements left to execute, the class table, the process stack, the memory store of each object and the local memory store of each process on the stack.

Lists `List<C>` are a special construct in the syntax of SMOL, which enforces that lists cannot be nested and avoids full generics, but allows lists to be treated as classes when it comes to typing and runtime semantics: A list type `List<C>` is treated as a class without methods and two fields: `C content` and `List<C> next`. In the sequel, we include lists whenever we refer to classes.

SMOL

```
1 class Room(Int size) end
2 class Building(List<Room> rooms, Int size, Street street)
3    Unit addRoom(Room room)
4      this.rooms = Cons(room, this.rooms);
5      this.size = this.size + room.size;
6    end
7 end
8 class Street(List<Building> buildings, String name)
9   Unit addBuilding(Building building)
10     this.buildings = Cons(building, this.buildings);
11     buildings.street = this;
12   end
13 end
14 main
15   Room r1 = new Room(10);
16   Room r2 = new Room(20);
17   Room r3 = new Room(30);
18   Building b1 = new Building(null, 0, null); b1.addRoom(r1);
19   Building b2 = new Building(null, 0, null); b2.addRoom(r2); b2.addRoom(r3);
20   Street s1 = new Street(null, "Problemveien");
21   s1.addBuilding(b1); s1.addBuilding(b2);
22 end
```

Figure 7 A SMOL program $\mathsf{Prog}_{\mathsf{street}}$ for urban infrastructure.

We start with the formal definition of a *class table*, which represents static information about the fields and methods of the classes defined in a program.

▶ **Definition 3** (Class Table). *The* class table CT *is a map from class names to sets of field declarations and method declarations. The lists of fields and methods of the (instantiations of the) classes specified by* CT *can be accessed, for a class* `C`*, via functions* $\mathsf{fields}_{\mathsf{CT}}(\mathtt{C})$ *and* $\mathsf{methods}_{\mathsf{CT}}(\mathtt{C})$ *respectively. Functions* $\mathsf{vars}_{\mathsf{CT}}(\mathtt{C.m})$, $\mathsf{ret}_{\mathsf{CT}}(\mathtt{C.m})$ *and* $\mathsf{body}_{\mathsf{CT}}(\mathtt{C.m})$ *are used to access the list of variables, return type and body of a method* `m` *in* `C`*, respectively.*

In addition to the static information about a program captured in its class table, program states at runtime need to represent dynamically created information, including the program's objects and process call stack. Let a *domain element* (*DE*) be either a literal value (for a basic type) or an *object reference* (for a class). The formal representation of a program state is a *runtime configuration*, defined as follows.

▶ **Definition 4** (Runtime Configurations). *A* local store σ *is a map from variables to DEs and an* object store ρ *is a map from fields to DEs. Let* CT *be a class table and* `X` *range over object identifiers (the remaining terms are defined in Definition 1).* Configurations conf, objects obs *and* processes prs *are defined by the following grammar:*

$$\mathsf{conf} ::= \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs} \qquad \mathsf{rs} ::= \mathsf{Stmt} \mid \mathsf{Loc} \leftarrow \mathtt{stack;}\ \mathsf{Stmt} \qquad \mathsf{Cl} ::= \mathtt{C} \mid \mathtt{List<C>}$$
$$\mathsf{obs} ::= \overline{(\mathsf{Cl}, \rho)_{\mathtt{X}}} \qquad \mathsf{prs} ::= \overline{(\mathtt{m}, \mathtt{X}, \mathsf{rs}, \sigma)}$$

Besides the class table CT, a runtime configuration conf contains objects obs and processes prs. An object $(\mathsf{Cl}, \rho)_{\mathtt{X}}$ has a unique identifier (i.e., name) `X` and contains its class `C` (or a list `List<C>`) and the object's store ρ. A process $(\mathtt{m}, \mathtt{X}, \mathsf{rs}, \sigma)$ contains the name `m` of the method it is

$$
\begin{aligned}
\mathsf{fields}(\mathsf{CT}_{\mathsf{street}}, \texttt{Room}) &= \{\texttt{Int size}\} \\
\mathsf{fields}(\mathsf{CT}_{\mathsf{street}}, \texttt{Building}) &= \{\texttt{List<Room> rooms}, \texttt{Int size}, \texttt{Street street}\} \\
\mathsf{fields}(\mathsf{CT}_{\mathsf{street}}, \texttt{Street}) &= \{\texttt{List<Building> buildings}, \texttt{String name}\} \\
\mathsf{methods}(\mathsf{CT}_{\mathsf{street}}, \texttt{Room}) &= \emptyset \\
\mathsf{methods}(\mathsf{CT}_{\mathsf{street}}, \texttt{Building}) &= \{\texttt{Unit addRoom(Room room)}\ \mathsf{Stmt}_{\texttt{addRoom}}\ \textbf{end}\} \\
\mathsf{methods}(\mathsf{CT}_{\mathsf{street}}, \texttt{Street}) &= \\
&\quad \{\texttt{Unit addBuilding(Building building)}\ \mathsf{Stmt}_{\texttt{addBuilding}}\ \textbf{end}\} \\
\mathsf{vars}(\mathsf{CT}_{\mathsf{street}}, \texttt{Building}, \texttt{addRoom}) &= \{\texttt{Room room}\} \\
\mathsf{vars}(\mathsf{CT}_{\mathsf{street}}, \texttt{Street}, \texttt{addBuilding}) &= \{\texttt{Building building}\} \\
\mathsf{vars}(\mathsf{CT}_{\mathsf{street}}, \texttt{Entry}, \texttt{entry}) &= \emptyset \\
\mathsf{body}(\mathsf{CT}_{\mathsf{street}}, \texttt{Building}, \texttt{addRoom}) &= \mathsf{Stmt}_{\texttt{addRoom}} \\
\mathsf{body}(\mathsf{CT}_{\mathsf{street}}, \texttt{Street}, \texttt{addBuilding}) &= \mathsf{Stmt}_{\texttt{addBuidling}} \\
\mathsf{body}(\mathsf{CT}_{\mathsf{street}}, \texttt{Entry}, \texttt{entry}) &= \mathsf{Stmt}_{\mathsf{street}}
\end{aligned}
$$

Figure 8 Class table for program $\mathsf{Prog}_{\mathsf{street}}$ from Example 2.

executing, the identifier `X` of the object in which it executes, a runtime statement rs which remains to be executed and a local store σ. The list of processes in a configuration may be seen as a stack corresponding to nested method calls. To capture the transfer of return values between method calls at runtime, we use *runtime statements* rs, which extend the statements Stmt with an additional statement $\mathsf{Loc} \leftarrow \texttt{stack}$ that identifies the location Loc that is waiting for a return value from the next process on the stack. Each process on the stack, except for the top process, starts with this runtime statement.

The connection between surface and runtime syntax is established when execution starts: the program (in surface syntax) is translated into an *initial* runtime configuration, defined as follows.

▶ **Definition 5** (Initial Configuration). *Let* `E` *be an object identifier. The initial configuration of a program* Prog *is* $\mathsf{CT}_{\mathsf{Prog}}\ (\texttt{Entry}, \emptyset)_{\texttt{E}}\ (\texttt{entry}, \texttt{E}, \mathsf{Stmt}, \emptyset)$*, where* $\mathsf{CT}_{\mathsf{Prog}}$ *is the class table for* Prog*, extended with an additional class* `Entry` *that has a single, parameter-free method* `entry` *with the statement* Stmt *of the main block as its body.*[3]

In initial configurations, the empty sets denote the initially empty stores.

▶ **Example 6** (Initial Configuration). Figure 8 shows the class table $\mathsf{CT}_{\mathsf{street}}$ for the program $\mathsf{Prog}_{\mathsf{street}}$ from Example 2, where $\mathsf{Stmt}_{\texttt{m}}$ is the method body of `m` and $\mathsf{Stmt}_{\mathsf{street}}$ the statement of the main block. The initial configuration of $\mathsf{Prog}_{\mathsf{street}}$ is then $\mathsf{CT}_{\mathsf{street}}\ (\texttt{Entry}, \emptyset)_{\texttt{E}}\ (\texttt{entry}, \texttt{E}, \mathsf{Stmt}_{\mathsf{street}}, \emptyset)$.

At every point during execution, the state of a program can be represented by means of runtime syntax. We return to the rules that capture program execution in Section 5, when the full language including semantic reflection has been introduced.

[3] We assume, without loss of generality, that no program explicitly declares a class with name `Entry`.

4 Graph-Based State Semantics

Let us now consider the SMOL ontology, which describes the OWL classes and properties needed to describe the runtime configurations of executing SMOL programs, and then semantic lifting, a direct mapping that translates such runtime configurations into a set of triples. Semantic lifting allows a runtime configuration to be interpreted as a knowledge graph by serializing it in RDF, using the vocabulary introduced below, and adding the triples needed for domain linking.

4.1 An Ontology for SMOL

The SMOL-ontology[4] $\mathcal{K}_{\texttt{SMOL}}$ consists of a *language layer* that describes elements present in all programs, such as classes, fields and methods, and a *runtime layer* that describes the objects of a specific runtime configuration. Statements, expressions and processes are not lifted.

The IRIs of all entities in our ontology share a common prefix, which is added to the IRI by means of a function: $\cdot^{\texttt{SMOL}}$. For readability, we use the prefix `smol:` in examples, or omit the prefix altogether if it is clear from the context that we are concerned with the language layer.

During semantic lifting, two additional prefixes are used to distinguish knowledge about the program and about a specific runtime state. Given a program, the function $\cdot^{\texttt{prog}}$ (example prefix `prog:`) generates a fresh IRI based on the current program – two programs that share some code can still be distinguished this way. The function $\cdot^{\texttt{run}}$ (example prefix `run:`) generates fresh IRIs based on the current state. Two states during a run of the same program are thus lifted into separate entities, connected by the entities of the common lifted program.

▶ **Definition 7** (SMOL Ontology). *Let $\mathcal{K}_{\texttt{SMOL}}$ be the union of the axioms in Figures 9 and 10.*

The *language layer* consists of classes ($\texttt{Class}^{\texttt{SMOL}}$), methods ($\texttt{Method}^{\texttt{SMOL}}$) and fields ($\texttt{Field}^{\texttt{SMOL}}$). Each class has a string as its name ($\texttt{hasName}^{\texttt{SMOL}}$), and fields and methods are connected to the class in which they are declared. Fields and methods can be connected to more than one class, due to inheritance. All these concepts are disjoint. Finally, we define the classes $\texttt{Any}^{\texttt{SMOL}}$, $\texttt{Unit}^{\texttt{SMOL}}$ and $\texttt{List}^{\texttt{SMOL}}$. Figure 9 gives the axioms formally. We use an object property $\texttt{subClass}^{\texttt{SMOL}}$ to express inheritance between SMOL classes, thus avoiding interactions between inheritance in OWL and object-oriented programming.

The *runtime layer* consists of objects ($\texttt{Object}^{\texttt{SMOL}}$). An important individual introduced here is $\texttt{null}^{\texttt{SMOL}}$, which implements the type $\texttt{Any}^{\texttt{SMOL}}$. Membership of SMOL objects to SMOL classes is expressed through $\texttt{implements}^{\texttt{SMOL}}$. Figure 10 gives the axioms formally and introduces the $\texttt{links}^{\texttt{SMOL}}$ relation used for domain linkage. Note that we define its domain, but not its range, which depends on a specific application. The class $\texttt{MemoryEntry}^{\texttt{SMOL}}$ and the properties $\texttt{hasEntry}^{\texttt{SMOL}}$, $\texttt{hasValue}^{\texttt{SMOL}}$, $\texttt{hasPointer}^{\texttt{SMOL}}$, and $\texttt{entryOf}^{\texttt{SMOL}}$ are used to model the memory of an object, where $\texttt{hasPointer}^{\texttt{SMOL}}$ is used for fields of object type and $\texttt{hasValue}^{\texttt{SMOL}}$ for fields of a basic data type.

▶ **Example 8** (Semantically Lifted Memory). Consider two objects `o1` and `o2` of a class `C`, where a field `f` of `o1` points to `o2`. Semantic lifting will generate a graph where the prefixes mirror the origin of the different elements: the relations `hasEntry`, `entryOf` and `hasPointer` are from the ontology (prefixed by $\cdot^{\texttt{SMOL}}$), the field `f` is part of the program (prefixed by $\cdot^{\texttt{prog}}$), while the objects `o1` and `o2` and the memory entry `e1` are from the runtime configuration (prefixed by $\cdot^{\texttt{run}}$).

[4] Note that $\mathcal{K}_{\texttt{SMOL}}$ is a knowledge graph – the term ontology here expresses that it contains general knowledge, which is applicable to a whole range of programs and configurations.

OWL

```
Class: Class^SMOL
Class: Method^SMOL
Class: Field^SMOL

DataProperty: hasName^SMOL
  Domain: Class^SMOL and Method^SMOL and Field^SMOL Range: xsd:String
  Characteristics: Functional

ObjectProperty: subClass^SMOL Domain: Class^SMOL Range: Class^SMOL
  Characteristics: Transitive

ObjectProperty: hasMethod^SMOL Domain: Class^SMOL Range: Method^SMOL

ObjectProperty: hasField^SMOL Domain: Class^SMOL Range: Field^SMOL

Individual: Any^SMOL Types: Class^SMOL
Individual: List^SMOL Types: Class^SMOL Facts: subClass^SMOL Any^SMOL, hasName^SMOL "List"
Individual: Unit^SMOL Types: Class^SMOL Facts: subClass^SMOL Any^SMOL

AllDisjointClasses(Class^SMOL, Method^SMOL, Field^SMOL)
```

Figure 9 Axioms for the language layer of $\mathcal{K}_{\text{SMOL}}$.

OWL

```
Class: Object^SMOL
Class: MemoryEntry^SMOL

ObjectProperty: implements^SMOL
  Domain: Object^SMOL Range: Class^SMOL Characteristics: Functional

ObjectProperty: hasEntry^SMOL
  Domain: Object^SMOL Range: MemoryEntry^SMOL  Characteristics: InverseFunctional

ObjectProperty: hasPointer^SMOL
  Domain: MemoryEntry^SMOL Range: Object^SMOL Characteristics: Functional

DataProperty: hasValue^SMOL
  Domain: MemoryEntry^SMOL Characteristics: Functional

ObjectProperty: entryOf^SMOL
  Domain: MemoryEntry^SMOL Range: Field^SMOL  Characteristics: Functional

ObjectProperty: links^SMOL
  Domain: Object^SMOL Characteristics: Functional, InverseFunctional

Individual: null^SMOL Types: Object^SMOL Facts: implements^SMOL Any^SMOL
```

Figure 10 Axioms for the runtime layer of $\mathcal{K}_{\text{SMOL}}$.

RDF
```
run:o1 smol:hasEntry run:e1.
run:e1 smol:entryOf prog:f.
run:e1 smol:hasPointer run:o2.
```

An alternative design would here be to use the *punning* feature of OWL 2 [28, 30] that allows using the same URI for an individual, a property, and a class. These will be separate entities semantically, only sharing an identifier, and which one is meant can always be deduced from the context. We can thus let `prog:f` be both an OWL object and an OWL property. This approach, which we also used in Section 2, allows the following, more succinct lifting.

RDF
```
run:o1 prog:f run:o2.
```

There are no consequences for reasoning since the individual and the property will be kept apart. But this technique allows for more intuitive queries without the need for a specific query interface for, e.g., debugging. For these reasons, `SMOL` supports both kinds of semantic lifting;[5] we will continue to use punning in examples.

4.2 Domain Linkage

Before detailing the technical aspects of semantic lifting itself, we explain another novel aspect of `SMOL`: the **links** clause. The purpose of the **links** clause is to connect the program knowledge graph to the domain knowledge graph, thereby associating domain knowledge directly to the runtime state of `SMOL` programs. The **links** clause works similarly to `case` statements in imperative languages: it defines a sequence of guarded expressions, where each guard is a Boolean expression. Additionally, it contains an unguarded expression, which we represent by the guard `true`. The semantics of the **links** clause is that, during lifting, link guards are evaluated in the listed order, and the link expression of the first guard that evaluates to `true` is used to generate an additional axiom in the knowledge graph.

To this aim, we introduce expressions with holes and substitution of terms for holes in these expressions. Here, a hole is an expression that needs to be filled with a term before the expression can be evaluated (the terminology stems from Felleisen and Hieb's work on context-reduction semantics [23]). Let $\bullet$ denote a hole in an expression Expr and Expr$[X]$ the corresponding substitution of the hole by a term X. Thus, for example, $\bullet$ `!=` 5 is an expression with a hole and the substitution ($\bullet$ `!=` 5)[2] reduces to the expression 2 `!=` 5.

▶ **Definition 9** (Domain Linkage). *Let X be an object identifier,* `e` *a Boolean expression and* conf *a runtime configuration.*

- *A* link expression `le` *is an axiom with a hole for its subject.*
- *Let* `le`$[X]$ *denote the axiom obtained by filling the hole in* `le` *by* $X^{\texttt{run}}$.
- *A* guarded *link expression is a pair* (`e`, `le`).
- *A* domain linkage $\mathbb{L}$ *is a sequence of guarded link expressions.*

We denote by $\mathbb{L}[X, \mathsf{conf}]$ *the axiom* `le`$[X]$ *obtained by filling the hole in the first guarded link expression* (`e`, `le`) *in* $\mathbb{L}$ *such that* `e` *evaluates to* `true` *in the runtime configuration* conf.

For a given program, all link expressions in rule Linkage in the grammar of Definition 1 will form a domain linkage, where the last case **links** `le` is interpreted as the link expression (`true`, `le`).

[5] The implementation has an option to switch between the two kinds of lifting.

▶ **Example 10.** Consider a production by the rule Linkage in the grammar of Definition 1 of the form

links(e$_1$) le$_1$; ... **links**(e$_{n-1}$) le$_{n-1}$; **links** le$_n$;

This production gives rise to the domain linkage

$$\big((\texttt{e}_1, \texttt{le}_1), \ldots, (\texttt{e}_{n-1}, \texttt{le}_{n-1}), (\texttt{true}, \texttt{le}_n)\big)$$

Domain linkages can be associated with SMOL classes as well as with individual SMOL objects (by annotating the **new** constructor). Given a class C, we denote by links(C) its associated domain linkage. Similarly, given an object with identifier X, we represent by links(X) its associated domain linkage, which is by default that of its class. However, if an object has its own domain linkage, this linkage overrides the domain linkage of its class.

Since SMOL uses predicate object lists in Turtle syntax for link expressions without a subject, the holes are left implicit and the operation le[iri] is realized by simply concatenating iri as a prefix to the link expression le.

▶ **Example 11** (Domain Linkage)**.** Consider the following variant of the Building from Example 2, that links to the domain based on the accumulated size of all its rooms.

SMOL
```
1 class Building(List<Room> rooms, Int size, Street street)
2   links (this.size >= 100) "a domain:BigHouse.";
3   links "a domain:SmallHouse.";
4    Unit addRoom(Room room) ... end
5 end
```

The corresponding domain linkage for instances of class Building is defined by

$$\big((\texttt{this.size >= 100}, \texttt{"a domain:BigHouse."}), (\texttt{true}, \texttt{"a domain:SmallHouse."})\big)$$

Given an IRI domain : obj1 (which is not run:obj1, see Section 4.3) and a runtime configuration conf in which obj1.size = 20,

$$\begin{aligned}\mathbb{L}[\texttt{domain} : \texttt{obj1}, \texttt{conf}] &= \bullet\ \texttt{a domain:SmallHouse.}[\texttt{domain:obj1}] \\ &= \texttt{domain:obj1 a domain:SmallHouse.}\end{aligned}$$

since the first guard evaluates to false, and the (implicit) second guard evaluates to true in conf.

We denote by $\mathbb{L}_{\texttt{X}}$ the domain linkage for an object X. Figure 11 illustrates different semantic liftings of an object, depending on its state, domain linkage and **domain** annotations.[6]

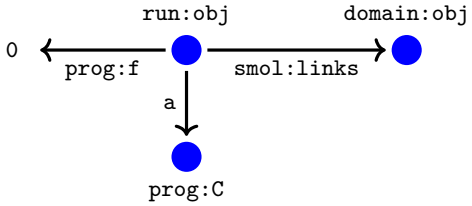
4.3 Semantic Lifting

We define a direct mapping to lift runtime configurations into knowledge graphs, extending the SMOL ontology $\mathcal{K}_{\texttt{SMOL}}$ of Definition 7. The availability of domain knowledge then enables the runtime state of the program to be accessed externally (i.e., via the knowledge graph), in terms of the vocabulary and axioms of the domain, formalized as an ontology. In order to connect

[6] The notation $\%f$ is analogous to non-answer variables in queries and replaced by the literal stored in the field at the moment of lifting. For simplicity, we omit this notation in our formalization (but it is implemented in the SMOL interpreter).

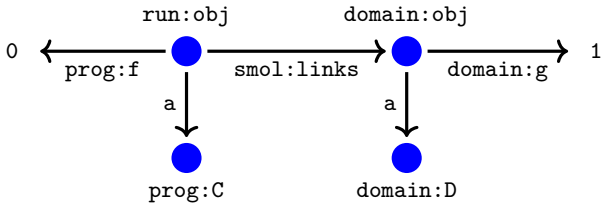
SMOL
```
// this.f == 0
class C(Int f)


end
```



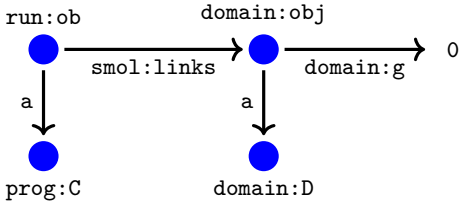
SMOL
```
// this.f == 0, this.g == 1
class C(Int f, domain Int g)
  links "a domain:D"

end
```

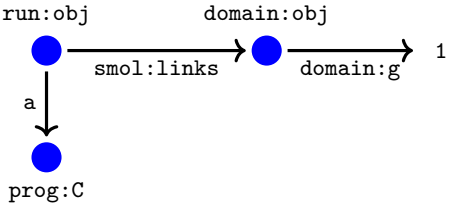


SMOL
```
// this.f == 0,
class C(hidden Int f)
  links "domain:g %f"

end
```



SMOL
```
// this.f == 1,
class C(hidden Int f)
  links(this.f > 0) "domain:g %f"
  links "a domain:D"
end
```



SMOL
```
// this.f == 0,
class C(hidden Int f)
 links(this.f > 0) "domain:g %f"
 links "a domain:D"
end
```

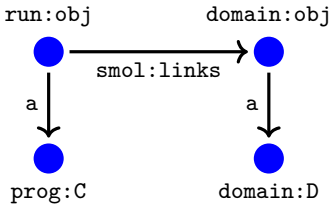


Figure 11 Dynamic variations of semantic lifting, depending on domain linkage and annotations. The `prog:f` and `domain:g` edges are short notation for the entities.

the resulting program knowledge graph to a domain knowledge graph, the domain knowledge graph needs to be a conservative extension [59] of $\mathcal{K}_{\texttt{SMOL}}$, to ensure that the domain knowledge cannot introduce inconsistencies in the lifted runtime configurations (assuming that the domain knowledge graph is consistent in the first place):

▶ **Definition 12** (Domain Knowledge Graph). *Domain knowledge is given as a knowledge graph* $\mathcal{K}_{\mathbf{domain}}$*, and a function* $\cdot^{\mathbf{domain}}$ *that adds a prefix to IRIs, such that* $\mathcal{K}_{\mathbf{domain}}$ *is a conservative extension of* $\mathcal{K}_{\texttt{SMOL}}$*.*

The direct mapping generates the remaining part of the knowledge graph, namely the graph lifted from the current runtime configuration. Recall from Example 8 how the different prefixes mirror the origin of the different lifted elements. The two layers have mutually exclusive prefixes, added by functions $\cdot^{\mathbf{prog}}$ and $\cdot^{\mathbf{run}}$.

▶ **Definition 13** (Direct Mapping). *Given a runtime configuration* $\texttt{conf} = \mathsf{CT}\ \mathsf{ob}_1 \ldots \mathsf{ob}_n\ \mathsf{prs}$*, the* direct mapping μ *is a function from runtime configurations to knowledge graphs defined as follows:*

$$\mu(\texttt{conf}) = \bigcup_{\texttt{C} \in \mathbf{dom}(\mathsf{CT})} \mu(\texttt{C}) \cup \bigcup_{1 \leq \texttt{X} \leq n} \left(\mu(\mathsf{ob}_\texttt{X}) \cup \mathbb{L}_\texttt{X}[\texttt{X}^{\mathbf{run}}, \texttt{conf}]\right) \cup \mathit{close}\ .$$

The mapping $\mu(\texttt{C})$ *of a class* `C` *is defined in Figure 12 and the mapping* $\mu(\mathsf{ob}_\texttt{X})$ *of an object with identifier* `X` *in Figure 13. The axiom set close is defined as follows. Let* $\texttt{C}_1, \ldots$ *be all classes in* CT*,* $\texttt{m}_1, \ldots$ *all methods in* CT*,* $\texttt{f}_1, \ldots$ *all fields, and* $\texttt{X}_1, \ldots$ *all object identifiers, then the following axioms are part of the ontology.*

OWL

```
Class^SMOL  EquivalentTo: { C_1^prog, ... }
Method^SMOL EquivalentTo: { m_1^prog, ... }
Field^SMOL  EquivalentTo: { f_1^prog, ... }
Object^SMOL EquivalentTo: { X_1^run, ... }
```

The axioms added by *close* are used to explicitly state all members of the classes in the `SMOL` ontology. Intuitively, these axioms ensure that despite an open world assumption, one cannot infer the existence of objects that must exist (according to the domain knowledge graph), unless they also exist in the given runtime configuration.

The lifting of classes in Figure 12 follows the structure of the class table. Inherited methods and fields are considered different between super- and subclass, as they are redeclared in the subclass. The lifting of objects in Figure 13 differentiates between fields holding values of basic data types and fields pointing to other objects, because of the distinction between data and object property in OWL. We assume that for every basic data type `T` there is an xsd equivalent that can be retrieved with $\mathsf{xsd}(\texttt{T})$, and analogously for literals.

We illustrate how the runtime configuration of a program can be accessed in terms of a formalized domain vocabulary in the following example.

▶ **Example 14** (Querying Runtime States with Domain Knowledge). Recall the class `Building` from Example 2:

SMOL

```
class Building(List<Room> rooms, domain Int size, Street street)
   Unit addRoom(Room room) ... end
end
```

OWL

```
Individual: C^prog
  Facts: a Class^SMOL, hasName^SMOL "C",
  [subClass^SMOL D^prog],      // if C extends D
  [subClass^SMOL Any^SMOL],    // otherwise
  hasMethod^SMOL m_1^prog, ..., hasMethod^SMOL m_n^prog,
  hasField^SMOL f_1^prog, ..., hasField^SMOL f_k^prog

Individual: m_1^prog Facts: a Method^SMOL, hasName^SMOL "m_1"
...
Individual: f_1^prog Facts: a Field^SMOL, hasName^SMOL "f_1"
...
```

Figure 12 The lifting of a class C with methods $\mathtt{m}_1,\ldots,\mathtt{m}_n$ and fields $\mathtt{f}_1,\ldots,\mathtt{f}_k$.

Now assume that a *villa* is a building with a surface of more than 300 square meters. This assumption can be expressed in the domain knowledge graph as follows:

OWL

```
domain:Villa EquivalentTo: domain:size some xsd:integer[<= 300]
```

Although villas are not defined in the SMOL program, objects in the runtime configuration of the program that qualify as villas can nevertheless be retrieved from the combined knowledge graph by the following query:

SPARQL

```
SELECT ?obj {?obj smol:links [a domain:Villa] }
```

The query returns the SMOL objects that are linked to villas.

Recall from Section 4.1 that fields may also be lifted as properties (so-called punning). The additional axioms are given in Figure 14, again differentiating between data and object properties.

OWL

```
Individual: X^run
    Facts: a Object^SMOL, implements^SMOL C^prog,
    links^SMOL X^domain,
    hasEntry^SMOL e_{f_i}^prog,
    ...  // for every class that is not annotated domain or hidden

Individual: X^domain
    hasEntry^SMOL e_{f_i}^prog,
    ...  // for every class that is annotated domain but not hidden

Individual: e_{f_1}^prog
    Facts: a MemoryEntry^SMOL,
    [hasPointer^SMOL ρ(f_1)^prog],      // if ρ(f_1) is an object identifier
    [hasValue^SMOL xsd(ρ(f_1))],        // otherwise
    entryOf^SMOL f_1^prog, ...
```

Figure 13 The lifting of an object $(\mathtt{C},\rho)_{\mathtt{X}}$, where class C has fields $\mathtt{f}_1,\ldots,\mathtt{f}_k$.

```
OWL
1 ObjectProperty: f_i^prog Domain: C^prog Range: D^prog  // if the field has type D
2 DataProperty: f_i^prog Domain: C^prog Range: xsd(T)    // if the  field has data type T
```

```
OWL
1 Individual: X^run
2     Facts: a Object^SMOL, implements^SMOL C^prog,
3     [f_1^SMOL ρ(f_1)^prog],          // if ρ(f_1) is an object identifier
4     [f_1^SMOL xsd(ρ(f_i))],          // otherwise
5     ...
```

Figure 14 Alternative liftings of objects and classes if fields are modeled as both object properties and individuals using punning.

5 Semantic Reflection

In this section, we explain semantic reflection by showing how a running SMOL program can interact directly with the knowledge graph obtained by semantic lifting from its own runtime configuration. We have seen in Section 4 how semantic lifting allows the representation of a program state in the knowledge graph to be controlled, using additional structures in the programming language to connect the program knowledge graph to a domain knowledge graph. Semantic lifting enables *external* queries to investigate a program state through a semantic, domain specific lens from the outside, which can be used for debugging or to access computation results after a program execution. In contrast, semantic reflection enables the program itself to directly interact with the semantically lifted runtime state and the domain knowledge during execution.

Semantic reflection is a powerful technique that enables semantic state access from *within* the program, which gives programs the ability to explore their own runtime state through a domain-specific lens, and to use this exploration to influence program behavior. Technically, we combine the semantic lifting of configurations during execution with language support to perform operations on the knowledge graph.

5.1 Language Support for Semantic Reflection

We consider language extensions that operate on knowledge graphs. These extensions only extend the grammar of SMOL (see Figure 5) with additional RHS expressions.

To allow dynamic, but type-safe queries, we consider expressions **access** to ask for objects that satisfy a SPARQL query, **member** to ask for objects that are members of an OWL concept, and **validate** to check if the knowledge graph satisfies a particular SHACL shape. In these queries, we use a slightly extended version of SPARQL by allowing, at every point in the grammar of SPARQL where a variable may occur in a graph pattern,[7] the use of a *parameter variable* $\%i$. These parameter variables are replaced by IRIs before the query is executed. This is analogous to SQL prepared statements in, e.g., Java libraries.[8] In particular, we require that graph patterns P in SPARQL SELECT queries are such that (1) the set of parameter variables form an interval $[\%1, \ldots, \%n]$ for some $n \in \mathbb{N}$ and (2) there is a query substitution mechanism, denoted $\mathtt{P}(v_1, \ldots, v_n)$, that syntactically replaces these variables by n values $v_1, \ldots, v_n$.

[7] See https://www.w3.org/TR/sparql11-query/#GraphPattern
[8] See https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html

SMOL

```
class Inspector()
 Unit inspect(String streetName)
  List<Building> over =
     access("SELECT ?x {?x smol:links [a domain:Villa];
                            prog:street [prog:name %1]. }",
            this.streetName);
  for b in over do
    this.inspectBuilding(b);
  end
 end
end
```

Figure 15 Using domain knowledge to influence the execution in SMOL.

▶ **Definition 15** (Extended Surface Syntax). *The grammar in Definition 1 is extended as follows:*

$$\text{RHS} ::= \ldots \mid \textbf{access}(\texttt{sparql}, \overline{\texttt{Expr}}) \mid \textbf{member}(\texttt{owl}) \mid \textbf{validate}(\texttt{shacl}) \qquad \textit{RHS Expressions}$$

where sparql *ranges over the extended SPARQL SELECT queries with one answer variable,* owl *over OWL concepts and* shacl *over SHACL shapes.*

The **access** expression returns a list of objects, resulting from the extended SPARQL query given as the first parameter. These objects *must* exist prior to the execution of this expression. The other parameters to this expression are query parameters; the query substitution mechanism reduces them to a standard SPARQL query. The **member** expression returns the list of objects which are members of the OWL concept in its parameter. The **validate** expression applies the SHACL shape in its parameter and returns a Boolean, depending on whether the knowledge graph of the semantic lifting satisfies this shape or not.

Before we formalize semantic reflection, we illustrate its use by an example to show how domain knowledge about the runtime configuration of a program can be accessed directly in the program.

▶ **Example 16** (Programmer Access to the Domain Knowledge Graph). Assume that we need to perform an inspection of all villas in a given street, continuing from Examples 2 and 14. The code in Figure 15 illustrates a possible implementation in SMOL using semantic reflection. It is left to the domain knowledge to define the meaning of `domain:Villa`, which can consequently be changed according to different scenarios outside of the SMOL program. In the query, the variable %1 is replaced by the literal passed as the second argument to the method.

The example shows how semantic lifting not only exposes the structure of the implementing runtime environment but adds domain knowledge, which one can access and use in the programs themselves by means of semantic reflection.

We now discuss how semantic reflection can be realized operationally by formalizing its behavior. To this aim, we define a semantics for the execution of SMOL programs that captures both semantic lifting and semantic reflection. We here only consider the essential aspects of these operations; the full structural operational semantics [68] is included in Appendix A.

Let us consider a transition relation $\mathsf{conf}_1 \rightarrow_{\mathsf{er}}^{\mathcal{K}_{\mathbf{domain}}} \mathsf{conf}_2$ defined by a set of transition rules, where conf_1 and conf_2 are runtime configurations of `SMOL`, er is a SPARQL entailment regime[9] and $\mathcal{K}_{\mathbf{domain}}$ is some domain knowledge according to Definition 12. Let $\mathsf{conf}_1 \rightsquigarrow_{\mathsf{er}}^{\mathcal{K}_{\mathbf{domain}}} \mathsf{conf}_2$ denote the *reachability* in the operational semantics, i.e., the transitive closure of the transition relation, and by $\mathsf{conf}_1 \Downarrow_{\mathsf{er}}^{\mathcal{K}_{\mathbf{domain}}} \mathsf{conf}_n$ the *maximal reflexive-transitive closure* of this relation.

We denote by $\mathsf{listify}(\mathit{des})$ an auxiliary function that takes a set *des* of domain elements and returns a `SMOL` list $\mathsf{obs}_{\mathtt{Y}}$ containing an object for each of the domain elements, where the subscript `Y` denotes the object identifier of the head of the list. The objects in the list are fresh in the usual sense of object creation: they have new and unique identifiers. The function listify fails (i.e., it is undefined) if the input list mixes different literal types, or mixes literals with object identifiers.

We first explain the behavior of **validate**. In this case, the next statement to be executed contains a **validate** expression with some shape `shacl`. After the transition, the **validate** expression is replaced by a (side-effect-free) assignment to the same location, but with the query-result res as its RHS. This Boolean literal results from evaluating the conformity of the lifted configuration together with the SMOL ontology and the domain knowledge graph. Otherwise, the objects and processes of the configuration are not changed.

▶ **Definition 17** (Semantics of **validate**). *Let* $\mathcal{K}_{\mathbf{domain}}$ *be a knowledge graph,* er *an entailment regime and* conf *a configuration of the form*

$$\mathsf{conf} = \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m}, \mathtt{X}, \mathsf{Loc} = \textbf{validate}(\mathtt{shacl}); \mathsf{Stmt}, \sigma)$$

where the next statement to execute in the top process contains a **validate** *expression. Let* res *be the result of checking the lifted configuration against the SHACL shape(s)* `shacl`*:*

$$\mathsf{res} = \mathsf{Sha}\big(\mathcal{K}_{\mathtt{SMOL}} \cup \mathcal{K}_{\mathbf{domain}} \cup \mu(\mathsf{conf}), \mathtt{shacl}\big)\ .$$

Recall that res *is a Boolean value in* `SMOL`*. The transition from* conf *is defined as*

$$\mathsf{conf} \rightarrow_{\mathsf{er}}^{\mathcal{K}_{\mathbf{domain}}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m}, \mathtt{X}, \mathsf{Loc} = \mathsf{res}; \mathsf{Stmt}, \sigma)\ .$$

The behavior of **member** is similar to **validate** in the sense that execution returns an object identifier `Y`, which is the head of a list of `SMOL` objects.

▶ **Definition 18** (Semantics of **member**). *Let* $\mathcal{K}_{\mathbf{domain}}$ *be a knowledge graph,* er *an entailment regime and* conf *a configuration of the form*

$$\mathsf{conf} = \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m}, \mathtt{X}, \mathsf{Loc} = \textbf{member}(\mathtt{owl}); \mathsf{Stmt}, \sigma)$$

where the next statement to execute in the top process contains a **member** *expression. Let* res *be the result of performing the membership query* `owl` *on the lifted configuration:*

$$\mathsf{res} = \mathsf{Mem}\big(\mathcal{K}_{\mathtt{SMOL}} \cup \mathcal{K}_{\mathbf{domain}} \cup \mu(\mathsf{conf}), \mathtt{owl}\big)\ ,$$

which is a set of IRIs. Let $\mathsf{obs}_{\mathtt{Y}} = \mathsf{listify}\big(\mathsf{res}\big)$ *be the representation of this set as a* `SMOL` *list. If* $\mathsf{obs}_{\mathtt{Y}}$ *is defined, then the transition from* conf *is defined as*

$$\mathsf{conf} \rightarrow_{\mathsf{er}}^{\mathcal{K}_{\mathbf{domain}}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{obs}_{\mathtt{Y}}\ \mathsf{prs}, (\mathtt{m}, \mathtt{X}, \mathsf{Loc} = \mathtt{Y}, \sigma)\ .$$

If $\mathsf{obs}_{\mathtt{Y}}$ *is not defined (see above), then the behavior of* **member** *is also not defined.*

[9] See `https://www.w3.org/TR/sparql11-entailment/`

We finally explain the behavior of **access**. In this case, the next statement to be executed contains an **access** expression with some query `sparql` and expressions $\mathsf{Expr}_1, \ldots, \mathsf{Expr}_n$. The expressions $\mathsf{Expr}_1, \ldots, \mathsf{Expr}_n$ are evaluated in the current state and their results substituted for the parameter variables in the query. The resulting query is evaluated using the semantically lifted configuration, producing a set of domain elements *des* from which a `SMOL` list with objects $\mathsf{obs}_\mathtt{Y}$ and head `Y` is constructed. These objects are then added to the configuration and the statement reduced to an assignment of `Y` into the target location.

▶ **Definition 19** (Semantics of **access**). *Let* $\mathcal{K}_{\mathbf{domain}}$ *be a knowledge graph,* er *an entailment regime and* conf *a configuration of the form,*

$$\mathsf{conf} = \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m}, \mathtt{X}, \mathsf{Loc} = \textbf{access}(\mathtt{sparql}, \mathsf{Expr}_1, \ldots, \mathsf{Expr}_n); \mathsf{Stmt}, \sigma)\ ,$$

where the next statement to execute in the top process contains an **access** *expression. Let* $[\![\mathsf{Expr}]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}}$ *denote the result of evaluating an expression* Expr *and* res *the result of performing the SPARQL query* `sparql` *on the semantically lifted configuration, with all parameter variables replaced by the literals resulting from the corresponding expressions:*

$$\mathsf{res} = \mathsf{Ans}_{\mathsf{er}}\big(\mathcal{K}_{\mathtt{SMOL}} \cup \mathcal{K}_{\mathbf{domain}} \cup \mathcal{K}_{\mathsf{conf}}, \mathtt{sparql}[[\![\mathsf{Expr}_1]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}} \ldots [\![\mathsf{Expr}_n]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}}]\big)\ ,$$

which is a set of IRIs.[10] *Let* $\mathsf{obs}_\mathtt{Y} = \mathsf{listify}(\mathsf{res})$ *be the representation of this set as a* `SMOL` *list. If* $\mathsf{obs}_\mathtt{Y}$ *is defined, then the transition from* conf *is defined as*

$$\mathsf{conf} \to^{\mathcal{K}_{\mathbf{domain}}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{obs}_\mathtt{Y}\ \mathsf{prs}, (\mathtt{m}, \mathtt{X}, \mathsf{Loc} = \mathtt{Y}, \sigma)\ .$$

If $\mathsf{obs}_\mathtt{Y}$ *is not defined (see above), then the semantics of* **access** *is not defined.*

5.2 Eliminating Runtime Failures for Semantically Reflected Programs

The clash of two different class models[11] and the interaction between the programming and semantic layers may be challenging for the programmer. We here consider static techniques to ensure that interaction between these layers happens correctly. At the level of syntax, we can enforce some constraints on statements with **access**, **member** and **validate** expressions to avoid programming errors; for example, the language extensions for semantic reflection should contain syntactically correct SPARQL queries, OWL concepts and SHACL shapes. A particular concern is that the answers to queries over a knowledge graph are generally (untyped) multisets of IRIs, whereas `SMOL` programs are otherwise typed. In this section, we consider the following failures that are specific to semantically reflected programs.

- **Representation Failure:** When executing an **access** expression, the query may return a set of IRIs that cannot be represented as values at runtime. For example, the query

  ```
  SELECT ?x {prog : obj1 ?x 1}
  ```

  returns a set of predicates, which cannot be translated to `SMOL` objects.
- **Location Failure:** While representation failures manifest when the semantic reflection is performed, a failure to respect the type of the target location may lead to later runtime errors. For example, a program could execute a query that returns string literals and then perform numerical operations on the elements of the result list. This will cause a delayed error, once the first string in the list is accessed and used for an operation expecting an integer.

[10] We remind that we only allow SPARQL queries with a single answer variable here, see Definition 15.
[11] Remark that the *impedance mismatch* (or semantic gap) between object-oriented and ontology/database class models is a general phenomenon [3], and not specific to semantic reflection.

SMOL
```
1 class C (String str) end
2 ...
3 C c = new C("a");
4 List<Int> res = access("SELECT ?x {?o prog:str ?x}");
5 print(res.content + 1); //runtime error
```

Assuming that the rest of the program is correct, the problem is that the query loads the results into a location of type `List<Int>`. If we assume that types are not represented at runtime, then this error occurs in line 5 when the data is processed, as the addition operator fails. Storing the results in line 4 will succeed.

- **Inconsistency:** If a semantically lifted state results in an inconsistent knowledge base, then query answering is not defined. As we lift the type of fields, the following program results in a query access over an inconsistent knowledge base. The knowledge graph contains the axiom $\texttt{D}^{\mathbf{prog}} \sqcap \texttt{C}^{\mathbf{prog}} \sqsubseteq \bot$, which stems from the class hierarchy. From the class table, the following axiom for the field `D.c` is generated: $\top \sqsubseteq \forall \texttt{D\_c}^{\mathbf{prog}}.\texttt{C}^{\mathbf{prog}}$. and the sole created object is lifted as an individual i with $\texttt{D}^{\mathbf{prog}}(i)$. These three axioms form an inconsistent knowledge graph.

SMOL
```
1 class C() end
2 class D(C c) end
3
4 main
5     D d = new D(null);
6     d.c = d;
7     List<C> l = access(...);
8 end
```

This is a different failure than location failure: while location failure leads to an error in the runtime semantics of the program, inconsistency leads to an error in the query answering. For example, in the above, the location failure does not lead to a runtime error because the field is never *read*.

5.2.1 A Type System for Semantic Reflection

A static type system *"makes sure a program does not go wrong"* [67], for some notion of *"going wrong"*. We present a type system for SMOL that eliminates errors related to representation failure, location failure and inconsistency. Specifically, the type system for SMOL ensures that semantic queries to the semantically lifted program return a list of IRIs and literals that can be represented by a value of the type of the target location (or a sub-type thereof) in the program. This is sufficient to guarantee consistency of the knowledge graph. The presented type system focuses on the program knowledge graph. Consequently, it does not cover domain linkage, which would require a deeper analysis depending on guard expressions – such an analysis has been left for future work.

We exploit the fact that each type in SMOL has a *direct* correspondence to a class in the knowledge graph, and tackle the three different kinds of semantic reflection as follows:

- **SHACL:** The **validate** expression should always return a Boolean if the expression's SHACL shape is syntactically well-formed. This can encode other reasoning or validation tasks as well [6, 7].
- **OWL:** Given a statement `l = member(C);`, where `l` has type `List<D>`, type checking is performed by concept subsumption: the parameter concept `C` must be a subsumed by `prog` : `D`. This can be checked directly using a reasoner.

- **SPARQL:** The most involved form of semantic reflection is querying with an **access** expression. Given a statement `l = access(Q);`, where `l` has type `List<D>`, type checking amounts to query containment under an entailment regime: Obviously, loading all elements of `prog : D` would be safe (i.e., representable in the runtime and respecting the type of the target location). This is equivalent to the query $Q_{\texttt{D}} = \texttt{SELECT ?x \{?x a prog : D\}}$. Consequently, we check whether the query returns a subset of $Q_{\texttt{D}}$, using the entailment regime for reasoning. We will introduce this kind of type checking in the following section.

These checks will be performed at compile time, i.e., without a specific state – it suffices to show that every reachable configuration is consistent with the `SMOL`-ontology $\mathcal{K}_{\texttt{SMOL}}$. If domain knowledge is used, it must be in the form of a conservative extension of this ontology [59].

We here focus on the handling of **access**. We do not detail the general setup and soundness proofs here; besides the cases for the semantic reflection rules, these are standard. For the full formal treatment of the type system, see Appendix B. We first introduce the typing environment that we use to keep track of field, variable and parameter types. Together with the class table, the typing environment provides context for typing judgments.

▶ **Definition 20** (Typing Environment). *A* typing environment Γ *is a partial function, mapping* locations *(fields, variables, method parameters) to types. The empty typing environment is denoted* $\emptyset$*.*

Given a program with a statement Stmt*, we use* Γ_{Stmt} *to denote the typing environment that maps (a) all fields of the class containing* Stmt *to their declared types, (b) all method parameters of the method containing* Stmt *to their declared types, (c) all variables declared before* Stmt *in this method to their declared types, and (d) is undefined for all other locations.*

Let $\Gamma, \mathsf{CT} \vdash^{\mathcal{K}_{\mathbf{domain}}}_{\mathsf{er}} \mathsf{Stmt}$ denote that a statement Stmt is well-typed in the context of an environment Γ, class table CT, domain knowledge graph $\mathcal{K}_{\mathbf{domain}}$ and entailment regime er. Similarly, let $\Gamma, \mathsf{CT} \vdash^{\mathcal{K}_{\mathbf{domain}}}_{\mathsf{er}} \mathsf{Expr} : \mathsf{Type}$ denote that an expression Expr has type Type in the given context.

▶ **Definition 21** (Typing Reflection). *Given a typing environment* Γ*, a class table* CT*, a domain knowledge graph* $\mathcal{K}_{\mathbf{domain}}$ *and an entailment regime* er*, the typing judgment*

$$\Gamma, \mathsf{CT} \vdash^{\mathcal{K}_{\mathbf{domain}}}_{\mathsf{er}} \texttt{List<c> v = access("SELECT ?obj \{P\}"}, \mathsf{Expr}_1, \ldots, \mathsf{Expr}_n\texttt{)}$$

holds if the following conditions can be satisfied:

- *The query parameters can be assigned types:* $\Gamma, \mathsf{CT} \vdash \mathsf{Expr}_i : \mathsf{Type}_i$ *for* $0 < i \leq n$
- *Query containment holds for the given types of the query parameter variables:*

$$\texttt{SELECT ?obj \{P. ?v}_1 \texttt{ a } \mu(\mathsf{Type}_1)\texttt{. .... ?v}_n \texttt{ a } \mu(\mathsf{Type}_n)\texttt{\}} \subseteq^{\mathcal{K}_{\texttt{SMOL}} \cup \mathcal{K}_{\mathbf{domain}}}_{\mathsf{er}} \texttt{SELECT ?obj \{?obj a c}^{\mathbf{prog}}\texttt{\}} .$$

Here, the first condition assigns types to all expressions that are used as parameters to the query. The derived types are then used to approximate the schema variable associated with this parameter in the second premise. To this aim, an *approximating triple* of the form $\texttt{?v}_i \texttt{ a } \mu(\mathsf{Type}_i)$ is generated for each parameter variable v_i typed with Type_i. The approximating triple expresses that the value used to instantiate the query must be of the given type. The second condition adds the approximating triples and checks (under the chosen entailment regime and background knowledge) whether the resulting query is contained in the query that retrieves all values of the type of the targeted location. In this case, every possible result of the query is a member of the type of the targeted location; i.e., the query only retrieves values of the correct type. We write $\vdash^{\mathcal{K}_{\mathbf{domain}}}_{\mathsf{er}} \mathsf{Prgm}$ to express that all statements within a program Prgm are well-typed.

The case for **member** is similar, but operates on OWL classes instead of SPARQL queries. The case for **validate** is straightforward; since SHACL queries take no input and return only true or false, the type of the expression must be a Boolean.

5.2.2 Optimizing Query Containment

The type system outlined in Section 5.2.1 is sound but not complete: it provides a fine-grained, but only sufficient condition for type safety of statements with **access** expressions, while necessity cannot be guaranteed since there are ABoxes that do not correspond to configurations.

Moreover, applicability of the type system is limited in practice by the fact that, as far as we are aware, there are no algorithms and tools for checking query containment over $\mathcal{SROIQ}(\mathbf{D})$ TBoxes under non-trivial entailment regimes. To overcome this issue, we consider a stronger sufficient condition, which is based on *concept subsumption* rather than query containment under entailment regimes. This approach is advantageous since concept subsumption is a main reasoning task for description logics, and there are practical systems (e.g., HermiT [27]) implementing efficient concept subsumption for the description logic underying OWL2 DL (i.e., $\mathcal{SROIQ}(\mathbf{D})$) and its fragments.

A unary query $\mathtt{Q}$ is *subsumed* by a concept C with respect to a knowledge graph (or TBox) $\mathcal{K}$, written $\mathtt{Q} \sqsubseteq^{\mathcal{K}} C$, if $s^{\mathcal{I}} \in C^{\mathcal{I}}$ for every certain answer s to $\mathtt{Q}$ over $\mathcal{K}$ and each model $\mathcal{I}$ of $\mathcal{K}$. In practice, we can syntactically construct a concept D from the query using a technique that guarantees the first subsumption ($\mathtt{Q} \sqsubseteq^{\mathcal{K}} D$ with respect to $\mathcal{K}$) to hold, and then check the second subsumption ($D \sqsubseteq^{\mathcal{K}} C$) by a description logic reasoner. A more specific (with respect to $\sqsubseteq^{\mathcal{K}}$) concept D ensures a more fine-grained sufficient condition for type safety. However, unless $\mathtt{Q}$ is equivalent (with respect to $\sqsubseteq^{\mathcal{K}}$) to a concept, there is no most specific concept D. Thus, there may be many techniques for constructing D from the query.

To be more concrete, we can consider a variation of this technique, applicable when the query $\mathtt{Q}$ is a conjunctive query (i.e., when the body consists of only basic graph patterns not mentioning any IRIs with special semantics) and the entailment regime is OWL2 DL (i.e., $\mathcal{SROIQ}(\mathbf{D})$). In this case, the containment $\mathtt{Q} \sqsubseteq^{\mathcal{K}} C$ boils down to, essentially, ABox reasoning over the body of the query seen as the ABox; in particular, we can just check, using a standard OWL2 reasoner, whether $I_{?obj}$ belongs to C over $\mathcal{K}$ extended with ABox part $\mathcal{A}_{\mathtt{Q}}$ that is obtained from the graph patterns of $\mathtt{Q}$ by replacing each variable $?x$ (including $?obj$) by a fresh IRI $I_{?x}$ (and translating them to an ABox in the standard way). Note that, formally, we do not have $\mathtt{Q} \sqsubseteq^{\mathcal{K}} D$ for the constructed ABox as D any more (even if we can translate the ABox to a concept using the standard internalization method), but the approach is still sound because the IRIs are fresh.

We can now state our type safety theorem that expresses two properties:

1. Program execution does not get stuck when using reflection, in particular not due to failure to translate the results of a query into internal data structures. (We here ignore reasons for failure that correspond to exceptions, such as null pointer access and division by zero.)
2. Every configuration reachable from a well-typed program lifts to a consistent knowledge graph.

Thus, even without reflection, the type system can give guarantees to programs that access a knowledge graph.

▶ **Theorem 1** (Type Safety). *Let* Prog *be a program that is well-typed with respect to* $\vdash_{\mathsf{er}}^{\mathcal{K}_{\mathbf{domain}}}$*, where* $\mathcal{K}_{\mathbf{domain}}$ *is a conservative extension of* $\mathcal{K}_{\mathtt{SMOL}} \cup \mu(\mathsf{CT}_{\mathsf{Prog}})$*. Every reachable configuration of* Prog *can be lifted to a consistent knowledge graph:*

$$\forall \mathsf{conf}.\ \mathsf{init}_{\mathsf{Prog}} \rightsquigarrow_{\mathsf{er}}^{\mathcal{K}_{\mathbf{domain}}} \mathsf{conf} \quad \rightarrow \quad \mu(\mathsf{conf}) \cup \mathcal{K}_{\mathtt{SMOL}} \cup \mathcal{K}_{\mathbf{domain}} \text{ is consistent.}$$

The proof is a standard inductive *subject reduction* proof [67], based on the structural operational semantics of SMOL (Appendix A) and a type system for runtime configurations (Appendix B).

6 Discussion

In this section, we discuss how the implementation of SMOL has been realized (Section 6.1), design choices for semantic lifting (Section 6.2) and applications of semantic reflection (Section 6.3).

6.1 An Interpreter for SMOL

SMOL has been implemented as an interpreter in an open-source project, available together with documentation and examples at www.smolang.org.[12] The interpreter is mainly written in Kotlin, so it compiles to Java class files and runs in the Java Virtual Machine (JVM) on most platforms, including Windows, macOS, and Linux. It uses ANTLR [63] to parse program files, which ensures that the SMOL grammar is followed strictly. For semantic data access, our implementation uses Apache Jena[13], OWL API [36], and ONT-API[14], and HermiT [27] is used as the default OWL reasoner.

Compared to the small language presented in this paper, the implemented language has some additional features that are orthogonal to semantic lifting, such as support for abstract classes, extended treatment of generics, support for loading SHACL shapes from files instead of string literals, further datatypes, a standard library, etc. In addition to the statements described in Definition 1, the full language supports local variables, a statement **destroy** for manual memory management, an extension to integrate simulation units [42] and some syntactic sugar for convenience (e.g., whole classes can be annotated with **hidden**).

In its simplest form, the interpreter takes a SMOL program as input and executes it by means of a process stack, a static table, and a memory heap. First, the interpreter clears the memory heap and scans the program to generate the initial program stack and the static table. Then it considers each subroutine from the stack and performs the required change to the memory heap until the stack is empty and the program execution is done. The project also includes a *Read-Eval-Print Loop* (REPL): an interactive environment that can be used to control and inspect the interpreter before, during, or after program execution. For example, the user can stop the program at any given state, query the state using SPARQL, change which sources and reasoners to use, check for consistency, or validate the data using SHACL.

Figure 16 gives an overview of how data from different sources is accessed semantically in our system. The available sources (left side) can be activated or deactivated independently. The active sources are combined into an integrated model that can be queried directly. If reasoning is required, then querying can either be done via the available inference model or via the OWL ontology interface. The static table and the heap, which are the two sources linked to the program state, are accessed virtually, i.e., statements are not materialized, but generated only when needed. These virtual sources use a guard mechanism to avoid traversing irrelevant parts of the internal data structure corresponding to the program state. The data access system is built with Jena, OWL API and ONT-API. The key parts of the data access system are detailed in the following.

6.1.1 Sources

Currently, the system supports four different data sources, but new sources can be added in the future as needed. Each of the four sources gives access to a particular set of statements:

[12] The version described in this work is available under `https://github.com/smolang/SemanticObjects/tree/prepare-1.0`.
[13] `https://jena.apache.org/`
[14] `https://github.com/owlcs/ont-api`

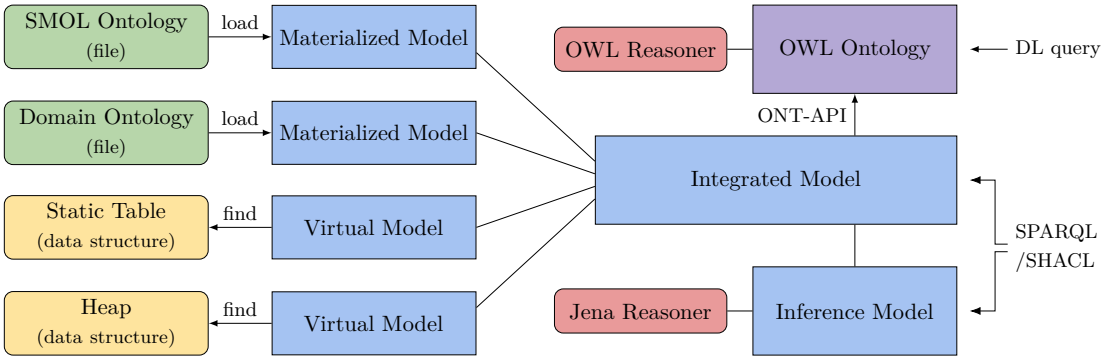


Figure 16 Diagram showing how data is accessed semantically.

- the `SMOL` *ontology* is a fixed OWL file describing the domain model for runtime configurations according to Definition 7;
- the *domain ontology* is an optional, static OWL file describing the model of the relevant domain (e.g., Geology in Section 2);
- the *static table* is an internal data structure containing static information about the current program, like the class hierarchy and each class' fields and methods; and
- the *heap* is an internal data structure containing all the objects constituting the current runtime configuration.

Both the `SMOL` ontology and the domain ontology are static and relatively small files, which are given as serializations of RDF. Hence, it is unproblematic to materialize their statements during the initialization of the system. The two remaining sources, on the other hand, are not provided as RDF statements, but as internal data structures coupled with a mapping to a corresponding RDF representation. For each of these two sources, this RDF representation is not materialized. Instead, the statements are accessed *virtually*; i.e., the statements are only generated when requested during query answering. While the static table remains static during runtime and is limited in size by the static parts of the program, the heap is dynamic and could potentially become very large. This shows the importance of accessing the heap virtually: materializing all RDF statements about the heap, which would need to be done every time is it accessed by a query, is very demanding.

6.1.2 Querying

The simplest way of accessing data is by means of a SPARQL query or by validating with SHACL directly on the integrated model. Alternatively, if a Jena reasoner is provided, it is possible to query with inference via the provided inference model. The third option is to send a description logic query to the available OWL ontology, which must be connected to a suitable OWL reasoner. This third approach is used by `SMOL`'s type checking mechanism. While SPARQL can query collections of RDF statements directly, DL queries instead require a set of OWL axioms. The translation from RDF to OWL in our system is done by OWL-API: OWL axioms are created when required, while the leftover statements are just translated to general assertion axioms. It should be clear that there is no limitation to who or what can access data, and when this can be done: queries can be posed externally by a user or internally by the program, and this can be done either before, during, or after the execution of the program.

6.1.3 Virtualization

Queries posed to the integrated model are distributed to the models that correspond to the different sources (see Figure 16). For the materialized models (here, the `SMOL` ontology and domain ontology), the query is simply evaluated over the available statements. For the virtual models, the system uses the corresponding mapping to generate answers. This mapping is manifested as an implementation of a search method *find*(t) for each source, which takes a search triple t as input and returns the set of statements matching this triple. For simple queries with just one triple, *find* only needs to be called directly once. For more complex queries, the query planner must first split the query into a set of multiple *find* calls and then combine the results from each such call into the final result set. In other words, the implementation of *find* is only responsible for traversing the relevant data structure and returning the statements matching t, while the query planner is responsible for transforming the query into *find* calls and combining the answers.

A naive way of implementing *find* could be to always traverse the whole internal data structure and collect statements matching t. Instead, our implementation carefully considers the search triple t and prevents the system from traversing the parts that will only lead to irrelevant statements. This is achieved by guards in the traversal by the interpreter, which is a simple control mechanisms that cannot be passed unless a given expression holds. For example, if we know that a given for-loop only generates statements of the form `(?v :y ?w)` and `(?v :z ?w)`, where `?v` and `?w` are variables, then there should be a guard in front of the loop to check if t matches either of the two forms. Placing guards into the code in a way that improves the efficiency of *find* requires a good overview of the data structure and the types of statements to which each part corresponds. It is worth noting that virtualization combined with reasoning does not work well in the current setup: many reasoners require initial materialization of all statements, which conflicts with the idea of virtual access.

6.2 Design Choices for Semantic Lifting

In this section, we discuss alternatives to the design of `SMOL`'s semantic lifting approach that has been formalised in this paper.

6.2.1 Abstraction

To emphasize semantic lifting, the design of `SMOL` supports *lifting by default* and the language offers hiding through explicit annotations in its semantic lifting mechanism. Most often, only a part of runtime configurations is actually queried in practice. Therefore, we have opted for a virtual model with a pull-based *find*-mechanism to lift parts of the heap upon need (discussed in Section 6.1.3). An alternative design, which we believe is a reasonable trade-off in most applications, would be to implement *hiding by default*, and only expose carefully selected pieces of information through the semantic lifting mechanism. Obviously, *hiding by default* can also profit from virtualization as outlined above.

Semantic lifting can also be integrated with an abstraction function. Taken together, hiding and abstraction would allow a more high-level representation of objects in the knowledge graph. Taken to the extreme, no actual fields would need to be lifted and an object could be represented in the knowledge graph solely though an abstraction function. Integrating abstraction in the semantic lifting process adds an additional layer of computation to the semantic lifting process. Pure methods can be used to operationally realize abstraction for the semantic lifting process (pure methods are methods without side-effects in object-oriented programming).

SMOL

```
class Building(List<Room> rooms, Int size, Street street)
  Unit addRoom(Room room)
    this.rooms = Cons(room, this.rooms);
  end
  rule domain Int size()
    Int res = 0;
    for r in rooms do
      res = res + r.size;
    end
    return res;
  end
end
```

Figure 17 Using pure methods in semantic lifting.

It is possible to make information that is implicit in the state of an object, explicit during the semantic lifting process by means of a similar computational layer. To materialize such implicit information in a program, one typically uses pure methods, as discussed above. For the purpose of semantic lifting, we could annotate such methods with a **rule** modifier, such that all annotated methods are executed on all objects from the class of the method during the lifting process.

▶ **Example 22** (Materializing implicit information during semantic lifting)**.** Consider a version of class `Building` from Example 2, in which instances of `Building` do not explicitly maintain their size in a field. The code of this version is given in Figure 17. Here, the size of a `Building` instance needs to be computed using the `size` method when needed, since it is not directly available. By annotating this method with **rule**, it is regarded as a field during semantic lifting; thus, the method is executed whenever the object is lifted and the result stored in a field `size` in the lifted object.

SMOL supported such computational semantic lifting in early versions (e.g., [44]), including a simple type system to ensure that these methods do not alter the state during lifting. The implementation called the method using the current function stack, executed it and stored the return value in the knowledge graph. This technique required arbitrary code to be executed; it was removed due to its low performance – while a powerful modeling tool, adding this computational layer to the semantic lifting process does not scale well for systems with many objects. Its role to derive information from the lifted state can either be done by a reflective architecture (see Section 6.3.1) or rule engine tools such as SWRL.[15]

6.2.2 Integrating Black-Box Components

For black-box components, the runtime state of a component will not generally be available for semantic lifting. In this case, we suggest to represent the state of the black-box component in the knowledge graph in terms of a semantically lifted interface, which provides a component descriptor and the input and output values that are exchanged between the program and the black-box component. This can be realized through a class definition that represents the component descriptor and a proxy object that exchanges input and output values between program and component.

[15] `https://www.w3.org/submissions/SWRL-FOL/`

SMOL

```
FMO[in Double y, out Double x] prey = simulate("Prey.fmu");
FMO[in Double x, out Double y] predator = simulate("Predator.fmu");
Int i = 0;
while (i++ <= iterations) do
   prey.y = predator.y;
   predator.x = prey.x;
   prey.tick(1.0);
   predator.tick(1.0);
end
```

Figure 18 A co-simulation of a prey-predator system.

Let us concretize this approach to the semantic lifting of black-box components by considering how functional mock-up units (FMUs) can be integrated into SMOL [42], thereby allowing numerical simulators to be embedded in SMOL programs. An FMU is a black-box component for a numerical simulator, as defined by the functional mock-up interface (FMI) [5], allowing simulation units and models to to exchanged for co-simulation [29]. An FMU is defined by a set of input and output ports. To perform the simulation, it provides a set of procedures to advance the simulation (and thus, update the output variables) for a certain amount of simulation time. The information about input and output ports, such as type and name, as well as other information, such as the tool used to generate the FMU or the external references to guidelines, are stored in the *FMU model information*. In SMOL, each FMU is handled as a special object, generated from its model description. The fields are names and typed after the input and output ports.

▶ **Example 23** (A Co-Simulation Scenario in SMOL and its Semantic Lifting)**.** Figure 18 is a simulation of a prey-predator system. It loads two FMUs, one each for prey and predators, which are stored in `Prey.fmu` and `Predator.fmu`. The type `FMO[in Double y, out Double x]` defines a functional mock-up object (FMO), which acts as a wrapper for the FMU and has two fields of type `Double`, one of which can only be written (`y`) and one only read ((`x`). This information is checked against the model information in the FMU file. The loop copies values between the simulators and calls the special `tick` method that advances simulation time by the provided parameter (here, one time unit).

The lifting of the configuration includes the lifting of the FMOs. Each such lifting contains the variables of the FMO (analogous to the lifting of fields of normal objects), their current values and the path of the loaded FMU.

6.2.3 Persistent State & Garbage Collection

Semantic reflection can retrieve objects that are no longer referenced in a program state, as long as they exist on the heap. This means that the result of a query to the semantically lifted program state may depend on the garbage collector; i.e., there is a race between the semantic lifting and runtime operations that are invisible to the programmer. For example, consider the following program:

SMOL
```
1 main
2     C c = new C();
3     c = null;
4     List<C> l = access("SELECT ?x WHERE { ?x a prog:C });
5     print(l); // non-null
6 end
```

A tracing garbage collector could remove the object created on Line 2 before the query is executed on Line 3, depending on the timing of the garbage collector. This renders the result of the query non-deterministic. We see two possible approaches to render semantic reflection deterministic in this context: disable garbage collection for objects reflected in the knowledge graph or force garbage collection before semantic lifting. For simplicity in SMOL, we opted for the former and did not implement an automatic garbage collector in the language interpreter so far. Instead, we have introduced a simple form of manual memory management – an object can be deallocated explicitly using a **destroy** statement. Note that the alternative, obtaining deterministic behavior of queries to the semantic layer by forcing a pass of the garbage collector before semantic lifting, can also be realized by restricting the results of a query to the reachable objects, thereby rendering the result of the query deterministic independent of when the garbage collector is applied.

The race between semantic lifting and garbage collections manifests itself if we consider objects only reachable through the knowledge graph as inaccessible. The underlying question is what we understand as inaccessible. In the example, the object is only accessible through the graph. One can reasonably argue that this can be considered inaccessible from the program's perspective (as it is a program object). One can equally argue that it is accessible, because the language provides a mechanism to access it through queries. Indeed, the first version of the geological case study relied on events being accessible through queries and did not store them in any variable.

SMOL
```
1 main
2     C c = new C();
3     destroy(c);
4     c = null
5     List<C> l = access("SELECT ?x WHERE { ?x a prog:C }); //empty
6 end
```

6.3 Applications

Semantic lifting, and its realization in SMOL, has been used and validated in several applications, which we describe in this section. Each of the applications is further detailed in the referenced publications. Throughout development, these applications have influenced the design of SMOL. As discussed above, the computational layer of semantic lifting (Section 6.2.1) was removed due to a lack of use and performance problems. Another removal was the lifting of the process stack. Originally, SMOL also lifted the full function stack, including local variables and process identifiers. However, this was removed to reduce the size of the lifted state and because it was rarely needed in applications.

6.3.1 Digital Twins

Semantic lifting has been used in case studies to enable a digital twin to self-adapt to changes in a twinned system. In this line of work, we exploited the ability to use graph queries on a knowledge graph that contains information about *both* the twinned system and the controlling digital twin

software. This has proven useful in several scenarios, and we give only a short description of the main idea here. For a comprehensive overview over the use of semantic lifting and reflection in digital twins, as well as more advanced patterns, we refer to [39, 47].

The first case study [45] considers a cyber-physical system consisting of a (simplified) building as the twinned system (the physical twin), and a SMOL program as the digital twin. The aim is to ensure that as rooms are added and removed from the building, the SMOL program automatically detects these changes and adapts the digital twin by removing or adding the objects that represent the rooms. Each SMOL object for a room contains an FMU that models its temperature.

To self-adapt, the semantically lifted SMOL program is compared against a so-called asset model, represented as a knowledge graph. The SMOL program that runs a *defect query* over the combined knowledge graph, where each such query expresses a relation between a lifted SMOL object and the part of the building that it models. If the defect query has a result, then it is either a SMOL object that must be removed (because the room it modeled was removed) or information about how to create a SMOL object to accommodate a new room.

This reflective digital twin architecture has been further applied and generalized to a greenhouse in the GreenhouseDT examplar [48]. Here, the physical twin is a greenhouse containing pumps and plants, where plants are regularly replaced or moved. In GreenhouseDT, SMOL is used as an orchestrator component for the digital twin: streams of sensor data and pump controllers are realized as external components, while the SMOL component acts as the semantically reflected system orchestrator. The reason for this decision is to separate data processing and numerical operations from operations on the semantic structure to reconfigure the digital twin components.

Reclassification and Dynamic Background Knowledge

The approach to semantic reflection as described here relies on the fact that the background knowledge is static, i.e., that the ontology provided to SMOL does not change during program execution. Only the lifted knowledge graph changes. In Digital Twins, and in general when the background knowledge contains information about a running/existing system and not just general knowledge about universals, the background knowledge can evolve as well.

Dynamic background knowledge is a problem in object-oriented programming, as this may cause that objects change their class after instantiation, which is known as *dynamic object reclassification* [17]. This is in particular important in the setting of self-adaptation, where objects need to be reclassified while retaining type safety. Sieve et al. [71] discuss a reclassification extension of SMOL, where an object changes its class, if the context it is linked to evolves. This declarative dynamic object reclassification has also been evaluated on the GreenhouseDT system and extended with type safety, but is so far not part of the main version of SMOL.

6.3.2 Simulation

This case study exploits semantic reflection in SMOL to facilitate the interpretation of states in a simulator, using domain terminology. The *geological simulator* of Qu et al. [70] uses the BFO-based GeoCore ontology [24]. The motivating example in Section 2 is a simplified version of this simulator. Using complex triggers, the SMOL program connects an advanced ontology with the simulation of geological process, without the need to extend the ontology – the new trigger concepts do not refer to the SMOL ontology. The application is able to reproduce results previously obtained manually by a team of geologists for the Ekofisk geological formation in under 10 minutes. As the so far biggest case study with SMOL, it also influences the design of the language the most. To enhance performance, the **hidden** keyword was introduced and the use of guards in domain linkage proved very useful to control the presence of kerogen depending on the object's state.

6.3.3 Semantic Lifting of JVM

In contrast to the SMOL-based applications above, the Java semantic debugger (sjdb) of Hauber [33] implements the semantic lifting of the Java Virtual Machine (JVM). It does not use SMOL, but implements semantic lifting to a mainstream language. Using such a language introduces additional challenges because the runtime state is not directly available or formalized. Thus, sjdb uses the debugging interface of JVM as the basis for semantic lifting – the lifting does not serialize the runtime state directly, but only the information exposed over this interface.

The sjdb tool consists of two parts. The jvm2owl library that is used for semantic lifting, and sjdb itself, which implements a debugger tool with breakpoints and a SPARQL interface to examine the state. Here, the breakpoints can also be semantic: execution is only halted if a certain query returns a non-empty set. The sjdb tool does not support semantic reflection, as it does not extend Java. For this reason, the programmer cannot control lifting without changing the code of jvm2owl and exclude certain parts of the language.

7 Related Work

Zhao et al. [78] have proposed translating programs, i.e., the static structure of types, variables, statements, etc., to knowledge graphs in order to simplify and integrate static analysis. Similarly, ontologies for the static structure of Java programs have been proposed by Kouneli et al. [51] and later Atzeni and Atzori [1]. Abstracting from concrete programming languages, de Aguiar et al. [15] describe the OOC-O ontology that aims to give an integrated view for multiple OO languages. The knowledge graphs produced by these approaches are similar to the static part of the SMOL translation, but runtime states are not considered.

The OPAL framework, proposed by Pattipati et al. [64], takes into account the runtime semantics by representing the control flow graph and static traces of C programs as triples. The mapping is based on a *static* analysis of the program and runtime states are still not represented. BOLD is an ontology-based log debugger for C programs developed by Pattipati et al. [65], building on OPAL. In BOLD, programs are instrumented in order to accumulate information about execution traces at runtime. Debugging then proceeds by querying this log information. In contrast to SMOL, only a part of the execution state is captured, and only at selected points in time, depending on the instrumentation. The gathered information cannot be accessed by the program, but it is available for debugging, similar to ideas outlined in our prior work [44]. On the other hand, the possibility to access information about a whole execution trace instead of only the current state is of course particularly useful for debugging, and this is not supported by SMOL in its current state. A recent extension of semantic lifting to traces [40] does not consider semantic state access, but focuses on runtime enforcement.

Connections between imperative programming languages and transition systems over knowledge graphs have been investigated in multiple lines of work, where the idea is to define languages that can operate directly on knowledge graphs through atomic actions. An early proposal is Golog [57], a language based on McCarthy's situation calculus [60], that uses first-order logic guards to examine and pick elements from its own state. Around the same time, Fagin et al. proposed to make explicit the distinction between what is the case in the world and what is known to a program, by means of epistemic (multi-)modal operators K_i, leading to the concept of *knowledge-based* programs [22]. This line of work is particularly interesting in multi-agent scenarios, where programs benefit not only from access to their own knowledge but also from reasoning about that of other programs. Zarrieß and Claßen [77] expand on this line of work by integrating description logic into a concurrent extension of Golog. They show how to verify CTL [12] properties with description logic assertions. In contrast to our work, knowledge is managed explicitly and programs do not reflect upon themselves.

A challenge for programs that can directly modify a knowledge graph is that an ABox may change in such a way that it violates the TBox, meaning that the system's state becomes inconsistent. Calvanese et al. [9] propose two operations `ASK` and `TELL` for transition systems defined *explicitly* over knowledge graphs. The `ASK` operator corresponds roughly to our **access**, while `TELL` performs a required action on the knowledge graph. This operation is based on the theory of knowledge base revision [49], in particular for DL-Lite knowledge bases [25, 26].

In contrast to this line of work, the transition system of `SMOL` is *implicit*, following the semantics of a fairly standard object-oriented language. The advantage of the `TELL` operator is clearly that state updates, like state access, closely match the semantic view. On the other hand, the advantage of our approach with `SMOL` is that well-established principles from programming languages carry over, avoiding to reinvestigate concepts such as modularity, runtime semantic structure and control flow for knowledge graphs. While all changes to the knowledge graph are global in the work of Calvanese et al. [9], global changes in `SMOL` only happen in the part of the knowledge graph inferred from user-provided axioms; the part inferred from the mapping only changes locally.

More generally, the effects of combining rule systems with description logics, and how to accommodate the differences in semantics, have been the object of much study. In particular, the work of Eiter et al. [21] concentrates on using rule systems as a programming language; technically, answer set programming is enhanced with access to knowledge graphs. The rules are similar to what is commonly used in non-monotonic logic programs, but rule bodies may also contain queries to the knowledge graph, possibly under default negation. In contrast to this line of work on rules and description logics, our work with `SMOL` has concentrated on the semantic lifting of a more 'mainstream' object-oriented language.

In contrast to the previously discussed work, we have not targeted a language that operates directly on description logic interpretations or knowledge graphs. Instead, we aim is to enhance a language similar to mainstream programming languages by semantic technologies.

Closest to our approach in this respect is the work on *ontology-mediated* programming of Dubslaff, Koopmann and Turhan [18, 19]. Instead of operating on a knowledge graph, they define the concept of an 'interface' between the program and the knowledge graph. Technically, the interface defines explicit mappings from program states to the description logic, and vice versa, where the interaction happens through a number of designated variables and 'hooks.' As underlying programming language, the authors use a stochastic guarded command language similar to PRISM [52], such that it is possible to perform probabilistic model checking on the 'ontologized programs.' In contrast to our work, the programming language provides neither semantic reflection nor typing. However, it is interesting to compare the interface mechanism itself to our work in more detail.

`SMOL` uses an implicit 'direct' mapping for semantic lifting, and does not require that the correspondence between program and knowledge graph is described in two directions. From the point of view of the program, the variables $Var_{\mathbf{O}}$ and hooks $H_{\mathbf{O}}$ of an interface can be compared to the variables used in **access** queries. The variables $Var_{\mathbf{O}}$ are those variables whose values are lifted into the knowledge graph, while $H_{\mathbf{O}}$ are concepts in the ontologies that are explicitly used inside the program. From the point of view of the knowledge graph, `SMOL` reflects the complete program state (including all variables) into the knowledge graph, but our implementation of virtualization ensures that only those parts needed for semantic state access are actually generated. We believe that the semantic lifting of `SMOL` subsumes the language concepts of ontology-mediated programming in terms of expressivity.

Ontologies have also been explored in the context of type systems for programming languages. Leinberger et al. [54] study DL concept expressions as static types in a λ-calculus, such that terms can be type-checked using SHACL constraints [56]. Existing programming languages can

be integrated with RDF data using the type systems of Paar and Vrandecic [62] and Leinberger et al. [55]. It is interesting to observe that the difference between ontologies and regular types is not just about taste: (a) concepts allow more expressive structure than type hierarchies and (b) classes in programming languages are designed by the user to fit the needs of its application, while the concepts of the domain are designed to accommodate the needs of a general domain. The connection to types has also been investigated through mappings [38] and code generation [72]. While this line of work attempts to unify two tools made for different tasks, our approach with `SMOL` is to propose a sensible interface.

Thimmaiah et al. [73] implement a simple (compared to sjdb) form of semantic lifting of Featherweight Java into Neo4j, but focus on efficiency of the subsequent data access instead of modeling or the connection between behavioral and ontological modeling. It provides no form of static analysis or type checking. To the best of our knowledge, their approach has not been applied to projects beyond performance evaluations.

8 Conclusion

Semantic reflection opens new perspectives on how semantic technologies and programming languages can be combined. Our work with `SMOL` introduces a clear separation of concerns between *computations* (in the imperative programming language) and *domain description* (in the ontology), and provides a clear interface between these concerns (queries and domain linkage). This way, we are able to reuse standard technologies and, thus, reduce the need to learn a new formalism for users. Furthermore, we provide basic tool and analysis support for this interface through a type system.

We believe that this research direction, at the intersection of semantic technologies and programming languages, opens up interesting possibilities for integrating domain knowledge and behavioral modeling. Complementing the foundational study presented here, we have also described first applications and case studies. Together, this work demonstrates the versatility and robustness of semantical reflection.

Beyond the technical level of knowledge graphs and imperative programs, semantic lifting also provides an opportunity to study the relation between knowledge evolution [69] and software evolution. Other kinds of interfaces between data and software artifacts, such as object-relational mappings, are known to be challenging to maintain [10, 11], but the interface between graph data and software has received less attention so far.

A possible direction for future work at the foundational level, is the extension of the type checking towards more dynamic queries, e.g., queries assembled at runtime using string operations, investigate the connection between semantic lifting and knowledge graph construction pipelines, and investigate semantic reflection for non-imperative programming languages, e.g., functional languages.

References

1 Mattia Atzeni and Maurizio Atzori. Codeontology: Rdf-ization of source code. In *Proc. International Semantic Web Conference (ISWC 2017)*, volume 10588 of *Lecture Notes in Computer Science*, pages 20–28. Springer, 2017. `doi:10.1007/978-3-319-68204-4_2`.

2 Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: An overview. In *Proc. International Workshop on Construction and Analysis of Safe, Secure, and Interoperable Smart Devices (CASSIS 2004)*, volume 3362 of *Lecture Notes in Computer Science*, pages 49–69. Springer, 2004. `doi:10.1007/978-3-540-30569-9_3`.

3 Selena Baset and Kilian Stoffel. Object-oriented modeling with ontologies around: A survey of existing approaches. *Int. J. Softw. Eng. Knowl. Eng.*, 28(11-12):1775–1794, 2018. `doi:10.1142/S0218194018400284`.

4 Knut Bjørlykke. Source rocks and petroleum geochemistry. *Petroleum Geoscience*, pages 339–348, 2010.

5 Torsten Blochwitz, Martin Otter, Johan Åkesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, Hans Olsson, and Antoine Viel. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Modelica Conference*, pages 173–184. The Modelica Association, 2012. doi:10.3384/ecp12076173.

6 Bart Bogaerts, Maxime Jakubowski, and Jan Van den Bussche. SHACL: A description logic in disguise. In *LPNMR*, volume 13416 of *Lecture Notes in Computer Science*, pages 75–88. Springer, 2022. doi:10.1007/978-3-031-15707-3_7.

7 Bart Bogaerts, Maxime Jakubowski, and Jan Van den Bussche. Expressiveness of SHACL features and extensions for full equality and disjointness tests. *Log. Methods Comput. Sci.*, 20(1), 2024. doi:10.46298/LMCS-20(1:16)2024.

8 Gilad Bracha and David M. Ungar. Mirrors: design principles for meta-level facilities of object-oriented programming languages. In John M. Vlissides and Douglas C. Schmidt, editors, *Proc. 19th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2004)*, pages 331–344. ACM, 2004. doi:10.1145/1028976.1029004.

9 Diego Calvanese, Giuseppe De Giacomo, et al. Actions and programs over description logic knowledge bases: A functional approach. In *Knowing, Reasoning, and Acting: Essays in Honour of Hector J. Levesque*. College Press, 2011.

10 Tse-Hsun Chen, Weiyi Shang, Zhen Ming Jiang, Ahmed E. Hassan, Mohamed N. Nasser, and Parminder Flora. Detecting performance anti-patterns for applications developed using object-relational mapping. In *ICSE*, pages 1001–1012. ACM, 2014. doi:10.1145/2568225.2568259.

11 Tse-Hsun Chen, Weiyi Shang, Jinqiu Yang, Ahmed E. Hassan, Michael W. Godfrey, Mohamed N. Nasser, and Parminder Flora. An empirical study on the practice of maintaining object-relational mapping code in java systems. In *MSR*, pages 165–176. ACM, 2016. doi:10.1145/2901739.2901758.

12 Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981. doi:10.1007/BFb0025774.

13 George P. Copeland and David Maier. Making smalltalk a database system. In *SIGMOD Conference*, pages 316–325. ACM Press, 1984. doi:10.1145/602259.602300.

14 Ole-Johan Dahl. The birth of object orientation: the Simula languages. In *Essays in Memory of Ole-Johan Dahl*, volume 2635 of *Lecture Notes in Computer Science*, pages 15–25. Springer, 2004. doi:10.1007/978-3-540-39993-3_3.

15 Camila Zacché de Aguiar, Ricardo de Almeida Falbo, and Vítor E. Silva Souza. OOCO: A reference ontology on object-oriented code. In *Conceptual Modeling (ER 2019)*, volume 11788 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2019. doi:10.1007/978-3-030-33223-5_3.

16 Crystal Chang Din, Leif Harald Karlsen, Irina Pene, Oliver Stahl, Ingrid Chieh Yu, and Thomas Østerlie. Geological multi-scenario reasoning. In *Norsk Informatikkonferanse (NIK 2019)*. Bibsys Open Journal Systems, Norway, 2019. URL: https://ojs.bibsys.no/index.php/NIK/article/view/640.

17 Sophia Drossopoulou, Ferruccio Damiani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Fickle : Dynamic object re-classification. In *Proc. 15th European Conference, on Object-Oriented Programming (ECOOP 2001)*, volume 2072 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2001. doi:10.1007/3-540-45337-7_8.

18 Clemens Dubslaff, Patrick Koopmann, and Anni-Yasmin Turhan. Ontology-mediated probabilistic model checking. In *Integrated Formal Methods (IFM 2019)*, volume 11918 of *Lecture Notes in Computer Science*, pages 194–211. Springer, 2019. doi:10.1007/978-3-030-34968-4_11.

19 Clemens Dubslaff, Patrick Koopmann, and Anni-Yasmin Turhan. Give inconsistency a chance: Semantics for ontology-mediated verification. In *Proc. Workshop on Description Logics (DL 2020)*, volume 2663 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020. URL: https://ceur-ws.org/Vol-2663/paper-9.pdf.

20 Clemens Dubslaff, Patrick Koopmann, and Anni-Yasmin Turhan. Enhancing probabilistic model checking with ontologies. *Formal Aspects Comput.*, 33(6):885–921, 2021. doi:10.1007/S00165-021-00549-0.

21 Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. *Artif. Intell.*, 172(12-13):1495–1539, 2008. doi:10.1016/J.ARTINT.2008.04.002.

22 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. Knowledge-based programs. *Distributed Comput.*, 10(4):199–225, 1997. doi:10.1007/S004460050038.

23 Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, 1992. doi:10.1016/0304-3975(92)90014-7.

24 Luan Fonseca Garcia, Mara Abel, Michel Perrin, and Renata dos Santos Alvarenga. The geocore ontology: A core ontology for general use in geology. *Comput. Geosci.*, 135:104387, 2020. doi:10.1016/J.CAGEO.2019.104387.

25 Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. On the update of description logic ontologies at the instance level. In *Proc. Conference on Artificial Intelligence (AAAI 2006)*, pages 1271–1276. AAAI Press, 2006. URL: http://www.aaai.org/Library/AAAI/2006/aaai06-199.php.

26 Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. On the approximation of instance level update and erasure in description logics. In *Proc. Conference on Artificial Intelligence (AAAI 2007)*, pages 403–408.

AAAI Press, 2007. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-063.php.

27 Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: An OWL 2 reasoner. *J. Autom. Reason.*, 53(3):245–269, 2014. doi:10.1007/S10817-014-9305-1.

28 Christine Golbreich and Evan Wallace. OWL 2 web ontology language new features and rationale (second edition). W3C recommendation, W3C, December 2012. https://www.w3.org/TR/2012/REC-owl2-new-features-20121211/.

29 Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: A survey. *ACM Comput. Surv.*, 51(3):49:1–49:33, 2018. doi:10.1145/3179993.

30 Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *J. Web Semant.*, 6(4):309–322, 2008. doi:10.1016/j.websem.2008.05.001.

31 Armin Haller, Krzysztof Janowicz, Simon Cox, Danh Le Phuoc, Kerry Taylor, and Maxime Lefrançois. Semantic sensor network ontology. W3C recommendation, W3C, 2017. URL: https://www.w3.org/TR/2017/REC-vocab-ssn-20171019/.

32 Andreas Harth, Tobias Käfer, Anisa Rula, Jean-Paul Calbimonte, Eduard Kamburjan, and Martin Giese. Towards representing processes and reasoning with process descriptions on the web. *TGDK*, 2(1):1:1–1:32, 2024. doi:10.4230/TGDK.2.1.1.

33 Anton W. Haubner. Inspecting Java program states with semantic web technologies. Master's thesis, Technische Universität Darmstadt, Darmstadt, 2022. The software developed as part of this thesis is available on GitHub. The Semantic Java Debugger: https://github.com/ahbnr/SemanticJavaDebugger The jdi2owl library: https://github.com/ahbnr/jdi2owl. doi:10.26083/tuprints-00022143.

34 Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. Chapman and Hall/CRC Press, 2010. doi:10.1201/9781420090512.

35 Matthew Horridge and Sean Bechhofer. The OWL API: A java API for working with OWL 2 ontologies. In *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. URL: https://ceur-ws.org/Vol-529/owled2009_submission_29.pdf.

36 Matthew Horridge and Sean Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011. doi:10.3233/SW-2011-0025.

37 Einar Broch Johnsen, Reiner Hähnle, Jan Schäfer, Rudolf Schlatte, and Martin Steffen. ABS: A core language for abstract behavioral specification. In *Formal Methods for Components and Objects (FMCO 2010)*, volume 6957 of *Lecture Notes in Computer Science*, pages 142–164. Springer, 2010. doi:10.1007/978-3-642-25271-6_8.

38 Aditya Kalyanpur, Daniel Jiménez Pastor, Steve Battle, and Julian A. Padget. Automatic mapping of OWL ontologies into Java. In *Proc. International Conference on Software Engineering & Knowledge Engineering (SEKE 2004)*, pages 98–103, 2004.

39 Eduard Kamburjan, Nelly Bencomo, Silvia Lizeth Tapia Tarifa, and Einar Broch Johnsen. Declarative lifecycle management in digital twins. In *Proc. International Conference on Engineering Digital Twins (EDTConf 2024)*. ACM, 2024. doi:10.1145/3652620.3688248.

40 Eduard Kamburjan and Crystal Chang Din. Runtime enforcement using knowledge bases. In *Proc. International Conference on Fundamental Approaches to Software Engineering (FASE 2023)*, volume 13991 of *Lecture Notes in Computer Science*, pages 220–240. Springer, 2023. doi:10.1007/978-3-031-30826-0_12.

41 Eduard Kamburjan and Sandro Rama Fiorini. On the notion of naturalness in formal modeling. In *Festschrift Reiner Hähnle*, volume 13360 of *Lecture Notes in Computer Science*, pages 264–289. Springer, 2022. doi:10.1007/978-3-031-08166-8_13.

42 Eduard Kamburjan and Einar Broch Johnsen. Knowledge structures over simulation units. In *Proc. Annual Modeling and Simulation Conference (ANNSIM 2022)*, pages 78–89. IEEE, 2022. doi:10.23919/ANNSIM55834.2022.9859490.

43 Eduard Kamburjan, Vidar Norstein Klungre, and Martin Giese. Never mind the semantic gap: Modular, lazy and safe loading of RDF data. In *Proc. Extended Semantic Web Conference (ESWC 2022)*, volume 13261 of *Lecture Notes in Computer Science*, pages 200–216. Springer, 2022. doi:10.1007/978-3-031-06981-9_12.

44 Eduard Kamburjan, Vidar Norstein Klungre, Rudolf Schlatte, Einar Broch Johnsen, and Martin Giese. Programming and debugging with semantically lifted states. In *Proc. Extended Semantic Web Conference (ESWC 2021)*, volume 12731 of *Lecture Notes in Computer Science*, pages 126–142. Springer, 2021. doi:10.1007/978-3-030-77385-4_8.

45 Eduard Kamburjan, Vidar Norstein Klungre, Rudolf Schlatte, S. Lizeth Tarifa Tapia, David Cameron, and Einar Broch Johnsen. Digital twin reconguration using asset models. In *Proc. International Conference on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2022)*, volume 13704 of *Lecture Notes in Computer Science*, pages 71–88. Springer, 2022. doi:10.1007/978-3-031-19762-8_6.

46 Eduard Kamburjan and Egor V. Kostylev. Type checking semantically lifted programs via query containment under entailment regimes. In *Proc. 34th Intl. Workshop on Description Logics (DL 2021)*, volume 2954 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021. URL: https://ceur-ws.org/Vol-2954/paper-19.pdf.

47 Eduard Kamburjan, Andrea Pferscher, Rudolf Schlatte, Riccardo Sieve, Silvia Lizeth Tapia Tarifa, and Einar Broch Johnsen. Semantic reflection and digital twins: A comprehensive overview. In Mike Hinchey and Bernhard Steffen, editors, *The Combined Power of Research, Education, and Dissemination - Essays Dedicated to Tiziana Margaria on the Occasion of Her 60th Birthday*, volume 15240 of *Lecture Notes in Com-*

puter Science, pages 129–145. Springer, 2025. doi:10.1007/978-3-031-73887-6_11.

48 Eduard Kamburjan, Riccardo Sieve, Chinmayi Prabhu Baramashetru, Marco Amato, Gianluca Barmina, Eduard Occhipinti, and Einar Broch Johnsen. GreenhouseDT: An exemplar for digital twins. In *Proc. International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2024)*, pages 175–181. ACM, 2024. doi:10.1145/3643915.3644108.

49 Hirofumi Katsuno and Alberto O. Mendelzon. On the difference between updating a knowledge base and revising it. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning (KR 1991)*, pages 387–394. Morgan Kaufmann, 1991.

50 Gregor Kiczales, Jim des Rivieres, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.

51 Aggeliki Kouneli, Georgia D. Solomou, Christos Pierrakeas, and Achilles Kameas. Modeling the knowledge domain of the Java programming language as an ontology. In *Advances in Web-Based Learning (ICWL 2012)*, volume 7558 of *Lecture Notes in Computer Science*, pages 152–159. Springer, 2012. doi:10.1007/978-3-642-33642-3_16.

52 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification (CAV 2011)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011. doi:10.1007/978-3-642-22110-1_47.

53 Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies. *Artif. Intell. Medicine*, 80:11–28, 2017. doi:10.1016/J.ARTMED.2017.07.002.

54 Martin Leinberger, Ralf Lämmel, and Steffen Staab. The essence of functional programming on semantic data. In *Proc. European Symposium on Programming (ESOP 2017)*, volume 10201 of *Lecture Notes in Computer Science*, pages 750–776. Springer, 2017. doi:10.1007/978-3-662-54434-1_28.

55 Martin Leinberger, Stefan Scheglmann, Ralf Lämmel, Steffen Staab, Matthias Thimm, and Evelyne Viegas. Semantic web application development with LITEQ. In *Proc. International Semantic Web Conference (ISWC 2014)*, volume 8797 of *Lecture Notes in Computer Science*, pages 212–227. Springer, 2014. doi:10.1007/978-3-319-11915-1_14.

56 Martin Leinberger, Philipp Seifer, Claudia Schon, Ralf Lämmel, and Steffen Staab. Type checking program code using SHACL. In *Proc. International Semantic Web Conference (ISWC 2019)*, volume 11778 of *Lecture Notes in Computer Science*, pages 399–417. Springer, 2019. doi:10.1007/978-3-030-30793-6_23.

57 Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *J. Log. Program.*, 31(1-3):59–83, 1997. doi:10.1016/S0743-1066(96)00121-5.

58 Barbara Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994. doi:10.1145/197320.197383.

59 Carsten Lutz, Dirk Walther, and Frank Wolter. Conservative extensions in expressive description logics. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 453–458. IJCAI, 2007. URL: http://ijcai.org/Proceedings/07/Papers/071.pdf.

60 John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, pages 463–502. Edinburgh University Press, 1969.

61 Manuel Nathenson and Marianne Guffanti. Geothermal gradients in the conterminous united states. *Journal of Geophysical Research: Solid Earth*, 93:6437–6450, 1988. doi:10.1029/JB093iB06p06437.

62 Alexander Paar and Denny Vrandecic. Zhi# - OWL aware compilation. In *Proc. Extended Semantic Web Conference (ESWC 2011)*, volume 6644 of *Lecture Notes in Computer Science*, pages 315–329. Springer, 2011. doi:10.1007/978-3-642-21064-8_22.

63 Terence Parr. *The definitive ANTLR 4 reference*. The Pragmatic Bookshelf, 2013.

64 Dileep Kumar Pattipati, Rupesh Nasre, and Sreenivasa Kumar Puligundla. OPAL: an extensible framework for ontology-based program analysis. *Softw. Pract. Exp.*, 50(8):1425–1462, 2020. doi:10.1002/spe.2821.

65 Dileep Kumar Pattipati, Rupesh Nasre, and Sreenivasa Kumar Puligundla. BOLD: an ontology-based log debugger for C programs. *Autom. Softw. Eng.*, 29(1):2, 2022. doi:10.1007/s10515-021-00308-8.

66 Kenneth E. Peters and Mary Rose Cassa. Applied source rock geochemistry. In *The Petroleum System —– From Source to Trap*. American Association of Petroleum Geologists, January 1994. doi:10.1306/M60585C5.

67 Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

68 Gordon Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61, 2004.

69 Axel Polleres, Romana Pernisch, Angela Bonifati, Daniele Dell'Aglio, Daniil Dobriy, Stefania Dumbrava, Lorena Etcheverry, Nicolas Ferranti, Katja Hose, Ernesto Jiménez-Ruiz, Matteo Lissandrini, Ansgar Scherp, Riccardo Tommasini, and Johannes Wachs. How does knowledge evolve in open knowledge graphs? *TGDK*, 1(1):11:1–11:59, 2023. doi:10.4230/TGDK.1.1.11.

70 Yuanwei Qu, Eduard Kamburjan, Anita Torabi, and Martin Giese. Semantically triggered qualitative simulation of a geological process. *Applied Computing and Geosciences*, 21, 2024. doi:10.1016/j.acags.2023.100152.

71 Riccardo Sieve, Eduard Kamburjan, Ferruccio Damiani, and Einar Broch Johnsen. Declarative dynamic object reclassification. In Jonathan Aldrich and Alexandra Silva, editors, *Proc. European Conference on Object-Oriented Programming (ECOOP 2025)*, volume 333 of *LIPIcs*, pages

29:1–29:31. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ECOOP.2025.29.

72 Graeme Stevenson and Simon Dobson. Sapphire: Generating Java runtime artefacts from OWL ontologies. In *Advanced Information Systems Engineering Workshops (CAiSE 2011)*, volume 83 of *Lecture Notes in Business Information Processing*, pages 425–436. Springer, 2011. doi:10.1007/978-3-642-22056-2_46.

73 Aditya Thimmaiah, Leonidas Lampropoulos, Christopher J. Rossbach, and Milos Gligoric. Object graph programming. In *ICSE*, pages 20:1–20:13. ACM, 2024. doi:10.1145/3597503.3623319.

74 Ingrid Chieh Yu, Irina Pene, Crystal Chang Din, Leif Harald Karlsen, Chi Mai Nguyen, Oliver Stahl, and Adnan Latif. Subsurface evaluation through multi-scenario reasoning. In Daniel Patel, editor, *Interactive Data Processing and 3D Visualization of the Solid Earth*, pages 325–355. Springer, 2021. doi:10.1007/978-3-030-90716-7_10.

75 Veruska Zamborlini and Giancarlo Guizzardi. On the representation of temporally changing information in OWL. In *Workshops Proceedings of the International Enterprise Distributed Object Computing Conference (EDOCW 2010)*, pages 283–292. IEEE Computer Society, 2010. doi:10.1109/EDOCW.2010.50.

76 Veruska Carretta Zamborlini and Giancarlo Guizzardi. An ontologically-founded reification approach for representing temporally changing information in owl. In *Logical Formalizations of Commonsense Reasoning (COMMONSENSE 2013)*, 2013. URL: http://www.commonsense2013.cs.ucy.ac.cy/docs/commonsense2013_submission_23.pdf.

77 Benjamin Zarrieß and Jens Claßen. Verification of knowledge-based programs over description logic actions. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 3278–3284. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/462.

78 Yue Zhao, Guoyang Chen, Chunhua Liao, and Xipeng Shen. Towards ontology-based program analysis. In *European Conference on Object-Oriented Programming, ECOOP 2016*, volume 56 of *LIPIcs*, pages 26:1–26:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ECOOP.2016.26.

A Full Runtime Semantics

The full syntax of SMOL is given by the grammar in Figure 19, including the primitives for semantic reflection. The runtime configurations (cf. Definition 4) are defined by the grammar

$$\begin{array}{llll} \mathsf{conf} ::= \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs} & \mathsf{rs} & ::= \mathsf{Stmt} \mid \mathsf{Loc} \leftarrow \texttt{stack;}\ \mathsf{Stmt} & \mathsf{Cl} ::= \texttt{C} \mid \texttt{List<C>} \\ \mathsf{obs} ::= \overline{(\mathsf{Cl}, \rho)_{\mathtt{X}}} & \mathsf{prs} & ::= \overline{(\mathtt{m}, \mathtt{X}, \mathsf{rs}, \sigma)}. & \end{array}$$

Here, σ ranges over local stores, i.e., maps from variables to DEs, ρ over object stores, i.e., maps from fields to DEs, CT over class tables, and X over object identifiers. The remaining terms are defined in Definition 1.

In the following, we present a Structural Operation Semantics (SOS) [68] for SMOL as a set of rules that defines transitions between runtime configurations. These rules formally define the relation $\rightarrow_{\mathsf{er}}^{\mathcal{K}}$ from Section 5.1 and have Definitions 17–19 as special cases in rules **(validate)**, **(member)**, and **(access)**. Expressions Expr are evaluated using an evaluation function $[\![\mathsf{Expr}]\!]_{\mathtt{X}}^{\sigma,\mathsf{obs}}$ with respect to an object identifier X to resolve the expression **this**, a local store σ to resolve local variables and a set of objects obs to resolve field accesses. To simplify the presentation, we will also allow object identifiers as the right-hand-sides of assignments; object identifiers simply evaluate to ehemselves.

We say that an object identifier is *fresh* if it does not appear in a runtime configuration. We group the rules into three parts: rules with global effect, rules for semantic reflection, and rules with local effect.

Rules with global effect. The rules in Figure 20 have a global effect; they either create new objects or manipulate the call stack.

- Rule **(new)** allocates a new object in the runtime configuration, initializing it and reduces to an assignment – thus, we do not need several rules for all forms of locations. The rule's first premise assigns to each field in the object memory ρ the evaluation of the corresponding parameter. The second premise creates a fresh object identifier X. The third premise ensures that the number of parameters is the same as the number of fields.

Prog	::= $\overline{\text{Class}}$ **main** Stmt **end**	Programs
Class	::= **class** C [**extends** C] ($\overline{\text{Field}}$) [Linkage] $\overline{\text{Met}}$ **end**	Classes
Type	::= `t` | `C` | `List<C>`	Types
Field	::= [**hidden** | **domain**] Type `f`	Fields
Linkage	::= $\overline{\textbf{links}(\text{Expr})\ \texttt{le};}$ **links** `le;`	Domain linkage
Met	::= Type `m`($\overline{\text{Type v}}$) Stmt **end**	Methods
Stmt	::= Loc = RHS; | **if** Expr **then** Stmt **else** Stmt **end**	Statements
	| Expr.`m`($\overline{\text{Expr}}$); | **skip**; | **while** Expr **do** Stmt **end**	
	| Type `v` = RHS; | Stmt Stmt | **return** Expr;	
RHS	::= **new** `C`($\overline{\text{Expr}}$) [Linkage] | Expr.`m`($\overline{\text{Expr}}$) | Expr	
	| **access**(`sparql`, $\overline{\texttt{Expr}}$) | **member**(`owl`) | **validate**(`shacl`)	RHS expressions
Expr	::= **this** | **null** | Loc | `a` | Expr *op* Expr | Expr == Expr | Expr != Expr	Expressions
Loc	::= Expr.`f` | `v`	Locations

Figure 19 Full Syntax of `SMOL`.

- Rule **(call)** deals with method calls. The rule is analogous to rule **(new)**, but modifies the stack instead of the runtime configuration. The rule's premises evaluate the target expression Expr to an object identifier, retrieves the class of this object from the runtime configuration, checks that the number of arguments is correct by a lookup in the class table (using vars). The rule creates an initial store σ' by evaluating the parameters, and adds a new process to the configuration. The old process has its active statement replaced by the waiting statement Loc ← `stack;` that records where the return value will be stored once the called method terminate.
- Rule **(return)** deals with **return** statements and removes one process from the stack, but also modifies the calling process by replacing its waiting statement Loc ← `stack;` by an assignment of the return value to the stored location Loc.

Rules for semantic reflection. The rules in Figure 21 correspond directly to Definitions 17–19. There are three analogous rules in the case that the target location is a declared variable.

Local effect without lifting. Finally, the rules in Figure 22 are standard programming constructs.

- Rules **(iftrue)** and **(iffalse)** reduce a branching statement to the first or second branch, depending on the evaluation of the guarding expression.
- The rule **(loop1)** unrolls the loop body once if the loop guard evaluates to true, while **(loop2)** removes the loop from the active statement and continues with the next statement.
- The rule **(assign1)** deals with assignment of side effect free expressions (and object identifiers) to fields. It evaluates Expr to an object identifier `Y` and updates its memory. Rules **(assign2)** and **(assign3)** update the local memory of the stack frame for new declared and
- Finally, **(skip)** merely removes a **skip** statement without further effect and **(callIn)** introduces a fresh variable to handle method calls without a target location.

$$\textbf{(new)}\ \frac{\bigwedge_{1\le i\le n}\rho(\mathtt{f}_i)=[\![\mathsf{Expr}_i]\!]^{\sigma,\mathsf{obs}}_{\mathtt{Y}} \qquad \mathtt{X}\ \text{fresh} \qquad |\mathsf{fields}_{\mathsf{CT}}(\mathtt{C})|=n}{\begin{array}{l}\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},\ (\mathtt{m},\mathtt{Y},\mathsf{Loc}=\textbf{new}\ \mathtt{C}(\mathsf{Expr}_1,\ldots,\mathsf{Expr}_n);\mathsf{Stmt},\sigma)\ \ \to^{\mathcal{K}}_{\mathsf{er}}\\ \qquad\qquad \mathsf{CT}\ \mathsf{obs}\ (\mathtt{C},\rho)_{\mathtt{X}}\ \mathsf{prs},\ (\mathtt{m},\mathtt{Y},\mathsf{Loc}=\mathtt{X};\mathsf{Stmt},\sigma)\end{array}}$$

$$\textbf{(call)}\ \frac{\begin{array}{c}\bigwedge_{1\le i\le n}\sigma'(\mathtt{v}_i)=[\![\mathsf{Expr}_i]\!]^{\sigma,\mathsf{obs}}_{\mathtt{Y}} \qquad |\mathsf{vars}(\mathsf{CT},\mathtt{C},\mathtt{m2})|=n\\ [\![\mathsf{Expr}]\!]^{\sigma,\mathsf{obs}}_{\mathtt{Y}}=\mathtt{X} \qquad (\mathtt{C},\rho)_{\mathtt{X}}\in\mathtt{obs} \qquad \mathsf{Stmt}'=\mathsf{body}(\mathsf{CT},\mathtt{C},\mathtt{m})\end{array}}{\begin{array}{l}\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},\ (\mathtt{m},\mathtt{Y},\mathsf{Loc}=\mathsf{Expr}.\mathtt{m2}(\mathsf{Expr}_1\ldots\mathsf{Expr}_n);\mathsf{Stmt},\sigma)\\ \to^{\mathcal{K}}_{\mathsf{er}}\ \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},\ (\mathtt{m},\mathtt{Y},\mathsf{Loc}\leftarrow\mathtt{stack};\mathsf{Stmt},\sigma),\ (\mathtt{m2},\mathtt{X},\mathsf{Stmt}',\sigma')\end{array}}$$

$$\textbf{(return)}\ \frac{[\![\mathsf{Expr}]\!]^{\sigma',\mathsf{obs}}_{\mathtt{Y}}=\mathsf{e}}{\begin{array}{l}\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},\ (\mathtt{m},\ \mathtt{Y},\ \mathsf{Loc}\leftarrow\mathtt{stack};\ \mathsf{Stmt},\sigma),\ (\mathtt{n},\mathtt{X},\textbf{return}\ \mathsf{Expr},\sigma')\\ \to^{\mathcal{K}}_{\mathsf{er}}\ \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},\ (\mathtt{m},\ \mathtt{Y},\ \mathsf{Loc}=\mathsf{e};\ \mathsf{Stmt},\sigma)\end{array}}$$

Figure 20 Rules with global effect: Object creation, method call, method return.

$$\textbf{(validate)}\ \frac{\begin{array}{c}\mathsf{res}=\mathsf{Sha}\big(\mathcal{K}_{\mathtt{SMOL}}\cup\mathcal{K}\cup\mathcal{K}_{\mathsf{conf}},\mathtt{shacl}\big)\\ \mathcal{K}_{\mathsf{conf}}=\mu\big(\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},(\mathtt{m},\ \mathtt{X},\ \mathsf{Loc}=\textbf{validate}(\mathtt{shacl});\ \mathsf{Stmt},\sigma)\big)\end{array}}{\begin{array}{l}\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},(\mathtt{m},\ \mathtt{X},\ \mathsf{Loc}=\textbf{validate}(\mathtt{shacl});\ \mathsf{Stmt},\sigma)\\ \to^{\mathcal{K}}_{\mathsf{er}}\ \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},(\mathtt{m},\ \mathtt{X},\ \mathsf{Loc}=\mathsf{res};\mathsf{Stmt},\sigma)\end{array}}$$

$$\textbf{(member)}\ \frac{\begin{array}{c}\mathsf{obs}_{\mathtt{Y}}=\mathsf{listify}\Big(\mathsf{Mem}\big(\mathcal{K}_{\mathtt{SMOL}}\cup\mathcal{K}\cup\mathcal{K}_{\mathsf{conf}},\mathtt{owl}\big)\Big)\\ \mathcal{K}_{\mathsf{conf}}=\mu\big(\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},(\mathtt{m},\ \mathtt{X},\ \mathsf{Loc}=\textbf{member}(\mathtt{owl});\ \mathsf{Stmt},\sigma)\big)\end{array}}{\begin{array}{l}\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},(\mathtt{m},\ \mathtt{X},\ \mathsf{Loc}=\textbf{member}(\mathtt{owl});\ \mathsf{Stmt},\sigma)\\ \to^{\mathcal{K}}_{\mathsf{er}}\ \mathsf{CT}\ \mathsf{obs}\ \mathsf{obs}_{\mathtt{Y}}\ \mathsf{prs},(\mathtt{m},\ \mathtt{X},\ \mathsf{Loc}=\mathtt{Y};\ \mathsf{Stmt},\sigma)\end{array}}$$

$$\textbf{(access)}\ \frac{\begin{array}{c}\mathsf{obs}_{\mathtt{Y}}=\mathsf{listify}(\mathit{des})\\ \mathit{des}=\mathsf{Ans}_{\mathsf{er}}\big(\mathcal{K}_{\mathtt{SMOL}}\cup\mathcal{K}\cup\mathcal{K}_{\mathsf{conf}},\mathtt{sparql}[[\![\mathsf{Expr}_1]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}}\ldots[\![\mathsf{Expr}_n]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}}]\big)\\ \mathcal{K}_{\mathsf{conf}}=\mu\big(\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},(\mathtt{m},\ \mathtt{X},\ \mathsf{Loc}=\textbf{access}(\mathtt{sparql},\mathsf{Expr}_1,\ldots,\mathsf{Expr}_n);\ \mathsf{Stmt},\sigma)\big)\end{array}}{\begin{array}{l}\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs},(\mathtt{m},\ \mathtt{X},\ \mathsf{Loc}=\textbf{access}(\mathtt{sparql},\mathsf{Expr}_1,\ldots,\mathsf{Expr}_n);\ \mathsf{Stmt},\sigma)\\ \to^{\mathcal{K}}_{\mathsf{er}}\ \mathsf{CT}\ \mathsf{obs}\ \mathsf{obs}_{\mathtt{Y}}\ \mathsf{prs},(\mathtt{m},\ \mathtt{X},\ \mathsf{Loc}=\mathtt{Y};\ \mathsf{Stmt},\sigma)\end{array}}$$

Figure 21 Rules for semantic reflection.

$$\textbf{(iftrue)}\ \frac{[\![\mathsf{Expr}]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}} = \mathtt{true}}{\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \textbf{if}\ \mathsf{Expr}\ \textbf{then}\ \mathsf{Stmt}_1\ \textbf{else}\ \mathsf{Stmt}_2\ \textbf{end}\ \mathsf{Stmt}, \sigma) \to^{\mathcal{K}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Stmt}_1\ \mathsf{Stmt}, \sigma)}$$

$$\textbf{(iffalse)}\ \frac{[\![\mathsf{Expr}]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}} = \mathtt{false}}{\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \textbf{if}\ \mathsf{Expr}\ \textbf{then}\ \mathsf{Stmt}_1\ \textbf{else}\ \mathsf{Stmt}_2\ \textbf{end}\ \mathsf{Stmt}, \sigma) \to^{\mathcal{K}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Stmt}_2\ \mathsf{Stmt}, \sigma)}$$

$$\textbf{(loop1)}\ \frac{[\![\mathsf{Expr}]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}} = \mathtt{true}}{\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \textbf{while}\ \mathsf{Expr}\ \textbf{do}\ \mathsf{Stmt}_1\ \textbf{end}\ \mathsf{Stmt}_2, \sigma) \to^{\mathcal{K}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Stmt}_1\ \textbf{while}\ \mathsf{Expr}\ \textbf{do}\ \mathsf{Stmt}_1\ \textbf{end}\ \mathsf{Stmt}_2, \sigma)}$$

$$\textbf{(loop2)}\ \frac{[\![\mathsf{Expr}]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}} = \mathtt{false}}{\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \textbf{while}\ \mathsf{Expr}\ \textbf{do}\ \mathsf{Stmt}_1\ \textbf{end}\ \mathsf{Stmt}_2, \sigma) \to^{\mathcal{K}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Stmt}_2, \sigma)}$$

$$\textbf{(assign1)}\ \frac{[\![\mathsf{Expr}]\!]^{\sigma,\ \mathsf{obs}\ (\mathtt{C},\rho)_{\mathtt{Y}}\ \mathsf{obs}'}_{\mathtt{X}} = \mathtt{Y} \qquad [\![\mathsf{Expr}']\!]^{\sigma,\ \mathsf{obs}\ (\mathtt{C},\rho)_{\mathtt{Y}}\ \mathsf{obs}'}_{\mathtt{X}} = \mathsf{e}}{\mathsf{CT}\ \mathsf{obs}\ (\mathtt{C},\rho)_{\mathtt{Y}}\ \mathsf{obs}'\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Expr}.\mathtt{f}{=}\mathsf{Expr}';\ \mathsf{Stmt}, \sigma) \to^{\mathcal{K}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ (\mathtt{C},\rho[\mathtt{f} \mapsto \mathsf{e}])_{\mathtt{Y}}\ \mathsf{obs}'\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Stmt}, \sigma)}$$

$$\textbf{(assign2)}\ \frac{[\![\mathsf{Expr}]\!]^{\sigma,\ \mathsf{obs}}_{\mathtt{X}} = \mathsf{e}}{\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathtt{v}\ =\ \mathsf{Expr};\ \mathsf{Stmt}, \sigma) \to^{\mathcal{K}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Stmt}, \sigma[\mathtt{v} \mapsto \mathsf{e}])}$$

$$\textbf{(assign3)}\ \frac{[\![\mathsf{Expr}]\!]^{\sigma,\ \mathsf{obs}}_{\mathtt{X}} = \mathsf{e}}{\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Type}\ \mathtt{v}\ =\ \mathsf{Expr};\ \mathsf{Stmt}, \sigma) \to^{\mathcal{K}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Stmt}, \sigma[\mathtt{v} \mapsto \mathsf{e}])}$$

$$\textbf{(skip)}\ \frac{}{\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \textbf{skip};\ \mathsf{Stmt}, \sigma) \to^{\mathcal{K}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Stmt}, \sigma)}$$

$$\textbf{(callIn)}\ \frac{\mathtt{v}\ \text{fresh} \qquad [\![\mathsf{Expr}]\!]^{\sigma,\mathsf{obs}}_{\mathtt{Y}} = \mathtt{X} \qquad (\mathtt{C},\rho)_{\mathtt{X}} \in \mathtt{obs} \qquad \mathsf{returnType}(\mathtt{C},\mathtt{m}) = \mathsf{Type}}{\mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Expr}.\mathtt{m}(\overline{\mathsf{Expr}});\ \mathsf{Stmt}, \sigma) \to^{\mathcal{K}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m},\ \mathtt{X},\ \mathsf{Type}\ \mathtt{v}\ =\ \mathsf{Expr}.\mathtt{m}(\overline{\mathsf{Expr}}), \sigma)}$$

Figure 22 Rules with a local effect that do not depend on semantic lifting: branching, iteration, assignment, variable declarations and calls without target variable.

B The Type System

We use a normalized form of SMOL in this section, in order to make the formalization of the type system more simple. In particular, (1) all expressions with side-effects target variable declarations, and (2) all methods end with a return statement. We further consider a Java-style type hierarchy that distinguishes basic and class types. Given a program Prog, we denote by $\texttt{C}_1$ *extends* $\texttt{C}_2$ that $\texttt{C}_1$ is declared to extend class $\texttt{C}_2$ in Prog.

▶ **Definition 24** (Type Hierarchy & Subtyping). *Given a program* Prog*, let* $\mathcal{T}$ *denote the associated type hierarchy, defined as follows:*

- `Object`, `Unit`, `Int`, `Bool`, $\top, \bot \in \mathcal{T}$ *and*
- `C` $\in \mathcal{T}$ *and* `List<C>` $\in \mathcal{T}$ *for all classes* `C`*.*

Subtyping is defined as the minimal partial order $\preceq$ *over* $\mathcal{T}$*, satisfying the following conditions:*

- $\forall T \in \mathcal{T}.\ T \preceq \top$,
- $\forall T \in \mathcal{T}.\ \bot \preceq T$,
- $\forall$`C`. `C` $\preceq$ `Object`,
- `List<C>` $\preceq$ `Object`,
- $\texttt{C}_1 \preceq \texttt{C}_2$ *if* $\texttt{C}_1$ *extends* $\texttt{C}_2$,
- `List<`$\texttt{C}_1$`>` $\preceq$ `List<`$\texttt{C}_2$`>` *if* $\texttt{C}_1 \preceq \texttt{C}_2$*, and*
- `Int`, `Unit`, `Bool` $\not\preceq$ `Object`

If there is no such partial order because of cycles in the *extends* relation, then type checking immediately fails. We write $\mathsf{defines}(\mathsf{CT}, \mathsf{Type})$ if type Type is either a basic type, or defined in the program, i.e., $\mathsf{Type} \in \mathbf{dom}(\mathsf{CT})$.

B.1 Typing Surface Syntax

We first describe the typing judgements and rules for programs, classes and methods, given in Figure 23. The typing hierarchy and class table are implicitly given, to avoid syntactic clutter.

Program Layer. The type judgement $\vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Prog}$ holds if the program Prog is type-safe with respect to knowledge base $\mathcal{K}$ and entailment regime er. In practice, we always use the SMOL ontology and the user provided domain knowledge here, i.e., $\mathcal{K} = \mathcal{K}_{\texttt{SMOL}} \cup \mathcal{K}_{\mathbf{domain}}$. The rule **(prog)** expresses simply that all classes and the main block must be well-typed with respect to the class table of the program and the given knowledge base and entailment regime.

Class Layer. The type judgement $\vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Class}$ holds if the class Class is type-safe with respect to knowledge base $\mathcal{K}$, and entailment regime er. The rule **(class)** checks that the extended class exists, that no declared field exists in a superclass, that all field types are defined and then type checks all methods.

Method Layer. The type judgement $\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Met}$ holds if the method Met is type-safe with respect to knowledge base $\mathcal{K}$ and entailment regime er. Here, the typing environment Γ captures the additional context for the type judgment, this typing environment consists of types for the fields of the surrounding class. The rule **(method)** again checks that all used types are defined and type checks the method body.

The type judgement for statements is given as $\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt} : \mathsf{Type} \rhd \Gamma'$, and expresses that the statement Stmt returns a value of type Type under environment Γ and updates the environment to Γ' (this update of the typing environmentis needed for variable declarations).[16] The type `Unit` is used for statements that do not return, but are still well-typed in the sense of containing no substatements that could cause a runtime error, as discussed.

[16] By slight abuse of notation, we consider **return** a statement here.

$$\textbf{(prog)}\ \frac{\emptyset \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt} : \mathtt{Unit} \rhd \Gamma \qquad \forall i \leq n.\ \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Class}_i}{\vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Class}_1 \dots \mathsf{Class}_n\ \textbf{main}\ \mathsf{Stmt}\ \textbf{end}}$$

$$\textbf{(class)}\ \frac{\begin{array}{c}\forall i \leq m.\ \{\mathtt{f}_1 \mapsto \mathsf{Type}_1, \dots, \mathtt{f}_n \mapsto \mathsf{Type}_n, \textbf{this} \mapsto \mathtt{C}\} \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Met}_i \qquad \mathtt{D} \in \mathbf{dom}(\mathsf{CT}) \\ \forall \mathtt{E}.\ \left(\mathtt{C} \prec \mathtt{E} \rightarrow \nexists T.\ T\ \mathtt{f}_i \in \mathsf{fields}(\mathsf{CT}, \mathtt{E})\right) \qquad \forall i \leq n.\ \mathtt{defines}(\mathsf{CT}, \mathsf{Type}_i)\end{array}}{\mathsf{CT} \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{class}\ \mathtt{C}\ \textbf{extends}\ \mathtt{D}(\mathsf{Type}_1\ \mathtt{f}_1, \dots, \mathsf{Type}_n\ \mathtt{f}_n)\ \mathsf{Met}_1 \dots \mathsf{Met}_m\ \textbf{end}}$$

$$\textbf{(method)}\ \frac{\begin{array}{c}\mathtt{defines}(\mathsf{CT}, \mathsf{Type}) \qquad \forall i \leq n.\ \mathtt{defines}(\mathsf{CT}, \mathsf{Type}_i) \\ \Gamma \cup \{\mathtt{v}_1 \mapsto \mathsf{Type}_1, \dots, \mathtt{v}_n \mapsto \mathsf{Type}_n\} \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt}\ \textbf{return}\ \mathsf{Expr};\ : \mathsf{Type}\end{array}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Type}\ \mathtt{m}(\mathsf{Type}_1\ \mathtt{v}_1, \dots, \mathsf{Type}_n\ \mathtt{v}_n)\ \mathsf{Stmt}\ \textbf{return}\ \mathsf{Expr};\ \textbf{end}}$$

Figure 23 Typing Rules for Programs, Classes and Methods.

$$\textbf{(T-fresh)}\ \frac{\begin{array}{c}\mathtt{v} \notin \mathbf{dom}\ \Gamma \qquad \Gamma' \vdash^{\mathcal{K}}_{\mathsf{er}} \mathtt{v}\ \mathtt{:=}\ \mathsf{RHS};\ : \mathtt{Unit} \\ \Gamma' = \Gamma[\mathtt{v} \mapsto \mathsf{Type}] \qquad \Gamma' \vdash^{\mathcal{K}}_{\mathsf{er}} \mathtt{Stmt} : \mathsf{Type}\end{array}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathtt{Type}\ \mathtt{v}\ \mathtt{:=}\ \mathsf{RHS};\ \mathtt{Stmt}\ : \mathtt{Unit}} \qquad \textbf{(T-wh)}\ \frac{\Gamma \vdash \mathtt{Expr} : \mathtt{Bool} \qquad \Gamma \vdash \mathtt{Stmt};\ \textbf{skip};\ : \mathsf{Type}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{while}\ \mathtt{Expr}\ \textbf{do}\ \mathtt{Stmt}\ \textbf{end} : \mathsf{Unit}}$$

$$\textbf{(T-if)}\ \frac{\begin{array}{c}\Gamma \vdash \mathtt{Expr} : \mathtt{Bool} \\ \Gamma \vdash \mathtt{Stmt}_1;\ \textbf{skip};\ : \mathsf{Type} \\ \Gamma \vdash \mathtt{Stmt}_2;\ \textbf{skip};\ : \mathsf{Type}\end{array}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{if}\ \mathtt{Expr}\ \textbf{then}\ \mathtt{Stmt}_1\ \textbf{else}\ \mathtt{Stmt}_2\ \textbf{end} : \mathsf{Type}} \qquad \textbf{(T-sequence)}\ \frac{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathtt{Stmt}_1 : \mathtt{Unit} \qquad \Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathtt{Stmt}_2 : \mathsf{Type}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathtt{Stmt}_1\ \mathtt{Stmt}_2 : \mathsf{Type}}$$

$$\textbf{(T-skip)}\ \frac{}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{skip};\ : \mathtt{Unit}}$$

$$\textbf{(T-return)}\ \frac{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathtt{Expr} : \mathsf{Type}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{return}\ \mathtt{Expr};\ : \mathsf{Type}} \qquad \textbf{(T-assign)}\ \frac{\Gamma \vdash \mathsf{Loc} : \mathsf{Type} \qquad \Gamma \vdash \mathsf{RHS} : \mathsf{Type}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Loc}\ \mathtt{:=}\ \mathsf{RHS};\ : \mathtt{Unit}}$$

Figure 24 Typing Rules for Statements.

Figure 24 gives the rules for statements. We first consider composed statements. Sequential composition is handled by two rules. Rule **(T-fresh)** updates the typing environment and continues with type checking the remainder of the program. Rule **(T-sequence)** first type checks the first and then the second statement in the sequential composition. The typing environment is not updated, because there is no rule for a variable declaration except **(T-fresh)** – if the first statement introduces a new variable, then rule **(T-sequence)** is *not* applicable. Rule **(T-if)** handles branching. The guard expression is typed as a Boolean, and both branches must have the same type. Note that the environment is not relevant – variables declared in one branch are not available afterward. Rule **(T-wh)** is analogous for loops. Rule **(T-skip)** always types the **skip** statement with `Unit`. Rule **(T-return)** types the return statement with the type of the returned expression. Rule **(T-assign)** is for assignments that do not declare new local variables. It, thus, does not update the environment. It type-checks the right-hand side expression. Assignability is ensured through subtyping, for which we have a special rule **(subtype)** for expressions (see below).

The remaining rules consider the typing of expressions and right-hand sides. Figure 25 gives the rules for the right-hand side expressions handling semantic state access. Rule **(T-validate)** always returns a Boolean. The reasoning behind **(T-acc)** is discussed in Section 5.2.1. Rule **(T-member)** uses an analogous check on the given DL concept to ensure that the returned objects can be represented at runtime.

$$\textbf{(T-validate)}\ \frac{}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{validate}(\texttt{shacl}) : \texttt{Bool}} \qquad \textbf{(T-member)}\ \frac{\texttt{dl} \sqsubseteq^{\mathcal{K}}_{\mathsf{er}} \texttt{C}^{\mathbf{prog}}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{member}(\texttt{dl}) : \texttt{List<C>}}$$

$$\textbf{(T-acc)}\ \frac{\forall i \leq n.\ \Gamma \vdash \mathsf{Expr}_i : \mathsf{Type}_i \qquad \texttt{SELECT ?obj \{P. ?v}_1\ \texttt{a}\ \mu(\mathsf{Type}_1).\ \ldots.\ \texttt{?v}_n\ \texttt{a}\ \mu(\mathsf{Type}_n)\} \subseteq^{\mathcal{K}}_{\mathsf{er}} \texttt{SELECT ?obj\{?obj a C}^{\mathbf{prog}}\texttt{\}}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{access}(\texttt{"SELECT ?obj\{P\}"}, \mathsf{Expr}_1, \ldots, \mathsf{Expr}_n) : \texttt{List<C>}}$$

Figure 25 Typing Rules for Right-Hand Sides.

$$\textbf{(literal-int)}\ \frac{}{\Gamma \vdash \texttt{n} : \mathsf{Int}} \qquad \textbf{(this)}\ \frac{\Gamma(\textbf{this}) = \mathsf{Type}}{\Gamma \vdash \textbf{this} : \mathsf{Type}} \qquad \textbf{(var)}\ \frac{\Gamma(\texttt{v}) = \mathsf{Type}}{\Gamma \vdash \texttt{v} : \mathsf{Type}}$$

$$\textbf{(literal-null)}\ \frac{\mathsf{Type} \preceq \texttt{Object}}{\Gamma \vdash \texttt{null} : \mathsf{Type}} \qquad \textbf{(compose)}\ \frac{\Gamma \vdash \mathsf{Expr} : \texttt{C} \qquad \mathsf{Type}\ \texttt{f} \in \mathsf{fields}_{\mathsf{CT}}(\texttt{C})}{\Gamma \vdash \texttt{Expr.f} : \mathsf{Type}} \qquad \textbf{(add)}\ \frac{\Gamma \vdash \texttt{Expr}_1 : \texttt{Int} \qquad \Gamma \vdash \texttt{Expr}_2 : \texttt{Int}}{\Gamma \vdash \texttt{Expr}_1 + \texttt{Expr}_2 : \texttt{Int}}$$

$$\textbf{(subtype)}\ \frac{\Gamma \vdash \mathsf{Expr} : \mathsf{Type}_1 \qquad \mathsf{Type}_1 \preceq \mathsf{Type}_2}{\Gamma \vdash \mathsf{Expr} : \mathsf{Type}_2}$$

$$\textbf{(T-call)}\ \frac{\Gamma \vdash \mathsf{Expr} : \texttt{C} \qquad \Gamma \vdash \mathsf{Expr}_i : \mathsf{Type}_i \text{ for all } i \leq n \qquad \mathsf{Type}\ \texttt{m}(\mathsf{Type}_1\ \texttt{f}_1, \ldots, \mathsf{Type}_n\ \texttt{f}_n) \in \mathsf{methods}_{\mathsf{CT}}(\texttt{C})}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Expr}.\texttt{m}(\mathsf{Expr}_1, \ldots, \mathsf{Expr}_n) : \mathsf{Type}}$$

$$\textbf{(T-new)}\ \frac{\mathsf{fields}_{\mathsf{CT}}(\texttt{C}) = \{\mathsf{Type}_1\ \texttt{f}_1, \ldots, \mathsf{Type}_n\ \texttt{f}_n\} \qquad \texttt{C} \in \mathbf{dom}(\mathsf{CT}) \qquad \Gamma \vdash \mathsf{Expr}_i : \mathsf{Type}_i \text{ for all } i \leq n}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Type}\ \textbf{new}\ \texttt{C}(\mathsf{Expr}_1, \ldots, \mathsf{Expr}_n) : \texttt{C}}$$

Figure 26 Typing rules for expressions and right hand sides.

The remaining typing rules for expressions are given in Figure 26. The typing judgement has the form $\Gamma \vdash \mathsf{Expr} : \mathsf{Type}$, with the intuitive meaning that under typing environment Γ the expression Expr has type Type. Rule **(this)** types the **this** literal with the carried type self. Rule **(literal-null)** types the **null** literal with any subtype of `Object`. Rule **(literal-int)** is representative for all typing rules for literals. It types every integer literal with `Int`. Rule **(var)** looks up the type of A variable in the typing environment. Rule **(compose)** types the sub-expression with some class `C`, and looks up the type of the field in the class table Rule **(add)** is representative for the underspecified set of expressions with operators. It types both sub-expressions with `Int`, and types the overall expression with `Int` as well. Rule **(T-call)** handles method calls, and its type is the return type of the method. The parameters require a more elaborate check: First, the target expression is typed with a class, then the class table is accessed to retrieve the abstract method parameters. Each of the expressions is then checked against the type of the corresponding abstract parameter. Furthermore, the number of concrete and abstract parameters must be the same. Rule **(T-new)** is analogous for object allocation.

B.2 Typing Runtime Syntax

We now consider the typing of runtime configurations, which amounts to typing all statements in the process stack and class table, and checking consistency constraints; e.g., every runtime return statement must have a corresponding statement to continue in the next lower process on the stack. Also, we need to generate the corresponding typing environments.

$$\textbf{(R-conf)}\ \frac{\forall i \leq n.\ \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Class}_i \quad \vdash \mathsf{obs} \quad \mathsf{obs} \vdash \mathsf{prs}}{\vdash \mathsf{obs}\ \mathsf{prs}} \qquad \textbf{(R-obs-1)}\ \frac{}{\vdash \emptyset} \qquad \textbf{(R-obs-2)}\ \frac{\forall i \leq n.\ \mathsf{CT} \vdash \mathsf{ob}_i}{\vdash \{\mathsf{ob}_1 \dots \mathsf{ob}_n\}}$$

$$\textbf{(R-obs-3)}\ \frac{\mathsf{fields}_{\mathsf{CT}}(\mathtt{C}) = \{\mathsf{Type}_{\mathtt{f}}\ \mathtt{f}\} \quad \mathbf{dom}\ \rho = \{\mathtt{f} \mid \mathsf{Type}_{\mathtt{f}}\ \mathtt{f} \in \mathsf{fields}(\mathtt{C})\} \quad \forall \mathtt{f} \in \mathbf{dom}\ \rho.\ \emptyset \vdash \rho(\mathtt{f}) : \mathsf{Type}_{\mathtt{f}}}{\mathsf{CT} \vdash (\mathtt{C}, \rho)_{\mathtt{X}}}$$

$$\textbf{(R-prs-1)}\ \frac{}{\mathsf{obs} \vdash \emptyset} \qquad \textbf{(R-prs-2)}\ \frac{\begin{array}{c}\mathsf{obs} \vdash \mathsf{prs} \quad \Gamma(\sigma, \rho, \mathtt{C}, \mathsf{obs}) \vdash \mathsf{Stmt} : \mathsf{Type} \rhd \Gamma' \\ \mathsf{Type}\ \mathtt{m}(\dots) \in \mathsf{methods}_{\mathsf{CT}}(\mathtt{C}) \quad (\mathtt{C}, \rho)_{\mathtt{X}} \in \mathsf{obs} \\ \mathsf{Stmt} \neq \mathsf{Stmt}';\ \textbf{return}\ \mathsf{Expr}; \quad \mathsf{Stmt} \neq \mathsf{Loc} \leftarrow \mathtt{stack}; \mathsf{Stmt'}\end{array}}{\mathsf{obs} \vdash \mathsf{prs}, (\mathtt{m}, \mathtt{X}, \mathsf{Stmt}, \sigma)}$$

$$\textbf{(R-prs-3)}\ \frac{\begin{array}{c}(\mathtt{C}_1, \rho_1)_{\mathtt{X}_1} \in \mathsf{obs} \quad (\mathtt{C}_2, \rho_2)_{\mathtt{X}_2} \in \mathsf{obs} \\ \mathsf{obs} \vdash \mathsf{prs} \quad \Gamma(\sigma_2, \rho_2, \mathtt{C}_2, \mathsf{obs}) \vdash \mathsf{Expr} : \mathsf{Type}(\mathsf{Loc}) \\ \Gamma(\sigma_1, \rho_1, \mathtt{C}_1, \mathsf{obs}) \vdash \mathsf{Stmt}_1 : \mathsf{Type}_1 \rhd \Gamma_1 \quad \Gamma(\sigma_2, \rho_2, \mathtt{C}_2, \mathsf{obs}) \vdash \mathsf{Stmt}_2 : \mathsf{Type}_2 \rhd \Gamma_2\end{array}}{\mathsf{obs} \vdash \mathsf{prs}, (\mathtt{m}_1, \mathtt{X}_1, \mathsf{Loc} \leftarrow \mathtt{stack}; \mathsf{Stmt}_1, \sigma_1),\ (\mathtt{m}_2, \mathtt{X}_2, \mathsf{Stmt}_2;\ \textbf{return}\ \mathsf{Expr};, \sigma_2)}$$

Figure 27 Typing rules for runtime configurations.

Figure 27 gives the typing rules for runtime configurations. Rule **(R-conf)** checks a whole configuration. The first premise type-checks all classes in the class table, the second the objects, assuming a well-typed class table, and the last checks the processes, using the well-typed class table and objects. As we will see, the first premise is not required if the configuration is reached from an initial configuration. Rule **(R-obs-1)** states that an empty set of objects is well typed, and rule **(R-obs-2)** decomposes checking a set of objects into checking each object in isolation. Rule **(R-obs-3)** checks a single object. Each field of the given class must have a value assigned in the memory, and the type of the value (checked as an expression) must be of the declared type. Rule **(R-prs-1)** states that an empty process stack is well-typed, and rule **(R-prs-2)** handles all processes on the stack with a non-return statement on top. It reduces to check the statement in the context generated from the heap and local memory with

$$\begin{aligned}\Gamma(\sigma, \rho, \mathtt{C}, \mathsf{obs}) = \ & \{\mathtt{v} \mapsto \mathsf{Type} \mid \mathtt{v} \in \mathbf{dom}\ \sigma, \emptyset \vdash \sigma(\mathtt{v}) : \mathsf{Type} \text{ or } (\mathsf{Type}, \rho')_{\sigma(\mathtt{v})} \in \mathsf{obs}\} \\ & \cup \{\mathtt{f} \mapsto \mathsf{Type} \mid \mathtt{f} \in \mathbf{dom}\ \rho, \emptyset \vdash \rho(\mathtt{f}) : \mathsf{Type} \text{ or } (\mathsf{Type}, \rho')_{\rho(\mathtt{f})} \in \mathsf{obs}\} \\ & \cup \{\textbf{this} \mapsto \mathtt{C}\}\end{aligned}$$

Finally, rule **(R-prs-3)** is concerned with process stacks where the next statement to be executed is a return; here, the next lower process must start with a continuation. Note that any process stack not matching these rules, is ill-typed.

B.3 Soundness

▶ **Lemma 25** (Initial State). *The initial configuration of a well-typed program is well-typed:*

$$\vdash \mathsf{Prog} \quad \Rightarrow \quad \vdash \mathsf{init}_{\mathsf{Prog}}\ .$$

Proof. We need to show that we can construct a proof tree to type check $\mathsf{init}_{\mathsf{Prog}}$ with the rule **(R-conf)** as its root, given a tree for Prog with **(prog)** as its root.

First premise: This follows from the second premise of rule **(prog)**, except the class `Entry`, which is well-typed if the main statement is well-typed, which is the first premise of **(prog)**.

Second premise: By definition there is only one object that is already created, which is of class `Entry`. Ergo, we must type is with rule **(R-obs-3)**. This class has no fields, thus the first premise trivially holds. The generated heap is empty, thus the second and third premises also hold.

Third premise: By definition there is only one process, which we must type with **(R-prs-2)**. As the main block is well-typed with Unit, which is the type of the generated `entry` method, the first two premises hold. The third premise holds trivially by generation. The forth one is guaranteed to hold as the identifier and class name are fixed for all initial configurations. The last premises hold as return and the waiting statement are not allowed in the main block. ◀

Before we show that every the lifting of well-typed configuration is consistent, we give an auxiliary structures as an intermediate steps.

▶ **Lemma 26.** *The lifting of a class table of every well-typed program is consistent:*

$$\vdash \mathsf{Prog} \quad \Rightarrow \quad \bigcup_{\mathtt{C} \in \mathsf{CT}_{\mathsf{Prog}}} \mu(\mathtt{C}) \cup \mathcal{K}_{\mathtt{SMOL}} \textit{ is consistent.}$$

Proof. Let $|\mathsf{Prog}|$ be the number of classes in a program. Let $|\mathsf{Class}|$ be the number of fields and methods within a class. We prove the theorem by induction on $n = |\mathsf{Prog}|$.

Induction hypothesis 1: $\forall n.\ |\mathsf{Prog}| = n \Rightarrow \big(\vdash \mathsf{Prog} \Rightarrow \bigcup_{\mathtt{C} \in \mathsf{CT}_{\mathsf{Prog}}} \mu(\mathtt{C}) \cup \mathcal{K}_{\mathtt{SMOL}} \text{ is consistent}\big)$

Induction base $n = 0$: In this case, the class table is empty and the lifting consist only of $\mathcal{K}_{\mathtt{SMOL}}$. This set of axioms is consistent, as is easily shown by inputing it into a DL reasoner.

Induction step $n > 0$: Before we continue, we observe the structure of the lifted class table: Besides additional axioms stemming from the subclass relation, it is saturated, i.e., no new axioms can be derived. Furthermore, it contains no counting axioms, as the only axioms that limit the number of members of a concept are in *close*. Thus, there are only 4 ways to make the knowledge graph inconsistent: (1) violating a domain axiom, (2) violating a range axiom, (3) violating a disjointness axiom, and (4) violating a set axiom from *close*. Note that all of these inconsistencies do involve more than one axiom, but each inconsistency must involve (at least) one of the above.

We remind that the axioms generated from lifting a class are as follows.

OWL

```
Individual: C^prog
  Facts: a Class^SMOL, hasName^SMOL "C",
  [subClass^SMOL D^prog],      // if C extends D
  [subClass^SMOL Any^SMOL],    // otherwise
  hasMethod^SMOL m_1^prog, ..., hasMethod^SMOL m_n^prog,
  hasField^SMOL f_1^prog, ..., hasField^SMOL f_k^prog

Individual: m_1^prog Facts: a Method^SMOL, hasName^SMOL "m_1"
...
Individual: f_1^prog Facts: a Field^SMOL, hasName^SMOL "f_1"
...
```

Let Class be any class in Prog, such that (by induction hypothesis 1) the other n classes are lifted to a consistent knowledge graph. We now proceed with an induction on $m = |\mathsf{Class}|$.

Induction hypothesis 2: $\forall m.\ |\mathsf{Class}| = m \Rightarrow \big(\mu(\mathsf{Class}) \cup \mathcal{K}_{\mathtt{SMOL}} \text{ is consistent}\big)$

Induction base $m = 0$: In this case the class `C` has no fields or methods.

- The first axiom generated connects the name to the IRI of the class using $\texttt{hasName}^{\texttt{SMOL}}$. The only interactions this axiom have is with (1) the domain axiom for this property, which is observed, because the lifting of CT states that $\texttt{C}^{\mathbf{prog}}\ \texttt{a}\ \texttt{Class}^{\texttt{SMOL}}$ explicitly, and (2) the range axiom, which is observed, as the name has the right data type, namely a `xsd:String`.
- The second group of axioms model the subtyping relation and the disjointness. These can only interact with each other, but all the subtyping relation axioms are consistent, as they form a tree by definition of the class system which excludes cycles, and the program is well-typed by the assumption of this theorem.
- The last axiom is the set axiom for $\texttt{Class}^{\texttt{SMOL}}$, which cannot cause an inconsistency as the membership of $\texttt{C}^{\mathbf{prog}}$ is state explicitly.

Induction step $m > 0$: We distinguish the cases for fields and methods.

Fields: The range and domain axioms cannot cause inconsistencies because the field is never used. The axioms for $\texttt{hasField}^{\texttt{SMOL}}$ and $\texttt{hasName}^{\texttt{SMOL}}$ again fulfill their range and domain axioms by construction and the last axiom $\texttt{f}^{\mathbf{prog}}\ \texttt{a}\ \texttt{Field}^{\texttt{SMOL}}$.

Methods: Analogous to fields. ◀

For our main theorem, we consider the following property that connects typability of runtime configurations and consistency of the corresponding knowledge bases.

▶ **Theorem 27** (Connection). *The lifting of every well-typed configuration is consistent:*

$\vdash \mathsf{conf} \Rightarrow \mu(\mathsf{conf})$ *is consistent.*

Proof. We have

$$\mu(\texttt{conf}) = \bigcup_{\texttt{C}\in\mathbf{dom}(\mathsf{CT})} \mu(\texttt{C}) \cup \bigcup_{1\leq \texttt{X}\leq n} \left(\mu(\mathsf{ob}_i) \cup \mathsf{links}(\texttt{X})[\texttt{X}^{\mathbf{run}}, \texttt{conf}]\right) \cup \mathit{close}$$

and, by Lemma 26, the lifted class table $(\bigcup_{\texttt{C}\in\mathbf{dom}(\mathsf{CT})} \mu(\texttt{C}) \cup \mathcal{K}_{\texttt{SMOL}})$ in itself is consistent. We show that (1) the lifting of objects is consistent, (2) that the lifting of objects is consistent with the lifted class table, and (3) both liftings are consistent with the closure axioms.

- We show that the following is consistent:

  $$\bigcup_{1\leq \texttt{X}\leq n} \left(\mu(\mathsf{ob}_i) \cup \mathsf{links}(\texttt{X})[\texttt{X}^{\mathbf{run}}, \texttt{conf}]\right) \cup \mathcal{K}_{\texttt{SMOL}} \ .$$

  Following the structure of Lemma 26, we consider each added axiom in isolation, and compare it with the range and domain axioms of the respective property. It is easy to see that the lifting follows domain and range, with the only critical point being the distinction between data and object property. The linkage is consistent due to being a conservative extension.
- We show that the union of of lifted class table and lifted objects is consistent:

  $$\bigcup_{\texttt{C}\in\mathbf{dom}(\mathsf{CT})} \mu(\texttt{C}) \cup \bigcup_{1\leq \texttt{X}\leq n} \left(\mu(\mathsf{ob}_i) \cup \mathsf{links}(\texttt{X})[\texttt{X}^{\mathbf{run}}, \texttt{conf}]\right) \cup \mathcal{K}_{\texttt{SMOL}} \ .$$

  The two axioms sets, which are consistent in themselves, interact only on class and field individuals. It is easy to see that the use of the $\texttt{entryOf}^{\texttt{SMOL}}$ and $\texttt{implements}^{\texttt{SMOL}}$ adhere to the domain and range axioms of the ontology.
- It remains to show that the union of the above with *close* is consistent. This is straigthforward: this set of axioms only defines classes in terms of sets of individuals. All these sets are disjoint, and there are no relevant subclass relations. There are no counting axioms in the SMOL ontology that could limit the number of individuals, and that the domain knowledge is a conservative extension and cannot introduce such restrictions other. Thus, there are no axioms that could be used to derive a contradiction. ◀

As our evaluation of side-effect and non-semantic expressions is underspecified, we impose the following restriction on it, which is standard and independent of semantic state access, as these constructs are handled by evaluation of statements, not expressions. We require the following property to connect typability of expression with their evaluation. As we underspecify all expressions except the ones related to retrieve objects, we only state this assumption for objects.

▶ **Assumption 28.** *If* $\Gamma \vdash \mathsf{Expr} : \mathtt{C}$, $\mathsf{CT} \vdash \mathsf{obs}$ *and* $(\mathtt{C}, \rho)_{\mathtt{X}} \in \mathsf{obs}$, *then either (1)* $[\![\mathsf{Expr}]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}} = \mathbf{null}$ *or (2)* $[\![\mathsf{Expr}]\!]^{\sigma,\mathsf{obs}}_{\mathtt{X}} = \mathtt{Y}$, *such that* $(\mathtt{D}, \rho)_{\mathtt{Y}} \in \mathsf{obs}$ *and* $\mathtt{D} \preceq \mathtt{C}$.

We now prove the subject reduction theorem and show that being well-typed is an invariant at runtime. We refrain from giving full formal details because besides the case for **access**, the system is a standard object-oriented language.

▶ **Lemma 29** (Subject Reduction). *Every transition from a well-typed configuration results in a well-typed configuration.*

$$\vdash \mathsf{conf} \wedge \mathsf{conf} \rightarrow^{\mathcal{K}}_{\mathsf{er}} \mathsf{conf}' \quad \Rightarrow \quad \vdash \mathsf{conf}' \ .$$

Proof. Case distinction on the rule used to make the transition.

- **Rule (iftrue):** We must show that for all Γ if

  $$\Gamma \vdash \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m}, \mathtt{X}, \mathbf{if}\ \mathsf{Expr}\ \mathbf{then}\ \mathsf{Stmt}_1\ \mathbf{else}\ \mathsf{Stmt}_2\ \mathbf{end}\ \mathsf{Stmt}, \sigma)$$

  then

  $$\rightarrow^{\mathcal{K}}_{\mathsf{er}} \mathsf{CT}\ \mathsf{obs}\ \mathsf{prs}, (\mathtt{m}, \mathtt{X}, \mathsf{Stmt}_1\ \mathsf{Stmt}, \sigma) \ .$$

  First, we observe that the type trees differ only in the statement, as the rule does not change the environment, objects or process. Thus, we only need to prove that if

  $$\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathbf{if}\ \mathsf{Expr}\ \mathbf{then}\ \mathsf{Stmt}_1\ \mathbf{else}\ \mathsf{Stmt}_2\ \mathbf{end}\ \mathsf{Stmt} :\ \mathsf{Type}$$

  then

  $$\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt}_1\ \mathsf{Stmt} :\ \mathsf{Type} \ .$$

  By assumption, we have the following derivation tree:

  $$\dfrac{\dfrac{\dfrac{(1)}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt}_2 :\ \mathsf{Type}} \quad \dfrac{(2)}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt}_1 :\ \mathsf{Type}}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathbf{if}\ \mathsf{Expr}\ \mathbf{then}\ \mathsf{Stmt}_1\ \mathbf{else}\ \mathsf{Stmt}_2\ \mathbf{end} :\ \mathsf{Type}}\ \textbf{(T-if)} \quad \dfrac{(3)}{\Gamma_2 \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt} :\ \mathsf{Type}}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathbf{if}\ \mathsf{Expr}\ \mathbf{then}\ \mathsf{Stmt}_1\ \mathbf{else}\ \mathsf{Stmt}_2\ \mathbf{end}\ \mathsf{Stmt} :\ \mathsf{Type}}\ \textbf{(T-sequence)}$$

  Here $(1), (2), (3)$ are closed derivation trees, $\mathbf{dom}\Gamma_3 \supseteq \mathbf{dom}\Gamma_2$, and Γ_3 and Γ_2 do agree in their image on the domain of Γ_2. It is easy to see that if a statement can be typed with Γ_2, then it can also be typed with Γ_3. Thus, we can construct the following derivation tree for the target judgement:

  $$\dfrac{\dfrac{\dfrac{(3)}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt}_1 :\ \mathsf{Type}}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt}_1 :\ \mathsf{Type}}\ \textbf{(T-weak)} \quad \dfrac{(2)}{\Gamma_2 \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt} :\ \mathsf{Type}}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt}_1\ \mathsf{Stmt} :\ \mathsf{Type}}\ \textbf{(T-sequence)}$$

- **Rules (iffalse), (loop1), (loop2):** Analogously to (iftrue), we just copy over the right subtrees in the rewritting.
- **Rule (assign1):** We must show that, under the conditions described by the premises, if

  $$\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Expr}.f := \mathsf{Expr}'; \ \mathsf{Stmt} : \ \mathsf{Type}$$

  and

  $$\mathsf{CT} \vdash (\mathtt{C}, \rho)_{\mathtt{Y}}$$

  then

  $$\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt} : \ \mathsf{Type}$$

  and

  $$\mathsf{CT} \vdash (\mathtt{C}, \rho[\mathtt{f} \mapsto \mathtt{e}])_{\mathtt{Y}} \ .$$

  By assumption, we have the following (slightly simplified) derivation tree

  $$\dfrac{\dfrac{(3)}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Expr}' : \mathsf{Type}_2 \rhd \Gamma'} \quad \dfrac{\dfrac{(2)}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Expr} : \ \mathtt{C}}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Expr}.f : \ \mathsf{Type}_1}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Expr}.f := \mathsf{Expr}'; : \mathsf{Unit}} \quad \dfrac{(1)}{\Gamma'' \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt} : \ \mathsf{Type}}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Expr}.f := \mathsf{Expr}'; \ \mathsf{Stmt} : \ \mathsf{Type}} \ \textsf{(T-sequence)}$$

  Obviously (1) is a derivation tree for $\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt} : \ \mathsf{Type}$. For the second statement, we must show that $\mathtt{e} : \mathsf{Type}_{\mathtt{f}}$, which follows from Assumption 28.
- **Rules (assign2), (assign3)]:** Analogous to (assign1).
- **Rule (skip):** We must show that if

  $$\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{skip}; \ \mathsf{Stmt} : \ \mathsf{Type}$$

  then

  $$\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt} : \ \mathsf{Type}$$

  By assumption, we have the following derivation tree

  $$\dfrac{\dfrac{}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{skip}; : \ \mathsf{Unit}} \ \textsf{(T-skip)} \quad \dfrac{(1)}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \mathsf{Stmt} : \ \mathsf{Type}}}{\Gamma \vdash^{\mathcal{K}}_{\mathsf{er}} \textbf{skip}; \ \mathsf{Stmt} : \ \mathsf{Type}} \ \textsf{(T-Sequence)}$$

  Thus, there is a derivation tree for (1), which is exactly the statement we need to show.
- **Rule (callIn):** This rule is not applicable in the small language, but it is trivial to see that the explicit check for the return type ensures the applicatbility of (T-call).
- **Rule (new):** We need to show if the whole configuration is well-typed and, thus, there derivation tree for the creation statement, rooted in (T-new), then there is one for the declaration for the declaration, rooted in (T-declare), and that the newly created object is well-typed.
  For the derivation tree we have, by construction, $\mathtt{Y} : \mathtt{C}$, and all subtrees can be copied over directly. For the newly created object, we must show that all evaluated values respect the type of the field they are assigned to. This is the same argument as for storing a single value in a field in (assign1) using Assumption 28.

- **Rule (call):** We must ensure that rule (R-prs-3) is applicable after the transition. Thus, we must ensure the required syntactic form, as the rest follows from the typability of the prior configuration (such as typability of the lower process stack). For this it suffices to observe that Stmt' ends in a **return** statement, as required by the rule.
- **Rule (return):** This removes exactly one pair of runtime stack statements, we must show that the resulting value is respecting the type of the expression, which is analogous to the above cases, as it is checked explicitly by (T-return), and this derivation subtree can be reused by (T-assign). Note that Stmt always ends in a **return** by definition, as it is always a method body, and, thus, we do not need to reason about its exact structure.
- **Rule (validate):** We must show that applicability of (T-validate) implies applicability of (T-declare) after the transition. By definition, Sha returns a Boolean literal, the other subtrees carry over directly.
- **Rule (member):** We must show that applicability of (T-member) implies applicability of (T-declare) after the transition. For this, we must show that we can listify the results of the membership query into a list of `C` elements.
  Only objects of `C` and its subclasses are described by $C^{\mathbf{prog}}$, the additional premise, explicitly checks that the membership query is on a concept that is a subconcept of $C^{\mathbf{prog}}$, i.e., a subset of objects of `C` and its subclasses. By Theorem 27 the original configuration is consistent, so the reasoner indeed does so. We remind the reader that we include the *close* axioms to ensure that no further individuals (that cannot be represented at runtime) can be added by the domain knowledge.Thus, every subconcept of $C^{\mathbf{prog}}$ is a subset of the individuals of representable objects of the required class.
- **Rule (access):** Analogous to (member), except that the argument is over query containment instead of concept subclassing. ◀

We obtain that for a well-typed program, every reachable state is well-typed.

▶ **Lemma 30** (Well-Typed Reachable States)**.**

$$\vdash \mathsf{Prog} \wedge \mathsf{init}_{\mathsf{Prog}} \rightsquigarrow_{\mathsf{er}}^{\mathcal{K}_{\mathbf{domain}}} \mathsf{conf} \quad \Rightarrow \quad \vdash \mathsf{conf}$$

Proof. Follows directly from Lemmas 25 and 29. ◀

We can now prove Theorem 1.

▶ **Theorem 1** (Type Safety)**.** *Let* Prog *be a program that is well-typed with respect to* $\vdash_{\mathsf{er}}^{\mathcal{K}_{\mathbf{domain}}}$*, where* $\mathcal{K}_{\mathbf{domain}}$ *is a conservative extension of* $\mathcal{K}_{\texttt{SMOL}} \cup \mu(\mathsf{CT}_{\mathsf{Prog}})$*. Every reachable configuration of* Prog *can be lifted to a consistent knowledge graph:*

$$\forall \mathsf{conf}.\ \mathsf{init}_{\mathsf{Prog}} \rightsquigarrow_{\mathsf{er}}^{\mathcal{K}_{\mathbf{domain}}} \mathsf{conf} \quad \rightarrow \quad \mu(\mathsf{conf}) \cup \mathcal{K}_{\texttt{SMOL}} \cup \mathcal{K}_{\mathbf{domain}} \textit{ is consistent.}$$

Proof. By Lemma 30, every reachable configuration is well-typed and by Theorem 27 every well-typed configuration is lifted to a consistent knowledge graph. ◀

Native Provenance Computation for Federated and Non-Federated SPARQL Queries

Zubaria Asma
FORTH-ICS, Heraklion, Crete, Greece
University of Crete, Heraklion, Crete, Greece

Daniel Hernández
University of Stuttgart, Stuttgart, Germany

Luis Galárraga
Inria, Rennes, France

Giorgos Flouris
FORTH-ICS, Heraklion, Crete, Greece

Irini Fundulaki
FORTH-ICS, Heraklion, Crete, Greece

Katja Hose
TU Wien, Wien, Austria

Abstract

The popularity of knowledge graphs (KGs) owes credit to their flexible data model, which is suitable for data integration from multiple sources. Several KG-based applications, such as trust assessment, view maintenance, or data valuation on dynamic data, rely on the ability to compute provenance explanations for query results. This need becomes more urgent in federated query processing systems, which allow the online consumption of heterogeneous and decentralized Web data. However, the problem of computing and interacting with provenance has received little attention, especially in the federated setting. On those grounds, this paper introduces the NPCS (Native Provenance Computation for SPARQL) approach, and its federated variant Fed-NPCS, that compute provenance for SPARQL query results. Both approaches build upon spm-semirings to annotate the results of monotonic and non-monotonic SPARQL queries with their provenance. Due to their reliance on query rewriting techniques, the approaches are directly applicable to already deployed SPARQL engines and federations using different reification schemes, including RDF-star. Our experimental evaluation shows that our novel query rewriting approach brings significant run-time improvements w.r.t. the state-of-the-art across both centralized and federated settings. In centralized settings, our tests on two popular SPARQL engines (GraphDB and Stardog) reveal substantial runtime gains over existing query rewriting solutions, enabling scalability to RDF graphs with billions of triples. In federated settings, our experiments on the FedShop benchmark with GraphDB show the viability of Fed-NPCS for federations with up to 200 sources.

2012 ACM Subject Classification Information systems → Data provenance; Information systems → Resource Description Framework (RDF)

Keywords and phrases native provenance computation, federated SPARQL queries, data provenance, NPCS, Fed-NPCS

Digital Object Identifier 10.4230/TGDK.4.1.4

Category Research

Supplementary Material *Software (Source Code, Dataset)*: `https://github.com/ZubariaForthAcc/NPCS_Fed-NPCS` [10]; archived at `swh:1:dir:c3726715a2159edc556ef0df042df5b515a906d6`

Acknowledgements This work was funded by the EU H2020 R&I Marie Skłodowska-Curie program, project KnowGraphs – 860801; the TAILOR project (EU Horizon 2020 R&I program, GA 952215); the COST Action CA19134; the German Research Foundation (DFG) under Germany's Excellence Strategy – EXC 2120/1 – 390831618; and the European Research Council (ERC) under the European Union's Horizon Europe Research and Innovation Programme, grant agreement No. 101200433, project META-LEARN.

Received 2025-06-06 **Accepted** 2026-04-14 **Published** 2026-05-21

1 Introduction

The fast advances in information extraction and knowledge graph (KG) construction have enriched the Web with vast amounts of structured, machine-readable data. These KGs, represented as RDF triples and queried through SPARQL endpoints, form the foundation of numerous intelligent applications such as semantic search, question answering, and digital assistants. By encoding real-world knowledge as triples in the form (s, p, o), knowledge graphs allow machines to "understand" and process complex information, driving many data-driven systems.

KGs usually aggregate data from independent and heterogeneous sources. One common approach for KG construction is to apply information extraction techniques, e.g., on the Web, to filter, cleanse, and centralize knowledge so that it can be queried on a single endpoint. Although this strategy guarantees full control over the data, it requires a lot of effort to keep the data up-to-date. On the other hand, federated query systems [2, 3, 14, 24, 29, 36, 39, 41, 42] allow users to query knowledge in a fully decentralized way, giving the impression of a large centralized KG. Federated approaches provide an abstraction layer that hides the complexity of fully distributed processing including tasks such as source selection and efficient query planning. This capability is essential for various downstream applications such as data integration platforms, search engines, or smart assistants, which require real-time access to fresh knowledge. Federated architectures are also crucial when third parties must hold control of the data for the sake of data privacy [43] or for legal reasons.

Given the heterogeneity of the data sources that contribute to modern KGs, the problem of identifying the *provenance* of query results is central – especially in the federated setting. The provenance of a query result is an expression that encodes the lineage of the data transformations and statements that contributed to that result. Provenance is of great value for KG providers because it streamlines maintenance tasks, such as source selection and view maintenance [20]. For data consumers, query provenance serves as an explanation for answers. This can be pivotal in use cases that need to assess data reliability, or manage access control and privacy [37], data valuation, trustworthiness, auditing [40], or data quality.

Among the existing formalisms to model query provenance, *how-provenance* is the most expressive [25]. In this model, the provenance of a query result is an algebraic expression in a *provenance semiring*. Consider, for instance, the following KG,

$$\{\ u_1\colon (\mathit{UK}, \mathit{capital}, \mathit{London}),\ u_2\colon (\mathit{London}, \mathit{in}, \mathit{UK}),\ u_3\colon (\mathit{London}, \mathit{a}, \mathit{City})\ \},$$

and the SPARQL query

$$\textsc{select}\ ?x\ \textsc{where}\ \{\ \{\ \mathit{UK}, \mathit{capital}, ?x\ \}\ \textsc{union}\ \{\ ?x\ \mathit{in}\ \mathit{UK}\ ;\ \mathit{a}\ \mathit{City}\ \}\ \}.$$

The answer to this query is *London*. How-provenance explains the presence of *London* in the result set with the polynomial expression $u_1 \oplus (u_2 \otimes u_3)$. This polynomial tells us that there are two ways to get *London* as a solution: either via u_1, or via the conjunction of u_2 and u_3. There exist different algebraic structures for provenance in the literature [16, 22, 25], but this paper considers spm-semirings [22] because they are designed for the semantics of SPARQL, including its non-monotonic fragment. We highlight that query provenance assumes the availability of identifiers for triples, in other words, it assumes that the KG has been reified using some scheme. Examples of reification schemes are RDF-star and named graphs.

In the centralized setting, there are essentially two main strategies to compute how-provenance for SPARQL queries. Methods such as TripleProv [44] opt for customized engines that compute provenance alongside query evaluation. Since provenance support is embedded in the engine, such solutions allow for advanced optimizations. On the downside, customized engines are not applicable to already deployed SPARQL endpoints. The other alternative is query rewriting [9, 30].

In this approach, SPARQL queries are rewritten so that the new query retrieves both the query solutions and the polynomials describing their provenance, potentially with some post-processing. This design provides the flexibility to be applied to any SPARQL endpoint on the Web at the price of a runtime overhead. Both of these approaches have been applied for the centralized setting; however, to the best of our knowledge, no system exists that allows the computation of how-provenance for query results in the federated setting.

In this paper, we first describe the *NPCS*[1] approach [9], a method to compute provenance for SPARQL queries via query rewriting. NPCS is the first fully native SPARQL solution for how-provenance computation, as previous approaches [30] required some sort of post-processing. Moreover, NPCS supports different data reification schemes, including RDF-star. NPCS is designed for centralized KGs; in this paper we introduce an extension of the method, called *Fed-NPCS*, which is applicable for queries run against federated systems. This new work includes an extension of the how-provenance formalism for federated settings, as well as a runtime evaluation of Fed-NPCS on FedShop [17], a challenging benchmark for SPARQL federated systems. To the best of our knowledge, this is the first work that addresses the computation of how-provenance explanations for SPARQL queries on federations.

Our experimental evaluation demonstrates that NPCS is consistently faster than SPARQL-prov [30], the state of the art in how-provenance in SPARQL for centralized KGs. Moreover, NPCS and Fed-NPCS provide provenance explanations with reasonable runtime overhead (around 10%) while scaling efficiently to large datasets with billions of triples and federations of up to 200 endpoints.

The rest of this paper is structured as follows. Section 2 provides an overview of related work in the areas of provenance and federated query processing, whereas Section 3 introduces some preliminary concepts that will be useful throughout the paper. Section 4 presents our query rewriting method and Section 5 describes its extension to federated SPARQL query processing systems. In Section 6, we report on our experimental evaluation. Finally, Section 7 concludes with a discussion of the implications of our work and an outline of directions for future research.

2 Related work

We survey the literature on provenance for query results along two axes: provenance models (Section 2.1) and provenance support for RDF/SPARQL engines (Section 2.2). For a survey on federated SPARQL systems and related benchmarks we refer the reader to [17, 26]. We highlight that, so far, no other work has addressed the problem of computing how-provenance for SPARQL federated queries.

2.1 Semirings and Provenance Models

Semirings were first used to model query provenance in the groundbreaking work by Green et al. [25]. This work proposed *commutative semirings* to annotate query results for selection, projection, join, and union queries for Datalog and the positive fragment of relational algebra. Commutative semirings cannot model provenance for non-monotonic operators such as the left-outer join and the difference [25], hence the algebraic structures were expanded to include a monus operator[2] that accounts for the relational difference [21]. Commutative semirings and their extensions model provenance as polynomial expressions. These expressions, called *how-provenance*,

[1] Native Provenance Computation for SPARQL

[2] Do not confuse monus with minus (the SPARQL operator). A monus operator is an operator on certain commutative monoids that are not groups (see [21]).

encode both the sources and the data transformations required to obtain (and sometimes exclude) a particular query answer. How-provenance is more expressive than other provenance models such as lineage [15] or why-provenance [13].

Damasio et al. [16] showed that by rewriting SPARQL queries into relational algebra, we can provide provenance annotations for SPARQL queries using *m-semirings*. However, Geerts et al. [22] showed that these can yield very long and complex provenance expressions, and thus developed the *spm-semirings* formalism (spm stands for SPARQL Minus) to overcome these limitations. Spm-semirings guarantee more compact explanations and offer native support for non-monotonic SPARQL operators such as OPTIONAL and MINUS. Since how-provenance polynomials are abstract annotations, they are useful to a handful of metadata management applications [18, 27] via the notion of commutation with homomorphisms.

2.2 Provenance-supported SPARQL engines

Wylot et al. [44] introduced TripleProv, a system to compute provenance annotations in the commutative semiring framework for queries with basic graph patterns, union, and the OPTIONAL operator. Due to its reliance on commutative semirings, TripleProv cannot guarantee commutation with homomorphisms for queries involving the non-monotonic OPTIONAL operator. Additionally, TripleProv uses a customized engine that organizes data into molecules – sort of indexes for star patterns – , and thus it cannot be used on already deployed SPARQL engines. This is why our approach resorts to query rewriting for how-provenance computation.

But approaches based on query rewriting are not rare at all. Perm [23] and GProM [8] are two examples. Such approaches are, however, tailored for relational databases. Hence, they are not applicable to SPARQL queries out of the box. Similarly to TripleProv, none of these methods can properly support non-monotonic SPARQL queries because Perm is based on the lineage model, and GProM relies on commutative semirings.

While the work of Geerts et al. [22] was the first to study provenance for the non-monotonic fragment of SPARQL, the first concrete method to compute how-provenance under the spm-semiring formalism was proposed by Hernández et al. [30]. They introduced SPARQLprov, a method based on query rewriting that can annotate query results with how-provenance polynomials for both monotonic and non-monotonic queries, establishing query rewriting as a practical approach for computing SPARQL how-provenance. Contrary to NPCS, SPARQLprov is not a 100% SPARQL solution because it relies on a subsequent decoding phase to compute the final provenance annotations from the results of the rewritten query. As our experimental evaluation shows, this decoding phase can incur prohibitive runtime overheads for non-selective queries.

3 Preliminaries

3.1 RDF-star

The following presentation follows the W3C Community Group Draft [28]. We assume the existence of three (pairwise disjoint) countably infinite sets: the set of IRIs $\mathbf{I}$, the set of blank nodes $\mathbf{B}$, and the set of literals $\mathbf{L}$. An RDF triple $t = (s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is a statement that consists of a *subject* s, a *predicate* p, and an *object* o. An RDF graph G is a set of RDF triples. The RDF-star data model extends RDF by allowing arbitrarily deep nesting of triples as subject or object arguments:

▶ **Definition 1** (RDF-star). *An RDF-star triple is a 3-tuple defined recursively as follows:*

- *Every RDF triple t is an RDF-star triple; and*
- *Given RDF-star triples t and t', and RDF terms $s \in (\mathbf{I} \cup \mathbf{B})$, $p \in \mathbf{I}$, and $o \in (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$, then the triples (t, p, o), (s, p, t), and (t, p, t') are also RDF-star triples.*

We write $\mathbf{T}$ to denote the set of all RDF-star triples. An RDF-star graph G is a finite set of RDF-star triples (i.e., $G \subseteq \mathbf{T}$). We write $\mathbf{G}$ to denote the set of all RDF-star graphs.

In a nutshell, RDF-star allows us to "say things" about statements, which endows RDF with native *reification* capabilities. This is crucial when computing how-provenance for query results because query provenance builds upon identifiers for triples in the graph.

▶ **Definition 2** (Federated Dataset). *Given a finite set* $Y \subseteq \mathbf{I}$*, called the set of* graph names*, a* federated dataset *over* Y *is a function* $D : Y \to \mathbf{G}$.

Intuitively, a federated dataset D associates IRIs with RDF-star graphs.

3.2 SPARQL-star

RDF-star graphs can be queried using the SPARQL-star language. The base of SPARQL-star queries is a countably infinite set $\mathbf{V}$ of variables – prefixed by the character '?' and disjoint with $\mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$. A triple pattern is defined recursively as a follows:

- A triple $(S, P, O) \in (\mathbf{V} \cup \mathbf{I} \cup \mathbf{B}) \times (\mathbf{V} \cup \mathbf{I}) \times (\mathbf{V} \cup \mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is a triple pattern.
- Given two triple patterns T_1 and T_2, the triples (T_1, P, O), (S, P, T_2), and (T_1, P, T_2) are SPARQL-star triple patterns.

Triple patterns in SPARQL-star queries are combined with operators (e.g., AND, UNION, SELECT) to define SPARQL-star queries.

▶ **Definition 3** (SPARQL-star Syntax). *The syntax of the* SPARQL-star algebra*, whose expressions are called queries, is defined recursively as follows. For each query we also define whether it is* local*,* remote*, or* fully remote*.*

- *An empty basic graph pattern* $\{\}$ *is a local query.*
- *A triple pattern* T *is a local query.*
- *Given an IRI* $y \in Y$*, and a local query* Q'*, the expression* $Q = (\text{SERVICE } y\ Q')$ *is a fully remote query.*
- *Given two queries* Q_1 *and* Q_2*, the expressions* $(Q_1 \text{ AND } Q_2)$*,* $(Q_1 \text{ UNION } Q_2)$*,* $(Q_1 \text{ DIFF } Q_2)$*, and* $(Q_1 \text{ OPTIONAL } Q_2)$ *are queries. If* Q_1 *and* Q_2 *are local, then these four queries are said to be local. If* Q_1 *and* Q_2 *are fully remote, then these four queries are said to be fully remote. Otherwise, if these queries combine a local and a fully remote query, they are said to be remote.*
- *A selection formula consists of a combination of atoms of the form* $A = B$ *and* bound(?x)*, where* $A, B \in \mathbf{I} \cup \mathbf{L}$ *and* $?x \in \mathbf{V}$*, with the logical connectives* $\wedge$*,* $\vee$*, and* $\neg$*. Given a query* Q' *and a selection formula* φ*, the expression* $Q = (Q' \text{ FILTER } \varphi)$ *is a query, which is local if* Q' *is local, fully remote if* Q' *is fully remote, and remote if* Q' *is remote.*
- *Given a query* Q' *and a finite set of variables* W*, the expression* $Q = (\text{SELECT } W \text{ WHERE } Q')$ *is a query, which is local if* Q' *is local, fully remote if* Q' *is fully remote, and remote if* Q' *is remote.*

▶ Note 4. Notice that this definition does not allow a SERVICE clause to include another SERVICE clause. We follow this design because according to the SPARQL 1.1. specification, a service clause can contain a single endpoint.

▶ **Example 5.** Let T_1 and T_2 be two triple patterns, and consider the following three queries:

- $Q_1 = (T_1 \text{ AND } T_2)$
- $Q_2 = (T_1 \text{ AND } (\text{SERVICE } y_2\ T_2))$
- $Q_3 = ((\text{SERVICE } y_1\ T_1) \text{ AND } (\text{SERVICE } y_2\ T_2))$

Query Q_1 is local; query Q_2 is remote but not fully-remote because the triple patterns T_1 does not occur in a remote subquery; and query Q_3 is fully-remote because both triple patterns occur in remote subqueries.

▶ Note 6. The syntax described in Definition 3 follows the syntax introduced by Perez et al. [38], and extended in Geerts et al. [22] with the operator SELECT, and by Buil-Aranda [12] with the operator SERVICE. Unlike them [22, 38], we call the difference operator DIFF because it differs from the standard MINUS operator [6, 34]. The operator DIFF is part of the standard SPARQL 1.1 algebra, and the operator MINUS is part of the standard SPARQL 1.1 query language syntax. Also, the operator MINUS is expressible with the other operators we include in Definition 3 [34]. An additional difference compared to the above works is that we syntactically classify queries in *local*, *remote* and *fully-remote* depending on where the triple patterns are evaluated. In this paper we build upon the how-provenance theoretical framework by Geerts et al. [22] who used the operator DIFF in their annotated SPARQL algebra (but called it MINUS).

▶ Note 7. The syntax of the SERVICE operator that we presented in Definition 3 does not allow variables as service names. We introduce this simplification because this feature is not needed to express execution plans for queries over federations.

A (solution) *mapping* is a partial function $\mu : \mathbf{V} \to (\mathbf{T} \cup \mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ where the domain of μ, denoted by $\text{dom}(\mu)$, is a finite set of variables. We write $\mu_\emptyset$ to denote the solution mapping with an empty domain, called the *empty solution mapping*. Two mappings μ_1 and μ_2 are said to be compatible, denoted $\mu_1 \sim \mu_2$, if $\mu_1(\texttt{?x}) = \mu_2(\texttt{?x})$ for each variable $\texttt{?x} \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$. Given a mapping μ, and a set of variables $W \subseteq \text{dom}(\mu)$, we write $\mu|_W$ to denote the mapping such that $\text{dom}(\mu|_W) = W$ and $\mu|_W \sim \mu$. Given a triple pattern T, and a mapping μ whose domain $\text{dom}(\mu)$ consists of all variables occurring in T, we write $\mu(T)$ to denote the RDF-star triple t resulting from replacing every variable ?x in T with $\mu(\texttt{?x})$. In short, the evaluation of a SPARQL query Q on an RDF dataset D and an RDF-star graph G, is defined as a function $[\![Q]\!]_{D,G}$ that returns a set of mappings. The details of SPARQL evaluation semantics are detailed in [7, 38]. We next present a definition of the semantics of SPARQL-star.

▶ **Definition 8** (SPARQL-star Semantics). *Given a SPARQL query Q, a federated dataset D, and an RDF-star graph G, we write $[\![Q]\!]_{D,G}$ to denote the set of mappings obtained from evaluating Q in D and G, which is defined recursively as follows:*

- *If $\{\}$ is an empty basic graph pattern, then $[\![\{\}]\!]_{D,G}$ is the set with a single mapping $\mu_\emptyset$ such that $\text{dom}(\mu) = \emptyset$.*
- *If T is a triple pattern, then $[\![T]\!]_{D,G}$ is the set of mappings μ such that $\text{dom}(\mu)$ is the set of variables occurring in T and $\mu(T) \in G$.*
- $[\![(\text{SERVICE } y\ Q)]\!]_{D,G} = [\![Q]\!]_{D,D(y)}$.
- *$[\![Q_1 \text{ AND } Q_2]\!]_{D,G}$ is the set of all mappings μ such that $\mu_1 \in [\![Q_1]\!]_{D,G}$ and $\mu_2 \in [\![Q_2]\!]_{D,G}$, $\mu_1 \sim \mu_2$, and $\mu = \mu_1 \cup \mu_2$.*
- *$[\![Q_1 \text{ UNION } Q_2]\!]_{D,G}$ is the set of all mappings μ such that $\mu \in [\![Q_1]\!]_{D,G}$ or $\mu \in [\![Q_2]\!]_{D,G}$.*
- *$[\![Q_1 \text{ DIFF } Q_2]\!]_{D,G}$ is the set of all mappings μ_1 such that $\mu_1 \in [\![Q_1]\!]_{D,G}$ and every mapping $\mu_2 \in [\![Q_2]\!]_{D,G}$ is incompatible with μ_1.*
- *$[\![Q_1 \text{ OPTIONAL } Q_2]\!]_{D,G}$ is the set of all mappings μ such that $\mu \in [\![Q_1 \text{ AND } Q_2]\!]_{D,G}$ or $\mu \in [\![Q_1 \text{ DIFF } Q_2]\!]_{D,G}$.*
- *$[\![Q \text{ FILTER } \varphi]\!]_{D,G}$ is the set of all mappings μ such that $\mu \in [\![Q]\!]_{D,G}$ and φ is a true formula according to mapping μ, denoted $\mu \models \varphi$.*

  We write $\mu \models \varphi$ if the value of the formula φ according to μ, denoted $\varphi[\mu]$ is 1. The value $\varphi[\mu]$ is one the values in $\{0, \frac{1}{2}, 1\}$ defined recursively as follows:
  - *Let φ be the a formula of the form $A = B$. If there is a variable in φ that is not in $\text{dom}(\mu)$, then $\varphi[\mu] = \frac{1}{2}$. Otherwise, the formula $\mu(\varphi)$ resulting from replacing the variables ?x in φ with $\mu(\texttt{?x})$ can have the forms $a = a$ or $a = b$, where a and b are two different values in the set $\mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$ (e.g., if φ is the formula $\texttt{?x} = a$ and $\mu(\texttt{?x}) = a$ then the $\mu(\varphi)$ is $a = a$). If $\mu(\varphi)$ is $a = a$ then $\varphi[\mu] = 1$. Otherwise, if $\mu(\varphi)$ is $a = b$ then $\varphi[\mu] = 0$.*

- *If* φ *has the form* bound(?x)*, then* $\varphi[\mu] = 1$ *if* ?x $\in \text{dom}(\mu)$*. Otherwise,* $\varphi[\mu] = 0$*.*
- $(\neg\varphi)[\mu] = 1 - \varphi[\mu]$*,* $(\varphi \wedge \psi)[\mu] = \min(\varphi[\mu], \psi[\mu])$*, and* $(\varphi \vee \psi)[\mu] = \max(\varphi[\mu], \psi[\mu])$*.*

- $[\![\text{SELECT } W \text{ WHERE } Q]\!]_{D,G}$ *is the set of all mappings* μ *such that there exists a mapping* $\mu' \in [\![Q]\!]_{D,G}$ *with* $\mu = \mu'|_W$*.*

▶ Note 9. It is not difficult to see that if a query Q is fully-remote, then the set $[\![Q]\!]_{D,G}$ does not depend on the graph G. Similarly, if query Q is local, then the set $[\![Q]\!]_{D,G}$ does not depend on the federated dataset D. Following this observation, we will abbreviate the aforementioned set as $[\![Q]\!]_D$ if Q is fully-remote, and as $[\![Q]\!]_G$ if Q is local.

Not all variables occurring in a query necessarily appear on the solutions. The problem of determining which variables can appear and which variables cannot appear is in general undecidable [12], but an approximation can be defined syntactically. A variable is said *in-scope* when it can appear in the solutions and *strongly bound* when it necessarily appears in all the solutions.

▶ **Definition 10** (SPARQL-star in-scope and strongly bound variables)**.** *Given a* SPARQL-star *query* Q*, the sets of variables that are* in-scope *and* strongly bound*, denoted respectively* inScope(Q) *and* stronglyBound(Q)*, are recursively defined as follows.*

- inScope($\{\}$) $= \emptyset$ *and* stronglyBound($\{\}$) $= \emptyset$*.*
- *Given a triple pattern* T*. The sets* inScope(T) *and* stronglyBound(T) *are equal to the set of variables occurring in* T*.*
- *Given the queries* Q_1 *and* Q_2*, the following identities define the in-scope variables in compound queries:*
  - inScope(SERVICE y Q_1) $=$ inScope(Q_1)*,*
  - inScope(Q_1 AND Q_2) $=$ inScope(Q_1) $\cup$ inScope(Q_2)*,*
  - inScope(Q_1 UNION Q_2) $=$ inScope(Q_1) $\cup$ inScope(Q_2)*,*
  - inScope(Q_1 OPTIONAL Q_2) $=$ inScope(Q_1) $\cup$ inScope(Q_2)*,*
  - inScope(Q_1 DIFF Q_2) $=$ inScope(Q_1)*,*
  - inScope(Q_1 FILTER φ) $=$ inScope(Q_1)*,*
  - inScope(SELECT W WHERE Q_1) $= W \cap$ inScope(Q_1)*.*
- *Given the queries* Q_1 *and* Q_2*, the following identities define the strongly bound variables in compound queries:*
  - stronglyBound(SERVICE y Q_1) $=$ stronglyBound(Q_1)
  - stronglyBound(Q_1 AND Q_2) $=$ stronglyBound(Q_1) $\cap$ stronglyBound(Q_2)*,*
  - stronglyBound(Q_1 UNION Q_2) $=$ stronglyBound(Q_1) $\cap$ stronglyBound(Q_2)*,*
  - stronglyBound(Q_1 OPTIONAL Q_2) $=$ stronglyBound(Q_1) $\cap$ stronglyBound(Q_2)*,*
  - stronglyBound(Q_1 DIFF Q_2) $=$ stronglyBound(Q_1)*,*
  - stronglyBound(Q_1 FILTER φ) $=$ stronglyBound(Q_1)*,*
  - stronglyBound(SELECT W WHERE Q_1) $= W \cap$ stronglyBound(Q_1)*.*

▶ Note 11. One may think that ?x $\in$ inScope(Q) implies that there exists one graph where there is a mapping $\mu \in [\![Q]\!]_G$ such that ?x $\in \text{dom}(\mu)$. However, this is not true. Indeed, the query

$$Q = (\{\} \text{ OPTIONAL } ((a, p, \texttt{?x}) \text{ DIFF } (a, p, \texttt{?x})))$$

includes the variable ?x in inScope(Q) and for every graph G, each $\mu \in [\![Q]\!]_G$ holds that ?x $\notin$ dom(μ). One may think something similar regarding the variables in stronglyBound(Q). That is, thinking that every variable ?x that occurs in the domain of all solution mappings of a query Q in every graph G must be a strongly bound. This happens to variable ?x in the query

$$Q' = ((a, p, \texttt{?x}) \text{ UNION } (\{\} \text{ FILTER } \neg(a = a))).$$

However, variable ?x $\notin$ stronglyBound(Q'). The problem of determining if a variable can occur in the domain of all or some of the mappings is in general undecidable [12]. Hence, the notions of inScope and stronglyBound are approximations of these more complex notions. The variables in inScope(Q) are a superset of the variables that occur in some mappings $\mu \in [\![Q]\!]_G$ for every graph G. The variables in stronglyBound(Q) are a subset of the variables that occur in all mappings $\mu \in [\![Q]\!]_G$ for every graph G.

3.3 Federated Query Processing in RDF/SPARQL

Federated querying consists of executing queries across multiple, potentially heterogeneous, RDF data sources, so that they are treated as a single and unified dataset.

Formally [1], a federation F is a pair (E, d) where E is the set of all data sources, and d is function that associates every data source $e \in E$ with an RDF graph. A second evaluation function $[\![\cdot]\!]_F$ is defined in terms of the evaluation function over graphs $[\![\cdot]\!]_G$. For a local SPARQL query Q, $[\![Q]\!]_F = [\![Q]\!]_G$ where the graph G, called the *union graph*, is defined as follows:

$$G = [\![Q]\!]_{\bigcup_{e \in E} d(e)}.$$

This ensures that relationships spanning multiple datasets are considered during evaluation, enabling integrated query processing across distributed datasets.

Notice that query Q is local. That is, Q does not contain SERVICE clauses. However, the federated query processing of Q consists of execution Q over a set of graphs from remote services. Since the SERVICE clause allows for remote query execution, one may think that a federated query processing can be implemented by rewriting the local query Q as a fully remote query. Indeed, the following subsection describes how federated SPARQL query engines decompose the local query Q into multiple queries that are remotely executed by rewriting Q as a single fully remote query that uses the SERVICE for the remote execution.

3.3.1 Data Integration and Query Decomposition

In federated querying, data integration requires handling schema heterogeneity across different RDF graphs. Query decomposition techniques decompose the original SPARQL query into subqueries. Each subquery is evaluated on a subset of the federated endpoints, and the results are merged based on join conditions, compatibility mappings, and downstream algebraic operators.

Queries can be decomposed in different ways. The simpler way is decomposing basic graph patterns into triple patterns to compute unions first and then joins. For each triple pattern in a basic graph pattern, the federated engine generates an intermediate relation that consists of the union of the results obtained from evaluating the triple pattern in all the data sources. These relations are then joined to obtain the results of the triple pattern.

▶ **Example 12.** Let Q be the following query asking for the cities of Germany that are not crossed by the Rhine river:

$$Q = (\text{SELECT } \{\texttt{?city}\} \text{ WHERE } (((\texttt{?city}, \mathsf{a}, \mathsf{City}) \text{ AND } (\texttt{?city}, \mathsf{country}, \mathsf{Germany})) \text{ DIFF } (\mathsf{Rhine}, \mathsf{crosses}, \texttt{?city}))).$$

To answer this query over a federated dataset $D = \{y_1 \mapsto G_1, y_2 \mapsto G_2\}$, we can first answer the following triple patterns over the federation:

$T_1 = (\texttt{?city}, \mathsf{a}, \mathsf{City})$,

$T_2 = (\texttt{?city}, \mathsf{country}, \mathsf{Germany})$,

$T_3 = (\mathsf{Rhine}, \mathsf{crosses}, \texttt{?city})$.

The federated computation of a triple pattern can be done by computing the triple on each SPARQL endpoint, and then computing the union of the results. For example,

$$[\![T_1]\!]_{G_1 \cup G_2} = [\![T_1]\!]_{G_1} \cup [\![T_1]\!]_{G_2}.$$

This union can be expressed as a query execution over the RDF-star dataset D with the SPARQL-star operator SERVICE and UNION.

$$[\![T_1]\!]_{G_1 \cup G_2} = [\![((\text{SERVICE } y_2\ T_1))\ \text{UNION}\ ((\text{SERVICE } y_1\ T_1))]\!]_D.$$

Following this approach, the whole federated execution of the query can be done by evaluating another query Q' over the federated dataset D (i.e., $[\![Q]\!]_{G_1 \cup G_2} = [\![Q']\!]_D$). Such a query Q' is the following:

$$\begin{aligned} Q' = (\text{SELECT}\ \{\texttt{?city}\}\ \text{WHERE}\ (((((&\text{SERVICE } y_2\ T_1))\ \text{UNION}\ ((\text{SERVICE } y_1\ T_1)))\ \text{AND} \\ &(((\text{SERVICE } y_2\ T_2))\ \text{UNION}\ ((\text{SERVICE } y_1\ T_2))))\ \text{DIFF} \\ &(((\text{SERVICE } y_2\ T_3))\ \text{UNION}\ ((\text{SERVICE } y_1\ T_3))))). \end{aligned}$$

Schwarte et al. [42] proposed a query decomposition that checks if two or more triple patterns are exclusive to one of the sources in the federation. If this is the case, they evaluate the basic graph pattern on that specific data source. The evaluation of a basic graph pattern in a single data source reduces the data transfer across the network required by the simple unions-first-joins-later approach.

▶ **Example 13.** Consider the queries Q and Q', and the federated dataset D described in Example 12. If we know that $[\![T_1]\!]_{G_2}$, $[\![T_2]\!]_{G_2}$, and $[\![T_3]\!]_{G_1}$ are empty, then the result of evaluating Q' in the federation will be equal to the result of evaluating the following query Q'' on D:

$$Q'' = (\text{SELECT}\ \{\texttt{?city}\}\ \text{WHERE}\ (((\text{SERVICE } y_1\ (T_1\ \text{AND}\ T_2)))\ \text{DIFF}\ ((\text{SERVICE } y_2\ T_3)))).$$

Aimonier-Davat et al. [3] proposed a joins-first-unions-later query decomposition. They noticed that, often, only a few combinations of sources return results. Following this observation, they decompose queries using into joins of answers to triple patterns. The union of these joins is then computed to obtain the answers of a basic graph pattern. Although, in general, the efficiency depends on the federation, Aimonier-Davat et al. showed that their decomposition outperforms others in some large-scale federations.

3.4 How-provenance in SPARQL

3.4.1 Semirings

A *commutative monoid* $\mathcal{M}$ is an algebraic structure $(M, +_{\mathcal{M}}, 0_{\mathcal{M}})$ such that $M \neq \emptyset$ is a set closed under a commutative and associate binary operation $+_{\mathcal{M}}$. The element $0_{\mathcal{M}}$ is the identity operand for $+_{\mathcal{M}}$. Given two commutative monoids $(K, +_{\mathcal{K}}, 0_{\mathcal{K}})$ and $(K, \times_{\mathcal{K}}, 1_{\mathcal{K}})$ such that $\times_{\mathcal{K}}$ is distributive over $+_{\mathcal{K}}$, and $0_{\mathcal{K}} \times_{\mathcal{K}} x = 0_{\mathcal{K}}$ (for every $x \in K$), we call the structure $\mathcal{K} = (K, +_{\mathcal{K}}, \times_{\mathcal{K}}, 0_{\mathcal{K}}, 1_{\mathcal{K}})$ a *commutative semiring*. An *spm-semiring* $(K, +_{\mathcal{K}}, \times_{\mathcal{K}}, -_{\mathcal{K}}, 0_{\mathcal{K}}, 1_{\mathcal{K}})$ extends a commutative semiring with a minus operation $-_{\mathcal{K}}$. This operator follows a set of axioms that allows us to model non-monotonic operations such as the relational difference. For more details about spm-semirings, we refer the reader to [22]. The algebraic expressions within an spm-semiring are used to annotate query solutions. To see how, we need to introduce the concepts of $\mathcal{K}$-relations and $\mathcal{K}$-graphs.

3.4.2 $\mathcal{K}$-relations and $\mathcal{K}$-graphs

Given a set A and an spm-semiring $\mathcal{K} = (K, +_\mathcal{K}, \times_\mathcal{K}, -_\mathcal{K}, 0_\mathcal{K}, 1_\mathcal{K})$, a function $f : A \to K$ is called a $\mathcal{K}$-*set* over A. The set $\text{supp}(f) = \{a \in A \mid f(a) \neq 0_\mathcal{K}\}$ is called the support of f. For every element $a \in A$, we call $f(a)$ the $\mathcal{K}$-*value* of a in f. If f has a finite support $\{a_1, \ldots, a_n\}$, we write $f = \{\!\!\{a_1 \mapsto k_1, \ldots, a_n \mapsto k_n\}\!\!\}$ to denote that $f(a_i) = k_i$, for $1 \leq i \leq n$.

Intuitively, $\mathcal{K}$-sets annotate the elements of a set A with values in the structure. By assuming that $\mathcal{K}$ is the structure of provenance annotations, this formalism is used to represent provenance of answer to queries or statements in an RDF graph. A $\mathcal{K}$-set Ω over the set of all possible SPARQL mappings is called a $\mathcal{K}$-relation, and a $\mathcal{K}$-set Γ over all possible RDF triples is called a $\mathcal{K}$-*graph*.

▶ Note 14. Note that a $\mathcal{K}$-set is a total function $f : A \to K$. The double braces in the notation $f = \{\!\!\{a_1 \mapsto k_1, \ldots, a_n \mapsto k_n\}\!\!\}$ tell us that the function f is not only defined over the set $\{a_1, \ldots, a_n\}$, which is the support of f, but over the whole set A. This notation omits the elements that are not in the support of f because we already know their values are $0_\mathcal{K}$.

▶ **Example 15.** Let $\mathcal{K} = (K, \oplus, \otimes, \ominus, 0, 1)$ be an spm-semiring with $K = \{u_1, u_2, u_3\}$, and let G and Q be the following RDF-star graph and query:

$$G = \{\ ((\mathsf{Alice}, \mathsf{likes}, \mathsf{pasta}), \mathsf{wasDerivedFrom}, u_1),\\ ((\mathsf{Alice}, \mathsf{likes}, \mathsf{pasta}), \mathsf{wasDerivedFrom}, u_2),\\ ((\mathsf{Alice}, \mathsf{livesIn}, \mathsf{Italy}), \mathsf{wasDerivedFrom}, u_3)\ \},$$

$$Q = (\mathsf{?x}, \mathsf{likes}, \mathsf{pasta})\ \text{AND}\ (\mathsf{?x}, \mathsf{livesIn}, \mathsf{Italy}).$$

It is easy to see that the predicate *wasDerivedFrom* defines a $\mathcal{K}$-graph Γ where the first and last triples are associated to the elements $u_1 \oplus u_2$ and u_3, and that $\mu = \{\mathsf{?x} \mapsto \mathsf{Alice}\}$ is a solution mapping for Q. The $\mathcal{K}$-relation $\Omega = \{\!\!\{\mu \mapsto (u_1 \oplus u_2) \otimes u_3\}\!\!\}$ associates Q's solutions to provenance annotation. Even when the domain of Ω is an infinite set of solution mappings, only the non-zero element for μ is needed to express the function Ω. This how-provenance annotation tells us that Alice is a query solution for Q as long as the triple identified by u_3 is present in conjunction with either the triples u_1 or u_2.

▶ **Definition 16.** *Given an spm-semiring $\mathcal{K}$, a SPARQL query Q consisting of a combination of triple patterns with the operators* AND, UNION, DIFF, FILTER, OPTIONAL *and* SELECT, *and a $\mathcal{K}$-graph Γ, we write $(\!|Q|\!)_\Gamma$ to denote the $\mathcal{K}$-relation obtained from evaluating Q in Γ, which is defined recursively for an arbitrary mapping μ as follows:*

- $(\!|\{\}|\!)_\Gamma(\mu) = 1$ *if* $\text{dom}(\mu) = \emptyset$, *and* $(\!|\{\}|\!)_\Gamma(\mu) = 0$ *if* $\text{dom}(\mu) \neq \emptyset$,
- $(\!|(s, p, o)|\!)_\Gamma(\mu) = \Gamma(\mu(s, p, o))$,
- $(\!|\text{SELECT}\ W\ \text{WHERE}\ Q|\!)_\Gamma(\mu) = \sum_{\mu' : \mu'|_W = \mu} (\!|Q|\!)_\Gamma(\mu')$,
- $(\!|Q\ \text{FILTER}\ \varphi|\!)_\Gamma(\mu) = (\!|Q|\!)_\Gamma(\mu) \times_\mathcal{K} 1_{\mu \models \varphi}$, *where* $1_{\mu \models \varphi} = 1$ *if* $\mu \models \varphi$ *and* $1_{\mu \models \varphi} = 0$, *otherwise*,
- $(\!|Q_1\ \text{UNION}\ Q_2|\!)_\Gamma(\mu) = (\!|Q_1|\!)_\Gamma(\mu) +_\mathcal{K} (\!|Q_2|\!)_\Gamma(\mu)$,
- $(\!|Q_1\ \text{AND}\ Q_2|\!)_\Gamma(\mu) = \sum_{\mu = \mu_1 \cup \mu_2} ((\!|Q_1|\!)_\Gamma(\mu_1) \times_\mathcal{K} (\!|Q_2|\!)_\Gamma(\mu_2))$,
- $(\!|Q_1\ \text{DIFF}\ Q_2|\!)_\Gamma(\mu) = (\!|Q_1|\!)_\Gamma(\mu) -_\mathcal{K} (\sum_{\mu' \sim \mu} (\!|Q_2|\!)_\Gamma(\mu'))$,
- $(\!|Q_1\ \text{OPTIONAL}\ Q_2|\!)_\Gamma(\mu) = (\!|Q_1\ \text{AND}\ Q_2|\!)_\Gamma(\mu) +_\mathcal{K} (\!|Q_1\ \text{DIFF}\ Q_2|\!)_\Gamma(\mu)$,

where $\sum$ denotes sums using the operation $+_\mathcal{K}$.

3.4.3 How-provenance polynomials

In Example 15 we presented how-provenance annotations. We next describe how a universal spm-semiring *generalizes* all the other spm-semirings for the annotated SPARQL algebra. To understand what is meant by the term "generalize", we next introduce the notions of *spm-homomorphism* and *commutation with homorphisms*.

▶ **Definition 17** (spm-homomorphism). *An* spm-homomorphism *between two spm-semirings* $\mathcal{K}_1 = (K, +_{\mathcal{K}_1}, \times_{\mathcal{K}_1}, -_{\mathcal{K}_1}, 0_{\mathcal{K}_1}, 1_{\mathcal{K}_1})$ *and* $\mathcal{K}_2 = (K, +_{\mathcal{K}_2}, \times_{\mathcal{K}_2}, -_{\mathcal{K}_2}, 0_{\mathcal{K}_2}, 1_{\mathcal{K}_2})$*, is a function* $h : K_1 \to K_2$ *that preserves the identities and the operations. That is:*

- $h(0_{\mathcal{K}_1}) = 0_{\mathcal{K}_2}$,
- $h(1_{\mathcal{K}_1}) = 1_{\mathcal{K}_2}$,
- $h(a +_{\mathcal{K}_1} b) = h(a) +_{\mathcal{K}_2} h(b)$,
- $h(a \times_{\mathcal{K}_1} b) = h(a) \times_{\mathcal{K}_2} h(b)$,
- $h(a -_{\mathcal{K}_1} b) = h(a) -_{\mathcal{K}_2} h(b)$.

▶ **Example 18.** Consider the following spm-semirings:

- The spm-semiring of natural numbers $(\mathbb{N}, +, \cdot, -, 0, 1)$, where $+$ and $\cdot$ are the usual sum and product of natural numbers, and $a - b = a$ if $b = 0$ and $a - b = 0$ if $b \neq 0$.
- The spm-semiring of boolean values $(\mathbb{B}, \vee, \wedge, \nrightarrow, 0, 1)$, where $\vee$, $\wedge$, and $\nrightarrow$ are the Boolean disjunction, conjunction and material nonimplication.

The function $h : \mathbb{N} \to \mathbb{B}$, defined as $h(k) = \min(1, k)$, is an spm-homomorphism.

So far, we have described when an spm-semiring is more general than another spm-semiring. Geerts et al. [22] constructed the most general semiring, called the universal spm-semiring. To define the universal spm-semiring, let $X = \{x_1, \ldots, x_n\}$ be a set of variables, and $\text{mon}(X)$ be the set of monomials over X. Then, let B_X and $\bar{B}_X$ be two sets disjoint from X, defined as follows: $B_X = \{b_\mu \mid \mu \in \text{mon}(X)\}$ and $\bar{B}_X = \{\bar{b}_\mu \mid \mu \in \text{mon}(X)\}$. We abbreviate the set $X \cup B_X \cup \bar{B}_X$ as X_B, and we write $\mathbb{N}[X_B]$ for the set of polynomials with coefficients in $\mathbb{N}$ and variables X_B.

We write $\cong$ to denote the congruence relation between polynomials in $\mathbb{N}[X_B]$. That is, if $p[X_B] \cong q[X_B]$ and $p'[X_B] \cong q'[X_B]$, then $p[X_B] + p'[X_B] \cong q[X_B] + q'[X_B]$ and $p[X_B] \cdot p'[X_B] \cong q[X_B] \cdot q'[X_B]$. In addition, $\cong$ is assumed to be the minimum congruence such that for all monomials $\mu \in \text{mon}(X)$, $b_\mu \cdot \bar{b}_\mu \cong 0$, $b_\mu + \bar{b}_\mu \cong 1$, $b_\mu + b_\mu \cong b_\mu$ and $\bar{b}_\mu + \bar{b}_\mu \cong \bar{b}_\mu$, $\bar{b}_m \cdot \mu \cong 0$, and $b_\mu \cdot b_\nu \cong b_\mu$ whenever $\mu = \nu \cdot \nu'$ for some monomial ν'. Since these last entities characterize the elements of B_X and $\bar{B}_X$ as Booleans, the quotient semiring of $\mathbb{N}[X_B]$ with respect to $\cong$ is called the *semiring of boolynomials*. Abusing of notation, we write $(\mathbb{N}_{\cong}[X_B], +, \cdot, [0], [1])$ for this quotient semiring, where the elements of $\mathbb{N}_{\cong}[X_B]$ are the equivalence classes $[p[X_B]] = \{p'[X_B] \mid p[X_B] \cong p'[X_B]\}$, and the operations are $[p[X_B]] + [q[X_B]] = [p[X_B] + q[X_B]]$ and $[p[X_B]] \cdot [q[X_B]] = [p[X_B] \cdot q[X_B]]$.

The semiring of boolynomials lacks of the difference operator. Geerts et al. [22] propose the following difference: $[p[X_B]] - [q[X_B]] = [p[X_B]] \cdot [p_q]$. To see the formal definition of boolynomial p_q see Geerts et al. [22] work. Intuitively, the boolynomial p_q should contain monomials $\bar{b}_\mu$ such that the product eliminates the monomials μ from boolynomial $p[X_B]$. Geerts et al. [22] proved that such a structure $(\mathbb{N}_{\cong}[X_B], +, \cdot, -, [0], [1])$ is the universal spm-semiring.

In what follows, we write $\text{ProvExp}(X)$ to denote the set of all the expressions generated with set of variables X, the operator symbols $\oplus$, $\otimes$, and $\ominus$, and the constants 0 and 1. The semantics of these expressions is given by associating every expression in $\text{ProvExp}(X)$ to an element in $\mathbb{N}_{\cong}[X_B]$ by mapping every variable $x \in X$ to an equivalence class $[X]$, the constants 0 and 1 with the equivalence classes $[0]$ and $[1]$, and the operators $\oplus$, $\otimes$, and $\ominus$ with the operators $+$, $\cdot$, and $-$ of the universal spm-semiring. The expressions computed by the methods proposed by Hernández et al. [30] and Asma et al. [9] represent thus elements in the universal spm-semiring.

We write $\text{ProvExp}_{\cong}(X)$ for the set of equivalence classes $[e]$ where $e \in \text{ProvExp}(X)$ and $e' \in [e]$ if and only if e and e' are associated to the same element in $\mathbb{N}_{\cong}[X_B]$. The structure $(\text{ProvExp}_{\cong}(X), \oplus, \otimes, \ominus, [0], [1])$ whose operators are $[a] \oplus [b] = [a \oplus b]$, $[a] \otimes [b] = [a \otimes b]$, and $[a] \ominus [b] = [a \ominus b]$ is thus an spm-semiring of equivalence classes of expressions representing elements in the universal spm-semiring.

▶ **Example 19.** Consider the spm-semiring $\text{ProvExp}_{\cong}(X)$, an arbitrary spm-semiring $\mathcal{K} = (K, +_{\mathcal{K}}, \times_{\mathcal{K}}, -_{\mathcal{K}}, 0_{\mathcal{K}}, 1_{\mathcal{K}})$, and a function $g : X \to K$. Let $h : \text{ProvExp}_{\cong}(X) \to \mathcal{K}$ be the function defined recursively as follows:

- $h([x]) = g(x)$ if $x \in X$.
- $h([0]) = 0_{\mathcal{K}}$ and $h([1]) = 1_{\mathcal{K}}$.
- $h([a \oplus b]) = h([a]) +_{\mathcal{K}} h([b])$, $h([a \otimes b]) = h([a]) \times_{\mathcal{K}} h([b])$, and $h([a \ominus b]) = h([a]) -_{\mathcal{K}} h([b])$.

The function h is an spm-homomorphism.

For readability, in what follows we will omit the brackets for the elements of $\text{ProvExp}_{\cong}(X)$. For example, we will write that a triple in a $\text{ProvExp}_{\cong}(X)$-graph is annotated with a variable x to indicate that it is annotated with the element $[x]$ in the spm-semiring over set $\text{ProvExp}_{\cong}(X)$.

▶ **Definition 20** (Commutation with homomorphisms)**.** *Given two spm-semirings $\mathcal{K}_1$ and $\mathcal{K}_2$. We say that the $\mathcal{K}_1$- and $\mathcal{K}_2$-annotated SPARQL algebras* commute with homomorphisms *if and only if for every spm-homomorphism $h : \mathcal{K}_1 \to \mathcal{K}_2$, every SPARQL query Q and every $\mathcal{K}_1$-graph Γ_1, it holds that $(\!|Q|\!)_{\Gamma_1} \circ h = (\!|Q|\!)_{\Gamma_1 \circ h}$, where $\circ$ denotes the composition of functions.*

▶ **Example 21.** Let Γ_1 be the $\text{ProvExp}_{\cong}(X)$-graph whose support has two triples annotated as follows: $\{\!|(\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}) \mapsto x_1, (\mathsf{Danube}, \mathsf{crosses}, \mathsf{Budapest}) \mapsto x_2|\!\}$, where x_1 and x_2 are elements in set X. Let $g : X \to \mathbb{N}$ be the function such that $g(x_1) = 3$ and $g(x_2) = 5$, and $h : \text{ProvExp}_{\cong}(X) \to \mathbb{N}$ be the spm-homomorphism defined on top of function g as it is described in Example 19. Then, $\Gamma_2 = \Gamma_1 \circ h$ is the $\mathbb{N}$-graph $\{\!|(\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}) \mapsto 3, (\mathsf{Danube}, \mathsf{crosses}, \mathsf{Budapest}) \mapsto 5|\!\}$. Let Q be the query (SELECT `?river` WHERE ((`?river`, crosses, `?city`))), asking for rivers crossing cities. Then, $(\!|Q|\!)_{\Gamma_1}$ is the $\mathbb{N}$-relation $\{\!|\mathsf{Danube} \mapsto x_1 \oplus x_2|\!\}$. Then $(\!|Q|\!)_{\Gamma_1} \circ h$ is the $\mathbb{N}$-relation $\{\!|\mathsf{Danube} \mapsto 8|\!\}$, which is equal to $(\!|Q|\!)_{\Gamma_2}$.

▶ Remark 22. Example 21 illustrates the generality of how-provenance polynomials. It is well-known that every pair of SPARQL algebras commute with homomorphisms. Also, by using the spm-homomorphism like the one described in Example 19, we can use how-provenance polynomials to symbolically operate elements of other spm-semirings. Thus, the how-provenance polynomials generated with the method we propose in the next section, NPCS, can be used in downstream tasks over other spm-semirings.

4 Provenance computation with NPCS

Having introduced all the preliminary concepts in the previous section, we now introduce our solution to compute how-provenance annotations for query solutions delivered by a single source. Section 5 will show how to port this method to a federated setting.

Given a finite set $X \subset \mathbf{I}$, an X-graph Γ, its corresponding RDF-star graph G, and a SPARQL query Q_O, NPCS rewrites Q_O into Q_R so that $[\![Q_R]\!]_G$ is a set of mappings with an additional variable that encodes the provenance polynomials. Thus, the output of Q_R represents the $\text{ProvExp}_{\cong}(X)$-relation $[\![Q_O]\!]_\Gamma$ that maps each solution to a how-provenance polynomial explanation. Those polynomials lie in an spm-semiring $(\text{ProvExp}_{\cong}(X), \oplus, \otimes, \ominus, 0, 1)$, where X is the set of triple identifiers in G. We highlight that computing provenance assumes that the triples in the graph are reified, i.e., identified. RDF-star is a natural way to do it, but as shown later, our approach supports any reification scheme for RDF data. The query rewriting process of NPCS is depicted in Figure 1.

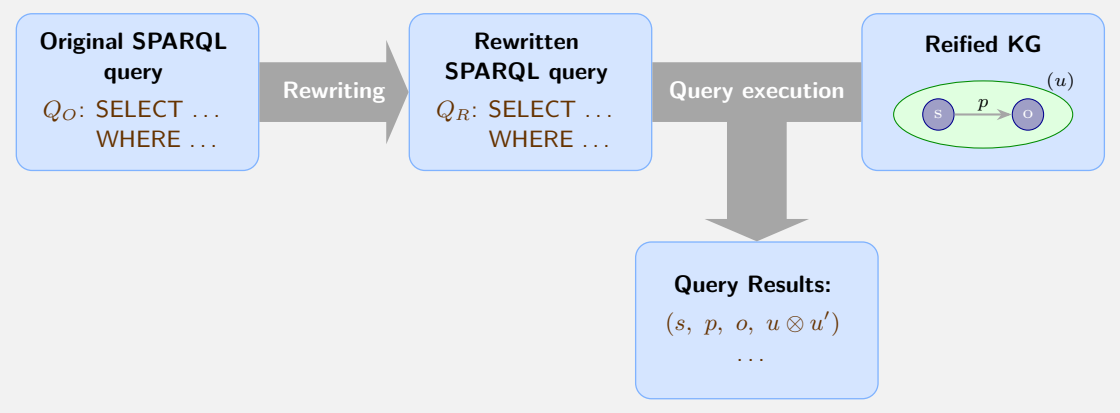


Figure 1 The query rewriting process for NPCS.

4.1 Our Query Rewriting in a Nutshell

Consider the graphs Γ and G and a query Q asking for people who like pasta and live in Italy (see Example 15). For pedagogical reasons, we rewrite our query Q as (P_1 AND P_2), where P_1 is the triple pattern (?x, likes, pasta) and P_2 is the triple pattern (?x, livesIn, Italy). We recall that our goal is to return the following result set:

$$\left[\begin{array}{c|c} \texttt{?x} & \texttt{?prov} \\ \hline \mathsf{Alice} & (u_1 \oplus u_2) \otimes u_3 \end{array}\right].$$

The column labeled ?prov stores the how-provenance of each of the query solutions. To compute such an expression, our strategy must rewrite the query such that the rewritten query retrieves the identifiers of the triples that match each of the triple patterns. However, this is not enough. For instance, the triple pattern $P_1 = (\texttt{?x}, \mathsf{likes}, \mathsf{pasta})$ has two matches, i.e., u_1 and u_2, that must be *grouped* into the expression $(u_1 \oplus u_2)$. This term tells us that the presence of at least one of those sources guarantees the inclusion of Alice in the result set. Finally, the groups extracted from each of the triple patterns must be combined with the $\otimes$ operator that explains the semantics of AND. We argue that to obtain the left-hand term of the product we can rewrite P_1 into the following sub-query:

$$\begin{array}{lll} P_1' = (\text{SELECT} & \texttt{?x}\ (\text{ProvAggSum}(\texttt{?prov}\oplus\otimes 1\oplus\odot)\ \text{AS}\ \texttt{?prov}\oplus\otimes 1) \\ \quad \text{WHERE} & \mathsf{Reify}((\texttt{?x}, \mathsf{likes}, \mathsf{pasta}), \texttt{?prov}\oplus\otimes 1\oplus\odot) \\ \quad \text{GROUP BY} & \texttt{?x}), \end{array}$$

▶ Note 23. For readability, we use the symbols $\oplus$ and $\otimes$ in the variables. Actually, NPCS writes the variable ?prov$\oplus\otimes$1 as ?provSumProdOne. These symbols are just a convention we follow to create fresh variables and avoid clashes between variable names.

The Reify function rewrites a triple pattern so that it matches the reification scheme used to encode the X-graph Γ as an RDF-star graph G. In our running example, the Reify expression is a shortcut for

$$((\texttt{?x}, \mathsf{likes}, \mathsf{pasta}), \mathsf{wasDerivedFrom}, \texttt{?prov}\oplus\otimes 1\oplus\odot).$$

The intermediate variable ?prov$\oplus\otimes$1$\oplus\odot$ is introduced to capture the identifiers of the triples that match P_1, whereas variable ?prov$\oplus\otimes$1 groups all the triple identifiers associated to a query solution, which explains the group clause on ?x. The function ProvAggSum, later explained, combines the different sources into a summation with the operator $\oplus$.

The signs $\oplus$, $\otimes$, and $\odot$ in the intermediate variable names are strings used to produce new variable names that do not clash with the original query variables. The names of the variables encode the different steps of the construction of the annotations. For instance, the sign $\odot$ at the end of a variable name tells us that the variable's bindings are triple identifiers. If this is preceded by a $\oplus$ sign, then those bindings will eventually be grouped into a summation by a subsequent step. Those results will be stored in a variable with the same prefix but without the suffix $\oplus\odot$. Furthermore, the $\otimes 1$ sign tells us that our results correspond to the first operand of a join operation, namely AND in SPARQL. If we apply the same logic to P_2 our rewriting for Q takes the following form:

$$
\begin{array}{lll}
Q' = (\ \text{SELECT} & \texttt{?x}\ (\text{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\
\quad \text{WHERE} & ((P_1'\ \text{AND}\ P_2')\ \text{BIND}\ (\text{ProvProd}(\texttt{?prov}\oplus\otimes 1, \texttt{?prov}\oplus\otimes 2)\ \text{AS}\ \texttt{?prov}\oplus)) \\
\quad \text{GROUP BY} & \texttt{?x}\ ).
\end{array}
$$

The operation ProvProd combines the expressions derived from the product's operands. We resort again to ProvAggSum to sum up all the ways to produce a solution mapping from a join operation. The operators ProvAggSum and ProvProd are defined in terms of the built-in SPARQL functions as follows.

▶ **Definition 24.** *Let* $\texttt{?x}$, $\texttt{?x}_1, \ldots,$ $\texttt{?x}_n$ *be variables. Then, we define the following SPARQL operators based on the built-in SPARQL functions* `concat` *and* `aggregate_concat`*:*

$$
\begin{array}{l}
\text{ProvAggSum}(\texttt{?x}) = \texttt{concat}(\ \texttt{"(}\oplus\texttt{"},\ \texttt{aggregate\_concat}(\texttt{?x}),\ \texttt{")"}\ ), \\
\text{ProvProd}(\texttt{?x}_1, \ldots, \texttt{?x}_n) = \texttt{concat}(\ \texttt{"(}\otimes\texttt{"},\ \texttt{?x}_1, \ldots, \texttt{?x}_n,\ \texttt{")"}\ ), \\
\text{ProvDiff}(\texttt{?x}_1, \texttt{?x}_2) = \texttt{concat}(\ \texttt{"(}\ominus\texttt{"},\ \texttt{?x}_1, \texttt{?x}_2,\ \texttt{")"}\ ).
\end{array}
$$

▶ **Example 25.** The expression ProvProd($\texttt{?prov}\oplus\otimes 1, \texttt{?prov}\oplus\otimes 2$) in the query Q' of our running example is evaluated against the answers of the query (P_1' AND P_2'), which are described in the following result set:

?x	?prov⊕⊗1	?prov⊕⊗2
Alice	(u_1)	(u_3)
Alice	(u_2)	(u_3)

Then, this expression generates a third provenance column for the variable $\texttt{?prov}\oplus$:

?x	?prov⊕⊗1	?prov⊕⊗2	?prov⊕
Alice	(u_1)	(u_3)	$(\otimes(u_1)(u_3))$
Alice	(u_2)	(u_3)	$(\otimes(u_2)(u_3))$

By aggregating these two results with the expression ProvAggSum($\texttt{?prov}\oplus$) we obtain a final provenance value annotated in variable `?prov`:

?x	?prov
Alice	$(\oplus(\otimes(u_1)(u_3))(\otimes(u_2)(u_3)))$

▶ Note 26. The expression $(\oplus(\otimes(u_1)(u_3))(\otimes(u_2)(u_3)))$ obtained in Example 25 is a how-provenance polynomial written in Polish notation, and is equivalent to the aforementioned how-provenance polynomial $(u_1 \oplus u_2) \otimes u_3$ we want to generate. We use Polish notation because it integrates seamlessly with the built-in SPARQL function `aggregate_concat`.

▶ Note 27. In this section we present a query rewriting from the SPARQL fragment described in Definition 16 to a SPARQL fragment that include operations that are not present in Definition 16. For example, the operation BIND. The rational of this discrepancy between both SPARQL fragments is that we assume we can use whatever we have at hand in an standard SPARQL engine to compute the how-provenance of a SPARQL query in a fragment in Definition 16, which corresponds to the one studied by Geerts et al. [22]. We assume that the reader is familiar with SPARQL, and we include the formal definitions of these additional operations in the proofs of the correctness of the method proposed in this paper.

4.2 Base Rewriting Rules

Having provided the intuition behind our query rewriting in Section 4.1, we now introduce our rewriting rules for arbitrary SPARQL queries.

▶ **Definition 28** (Base SPARQL-star query rewriting). *Let Q be a local SPARQL query,* `?prov` *a variable, and* Reify *a reification scheme. Then, the rewritten query for Q and variable* `?prov` *over scheme* Reify*, denoted $\beta(Q, \texttt{?prov})$, is defined recursively as follows:*

1. *If Q is an empty basic graph pattern, then $\beta(Q, \texttt{?prov})$ is the query* $(\{\}$ BIND $(1$ AS `?prov`$))$.
2. *If Q is a triple pattern (s, p, o), then $\beta(Q, \texttt{?prov})$ is the query*

$$\begin{array}{ll}(\text{SELECT} & \text{inScope}(Q)\ (\text{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\ \text{WHERE} & \text{Reify}((s,p,o), \texttt{?prov}\oplus) \\ \text{GROUP BY} & \text{inScope}(Q)\ ).\end{array}$$

3. *If Q is $(Q_1$ AND $Q_2)$, then $\beta(Q, \texttt{?prov})$ is the query*

$$\begin{array}{ll}(\text{SELECT} & \text{inScope}(Q)\ (\text{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\ \text{WHERE} & (\beta(Q_1, \texttt{?prov}\oplus\otimes 1)\ \text{AND}\ \beta(Q_2, \texttt{?prov}\oplus\otimes 2)) \\ & \text{BIND}\ (\text{ProvProd}(\texttt{?prov}\oplus\otimes 1, \texttt{?prov}\oplus\otimes 2)\ \text{AS}\ \texttt{?prov}\oplus) \\ \text{GROUP BY} & \text{inScope}(Q)\ ).\end{array}$$

4. *If Q is $(P_1$ UNION $P_2)$, then $\beta(Q, \texttt{?prov})$ is the query*

$$\begin{array}{ll}(\text{SELECT} & \text{inScope}(Q)\ (\text{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\ \text{WHERE} & (\beta(Q_1, \texttt{?prov}\oplus)\ \text{UNION}\ \beta(Q_2, \texttt{?prov}\oplus)) \\ \text{GROUP BY} & \text{inScope}(Q)\ ).\end{array}$$

5. *If Q is $(Q_1$ DIFF $Q_2)$, then let ν a variable substitution that substitutes with fresh variables the variables in $\text{dom}(Q_1) \cap \text{dom}(Q_2)$ that are not strongly bound in query Q_1. Then, $\beta(Q, \texttt{?prov})$ is the query*

$$\begin{array}{ll}(\text{SELECT} & \text{inScope}(Q)\ (\text{ProvDiff}(\texttt{?prov}\ominus 1, \text{ProvAggSum}(\texttt{?prov}\ominus 2\oplus))\ \text{AS}\ \texttt{?prov}) \\ \text{WHERE} & (\beta(Q_1, \texttt{?prov}\ominus 1)\ \text{OPTIONAL}_{C_\nu}\ \beta(\nu(Q_2), \texttt{?prov}\ominus 2\oplus)) \\ \text{GROUP BY} & \text{inScope}(Q) \cup \{\texttt{?prov}\ominus 1\}\ ).\end{array}$$

6. *If Q is* (SELECT W WHERE Q')*, then $\beta(Q, \texttt{?prov})$ is the query*

$$\begin{array}{ll}(\text{SELECT} & W\ (\text{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\ \text{WHERE} & \beta(Q', \texttt{?prov}\oplus) \\ \text{GROUP BY} & W\ ).\end{array}$$

7. *If Q is $(Q'$ FILTER $\varphi)$, then $\beta(Q, \texttt{?prov})$ is the query $(\beta(Q', \texttt{?prov})$ FILTER $\varphi)$.*

▶ Note 29. The functions inScope, ProvAggSum, ProvDiff, ProvProd, and Reify are not SPARQL algebra operators (like AND and UNION), but are used to define actual SPARQL expressions. For example, inScope(Q) must be replaced by the set of variables that are in scope of query Q, and ProvAggSum(?prov$\oplus$) must be replaced by the actual SPARQL expression according to Definition 24.

▶ Note 30. We omit the rewriting rule for the OPTIONAL operator because this operator can be written in terms of AND, UNION and DIFF. To be precise, the query P_1 OPTIONAL P_2 is equivalent to the query (P_1 DIFF P_2) UNION (P_1 AND P_2).

▶ Note 31. Since the DIFF operation requires tracking the provenance of both operands, DIFF translates to an OPTIONAL operation, more precisely, to an OPTIONAL$_{C_v}$ operation, which extends OPTIONAL by renaming variables in the optional pattern with fresh ones. This renaming discards undesired bindings produced in the subtrahend while tracking the provenance. For example, consider the patterns P_1(?x, ?y) and P_2(?x, ?y, ?z), whose in-scope variables are indicated in the parenthesis, and let $\mu_1 = \{?x \mapsto a\}$ and $\mu_2 = \{?x \mapsto a, ?y \mapsto b, ?z \mapsto c\}$ be two solutions for the patterns P_1 and P_2. Our goal is to design an operation that returns a mapping with all variable bindings for variables ?x and ?y from pattern P_1 but excluding variable bindings from pattern P_2. The challenge is that it does not suffices simply discarding the variable ?z, which occurs only in pattern P_2, but also considering the variable ?y, which is unbound in pattern P_1. If instead of OPTIONAL$_{C_v}$, the rule for DIFF (rule 5) used OPTIONAL, the query would return the provenance for the mapping $\{?x \mapsto a, ?y \mapsto b\}$ instead of the mapping μ_1. That we would returned a value for variable ?y that is bound in pattern P_2 but not in pattern P_1. Instead, if ?x is strongly bound for P_1 (i.e., always bound in its answers) and ?y is not, then the operation P_1 OPTIONAL$_{C_v}$ P_2 is:

$$((P(?x, ?y) \text{ OPTIONAL } P(?x, ?u, ?z)) \\ \text{FILTER } (\neg \text{bound}(?y) \vee \neg \text{bound}(?u) \vee ?y = ?u)).$$

where $?u = \nu(?y)$ is a fresh variable introduced by the variable renaming function $\nu : \mathbf{V} \to \mathbf{V}$. The result of this operation for the aforementioned mappings is the mapping $\{?x \mapsto a, ?u \mapsto b, ?z \mapsto c\}$. Since variables ?u and ?z are not in-scope variables of pattern P_1, can be discarded to obtain the actual answer $\mu = \{?x \mapsto a\}$ of the query (Q_1 DIFF Q_2).

Formally, the operation (Q_1 OPTIONAL$_{C_\nu}$ Q_2) has the form ((Q_1 OPTIONAL $\nu(Q_2)$) FILTER C_ν) where ν is a function mapping in-scope variables in Q_1 that are not strongly-bound to fresh variables, $\nu(Q_2)$ is the query resulting from renaming in Q_2 the variables $?y \in \text{dom}(\nu)$ with $\nu(?y)$, and C_ν is a conjunctions of selection conditions that check the compatibility between the values of such pairs of variables ?y and $\nu(?y)$ (i.e., formulas of the form $\neg \text{bound}(?y) \vee \neg \text{bound}(\nu(?y)) \vee ?y = \nu(?y)$).

4.3 Correctness criteria

To define correctness of the query rewriting described in Definition 28 we need the notions of *soundness* and *completeness*. A query rewriting is sound when the rewritten query returns right polynomial expressions, and complete if it returns polynomial expressions for all mappings with non-zero polynomials.

▶ **Definition 32** (Soundness and Completeness). *Let* Reify *be a function that implements a reification scheme, and γ be a function that receives a SPARQL query Q and a variable* ?prov $\notin$ inScope(Q), *and returns a SPARQL query $\gamma(Q, ?prov)$ with* inScope($\gamma(Q, ?prov)$) = inScope(Q) $\cup$ {?prov}. *Let Γ be an X-graph, and G =* Reify(Γ) *the RDF-star graph resulting from applying the* Reify *function to each triple in Γ in order to encode Γ's triple annotations. Then:*

- *γ is called* sound *for* Reify *if, for every answer of the rewritten query $\mu \cup \{?prov \mapsto e\}$ in $[\![\gamma(Q, ?prov)]\!]_G$, e is an expression for the polynomial $(\!|Q|\!)_\Gamma(\mu)$.*
- *γ is called* complete *for* Reify *if, for every mapping μ such that $(\!|Q|\!)_\Gamma(\mu)$ is a non-zero polynomial, there exists an expression e such that $\mu \cup \{?prov \mapsto e\} \in [\![\gamma(Q, ?prov)]\!]_G$.*

▶ **Theorem 33.** *Let* Reify *be a reification scheme and β be the function described in Definition 28. Then, function β is sound and complete for the reification scheme* Reify*.*

Proof. It can be shown by induction on the query structure. There are two base cases. First, when the query Q is an empty basic graph pattern, then we know that for every annotated graph Γ, $(\!|Q|\!)_\Gamma = \{\!|\mu_\emptyset \mapsto 1|\!\}$. Similarly, for every graph G, $[\![(\{\} \text{ BIND } (1 \text{ AS } \texttt{?prov}))]\!]_G = \{\{\texttt{?prov} \mapsto 1\}\}$, which corresponds to the $\mathcal{K}$-relation $\{\!|\mu_\emptyset \mapsto 1|\!\}$. Second, when the query is a triple pattern T, then $(\!|Q|\!)_\Gamma = \{\!|\mu_t \mapsto \Gamma(t)|\!\}$, where $\mathrm{dom}(\mu_t)$ are the variables occurring in T and $\mu_t(T) = t$. If G is a corresponding graph for Γ, then $[\![\mathsf{Reify}(T, \texttt{?prov}\oplus)]\!]_G$ has the form $\{\mu_t \cup \{\texttt{?prov}\oplus \mapsto k_1\}, \ldots, \mu_t \cup \{\texttt{?prov}\oplus \mapsto k_n\}\}$, where $\sum_1^n k_i = \Gamma(t)$. This sum is required because regular graphs G representing an annotated graph Γ can include the same triple t multiple times (e.g., when they are the result of aggregating multiple data sources). The function ProvAggSum(?prov$\oplus$) in query $\beta(Q, \texttt{?prov})$ will then return the expression $(\oplus\, k_1 \ldots k_n)$, which is the expression for the polynomial $\Gamma(t)$. Hence, $[\![\beta(Q, \texttt{?prov})]\!]_G = \{\mu_t \cup \{\texttt{?prov} \mapsto (\oplus\, k_1 \ldots k_n)\}\}$, which corresponds to the expected $\mathcal{K}$-relation.

We next consider the non-base cases:

Case $Q = (Q_1 \text{ AND } Q_2)$. For every mapping μ and annotated graph Γ,

$$(\!|(Q_1 \text{ AND } Q_2)|\!)_\Gamma(\mu) = \sum_{\mu=\mu_1\cup\mu_2} ((\!|Q_1|\!)_\Gamma(\mu_1) \otimes (\!|Q_2|\!)_\Gamma(\mu_2)).$$

Similarly for a corresponding regular graph G for the annotated graph Γ, the set $[\![\beta(Q, \texttt{?prov})]\!]_G$ has the form $\{\mu'_1 \cup \{\texttt{?prov} \mapsto k_1\}, \ldots, \mu'_m \cup \{\texttt{?prov} \mapsto k_m\}\}$, where $\mu'_1, \ldots, \mu'_m$ are distinct mappings. By construction, for $1 \le j \le m$, k_j has the form $(\oplus\, (\otimes\, k_1^1\ k_2^1) \cdots (\otimes\, k_1^p\ k_2^p))$, and for $1 \le i \le p$, there exists two mappings μ_1^i and μ_2^i such that

- $\mu_1^i \cup \{\texttt{?prov}\oplus\otimes 1 \mapsto k_1^i\} \in [\![\beta(Q_1, \texttt{?prov}\oplus\otimes 1)]\!]_G$,
- $\mu_2^i \cup \{\texttt{?prov}\oplus\otimes 2 \mapsto k_2^i\} \in [\![\beta(Q_2, \texttt{?prov}\oplus\otimes 2)]\!]_G$.

By the induction hypothesis, the query rewriting is sound for the queries Q_1 and Q_2. Thus, k_1^i and k_2^i represent $(\!|Q_1|\!)_\Gamma(\mu_1^i)$ and $(\!|Q_1|\!)_\Gamma(\mu_2^i)$, respectively. The rewriting is sound if the expression $(\oplus\, (\otimes\, k_1^1\ k_2^1) \cdots (\otimes\, k_1^p\ k_2^p))$ represents $(\!|Q|\!)_\Gamma(\mu'_j)$ if it contains all then non-zero expressions for the mappings μ_1 and μ_2 such that $\mu'_j = \mu_1 \cup \mu_2$. This requirement holds by the induction hypothesis: the query rewriting is complete for the queries Q_1 and Q_2. Hence, the rewriting is sound.

To prove the completeness, we need to show that every mapping μ in the support of $(\!|Q|\!)_\Gamma$ is one of the aforementioned mappings $\mu'_1, \ldots, \mu'_m$. Since $\mu \in \mathrm{supp}((\!|Q|\!)_\Gamma)$, there exist at least two mappings $\mu_1 \in \mathrm{supp}((\!|Q_1|\!)_\Gamma)$ and $\mu_2 \in \mathrm{supp}((\!|Q_2|\!)_\Gamma)$ such that $\mu = \mu_1 \cup \mu_2$. By the induction hypothesis, these mappings appear in the solutions of queries $\beta(Q_1, \texttt{?prov}\oplus\otimes 1)$ and $\beta(Q_2, \texttt{?prov}\oplus\otimes 2)$, respectively. Then, μ appears in the solutions of query $(\beta(Q_1, \texttt{?prov}\oplus\otimes 1) \text{ AND } \beta(Q_2, \texttt{?prov}\oplus\otimes 2))$. Then, μ appears in the solutions of query $\beta(Q, \texttt{?prov})$. Hence, the rewriting is complete.

Case $Q = (Q_1 \text{ UNION } Q_2)$. For every mapping μ and annotated graph Γ,

$$(\!|(Q_1 \text{ UNION } Q_2)|\!)_\Gamma(\mu) = (\!|Q_1|\!)_\Gamma(\mu) \oplus (\!|Q_2|\!)_\Gamma(\mu).$$

Similarly for a corresponding regular graph G for the annotated graph Γ, the set $[\![\beta(Q, \texttt{?prov})]\!]_G$ has the form $\{\mu'_1 \cup \{\texttt{?prov} \mapsto r_1\}, \ldots, \mu'_m \cup \{\texttt{?prov} \mapsto k_m\}\}$, where $\mu'_1, \ldots, \mu'_m$ are distinct mappings. By construction, for $1 \le j \le m$, k_j has either the form $(\oplus\, r_1\ r_2)$, $(\oplus\, r_1)$, or $(\oplus\, r_2)$, and

- $\mu'_j \cup \{\texttt{?prov}\oplus \mapsto r_1\} \in [\![\beta(Q_1, \texttt{?prov}\oplus)]\!]_G$ if and only if $k_j = (\oplus\, r_1\ r_2)$ or $k_j = (\oplus\, r_1)$,
- $\mu'_j \cup \{\texttt{?prov}\oplus \mapsto r_2\} \in [\![\beta(Q_2, \texttt{?prov}\oplus)]\!]_G$ if and only if $k_j = (\oplus\, r_1\ r_2)$ or $k_j = (\oplus\, r_2)$.

By the induction hypothesis, the query rewriting is sound and complete for the queries Q_1 and Q_2. Thus, if defined, r_1 and r_2 represent $(\!|Q_1|\!)_\Gamma(\mu'_j)$ and $(\!|Q_2|\!)_\Gamma(\mu'_j)$, respectively. If r_1 or r_2 is undefined, the respective value of $(\!|Q_1|\!)_\Gamma(\mu'_j)$ or $(\!|Q_2|\!)_\Gamma(\mu'_j)$ is 0. Hence, the rewriting is sound.

To prove the completeness, we need to show that every mapping μ in the support of $(\!|Q|\!)_\Gamma$ is one of the aforementioned mappings $\mu'_1, \ldots, \mu'_m$. Since $\mu \in \mathrm{supp}((\!|Q|\!)_\Gamma)$, it must happen that $\mu \in \mathrm{supp}((\!|Q_1|\!)_\Gamma)$ or $\mu \in \mathrm{supp}((\!|Q_2|\!)_\Gamma)$. By the induction hypothesis, μ appears in the solutions of queries $\beta(Q_1, \texttt{?prov}\oplus)$ or $\beta(Q_2, \texttt{?prov}\oplus)$. Then, μ appears in the solutions of query $(\beta(Q_1, \texttt{?prov}\oplus)$ UNION $\beta(Q_2, \texttt{?prov}\oplus))$. Then, μ appears in the solutions of query $\beta(Q, \texttt{?prov})$. Hence, the rewriting is complete.

Case $Q = (Q_1$ DIFF $Q_2)$. For every mapping μ and annotated graph Γ,

$$(\!|(Q_1 \text{ DIFF } Q_2)|\!)_\Gamma(\mu) = (\!|Q_1|\!)_\Gamma(\mu) \ominus \sum_{\mu \sim \mu_2} ((\!|Q_2|\!)_\Gamma(\mu_2)).$$

Similarly for a corresponding regular graph G for the annotated graph Γ, the set $[\![\beta(Q, \texttt{?prov})]\!]_G$ has the form $\{\mu'_1 \cup \{\texttt{?prov} \mapsto k_1\}, \ldots, \mu'_m \cup \{\texttt{?prov} \mapsto k_m\}\}$, where $\mu'_1, \ldots, \mu'_m$ are distinct mappings. By construction, for $1 \leq j \leq m$, k_j has the form $(\ominus\ k\ (\oplus\ k_1 \cdots k_p))$, and

- $\mu'_j \cup \{\texttt{?prov}\ominus 1 \mapsto k\} \in [\![\beta(Q_1, \texttt{?prov}\ominus 1)]\!]_G$,
- for $1 \leq i \leq p$, there is a mapping μ_i, such that $\mu_i \cup \{\texttt{?prov}\ominus 2\oplus \mapsto k_i\} \in [\![\beta(Q_2, \texttt{?prov}\ominus 2\oplus)]\!]_G$ and $\mu_i \sim \mu'_j$.

By the induction hypothesis, the query rewriting is sound and complete for the queries Q_1 and Q_2. Thus, k and $(\oplus\ k_1 \ldots k_p)$ represent $(\!|Q_1|\!)_\Gamma(\mu'_j)$ and $\sum_{\mu_i \sim \mu'_j} (\!|Q_2|\!)_\Gamma(\mu_i)$. Hence, the rewriting is sound.

To prove the completeness, we need to show that every mapping μ in the support of $(\!|Q|\!)_\Gamma$ is one of the aforementioned mappings $\mu'_1, \ldots, \mu'_m$. Since $\mu \in \mathrm{supp}((\!|Q|\!)_\Gamma)$, also $\mu \in \mathrm{supp}((\!|Q_1|\!)_\Gamma)$. By the induction hypothesis, μ appears in the solutions of queries $\beta(Q_1, \texttt{?prov}\ominus 1)$. Then, μ appears in the solutions of query $\beta(Q, \texttt{?prov})$. Hence, the rewriting is complete.

Case $Q = ($SELECT W WHERE $Q_1)$. For every mapping μ and annotated graph Γ,

$$(\!|(\text{SELECT } W \text{ WHERE } Q_1)|\!)_\Gamma(\mu) = \sum_{\mu' : \mu'|_W = \mu} (\!|Q_1|\!)_\Gamma(\mu').$$

Similarly for a corresponding regular graph G for the annotated graph Γ, the set $[\![\beta(Q, \texttt{?prov})]\!]_G$ has the form $\{\mu'_1 \cup \{\texttt{?prov} \mapsto k_1\}, \ldots, \mu'_m \cup \{\texttt{?prov} \mapsto k_m\}\}$, where $\mu'_1, \ldots, \mu'_m$ are distinct mappings. By construction, for $1 \leq j \leq m$, k_j has the form $(\oplus\ k_1 \ldots k_p)$, and for $1 \leq i \leq p$, there exists a mapping μ_i such that $\mu_i|_W = \mu'_j$ and $\mu_i \cup \{\texttt{?prov}\oplus \mapsto k_i\} \in [\![\beta(Q_1, \texttt{?prov}\oplus]\!]_G$. By the induction hypothesis, the query rewriting is sound and complete for query Q_1. Thus, each k_i represents $(\!|Q_1|\!)_\Gamma(\mu_i)$. Hence, the rewriting is sound.

To prove the completeness, we need to show that every mapping μ in the support of $(\!|Q|\!)_\Gamma$ is one of the aforementioned mappings $\mu'_1, \ldots, \mu'_m$. Since $\mu \in \mathrm{supp}((\!|Q|\!)_\Gamma)$, there exists at least one mapping $\mu_1 \in \mathrm{supp}((\!|Q_1|\!)_\Gamma)$ such that $\mu = \mu_1|_W$. By the induction hypothesis, mapping μ_1 appears in the solutions of query $\beta(Q_1, \texttt{?prov}\oplus)$. Then, μ appears in the solutions of query $\beta(Q, \texttt{?prov})$. Hence, the rewriting is complete.

Case $Q = (Q_1$ FILTER $\varphi)$. For every mapping μ and annotated graph Γ,

$$(\!|(Q_1 \text{ FILTER } \varphi)|\!)_\Gamma(\mu) = (\!|Q_1|\!)_\Gamma(\mu) \otimes 1_{\mu \models \varphi}.$$

Similarly for a corresponding regular graph G for the annotated graph Γ, the set $[\![\beta(Q, \texttt{?prov})]\!]_G$ has the form $\{\mu'_1 \cup \{\texttt{?prov} \mapsto k_1\}, \ldots, \mu'_m \cup \{\texttt{?prov} \mapsto k_m\}\}$, where $\mu'_1, \ldots, \mu'_m$ are distinct mappings. By construction, for $1 \leq j \leq m$, k_j, $\mu'_j \cup \{\texttt{?prov} \mapsto k_i\} \in [\![\beta(Q_1, \texttt{?prov}]\!]_G$ and $\mu_j \models \varphi$. By the induction hypothesis, the query rewriting is sound for query Q_1. Thus, k_j represents the $(\!|Q_1|\!)_\Gamma(\mu'_j)$. Hence, the rewriting is sound.

To prove the completeness, we need to show that every mapping μ in the support of $(\!|Q|\!)_\Gamma$ is one of the aforementioned mappings $\mu'_1, \ldots, \mu'_m$. Since $\mu \in \text{supp}((\!|Q|\!)_\Gamma)$, $\mu \in \text{supp}((\!|Q_1|\!)_\Gamma)$ and $\mu \models \varphi$. By the induction hypothesis, mapping μ appears in the solutions of query $\beta(Q_1, \texttt{?prov})$. Since $\mu \models \varphi$, μ appears in the solutions of query $(\beta(Q_1, \texttt{?prov})$ FILTER $\varphi)$, which are the solutions of query $\beta(Q, \texttt{?prov})$. Hence, the rewriting is complete.

We have proved that the rewriting is sound and complete for the two base cases and all the inductive cases. ◀

We highlight that for queries of the form Q_1 DIFF Q_2, NPCS also returns why-not provenance explanations of the form $k_1 \ominus k_2$ (k_1 and k_2 are polynomials) for the bindings that match both Q_1 and Q_2. The polynomial k_2 tells us which sources must be removed from the graph so that the corresponding binding becomes a query solution.

4.4 Query Rewriting Optimizations

If we look at our example rewritten query Q' described in Section 4.1, we can notice that this query includes a GROUP BY clause, and two subqueries, namely P'_1 and P'_2, each of which also includes a GROUP BY clause. In Definition 35 we describe an alternative query rewriting that produces equivalent polynomial expressions, but reduces the number of aggregate operations.

▶ **Definition 34.** *A* sum-query Q *is a query such that the query rewriting* β *described in Definition 28 returns a query of the form:*

$$\begin{array}{lll}\beta(Q, \texttt{?prov}) = (\ \text{SELECT} & \text{inScope}(Q)\ (\text{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\ \text{WHERE} & T \\ \text{GROUP BY} & \text{inScope}(Q)\ ).\end{array}$$

We call query T *the* pattern *of* $\beta(Q, \texttt{?prov})$.

Note that, according to Definition 28, sum-queries are all queries that match the rules for the triple patterns and the operators AND, UNION, and SELECT (rewriting rules 2–4, and 6).

▶ **Definition 35.** *Let* Q *be a SPARQL query,* ?prov *be a variable, and* Reify *a reification scheme. Then, the rewritten query for* Q *and variable* ?prov *over scheme* Reify*, denoted* $\beta(Q, \texttt{?prov})$*, is defined recursively as is specified in Definition 28, but the following rules are applied when possible:*

1. *If* Q *is* $(Q_1$ AND $\cdots$ AND $Q_n)$*, and* $Q_1, \ldots, Q_n$ *are sum-queries, such that for* $1 \leq i \leq n$*, the pattern of* $\beta(Q_i, \texttt{?prov}\oplus\otimes i)$ *is* T_i*, then* $\beta(Q, \texttt{?prov})$ *is the query*

$$\begin{array}{lll}(\ \text{SELECT} & \text{inScope}(Q)\ (\text{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\ \text{WHERE} & ((\ T_1\ \text{AND} \cdots \text{AND}\ T_n) \\ & \quad \text{BIND}\ (\text{ProvProd}(\texttt{?prov}\oplus\otimes 1, \ldots, \texttt{?prov}\oplus\otimes n)\ \text{AS}\ \texttt{?prov}\oplus) \\ \text{GROUP BY} & \text{inScope}(Q)\ ).\end{array}$$

2. *If* Q *is* $(Q_1$ UNION $\cdots$ UNION $Q_n)$*, where* $Q_1, \ldots, Q_n$ *are sum-queries, and for* $1 \leq i \leq n$*, the pattern of* $\beta(Q_i, \texttt{?prov}\oplus)$ *is* T_i*, then* $\beta(Q, \texttt{?prov})$ *is the query*

$$\begin{array}{lll}(\ \text{SELECT} & \text{inScope}(Q)\ (\text{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\ \text{WHERE} & (T_i\ \text{UNION} \cdots \text{UNION}\ T_n) \\ \text{GROUP BY} & \text{inScope}(Q)\ ).\end{array}$$

3. *If Q is $(Q_1$ DIFF $Q_2)$, and ν is a variable substitution that substitutes with fresh variables the variables in $\mathrm{dom}(Q_1) \cap \mathrm{dom}(Q_2)$ that are not strongly bound in Q_1, and Q_2 is a sum-query, then $\beta(Q, \texttt{?prov})$ is the query*

$$\begin{array}{ll}(\text{ SELECT} & \mathrm{inScope}(Q)\ (\mathrm{ProvDiff}(\texttt{?prov}\ominus 1, \mathrm{ProvAggSum}(\texttt{?prov}\ominus 2\oplus))\ \text{AS}\ \texttt{?prov}) \\ \text{WHERE} & (\beta(Q_1, \texttt{?prov}\ominus 1)\ \text{OPTIONAL}_{C_\nu}\ \beta(\nu(Q_2), \texttt{?prov}\ominus 2\oplus)) \\ \text{GROUP BY} & \mathrm{inScope}(Q) \cup \{\texttt{?prov}\ominus 1\}\ ).\end{array}$$

4. *If Q is* (SELECT W WHERE Q')*, where Q' is a sum-query, then $\beta(Q, \texttt{?prov})$ is*

$$\begin{array}{ll}(\text{ SELECT} & W\ (\mathrm{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\ \text{WHERE} & \beta(Q', \texttt{?prov}\oplus) \\ \text{GROUP BY} & W\ ).\end{array}$$

▶ Note 36. In the first two rules of Definition 35, we omitted the parenthesis for sequences of operations AND and UNION, because these operators are associative. Intuitively, the associativity of these operators allows considering the binary operation as a single variadic operation with a single GROUP BY clause.

▶ **Example 37.** Consider the query Q from Example 15. Then, according to the query rewriting described in Definition 35 the rewritten query of Q is:

$$\begin{array}{lll}\beta(Q, \texttt{?prov}) = (\text{ SELECT} & & \texttt{?x}\ (\mathrm{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\ & \text{WHERE} & ((\ \mathsf{Reify}(\texttt{?x}, \mathsf{likes}, \mathsf{pasta}, \texttt{?prov}\oplus\otimes 1)\ \text{AND} \\ & & \quad \mathsf{Reify}(\texttt{?x}, \mathsf{livesIn}, \mathsf{Italy}, \texttt{?prov}\oplus\otimes 2)) \\ & & \ \text{BIND}\ (\mathrm{ProvProd}(\texttt{?prov}\oplus\otimes 1, \texttt{?prov}\oplus\otimes 2)\ \text{AS}\ \texttt{?prov}\oplus)) \\ & \text{GROUP BY} & \texttt{?x}\ ).\end{array}$$

This query has only one GROUP BY clause whereas the query Q' at the end of Section 4.1 (generated using the rewriting of Definition 28) has three.

▶ **Theorem 38.** *Let* Reify *be a reification scheme, and β be the function described in Definition 35. Then, function β is sound and complete for the reification scheme* Reify.

Proof. We prove this theorem by induction on the query structure. Since we already proved that the rewriting in Definition 28 is sound and complete, it suffices proving that each of the new rewriting rules in Definition 35 produce the same results as the rewriting in Definition 28.

Case $Q = (Q_1$ AND $\cdots$ AND $Q_n)$. It suffices to prove this case by induction on n. For the base case $n = 2$, both query rewritings produce the same query. For the inductive case, we abbreviate $(Q_1$ AND $\cdots$ AND $Q_{n-1})$ as $Q_{1,n-1}$. Let Q'_1 be $\beta(((Q_{1,n-1}$ AND $Q_n)$ AND $Q_{n+1}), \texttt{?prov})$ according to Definition 35, and Q'_2 be $\beta((Q_{1,n-1}$ AND Q_n AND $Q_{n+1}), \texttt{?prov})$ according to Definition 35. It is not difficult to see that every mapping μ that appears in the answers to query Q'_1 appears in the answers to query Q'_2, and vice versa. Assume that $\mu \cup \{\texttt{?prov} \mapsto k_1\} \in [\![Q'_1]\!]_G$ and $\mu \cup \{\texttt{?prov} \mapsto k_2\} \in [\![Q'_2]\!]_G$. Then, we need to prove that the expressions k_1 and k_2 are equivalent. Let Ω_a, Ω_b, and Ω_c be the respective set of mappings that appear in the support of $(\!|Q_{1,n-1}|\!)_G$, $(\!|Q_{n-1}|\!)_G$, and $(\!|Q_n|\!)_G$. By construction,

$$k_1 = \sum_{\substack{\mu_{ab}\in\Omega_a \bowtie \Omega_b \\ \mu_c\in\Omega_c \\ \mu_{ab}\sim\mu \\ \mu_c\sim\mu}} \Big(\otimes \sum_{\substack{\mu_a\in\Omega_a \\ \mu_b\in\Omega_b \\ \mu_a\sim\mu_{ab} \\ \mu_b\sim\mu_{ab}}} (\otimes\ k_a\ k_b)\ k_c\Big), \qquad k_2 = \sum_{\substack{\mu_a\in\Omega_a \\ \mu_b\in\Omega_b \\ \mu_c\in\Omega_c \\ \mu_a\sim\mu \\ \mu_b\sim\mu \\ \mu_c\sim\mu}} (\otimes\ k_a\ k_b\ k_c),$$

where the $\sum$ denote the expressions $(\oplus\ \ldots)$, $k_a = \Omega_a(\mu_a)$, $k_b = \Omega_b(\mu_b)$, and $k_c = \Omega_c(\mu_c)$. By construction, $\mu_{ab} = \mu_a \cup \mu_b$. Hence, k_1 and k_2 are equivalent.

Case $Q = (Q_1 \text{ UNION} \cdots \text{UNION} Q_n)$. It suffices to prove this case by induction on n. For the base case $n = 2$, both query rewritings produce the same query. For the inductive case, we abbreviate $(Q_1 \text{ UNION} \cdots \text{UNION } Q_{n-1})$ as $Q_{1,n-1}$. Let Q'_1 be $\beta(((Q_{1,n-1} \text{ UNION } Q_n) \text{ UNION } Q_{n+1}), \texttt{?prov})$ according to Definition 35, and Q'_2 be $\beta((Q_{1,n-1} \text{ UNION } Q_n \text{ UNION } Q_{n+1}), \texttt{?prov})$ according to Definition 35. It is not difficult to see that every mapping μ that appears in the answers to query Q'_1 appears in the answers to query Q'_2, and vice versa. Assume that $\mu \cup \{\texttt{?prov} \mapsto k_1\} \in [\![Q'_1]\!]_G$ and $\mu \cup \{\texttt{?prov} \mapsto k_2\} \in [\![Q'_2]\!]_G$. Then, we need to prove that the expressions k_1 and k_2 are equivalent. Let Ω_a, Ω_b, and Ω_c be the respective set of mappings that appear in the support of $(\!|Q_{1,n-1}|\!)_G$, $(\!|Q_{n-1}|\!)_G$, and $(\!|Q_n|\!)_G$. By construction,

$$k_1 = (\oplus\ (\oplus\ k_a\ k_b)\ k_c)\,, \qquad\qquad k_2 = (\oplus\ k_a\ k_b\ k_c),$$

where $k_a = \Omega_a(\mu)$, $k_b = \Omega_b(\mu)$, $k_c = \Omega_c(\mu)$, and if any of the terms k_a, k_b, and k_c is 0, then it does not necessarily appears in the expression. Hence, k_1 and k_2 are equivalent.

Case $Q = (Q_1 \text{ DIFF } Q_2)$. Let Q'_1 be $\beta(Q, \texttt{?prov})$ according to Definition 35, and Q'_2 be $\beta(Q, \texttt{?prov})$ according to Definition 35. It is not difficult to see that every mapping μ that appears in the answers to query Q'_1 appears in the answers to query Q'_2, and vice versa. Assume that $\mu \cup \{\texttt{?prov} \mapsto k_1\} \in [\![Q'_1]\!]_G$ and $\mu \cup \{\texttt{?prov} \mapsto k_2\} \in [\![Q'_2]\!]_G$. Then, we need to prove that the expressions k_1 and k_2 are equivalent. By construction,

$$k_1 = (\ominus\ k\ \ (\oplus\ (\oplus\ a_1)\ \ldots\ (\oplus\ a_n)))\,, \qquad\qquad k_2 = (\ominus\ k\ (\oplus\ a_1\ \ \ldots\ \ a_n))\,,$$

where, k is the how-provenance of mapping μ for query Q_1, and for $1 \leq j \leq n$, a_j is a sequence of the form $k_1 \ldots k_m$ that represents the how-provenance of a solution of the pattern of the sum-query Q_2. Hence, k_1 and k_2 are equivalent.

Case $Q = (\text{SELECT } \boldsymbol{W} \text{ WHERE } Q_1)$. Let Q'_1 be $\beta(Q, \texttt{?prov})$ according to Definition 35, and Q'_2 be $\beta(Q, \texttt{?prov})$ according to Definition 35. It is not difficult to see that every mapping μ that appears in the answers to query Q'_1 appears in the answers to query Q'_2, and vice versa. Assume that $\mu \cup \{\texttt{?prov} \mapsto k_1\} \in [\![Q'_1]\!]_G$ and $\mu \cup \{\texttt{?prov} \mapsto k_2\} \in [\![Q'_2]\!]_G$. Then, we need to prove that the expressions k_1 and k_2 are equivalent. By construction,

$$k_1 = (\oplus\ (\oplus\ a_1)\ \ldots\ (\oplus\ a_n))\,, \qquad\qquad k_2 = (\oplus\ a_1\ \ \ldots\ \ a_n),$$

where, for $1 \leq j \leq n$, a_j is a sequence of the form $k_1 \ldots k_m$ that represents the how-provenance of a solution of the pattern of the sum-query Q_2. Hence, k_1 and k_2 are equivalent. ◀

5 How-provenance of federated SPARQL query computation

In the previous section we have described NPCS, an engine-agnostic method to compute provenance polynomials that explain the answers of SPARQL queries run against a single SPARQL endpoint. They are therefore suitable for a centralized setting. In a federated setting, a query is executed on multiple SPARQL endpoints. Keeping track of the endpoints that contribute to each answer has multiple potential applications: for example, we can use that information to optimize query execution, to control the access to information on materialized views, and to assign trust levels to the retrieved answers. With this in mind, we present an extension to the NPCS system for the federated context, called Fed-NPCS.

5.1 An algebra for Federated Query Computation

We start by introducing an algebraic structure and a set of expressions that extend our how-provenance polynomials to provide explanations for the answers of SPARQL queries run on a federation. The how-provenance polynomials, discussed in the previous section and proposed by Geerts et al. [22] provided the semantics for the expressions in the set $\mathrm{ProvExp}(X)$. These expressions combine statement identifiers from a finite set $X \subset \mathbf{I}$ with the operations $\oplus$, $\otimes$, and $\ominus$. We wrote $(\mathrm{ProvExp}_{\cong}(X), \oplus, \otimes, \ominus, [0], [1])$ for the spm-semiring these expressions define. We now show that this algebra can also be used to record the endpoints where the statements are found. Indeed, instead of using how-provenance expressions in $\mathrm{ProvExp}(X)$ we can consider expressions in $\mathrm{ProvExp}(Y \times X)$ where each pair $(y, x) \in Y \times X$ consists of an element $y \in Y$ that identifies a SPARQL endpoint and an element $x \in X$ that identifies a statement. The following example illustrates the use of these pairs in provenance polynomials.

▶ **Example 39.** Let G_1 and G_2 be two RDF-star graphs defined as follows:

$$\begin{aligned}
G_1 = \{&((\mathsf{Ulm}, \mathsf{a}, \mathsf{City}), \mathsf{wasDerivedFrom}, x_1),\\
&((\mathsf{Ulm}, \mathsf{country}, \mathsf{Germany}), \mathsf{wasDerivedFrom}, x_2),\\
&((\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}), \mathsf{wasDerivedFrom}, x_3),\\
&((\mathsf{Budapest}, \mathsf{a}, \mathsf{City}), \mathsf{wasDerivedFrom}, x_4),\\
&((\mathsf{Budapest}, \mathsf{country}, \mathsf{Hungary}), \mathsf{wasDerivedFrom}, x_5)\}.\\
G_2 = \{&((\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}), \mathsf{wasDerivedFrom}, x_3),\\
&((\mathsf{Danube}, \mathsf{crosses}, \mathsf{Budapest}), \mathsf{wasDerivedFrom}, x_6)\}.
\end{aligned}$$

Graphs G_1 and G_2 can share statements and identifiers. For example, "The Danube crosses Ulm" is stated in both graphs, and annotated with the IRI x_3. To use the non-federated NPCS framework we should first merge these two graphs into one. By naming the graph G_1 as y_1 and the graph G_2 as y_2, we can define the following RDF-star graph as the merge of both graphs:

$$\begin{aligned}
G = \{&((\mathsf{Ulm}, \mathsf{a}, \mathsf{City}), \mathsf{wasDerivedFrom}, (y_1, x_1)),\\
&((\mathsf{Ulm}, \mathsf{country}, \mathsf{Germany}), \mathsf{wasDerivedFrom}, (y_1, x_2)),\\
&((\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}), \mathsf{wasDerivedFrom}, (y_1, x_3)),\\
&((\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}), \mathsf{wasDerivedFrom}, (y_2, x_3)),\\
&((\mathsf{Budapest}, \mathsf{a}, \mathsf{City}), \mathsf{wasDerivedFrom}, (y_1, x_4)),\\
&((\mathsf{Budapest}, \mathsf{country}, \mathsf{Hungary}), \mathsf{wasDerivedFrom}, (y_1, x_5)),\\
&((\mathsf{Danube}, \mathsf{crosses}, \mathsf{Budapest}), \mathsf{wasDerivedFrom}, (y_2, x_6))\}.
\end{aligned}$$

In an abuse of notation, each pair (y_i, x_j) in the merged graph G denotes an IRI that can be used to identify the statement, as we already did with NPCS. The corresponding $(Y \times X)$-graph for G is the graph Γ defined as follows:

$$\begin{aligned}
\Gamma = \{\!\!\{&(\mathsf{Ulm}, \mathsf{a}, \mathsf{City}) && \mapsto (y_1, x_1),\\
&(\mathsf{Ulm}, \mathsf{country}, \mathsf{Germany}) && \mapsto (y_1, x_2),\\
&(\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}) && \mapsto (y_1, x_3),\\
&(\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}) && \mapsto (y_2, x_3),\\
&(\mathsf{Budapest}, \mathsf{a}, \mathsf{City}) && \mapsto (y_1, x_4),\\
&(\mathsf{Budapest}, \mathsf{country}, \mathsf{Hungary}) && \mapsto (y_1, x_5),\\
&(\mathsf{Danube}, \mathsf{crosses}, \mathsf{Budapest}) && \mapsto (y_2, x_6)\}\!\!\}.
\end{aligned}$$

Let Q_O be the following query, asking for countries with a city crossed by the Danube river:

(SELECT {?country}
WHERE ((?city, a, City) AND (?city, country, ?country) AND (Danube, crosses, ?city))).

By executing the query Q_O on graph Γ, the answers Germany and Hungary are explained by the following how-provenance polynomials:

$$(\!|Q_O|\!)_\Gamma(\{\texttt{?country} \mapsto \mathsf{Germany}\}) = (y_1, x_1) \otimes (y_1, x_2) \otimes ((y_1, x_3) \oplus (y_2, x_3)),$$
$$(\!|Q_O|\!)_\Gamma(\{\texttt{?country} \mapsto \mathsf{Hungary}\}) = (y_1, x_4) \otimes (y_1, x_5) \otimes (y_2, x_6).$$

As described in Section 4, the how provenance of these solution mappings can be computed with NPCS by rewriting the query Q_O into a query Q_R, and then executing query Q_R on the graph Γ.

▶ Note 40. Example 39 shows that, by merging all graphs, we can reuse NPCS to compute how-provenance over a federation of SPARQL endpoints. However, this merging of graphs precludes the direct use of the federation and motivates the introduction of Fed-NPCS, an extension of NPCS to compute how-provenance on federated SPARQL services. To define Fed-NPCS we extend the spm-semiring structure so as to include the pairs (y_i, x_j) that appear in the example. Concretely, these pairs will be encoded with an operation $y_i \circledast x_j$ that tell us that the computation of an answer uses a statement identified as x_j and retrieved from a service y_i.

▶ **Definition 41** (Federated spm-semiring). *Let $\mathcal{M} = (M, \oplus\!\!\!\cdot, \circledast, 0_\mathcal{M}, 1_\mathcal{M})$ be a structure where $(M, \oplus\!\!\!\cdot, 0_\mathcal{M})$ is a commutative zero-sum-free monoid, $(M, \circledast, 1_\mathcal{M})$ is a zero-sum-free monoid, and $\mathcal{K} = (K, +_\mathcal{K}, \times_\mathcal{K}, -_\mathcal{K}, 0_\mathcal{K}, 1_\mathcal{K})$ be an spm-semiring. An spm-semiring $(R, \oplus, \otimes, \ominus, 0, 1)$ is said to be a* federated spm-semiring *over $\mathcal{M}$ and $\mathcal{K}$ if there exists a function $f : M \times K \to R$ such that:*

- *$f(0_\mathcal{M}, k) = 0$ and $(m, 0_\mathcal{K}) = 0$ for every $k \in K$ and $m \in M$.*
- *$f(1_\mathcal{M}, 1_\mathcal{K}) = 1$.*
- *For every $m \in M$ and $k_1, k_2 \in K$,*
  - *$f(m, k_1 +_\mathcal{K} k_2) = f(m, k_1) \oplus f(m, k_2)$.*
  - *$f(m, k_1 \times_\mathcal{K} k_2) = f(m, k_1) \otimes f(m, k_2)$.*
  - *$f(m, k_1 -_\mathcal{K} k_2) = f(m, k_1) \ominus f(m, k_2)$.*
- *For every $m_1, m_2 \in M$ and $k \in K$, $f(m_1 \oplus\!\!\!\cdot\, m_2, k) = f(m_1, k) \oplus f(m_2, k)$*
- *For every element $r \in R$ there is a finite set $\{(m_1, k_1), \ldots, (m_n, k_n)\} \subseteq M \times K$ such that $r = f(m_1, k_1) \oplus \cdots \oplus f(m_n, k_n)$.*

$\mathcal{M}$ is called the fed-structure *and $\mathcal{K}$ is called the* how-structure *of the federated spm-semiring R.*

▶ **Definition 42** (Federated how-provenance expressions). *Let Y and X be two disjoint sets of IRIs. We write* $\mathrm{FedProvExp}(X, Y)$ *for the set of expressions, called* federated how-provenance expressions*, defined recursively as follows:*

- *$0 \in \mathrm{FedProvExp}(X, Y)$, $1 \in \mathrm{FedProvExp}(X, Y)$, and $X \subseteq \mathrm{FedProvExp}(X, Y)$.*
- *We call* service-expressions *to the expressions combining elements in set $Y \cup \{0, 1\}$ with the operators $\oplus\!\!\!\cdot$ and $\circledast$. If $a \in \mathrm{FedProvExp}(X, Y)$ and s is a service-expression, then $s \circledast a \in \mathrm{FedProvExp}(X, Y)$.*
- *If $a, b \in \mathrm{FedProvExp}(X, Y)$ then $a \oplus b$, $a \otimes b$, and $a \ominus b$ are in $\mathrm{FedProvExp}(X, Y)$.*

The semantics of federated how-provenance expressions is defined by mapping them to a federated spm-semiring $(R, \oplus, \otimes, \ominus, 0, 1)$ with fed-structure $\mathcal{M}$ and how-structure $\mathcal{K}$. Such a mapping h is defined by two mappings $h_Y : Y \to \mathcal{M}$ and $h_X : Y \to \mathcal{K}$ as follows:

- $h(0) = 0$ and $h(1) = 1$.
- If $x \in X$ then $h(x) = h_X(x)$.
- If $y \in Y$ then $h(y) = h_Y(y)$.

- If s_1 and s_2 are service-expressions, then $h(s_1 \oplus s_2) = h_Y(s_1) \oplus h_Y(s_2)$ and $h(s_1 \circledast s_2) = h_Y(s_1) \circledast h_Y(s_2)$.
- If a and b are federated how-provenance expressions, then $h(a \oplus b) = h_Y(a) \oplus h_Y(b)$, $h(a \otimes b) = h_Y(a) \otimes h_Y(b)$, and $h(a \ominus b) = h_Y(a) \ominus h_Y(b)$.

▶ **Example 43.** If we replace the pairs (y_i, x_j) in Example 39 with expressions of the form $y_i \circledast x_j$, then we can codify the polynomials on that example with the following federated how-provenance expressions:

$$(\!|Q_O|\!)_\Gamma(\{\texttt{?country} \mapsto \mathsf{Germany}\}) = (y_1 \circledast x_1) \otimes (y_1 \circledast x_2) \otimes ((y_1 \circledast x_3) \oplus (y_2 \circledast x_3)),$$
$$(\!|Q_O|\!)_\Gamma(\{\texttt{?country} \mapsto \mathsf{Hungary}\}) = (y_1 \circledast x_4) \otimes (y_1 \circledast x_5) \otimes (y_2 \circledast x_6).$$

Observe that the first expression in Example 43 includes the sub-expression $(y_1 \circledast x_3) \oplus (y_2 \circledast x_3)$. Intuitively, this sub-expression tells us that the mapping can be alternatively computed using the statement x_3 on service y_1 or on service y_2. According to Definition 41, this expression can be abbreviated as $(y_1 \oplus y_2) \circledast x_3$.

5.2 Annotated Answers for Federated SPARQL queries

So far, we presented an algebraic structure for queries in the federated setting. Now, we will show how to extend the annotated SPARQL algebra by Geerts et al. [22] for the SERVICE operator.

Recall that given an spm-semiring $\mathcal{K}$ over a set of elements K, a K-graph Γ is a function that maps every RDF triple t to a value $\Gamma(t) \in K$. Similarly, we define a federated dataset as a collection of multiple K-graphs.

▶ **Definition 44** (Annotated Federated Dataset)**.** *Let $\mathcal{R} = (R, \oplus, \otimes, \ominus, 0, 1)$ be a federated spm-semiring, $\mathcal{K}$ be the how-structure of $\mathcal{R}$, and Y be a finite set of elements of the fed-structure of $\mathcal{R}$ such that Y is disjoint with $\{0, 1\}$. An* annotated federated dataset *is a partial function Δ that maps each element $y \in Y$ to a $\mathcal{K}$-graph. The set Y is called the set of* service names *of the annotated federated dataset.*

▶ **Example 45.** The graphs G_1 and G_2 of Example 39 encode the following annotated X-graphs:

$$\begin{aligned}
\Gamma_1 = \{\!\!\{ &(\mathsf{Ulm}, \mathsf{a}, \mathsf{City}) \mapsto x_1,\\
&(\mathsf{Ulm}, \mathsf{country}, \mathsf{Germany}) \mapsto x_2,\\
&(\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}) \mapsto x_3,\\
&(\mathsf{Budapest}, \mathsf{a}, \mathsf{City}) \mapsto x_4,\\
&(\mathsf{Budapest}, \mathsf{country}, \mathsf{Hungary}) \mapsto x_5 \}\!\!\},\\
\Gamma_2 = \{\!\!\{ &(\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}) \mapsto x_3,\\
&(\mathsf{Danube}, \mathsf{crosses}, \mathsf{Budapest}) \mapsto x_6 \}\!\!\}.
\end{aligned}$$

The function $\Delta = \{y_1 \mapsto \Gamma_1, y_2 \mapsto \Gamma_2\}$ is an annotated federated dataset.

▶ Remark 46. Recall that there are two ways to query a federated dataset. The first way, described in Section 3.3, consists of evaluating a local query (i.e., a query without SERVICE clauses) over the union of all graphs in the federated dataset. The second way consists of using SERVICE clauses to indicate that certain parts of the query should be evaluated remotely. Hence, in the second way, queries are not local, but remote or fully remote.

Geerts et al. [22] defined an annotated SPARQL algebra (see Definition. 16), which only considers local queries. This algebra can be extended to the evaluation of queries over annotated federated datasets by indicating that one of the services in the federation is used for the local

evaluation via the parameter G of the semantics (see Definition 8). Definition 47 does so by extending Definition 16 for federated datasets and SERVICE clauses and it is inspired by the formalization of the semantics of federated query processing by Buil-Aranda [11]. Hence, it defines an annotated SPARQL algebra which gives a semantics to local, remote and fully remote SPARQL queries over federated datasets.

▶ **Definition 47** (Semantics of the Federated Annotated SPARQL Algebra). *Given an annotated federated dataset Δ, an element $y \in Y$, and a mapping μ, the semantics of FedSPARQL queries is defined recursively as follows:*

- $(\!|(s,p,o)|\!)_{\Delta,y}(\mu) = \Delta(y)(\mu(s,p,o))$,
- $(\!|\text{SELECT } W \text{ WHERE } Q|\!)_{\Delta,y}(\mu) = \bigoplus_{\mu':\mu'|_W=\mu} (\!|Q|\!)_{\Delta,y}(\mu')$,
- $(\!|Q \text{ FILTER } \varphi|\!)_{\Delta,y}(\mu) = (\!|Q|\!)_{\Delta,y}(\mu) \otimes 1_{\mu\models\varphi}$,
- $(\!|Q_1 \text{ UNION } Q_2|\!)_{\Delta,y}(\mu) = (\!|Q_1|\!)_{\Delta,y}(\mu) \oplus (\!|Q_2|\!)_{\Delta,y}(\mu)$,
- $(\!|Q_1 \text{ AND } Q_2|\!)_{\Delta,y}(\mu) = \bigoplus_{\mu=\mu_1\cup\mu_2} ((\!|Q_1|\!)_{\Delta,y}(\mu_1) \otimes (\!|Q_2|\!)_{\Delta,y}(\mu_2))$,
- $(\!|Q_1 \text{ DIFF } Q_2|\!)_{\Delta,y}(\mu) = (\!|Q_1|\!)_{\Delta,y}(\mu) \ominus (\bigoplus_{\mu'\sim\mu} (\!|Q_2|\!)_{\Delta,y}(\mu'))$,
- $(\!|Q_1 \text{ OPTIONAL } Q_2|\!)_{\Delta,y}(\mu) = (\!|Q_1 \text{ AND } Q_2|\!)_{\Delta,y}(\mu) \oplus (\!|Q_1 \text{ DIFF } Q_2|\!)_{\Delta,y}(\mu)$,
- $(\!|(\text{SERVICE } y' \ Q)|\!)_{\Delta,y}(\mu) = y' \circledast (\!|Q|\!)_{\Delta,y'}(\mu)$,

where $\bigoplus$ denotes sums using the operation $\oplus$, $\sim$ denotes mapping compatibility, and $\mu'|_W$ is the projection of mapping μ' on the variables in W. Two mappings μ and μ' are compatible if $\mu(\texttt{?x}) = \mu'(\texttt{?x})$ for every variable $\texttt{?x} \in \text{dom}(\mu) \cap \text{dom}(\mu')$.

Definition 47 differs from the definition of the annotated SPARQL algebra in three cases. The first case is the evaluation of a triple pattern over a federated graph $\Gamma = \Delta(y)$. The second case defines the SERVICE operation. The current graph identified by y is replaced by another graph y', and this replacement is annotated in the how-provenance expression with the operation $\circledast$.

▶ **Example 48.** Consider the annotated federated dataset Δ from Example 45, and let Q be the query

$$((\texttt{?city}, \mathsf{a}, \mathsf{City}) \text{ AND } (\text{SERVICE } y_2 \ (\mathsf{Danube}, \mathsf{crosses}, \texttt{?city}))),$$

and μ be the mapping $\{\texttt{?city} \mapsto \mathsf{Ulm}\}$. Then, the first triple pattern is locally evaluated on the annotated graph $\Delta(y_1) = \Gamma_1$:

$$(\!|(\texttt{?city}, \mathsf{a}, \mathsf{City})|\!)_{\Delta,y_1}(\mu) = \Delta(y_1)(\mu(\texttt{?city}, \mathsf{a}, \mathsf{City})) = x_1.$$

The service clause in the second part of the query is remotely evaluated on the annotated graph $\Delta(y_2) = \Gamma_2$:

$$\begin{aligned}(\!|(\text{SERVICE } y_2 \ (\mathsf{Danube}, \mathsf{crosses}, \texttt{?city}))|\!)_{\Delta,y_1}(\mu) &= y_2 \circledast (\!|(\mathsf{Danube}, \mathsf{crosses}, \texttt{?city})|\!)_{\Delta,y_2}(\mu) \\ &= y_2 \circledast \Delta(y_2)(\mu(\mathsf{Danube}, \mathsf{crosses}, \texttt{?city})) \\ &= y_2 \circledast x_3.\end{aligned}$$

Then, $(\!|Q|\!)_{\Delta,y_1}(\mu) = x_1 \otimes y_2 \circledast x_3$.

5.3 How-Provenance for Federated Query Evaluation

In the preliminaries (Section 3.3), we described the evaluation of a local SPARQL query Q over multiple RDF graphs $G_1, G_2, \ldots, G_n$, as the evaluation of the query over the union of these graphs. That is, the result is the set of mappings

$$[\![Q]\!]_{\bigcup_{i=1}^n G_i}.$$

We called this process the *federated query evaluation* of query Q. However, this definition does not consider the how-provenance of the computed solutions. In Section 5.2, we introduced the federated annotated SPARQL algebra to provide a theoretical framework for how-provenance on federated SPARQL endpoints using the SERVICE clause. In this section, we will connect the notion of federated query evaluation with the federated annotated SPARQL algebra by presenting a simple query rewriting from local queries to remote queries. Doing so, we will formalize the notion of federated how-provenance for the federated query evaluation.

▶ **Definition 49** (Federated Graph)**.** *Let Δ be an annotated federated dataset with set of service names Y. The* federated graph *of Δ is the annotated graph, denoted* $\mathrm{FedGraph}(\Delta)$*, such that for every triple t,* $\mathrm{FedGraph}(\Delta)(t) = \bigoplus_{y \in Y} y \circledast \Gamma(t)$.

▶ **Example 50.** Intuitively, the annotated graph of an annotated federated dataset combines the annotations of the triples using the operation $\oplus$ to indicate the alternative explanations of each triple. The annotated graph for the annotated federated dataset Δ in Example 45 is the following:

$$\begin{aligned}\mathrm{FedGraph}(\Delta) = \{&(\mathsf{Ulm}, \mathsf{a}, \mathsf{City}) \mapsto y_1 \circledast x_1,\\ &(\mathsf{Ulm}, \mathsf{country}, \mathsf{Germany}) \mapsto y_1 \circledast x_2,\\ &(\mathsf{Danube}, \mathsf{crosses}, \mathsf{Ulm}) \mapsto (y_1 \circledast x_3) \oplus (y_2 \circledast x_3),\\ &(\mathsf{Budapest}, \mathsf{a}, \mathsf{City}) \mapsto y_1 \circledast x_4,\\ &(\mathsf{Budapest}, \mathsf{country}, \mathsf{Hungary}) \mapsto y_1 \circledast x_5,\\ &(\mathsf{Danube}, \mathsf{crosses}, \mathsf{Budapest}) \mapsto y_2 \circledast x_6\}.\end{aligned}$$

▶ **Definition 51** (Federated Query)**.** *Let Δ be an annotated federated dataset with set of service names $Y = \{y_1, \ldots, y_n\}$, and Q be a local query. The* federated query *of Q over Δ, denoted* $\mathrm{FedQuery}_\Delta(Q)$*, is the query that results from replacing in Q every triple pattern t with the query*

$$((\text{SERVICE } y_1\ t) \text{ UNION } \cdots \text{ UNION } (\text{SERVICE } y_n\ t)).$$

Intuitively, the federated query combines the evaluation of each triple pattern in the different SPARQL endpoints of an annotated federated dataset. The following lemma connects the evaluation of queries on a federated graph with the evaluation of queries in an annotated federated dataset.

▶ **Lemma 52.** *Let Δ be an annotated federated dataset with set of service names Y, and Q be a local SPARQL query. Then, for every mapping μ, and service name $y \in Y$.*

$$(\!|Q|\!)_{\mathrm{FedGraph}(\Delta)}(\mu) = (\!|\mathrm{FedQuery}_\Delta(Q)|\!)_{\Delta,y}(\mu).$$

Proof. It follows directly from definitions 47 and 51. ◀

▶ **Example 53.** Consider the annotated federated dataset $\Delta = \{y_1 \mapsto \Gamma_1, y_2 \mapsto \Gamma_2\}$, the local query

$$Q = ((\texttt{?city}, \mathsf{a}, \mathsf{City}) \text{ AND } (\mathsf{Danube}, \mathsf{crosses}, \texttt{?city})),$$

and the mapping $\mu\{\texttt{?city} \mapsto \mathsf{Ulm}\}$. Then, the polynomial annotation of Q in the federated graph of Δ (see Example 50) is

$$(\!|Q|\!)_{\mathrm{FedGraph}(\Delta)}(\mu) = (y_1 \circledast x_1) \otimes ((y_1 \circledast x_3) \oplus (y_2 \circledast x_3)).$$

Alternatively, the federated query for Q is:

$$\begin{aligned}\text{FedQuery}_Q(\Delta) = (((&\text{SERVICE } y_1\ (\texttt{?city}, \mathsf{a}, \mathsf{City})) \text{ UNION}\\ &(\text{SERVICE } y_2\ (\texttt{?city}, \mathsf{a}, \mathsf{City}))) \text{ AND}\\ ((&\text{SERVICE } y_1\ (\mathsf{Danube}, \mathsf{crosses}, \texttt{?city})) \text{ UNION}\\ &(\text{SERVICE } y_2\ (\mathsf{Danube}, \mathsf{crosses}, \texttt{?city}))).\end{aligned}$$

Then, evaluating query $\text{FedQuery}_Q(\Delta)$ on dataset Δ we obtain:

$$(\!|\text{FedQuery}_Q(\Delta)|\!)_\Delta(\mu) = (y_1 \circledast x_1) \otimes ((y_1 \circledast x_3) \oplus (y_2 \circledast x_3)).$$

▶ Note 54. The main implication of Lemma 52 is that we can evaluate the provenance of the evaluation of a local query over an annotated federated dataset without the need of transforming the annotated federated dataset into an annotated graph, but by rewriting the local query. Also, the equivalence of the evaluation of the rewritten query with the evaluation of the local query tells us that the federated evaluation follows the framework of spm-semirings in the non-federated setting.

5.4 Federated Provenance Computation

We have all the fundamentals to extend NPCS for federated provenance computation. By using a query rewriting, NPCS employs a middleware that takes a SPARQL query and generates a new query designed to retrieve provenance information. The resulting query seamlessly executes on a singular instance of a SPARQL endpoint, yielding the desired results along with their how-provenance algebraic expressions. Expanding upon this foundation, our extended approach, *Fed-NPCS*, can be integrated into federation engines. Federation engines, such as FedX [42] and FedUP [3], rewrite an *input query* into a set of subqueries that are executed in different endpoints of the federation. To this end, these federated engines execute queries in two phases. First, in a planning phase, they execute queries or use summaries to elaborate a plan to retrieve data from the different endpoints. This plan includes the definition of subqueries, the endpoints where these subqueries will be evaluated, and the operations to combine the retrieved answers. Second, the engines execute the plans. Our approach, Fed-NPCS, translates the plan generated by these federated engines into a single SPARQL query, called the *federated plan query*. The federated plan query includes SERVICE clauses that encode the execution of the subqueries in the corresponding endpoints, and SPARQL operations to encode the combination of the answers from the subqueries.

The process described so far does not include the federated how-provenance computation but the generation of the federated plan query that can compute the answers to the original query on the federation. To compute the how-provenance data, Fed-NPCS rewrites the federated plan query into a query called the *federated provenance query*. Like in NPCS, Fed-NPCS's federated provenance query is designed to retrieve provenance information.

▶ Remark 55. Unlike the NPCS system, which considers two queries, the Fed-NPCS considers three queries: (1) the input query (Q_O) is the original query sent to NPCS; (2) the federated plan query (Q_F) is the plan generated by the federation engine (e.g., FedX or FedUP) to compute the answers of the input query; and (3) the federated provenance query (Q_R) is the query designed to retrieve provenance information on top of the federated query plan.

▶ Remark 56. Lemma 52 showed that the local query Q_O can be evaluated in a federation of annotated graphs Δ by rewriting to the query $\text{FedQuery}_\Delta(Q_O)$. This query $\text{FedQuery}_\Delta(Q_O)$ is equivalent to the query Q_F generated by a federation engine. The difference is that $\text{FedQuery}_\Delta(Q_O)$

is a simple query that satisfies Lemma 52, and query Q_F is optimized for a fast execution. Heling and Acosta [29] showed that naive decompositions as the one presented in this work are sound and complete for exclusive groups. For arbitrary decompositions, such guarantees are not proven.

5.4.1 Federated Datasets

The aforementioned annotated federated dataset Δ provides a theoretical framework for the notion of how-provenance in the federated setting. In practice, federated engines do not operate over an annotated federated dataset Δ consisting of annotated graphs Γ, but over regular federated datasets D consisting of regular graphs G. Thus, to compute the how-provenance, the input is a regular dataset D which encodes Δ. Like NPCS, Fed-NPCS requires reification to encode each annotated graph Γ with a regular graph G. In this subsection we describe such an encoding.

▶ **Definition 57** (Encoding of the annotated federated dataset)**.** *Let X and Y be two disjoint sets of IRIs, Δ be an annotated federated dataset with set of service names Y, where for each $y \in Y$, $\Delta(y)$ is an X-graph. Given a reification scheme* Reify*, the* encoding of Δ *with the reification scheme* Reify *is the federated dataset D that maps each IRI $y \in Y$ to the RDF-star graph G defined as follows:*

$$G = \{\mathsf{Reify}(t, \Gamma(t)) \mid t \in \mathrm{supp}(\Gamma)\},$$

where Γ is the X-graph $\Delta(y)$.

▶ **Example 58.** Consider the reification scheme $\mathsf{Reify}(t, x) = (t, \mathsf{wasDerivedFrom}, x)$, and the federated dataset Δ described in Example 45. Then, the encoding of Δ with the reification scheme Reify is the RDF-star dataset $D = \{y_1 \mapsto G_1, y_2 \mapsto G_2\}$, where G_1 and G_2 are the RDF-star graphs described in Example 39.

5.4.2 Query rewriting design

Recall (Section 4) that NPCS computes how-provenance of a query Q_O over an RDF graph G that uses reification and that results in an X-graph Γ. The provenance polynomial expression is bound to an additional SPARQL variable `?prov` for each mapping μ in the support of $(\!|Q_O|\!)_\Gamma$. To this end, NPCS translates the local SPARQL query Q_O into another local SPARQL query Q_R whose answers are extended with provenance annotations. Formally, the answers in the set $[\![Q_R]\!]_G$ have the form $\mu \cup \{\texttt{?prov} \mapsto k\}$, where k is the how-provenance polynomial of μ (i.e., $(\!|Q_O|\!)_\Gamma(\mu) = k$). Similarly, our goal is to define a method, called Fed-NPCS, that rewrites every local SPARQL query Q_O into a fully remote SPARQL query Q_R such that the answers of Q_R over a federated dateset D have the form $\mu \cup \{\texttt{?prov} \mapsto k\}$ and $(\!|Q_O|\!)_\Delta(\mu) = k$, where Δ is the annotated federated dataset encoded by the federated dataset D.

Fed-NPCS uses two steps to translate the original query Q_O into the query Q_R that computes the provenance:

1. In the first step, Fed-NPCS translates the original query Q_O into an intermediate query Q_F which returns the expected answers of query Q_O over a federated dataset. For example, this intermediate query can be $Q_F = \mathrm{FedQuery}_\Delta(Q_O)$ (see Definition 51). Indeed, by Lemma 52, query Q_F returns the same answer as the federated evaluation of query Q_O. The intermediary query Q_F used in this work uses a naive decomposition that is proven to be sound and complete for exclusive groups [29].
2. In the second step, Fed-NPCS translates Q_F into the desired query Q_R that computes the provenance. This step is equivalent to what NPCS does on non-federated queries.

In the remainder of this section we will describe these two steps.

Step 1: The federated engine plan

FedQuery$_\Delta(Q_O)$ is a simple alternative to the definition of the intermediate query Q_F. However, this query does not take advantage of the distribution of the data across the endpoints in the federation. Federated query engines, such as FedX [42] and FedUP [3], define more efficient query plans for the federated query evaluation, which can be encoded into intermediate queries Q_F to improve the efficiency of Fed-NPCS.

These query plans reflect strategic decisions informed by the structure and content of the underlying federation. In practice, federation engines must determine which endpoints are likely to return useful results for each triple pattern or basic graph pattern. To do so, they rely on techniques such as metadata inspection, runtime source probing, or precomputed summaries to avoid contacting irrelevant federation members. This process, often referred to as *source selection*, plays a critical role in scaling to federations with a large number of endpoints.

After identifying relevant sources, federation engines apply *query decomposition* to assign subqueries to endpoints and construct an execution plan that minimizes intermediate result sizes and total query latency. Fed-NPCS translates these assignments subqueries-endpoint to SERVICE clauses.

▶ **Example 59.** Let $Q_O = (T_1 \text{ AND } T_2 \text{ AND } T_3)$ be a SPARQL query composed of the three triple patterns T_1, T_2, and T_3. Then, the simplest way to define a intermediate query Q_F is using the federated query FedQuery$_\Delta(Q_O)$. If Δ includes three service names, y_1, y_2, and y_3, then Q_F is:

$$\begin{aligned} Q_F = (&((\text{SERVICE } y_1\ T_1) \text{ UNION } (\text{SERVICE } y_2\ T_1) \text{ UNION } (\text{SERVICE } y_3\ T_1)) \text{ AND} \\ &((\text{SERVICE } y_1\ T_2) \text{ UNION } (\text{SERVICE } y_2\ T_2) \text{ UNION } (\text{SERVICE } y_3\ T_2)) \text{ AND} \\ &((\text{SERVICE } y_1\ T_3) \text{ UNION } (\text{SERVICE } y_2\ T_3) \text{ UNION } (\text{SERVICE } y_3\ T_3))). \end{aligned}$$

If we are informed that service y_3 has no answers for triple patterns T_1 and T_2, and that services y_1 and y_2 have no answers for triple pattern T_3, then we can rewrite Q_F as follows:

$$\begin{aligned} Q'_F = (&((\text{SERVICE } y_1\ T_1) \text{ UNION } (\text{SERVICE } y_2\ T_1)) \text{ AND} \\ &((\text{SERVICE } y_1\ T_2) \text{ UNION } (\text{SERVICE } y_2\ T_2)) \text{ AND} \\ &(\text{SERVICE } y_3\ T_3)). \end{aligned}$$

If we additionally know that each answer μ_1 of triple pattern T_1 in y_1 is incompatible with all the answers of triple pattern T_2 in service y_2, and similarly, each answer μ_2 to triple pattern T_1 in y_2 is incompatible with all the answers to triple pattern T_2 in service y_1, then we can rewrite Q_F as follows:

$$\begin{aligned} Q''_F = (&((\text{SERVICE } y_1\ (T_1 \text{ AND } T_2))) \text{ UNION} \\ &((\text{SERVICE } y_2\ (T_1 \text{ AND } T_2)))) \text{ AND} \\ &(\text{SERVICE } y_3\ T_3)). \end{aligned}$$

Queries Q'_F and Q''_F require less time to be executed than query Q_F because they avoid unnecessary pattern evaluation, and reduce network data transfer because of evaluating the joins in the service for which the basic graph patterns are exclusive.

▶ Remark 60. The queries Q_F, Q'_F, and Q''_F in Example 59 are not equivalent in general, but under the information described in the example. Federated engines use availability and pattern compatibility information across endpoints to generate efficient query plans. These efficient plans discard services that are known to do not return answers to a triple pattern or pairs of services that are known to do not produce joinable mappings for a given AND operation.

Step 2: Query Rewriting

After generating an intermediate query Q_F, which includes SERVICE operators, the next step is to rewrite Q_F into a query Q_R that computes the provenance. The difference between a local and a federated query provenance computing lies in the use of the SERVICE operator, which directs portions of the query to be evaluated on external SPARQL endpoints. Our rewriting must ensure that each part of the federated query can be rewritten to preserve both the provenance of intermediate results and the aggregation of these results across sources. To define this query rewriting we first define an auxiliary function that annotates the data from remote services with the service name.

▶ **Definition 61.** *Given a SPARQL variable* `?prov` *and an IRI* y*, the expression* $\text{ServiceOp}(y, \texttt{?prov})$ *is defined as follows:*

$$\text{ServiceOp}(y, \texttt{?prov}) = \texttt{concat("(} \circledast \texttt{", } y\texttt{, ?prov, ")")}.$$

Like the functions in Definition 24, the expression in Definition 61 defines a function to combine polynomial expressions. In particular, the function ServiceOp combines a polynomial with the service name using the service operator.

▶ **Definition 62** (Base Fed-NPCS query rewriting)**.** *Let* Q_F *be a SPARQL query,* `?prov` *a variable, and* Reify *a reification scheme. Then, the rewritten query for* Q_F *and variable* `?prov` *over scheme* Reify*, denoted* $\beta(Q_F, \texttt{?prov})$*, is defined recursively as follows:*

- *If* Q_F *matches one of the rewriting rules in Definition 28 (i.e., the NPCS query rewriting), then* $\beta(Q_F, \texttt{?prov})$ *is the result of applying that rule and then recursively applying rewriting function* β*.*
- *If* Q_F *has the form* (SERVICE y Q) *then* $\beta(Q_F, \texttt{?prov})$ *is the query*

$$\begin{array}{ll}(\ \text{SELECT} & \text{inScope}(Q) \cup \{\texttt{?prov}\} \\ \ \text{WHERE} & ((\text{SERVICE } y\ \beta(Q, \texttt{?prov}\circledast))\ \text{BIND}\ (\text{ServiceOp}(y, \texttt{?prov}\circledast)\ \text{AS}\ \texttt{?prov}))).\end{array}$$

▶ **Example 63.** Consider the federated dataset $D = \{y_1 \mapsto G_1, y_2 \mapsto G_2\}$ from Example 45, and the query

$$Q_F = ((\text{SERVICE } y_1\ (\mathsf{Danube}, \mathsf{crosses}, \texttt{?city}))\ \text{UNION}\ (\text{SERVICE } y_2\ (\mathsf{Danube}, \mathsf{crosses}, \texttt{?city}))).$$

By the rule for the UNION operator (in Definition 28), the rewritten query for Q_F is

$$\begin{array}{lll}\beta(Q_F, \texttt{?prov}) = (\ \text{SELECT} & \texttt{?city}\ (\text{ProvAggSum}(\texttt{?prov}\oplus)\ \text{AS}\ \texttt{?prov}) \\ \ \text{WHERE} & (P_1\ \text{UNION}\ P_2) \\ \ \text{GROUP BY} & \texttt{?city}\ ),\end{array}$$

where, for $i \in \{1, 2\}$, P_i is the query $\beta((\text{SERVICE } y_i\ T), \texttt{?prov}\oplus)$ and T is the triple pattern $(\mathsf{Danube}, \mathsf{crosses}, \texttt{?city})$. These queries P_i are computed using the rule for the SERVICE operator:

$$\begin{array}{ll}P_i = (\ \text{SELECT} & \texttt{?city ?prov}\oplus \\ \ \text{WHERE} & ((\text{SERVICE } y_i\ \beta(T, \texttt{?prov}\oplus\circledast))\ \text{BIND}\ (\text{ServiceOp}(y_i, \texttt{?prov}\oplus\circledast)\ \text{AS}\ \texttt{?prov}\oplus))).\end{array}$$

The query $\beta(T, \texttt{?prov}\oplus\circledast)$ is defined by the rule for triple patterns

$$\begin{array}{lll}\beta(T, \texttt{?prov}\oplus\circledast) = (\ \text{SELECT} & \texttt{?city}\ (\text{ProvAggSum}(\texttt{?prov}\oplus\circledast\oplus)\ \text{AS}\ \texttt{?prov}\oplus\circledast) \\ \ \text{WHERE} & \mathsf{Reify}((\mathsf{Danube}, \mathsf{crosses}, \texttt{?city}), \texttt{?prov}\oplus\circledast\oplus) \\ \ \text{GROUP BY} & \texttt{?city}\ ).\end{array}$$

Then, the query $\beta(T, \texttt{?prov}\oplus\circledast)$ returns the following answers in each service:

$$[\![\beta(T, \texttt{?prov}\oplus\circledast)]\!]_{D,y_1} = \left[\begin{array}{c|c} \texttt{?city} & \texttt{?prov}\oplus\circledast \\ \hline \mathsf{Ulm} & (\oplus\ (x_3)) \end{array}\right],$$

$$[\![\beta(T, \texttt{?prov}\oplus\circledast)]\!]_{D,y_2} = \left[\begin{array}{c|c} \texttt{?city} & \texttt{?prov}\oplus\circledast \\ \hline \mathsf{Ulm} & (\oplus\ (x_3)) \\ \mathsf{Budapest} & (\oplus\ (x_6)) \end{array}\right].$$

Then, the results of evaluating queries P_1 and P_2 are:

$$[\![\beta(P_1, \texttt{?prov}\oplus)]\!]_D = \left[\begin{array}{c|c} \texttt{?city} & \texttt{?prov}\oplus \\ \hline \mathsf{Ulm} & (\circledast\ y_1\ (\oplus\ (x_3))) \end{array}\right],$$

$$[\![\beta(P_2, \texttt{?prov}\oplus)]\!]_D = \left[\begin{array}{c|c} \texttt{?city} & \texttt{?prov}\oplus \\ \hline \mathsf{Ulm} & (\circledast\ y_2\ (\oplus\ (x_3))) \\ \mathsf{Budapest} & (\circledast\ y_2\ (\oplus\ (x_6))) \end{array}\right].$$

Combining these results we obtain:

$$[\![\beta(Q_F, \texttt{?prov})]\!]_D = \left[\begin{array}{c|c} \texttt{?city} & \texttt{?prov} \\ \hline \mathsf{Ulm} & (\oplus\ \ (\circledast\ y_1\ (\oplus\ (x_3)))\ (\circledast\ y_2\ (\oplus\ (x_3)))) \\ \mathsf{Budapest} & (\oplus\ (\circledast\ y_2\ (\oplus\ (x_6)))) \end{array}\right].$$

Thus, the values for variable `?prov` for solutions where variable `?city` takes the values Ulm and Budapest are the Polish notation expressions for the polynomials $(y_1 \circledast x_3) \oplus (y_2 \circledast x_3)$ and $y_2 \circledast x_6$.

Like in NPCS (see Theorem 33), the correctness of the Fed-NPCS's query rewriting is based on the underlying algebra of queries evaluated over annotated datasets.

▶ **Theorem 64.** *Let* Reify *be a reification scheme, and β be the function described in Definition 62. Then, function β is sound and complete for the reification scheme* Reify.

Proof. It can be shown by induction on the structure of the query. Excluding the rule with the SERVICE operator, the proof is the same as for Theorem 33. Let us review the new case and assume a federated dataset D for the corresponding annotated dataset Δ. If a query Q_F has the form (SERVICE y Q) then $\beta(Q_F, \texttt{?prov})$ is the query:

$$\begin{array}{rl} \beta(Q_F, \texttt{?prov}) = (\ \text{SELECT} & \text{inScope}(Q) \cup \{\texttt{?prov}\} \\ \text{WHERE} & ((\text{SERVICE}\ y\ \beta(Q, \texttt{?prov}\circledast))\ \text{BIND}\ (\text{ServiceOp}(y, \texttt{?prov}\circledast)\ \text{AS}\ \texttt{?prov}))). \end{array}$$

First, to show that β is sound, assume that mapping $\mu \cup \{\texttt{?prov} \mapsto y \circledast k\}$ is an answer to query $\beta(Q_F, \texttt{?prov})$. By construction, mapping $\mu \cup \{\texttt{?prov} \mapsto k\}$ is then an answer to query $\beta(Q, \texttt{?prov}\circledast)$. By induction, $(\!|Q|\!)_{\Delta,y}(\mu) = k$. By the semantics of the SERVICE operator (see Definition 47), $(\!|Q_F|\!)_{\Delta,y}(\mu) = y \circledast k$. Hence, β is sound.

Second, to show that β is complete, we use the inverse reasoning. By induction, the answers to query $\beta(Q, \texttt{?prov}\circledast)$ include all mappings $\mu \cup \{\texttt{?prov}\circledast \mapsto k\}$ such that $(\!|Q|\!)_{\Delta,y}(\mu) = k$ and $k \neq 0$. Since $y \circledast 0 = 0$, the answers to query $\beta(Q_F, \texttt{?prov})$ include all solutions $\mu \cup \{\texttt{?prov} \mapsto y \circledast k\}$ such that $y \circledast k \neq 0$. ◀

So far, we have presented a sound and complete base query rewriting to compute provenance for queries evaluated on federated SPARQL endpoints. However, this base query rewriting generates some redundant operations that can be further simplified. Fed-NPCS applies the same techniques used by NPCS and described in Section 4.4 for optimization.

▶ **Example 65.** Consider the query Q_F from Example 63. Wrapping up, the rewritten query $\beta(Q_F, \texttt{?prov})$ is

```
( SELECT   ?city (ProvAggSum(?prov⊕) AS ?prov)
  WHERE    (( SELECT  ?city ?prov⊕
               WHERE   ((SERVICE y1
                           ( SELECT     ?city (ProvAggSum(?prov⊕⊛⊕) AS ?prov⊕⊛)
                             WHERE      Reify((Danube, crosses, ?city), ?prov⊕⊛⊕)
                             GROUP BY   ?city ))
                          BIND (ServiceOp(y1, ?prov⊕⊛) AS ?prov⊕)))
             UNION
             ( SELECT  ?city ?prov⊕
               WHERE   ((SERVICE y2
                           ( SELECT     ?city (ProvAggSum(?prov⊕⊛⊕) AS ?prov⊕⊛)
                             WHERE      Reify((Danube, crosses, ?city), ?prov⊕⊛⊕)
                             GROUP BY   ?city ))
                          BIND (ServiceOp(y2, ?prov⊕⊛) AS ?prov⊕))))
  GROUP BY ?city ).
```

Following the optimization rules, we can rewrite this query as follows:

```
( SELECT   ?city (ProvAggSum(?prov⊕) AS ?prov)
  WHERE    (((SERVICE y1  Reify((Danube, crosses, ?city), ?prov⊕⊛))
               BIND (ServiceOp(y1, ?prov⊕⊛) AS ?prov⊕))
             UNION
             ((SERVICE y2  Reify((Danube, crosses, ?city), ?prov⊕⊛))
               BIND (ServiceOp(y2, ?prov⊕⊛) AS ?prov⊕))
  GROUP BY ?city ).
```

In the spirit of the simplification rule no. 2 in Definition 35 (for chains of UNION operators), we can remove two SELECT operations and obtained a simpler provenance query.

To summarize, our query rewriting framework, Fed-NPCS, extends NPCS to federated queries by introducing the ⊛ operator to track the provenance of intermediate results obtained from different SPARQL endpoints. The extension preserves the soundness and completeness of NPCS, ensuring correct results even when querying across multiple sources. By leveraging existing base rewriting rules and adapting them to handle federations, our approach provides a robust mechanism for executing and combining federated SPARQL queries with annotated provenance information.

6 Evaluation

Our evaluation is structured in two parts. In the first part, i.e., Section 6.1, we evaluate NPCS, our how-provenance solution for SPARQL queries on centralized knowledge graphs using two state-of-the-art benchmark datasets and two SPARQL engines. Then, Section 6.2 provides an evaluation of Fed-NPCS on the FedShop federation benchmark [17] that provides federations of different sizes.

6.1 Evaluation on Centralized KGs (NPCS)

We conducted an extensive evaluation of NPCS's viability for computing how-provenance by assessing the runtime overhead incurred by the rewritten queries on centralized settings. This is measured by comparing the runtime between the original queries without provenance annotations and the queries obtained with our approach.

6.1.1 Experimental Setup

Environment. NPCS was implemented in Java, using the Java Development Kit (JDK) version 11. All the experiments were conducted on a computer with an AMD EPYC 7281 16-core processor, 256GB of RAM, and an 8 TB HDD disk running Ubuntu 18.04.6 LTS. We evaluated NPCS on two widely used RDF/SPARQL engines with support for RDF-star, namely GraphDB[3] (version 10.2.0) and Stardog[4] (version 9.1.0). For all our experiments, we set a timeout of 350 seconds for individual query executions, to ensure consistent results, and reported the average response time of the queries over five executions in a cold setting, i.e., after clearing the disk cache.

Competitor. We compare NPCS with SPARQLprov [30], a state-of-the-art solution for how-provenance in SPARQL, which is also based on query rewriting. We used the implementation provided with the paper [30], and extended it to support the RDF-star reification scheme. SPARQLprov and NPCS compute the same provenance polynomials since they both rely on spm-semirings.

Synthetic Workload. We employed the Watdiv [4] performance benchmark specifically designed for RDF/SPARQL engines. Watdiv provides a data generator that can produce synthetic datasets of varying sizes. Additionally, WatDiv includes 20 SELECT query templates, each comprising 10 instantiated queries. The query templates are categorized into four types: linear queries (L), star queries (S), snowflake-shaped queries (F), and complex queries (C). They are all monotonic queries. We therefore introduced five additional non-monotonic query templates (O) as proposed by [30]. These non-monotonic queries were created by enclosing one of the triple patterns in the linear queries with an OPTIONAL clause. The triple pattern to be enclosed was chosen randomly to ensure a diverse set of non-monotonic query templates.

We evaluated NPCS on the 10M-triple and 100M-triple Watdiv datasets that we reified using the RDF-star and named graphs reification schemes. We excluded the standard reification from the evaluation as it exhibits the worst performance according to [30]. Moreover, we created a 200M-triple dataset by duplicating every triple of the 100M-triple dataset and assigning a second provenance identifier to the duplicates. This dataset simulates a challenging case where triples have been extracted from more than one source.

Real Workload. We tested NPCS and SPARQLprov on the WDBench benchmark [5], which provides real-world data. The benchmark uses 15.2 billion triples encoded using the Wikidata reification scheme from a 2023 Wikidata dump. The benchmark provides more than 800 queries consisting of simple BGPs, some of them with OPTIONAL clauses. We took a sample of 150 queries consisting of 50 single-triple-pattern queries, 50 non-monotonic queries (with OPTIONAL), and 50 monotonic queries with more than one triple pattern. The queries were randomly chosen.

6.1.2 Results

Synthetic Workload. Figure 2 compares the execution times of the original query with those of the rewritten queries produced by NPCS and our competitor SPARQLprov on RDF-star data when using GraphDB and Stardog. We measure the runtimes on the 10M and 100M Watdiv datasets. We first notice that in all cases, rewriting the query to compute how-provenance incurs a performance overhead regarding the execution of the original query. Not surprisingly, the overhead

[3] https://graphdb.ontotext.com/
[4] https://www.stardog.com/

increases with data size, but its behaviour also depends on the query engine. For instance, NPCS's overhead ranges from 20% to 30% in GraphDB, and from 25% to 50% in Stardog. Figure 2 highlights that runtime across query templates exhibits higher variability in Stardog compared to GraphDB. Regardless of the data size and the query engine, templates C3 and F5 are by far the most challenging, and make our competitor SPARQLprov timeout on both GraphDB and Stardog. The complexity of C3 is explained by its large number of intermediate results, whereas for F5 it is caused by the large number of solutions.

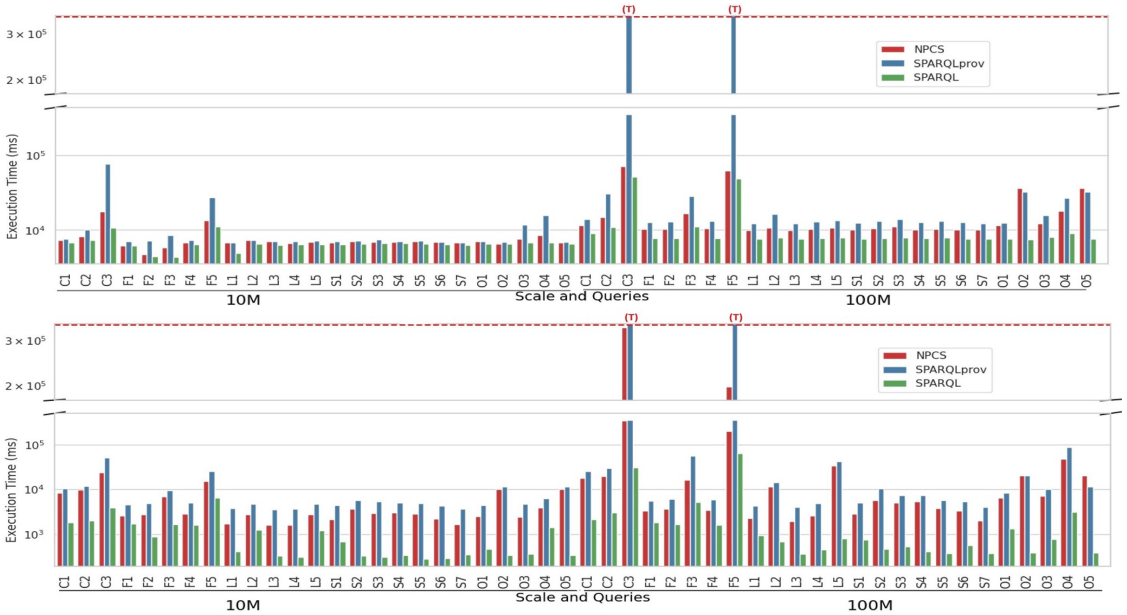


Figure 2 Query execution times on Watdiv 10M and 100M reified with RDF-star on GraphDB (top) and Stardog (bottom). The "SPARQL" label indicates the performance of the query engine without any provenance support.

When we compare the query rewriting strategies, we notice the NPCS consistently outperforms SPARQLprov in 98 out of our 100 studied cases. Also, NPCS is on average 25 times faster than SPARQLprov. One can explain this performance difference by the fact that NPCS is a fully native SPARQL solution, whereas SPARQLprov relies on a post-hoc decoding phase to compute the provenance polynomials. Like NPCS, SPARQLprov rewrites the query to extract provenance information. Unlike our approach, SPARQLprov encodes the structure of the how-provenance annotations in additional columns in the result set. Those additional columns can be numerous and encode the structure of the provenance polynomials. Decoding that information requires running additional group and aggregation operations. Hence, the runtime of this decoding phase is proportional to the number of query solutions times the maximal depth of the operator trees of the provenance annotations. That explains why SPARQLprov times out for query template F5, which is by far the template with the highest number of query solutions (173.6K solutions on average). NPCS, in contrast, carries out the grouping operations during query evaluation, which not only leverages the engine optimizations for grouping, but also makes it easier to deploy in real-world settings.

Despite NPCS's clear runtime advantage, SPARQLprov can exhibit comparable or better performance on very selective queries. This is demonstrated by the runtimes for queries O1, O2, and O5. In cases such as query templates O2 and O5 on GraphDB, NPCS's strategy of evaluating grouping operations in the SPARQL engine does not pay off. This is so because the queries and their constituent triple patterns are very selective.

Figure 3 shows the results for the rewritten queries on the 200M dataset on GraphDB. We observe similar trends as in the 100M scenario, except that SPARQLprov also times out on query templates C2 and F4. We omit the results for Stardog as they exhibit similar behavior as in the 100M dataset.

Finally, we evaluate NPCS on a different reification scheme, namely the popular named graphs strategy. The results are depicted in Figure 4 for the 10M and 100M datasets on GraphDB. We observe the same trends as for the RDF-star reification, that is, NPCS outperforms SPARQLprov consistently in 48 out of 50 studied cases. This shows that our approach is insensitive to the data reification scheme, which makes it applicable to any standard RDF/SPARQL engine. Similar results are observed for Stardog.

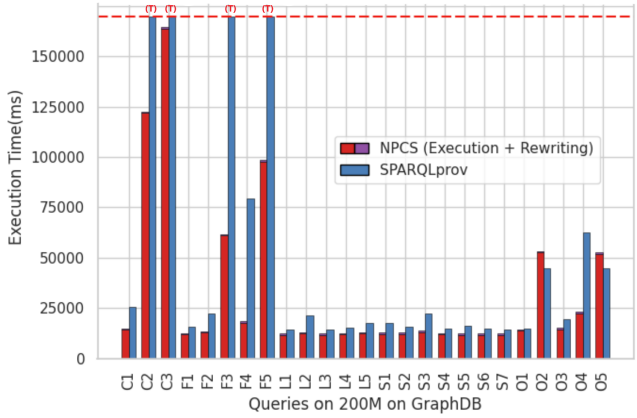


Figure 3 Query execution times on Watdiv 200M reified with RDF-star on GraphDB.

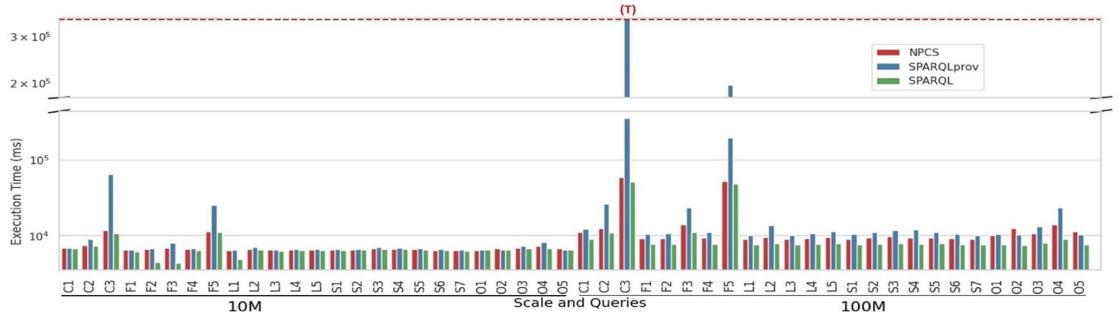


Figure 4 Query execution times on Watdiv 10M and 100M reified as named graphs on GraphDB.

Real Workload. We evaluate NPCS on WDBench. Figure 5 shows the results for GraphDB and Stardog. Each dot in the plot represents the execution of a query, either the original query or a rewritten query by NPCS or by SPARQLprov. Queries are plotted on the x-axis by increasing number of solutions, and the y-axis represents the execution time. We verify the same trend for both engines, namely that SPARQLprov's query rewriting induces a much larger overhead than NPCS's. While the overhead increases with the number of query results for both methods, it is more pronounced for SPARQLprov. This makes SPARQLprov time out when the number of results is above 700K on Stardog (on GraphDB timeouts start after 1M solutions). We also observe that for queries with a few thousand results executed on Stardog, NPCS's overhead can be minimal.

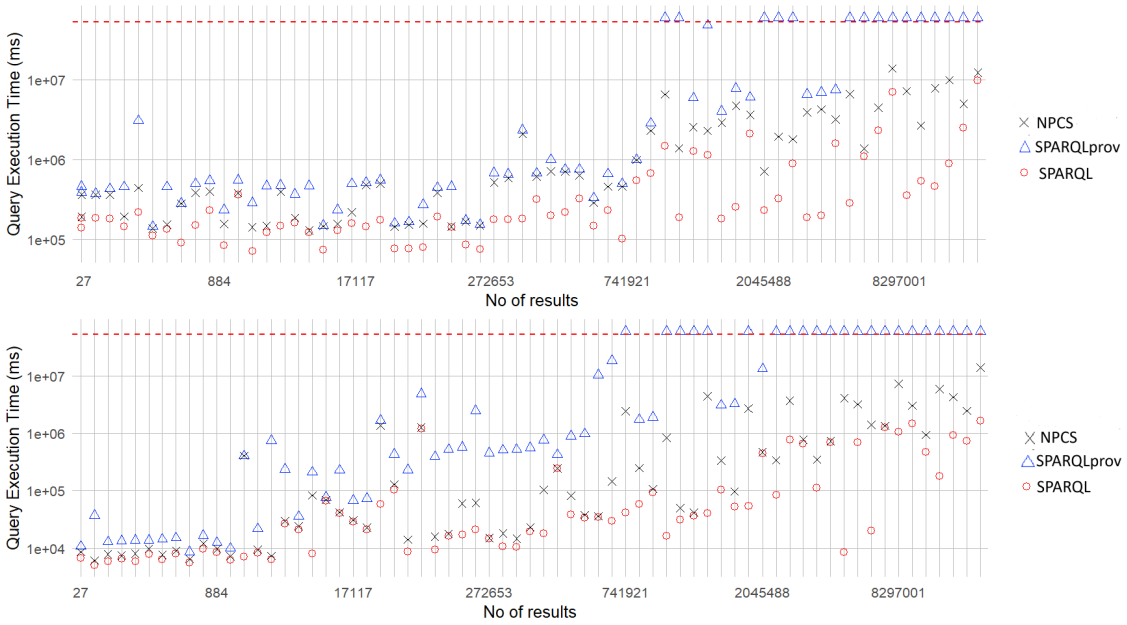


Figure 5 Number of results vs. query execution time for the WDBench queries run on Wikidata, stored in GraphDB (top) and Stardog (bottom), using the Wikidata reification scheme.

6.2 Evaluation on a Federated Setting (Fed-NPCS)

6.2.1 Experimental Setup

Environment. Fed-NPCS was implemented in Java, using the Java Development Kit (JDK) version 11. All the experiments were conducted on the same hardware environment used in the centralized evaluation: a computer with an AMD EPYC 7281 16-core processor, 256GB of RAM, and an 8 TB HDD disk running Ubuntu 18.04.6 LTS. In this evaluation we tested Fed-NPCS on GraphDB[5] (version 10.2.0) as it exhibits better performance than Stardog in Figure 5. To simulate a federated environment, we deployed multiple endpoints hosting different RDF datasets in a distributed manner. Each dataset was hosted on separate instances of the engine, configured to act as individual SPARQL services accessible via the `SERVICE` keyword. The evaluation considered varying the number of distributed endpoints, ranging from 20 to 200, to assess the scalability and efficiency of the rewritten federated queries. As with the centralized evaluation, a timeout of 350 seconds was enforced to individual query executions. For consistency, we report the average response time over five executions in a cold setting (with the disk cache cleared).

Query Workload. We used the FedShop benchmark [17], which is specifically designed to evaluate the scalability of federated RDF/SPARQL query engines. The FedShop schema replicates a virtual catalog across autonomous vendors and rating sites, linking local and global entities using `owl:sameAs` statements, and generates federations of varying sizes using schema-based data generators while maintaining local independence and global interoperability. FedShop provides pre-configured federations and a comprehensive query workload, which we employed for our evaluation. We generated 10 federations $F(N)$ with sizes $N = \{20, 40, \ldots, 200\}$. Each federation has as many vendors as reviewing sites. For instance, $F(200)$ consists of 100 vendors and 100 review sites. All necessary instructions to install, configure, and run the benchmark are available on the FedShop GitHub repository[6].

[5] `https://graphdb.ontotext.com/`
[6] `https://github.com/GDD-Nantes/FedShop.git`

The FedShop's query generator was used to create a workload of 120 queries, each derived from 12 template queries and executed across our 10 federations – from $F(20)$ to $F(200)$ – providing a robust testing environment. For sub-query decomposition and source selection, the FedShop benchmark relies on FedX [42], which follows a unions-first-joins-later (i.e., unions-over-joins) decomposition refined with FedX's exclusive-groups optimization. FedShop provides pre-computed Reference Source Assignments (RSA) on top of this decomposition, which we used as the basis for our evaluation.

The FedShop benchmark is designed to assess the performance and scalability of query federation engines across multiple sources. It categorizes queries based on their data location patterns, namely into, single-domain, multi-domain, and cross-domain queries. These are based on the queries' join variables and on how they combine data from the different sources, and can be explained by the structure of the provenance explanations of their solutions. Single-domain queries are queries without global join variables and with a bounded subject in at least one triple pattern. Since that bounded subject appears only in one of the endpoints, this allows the query to be executed on a single endpoint using one SERVICE clause, as seen for query templates Q9, Q11, and Q12. The provenance of their solutions consists of polynomials with a single $\circledast$ operator, e.g., $e_1 \circledast (s_1 \otimes s_2)$ with endpoint e_1. Multi-domain queries also lack global join variables but do not include bounded subjects, requiring all triple patterns to be grouped into one SERVICE clause, with results potentially retrieved from multiple endpoints. Unlike single-domain queries the resulting provenance explanations include summations of polynomials where each term in the sum is labeled with a different source using the $\circledast$ operator, e.g., $e_1 \circledast (s_1 \otimes s_2) \oplus e_2 \circledast (s_3 \otimes s_4)$. This is the case for query templates Q1, Q2, Q3, Q4, Q6, Q8, and Q10. Cross-domain decomposition involves queries with global join variables linking data across multiple datasets, which requires multiple SERVICE clauses that combine results from several endpoints, as in query templates Q5 and Q7. This translates into how-provenance explanations that contain products labeled with different sources, e.g., $e_1 \circledast (s_1 \otimes s_2) \otimes e_2 \circledast (s_3 \otimes s_4)$.

For our evaluation, we relied entirely on FedShop's query generation, decomposition, and execution framework, enabling a straightforward comparison across varying federation sizes. The benchmark efficiently manages the minimal source selection over federated datasets and provides valuable insights into the performance of our approach. This corresponds to Step 1 in Section 5.4.2; we therefore evaluate the overhead of rewriting the federated plan to augment its results with how-provenance (Step 2).

6.2.2 Results

We evaluated the performance of Fed-NPCS using the FedShop benchmark on a subset of 12 queries. Out of these, four were optional queries (Q2, Q3, Q7, and Q8). Each query consists of nine subqueries, and each subquery was executed 10 times, corresponding to the number of endpoints (from 20 to 200) involved in the execution. Since there is no direct competitor to Fed-NPCS, the only meaningful comparison is with a baseline federated SPARQL plan. This comparison allows us to assess the performance overhead introduced by provenance in a federated setting and to evaluate the scalability of Fed-NPCS with an increasing number of endpoints.

Fed-NPCS demonstrated higher execution times compared to SPARQL in most cases, which can be attributed to the additional overhead of query rewriting. As shown in Figure 8, this overhead consists of two components: (i) query rewriting time, which is minimal across all evaluated configurations (typically $<1\%$ of total runtime at larger scales), and (ii) query execution time on the reified knowledge graph, which dominates the total runtime. To avoid visual distortion that arises from stacked bar charts on a logarithmic scale, Figure 8 presents these two components as percentage contributions on a linear scale, where rewriting time represents at most 15% of total

runtime (WatDiv–GraphDB named graph reification at 10M triples) and decreases substantially as data scale increases. However, as the number of endpoints increased, Fed-NPCS showed a more stable performance pattern, particularly at handling a larger number of results. Below, we provide the performance outcomes for selected queries.

Impact of the number of endpoints. Figure 6 illustrates the execution time in milliseconds for both the SPARQL and Fed-NPCS as the number of endpoints varies from 20 to 200. Each data point represents the average execution time for 12 queries across five runs[7].

The results show a clear trend: as the number of endpoints increases, both SPARQL and Fed-NPCS experience a gradual increase in execution time. Not surprisingly Fed-NPCS takes longer to execute than SPARQL across all federation sizes due to the additional overhead associated with provenance computation, which requires extra resources for tracking and processing query provenance data.

Despite this overhead, Fed-NPCS exhibits a relatively stable and predictable growth pattern in execution time. Both systems demonstrate near-linear scalability as the endpoint count rises. Notably, the gap between SPARQL and Fed-NPCS widens slightly at higher endpoint counts as query processing on larger federations incurs the exchange of more query results – both intermediate and final. This overhead reaches 10.3% for the 200-source federation.

In short, while Fed-NPCS incurs higher execution costs than SPARQL due to provenance computation, its consistent scalability suggests it can still be viable for large federations where provenance tracking is required. This comparison highlights the trade-off between execution speed and the added value of provenance tracking in distributed query processing scenarios.

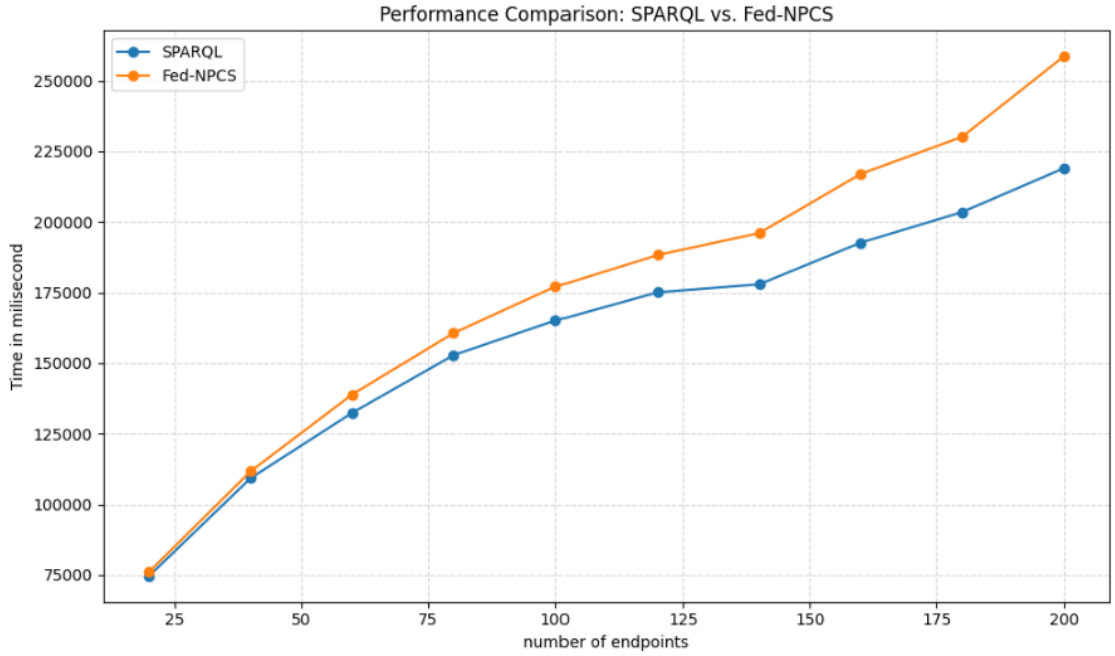


Figure 6 Average execution time of the different FedShop query templates (y-axis) vs. the size of the federation (x-axis).

[7] In reality we run each query five times and dropped the first measurement as it reports the response time in a cold setting.

Individual query analysis. Figure 7 presents the execution time comparison between the baseline SPARQL engine and Fed-NPCS across 12 queries on 200 endpoints. The y-axis is in logarithmic scale and is also split to accommodate for the wide range of execution times.

Notably, for complex queries (such as Q5 and Q6), Fed-NPCS exhibits substantially higher execution times compared to plain SPARQL. However, for less intensive queries (like Q8-Q12 that are not cross-domain), the difference in execution times is less pronounced.

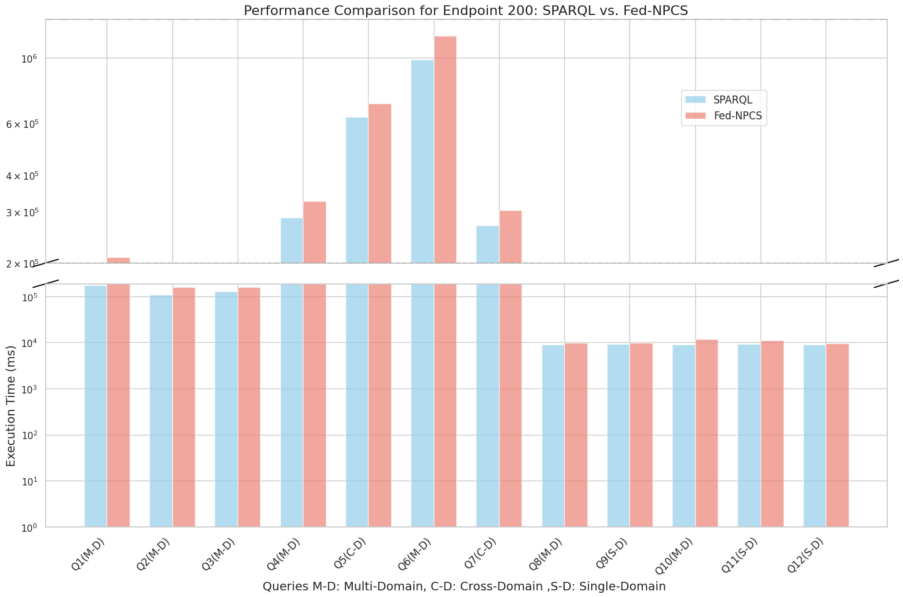


Figure 7 Average query template runtime for the original SPARQL queries vs. the Fed-NPCS queries on a FedShop federation with 200 sources.

The performance of individual queries reflects varying levels of complexity and data distribution. Notably, Q6 shows the highest execution time among all queries, as it is a multi-domain query involving the largest number of results. The high volume of intermediate results significantly contributes to the execution time for both SPARQL and Fed-NPCS, though the latter incurs additional overhead due to provenance computation.

The cross-domain queries, such as Q5 and Q7, also exhibit longer execution times as they involve extensive data joining across federated endpoints. The performance of the multi-domain queries Q1-Q4 and Q8, is mainly explained by their respective numbers of intermediate and final results. For example, queries with more intermediate results tend to require more computation, leading to higher execution times.

Conversely, the single-domain queries, such as Q10, Q11, and Q12, are simpler and thus exhibit the shortest execution times. Both SPARQL and Fed-NPCS perform comparably on these queries, with minimal overhead observed in Fed-NPCS. This indicates that, while the Fed-NPCS introduces additional computation, its impact is less pronounced for simpler queries that do not involve complex joins or provenance tracking across domains.

Overall, while the Fed-NPCS consistently incurs additional overhead due to provenance computation, the resulting system scales to complex multi-domain and cross-domain queries. This makes it a suitable choice for large-scale federated environments where provenance tracking is a priority.

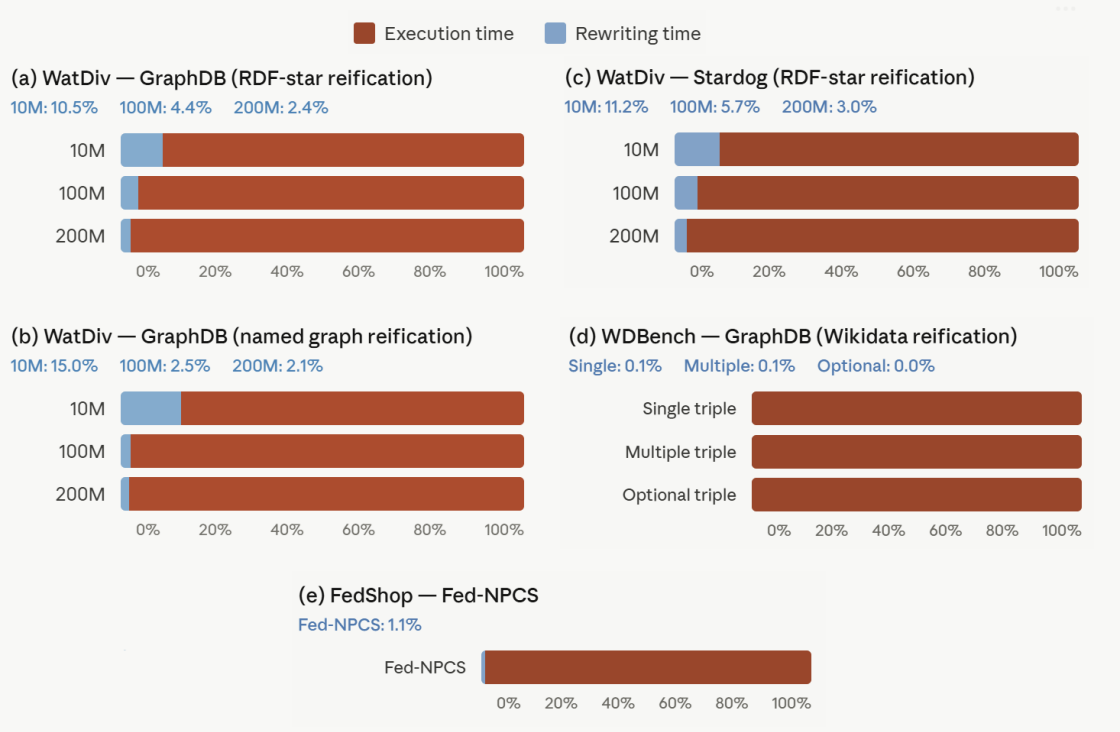


Figure 8 Breakdown of total query runtime into rewriting time (blue) and execution time (red), expressed as percentages. Rewriting time is consistently negligible, particularly at larger data scales.

7 Conclusions

We have proposed NPCS, a novel query rewriting method to compute how-provenance annotations for SPARQL query results. To the best of our knowledge, NPCS is the first 100% SPARQL-based solution for how-provenance. NPCS can be easily applied to standard and already deployed RDF/SPARQL engines, without the need for customized extensions or post-processing steps. Moreover, we introduce NPCS's federated counterpart, Fed-NPCS, which ports this idea to federations of SPARQL endpoints; this is also a novel extension, as, to our knowledge, there is currently no other engine allowing the computation of provenance information over federations.

Our experimental evaluation on synthetic and real data shows that NPCS's native SPARQL rewriting outperforms the state of the art in how-provenance for SPARQL queries. The performance gains provided by our method allow us to compute provenance annotations for millions of query results on knowledge graphs with billions of triples. This makes NPCS attractive for ETL processes on large volumes of data – a common scenario for multi-source KG construction and OLAP for KGs [19, 31–33, 35]. But NPCS's performance also makes it feasible to compute how-provenance explanations in federated settings. As shown by our evaluation of Fed-NPCS on the FedShop benchmark, our approach scales to federations with up to 200 sources. We expect this work to open new avenues in the fields of federated query optimization, access control, and data quality.

In this paper the result of a SPARQL query has a column with character strings which represent expressions in ProvExp(X) in Polish notation. For example, a mapping μ can be annotated with an expression $(\oplus\ u_1\ (\otimes\ u_2\ u_3))$ where u_1, u_2, and u_3 are URLs identifying statements. To apply an homomorphism to a spm-semiring $\mathcal{K}$ we can replace these URLs with the respective elements in $\mathcal{K}$ and then apply the corresponding operations on $\mathcal{K}$. This procedure is less efficient than using built-in operations during the query evaluation. For example, for the natural numbers spm-semiring, we can redefine the operations ProvAggSum, ProvProd, and ProvDiff as follows:

$$\text{ProvAggSum}(\texttt{?x}) = \text{sum}(\texttt{?x}),$$
$$\text{ProvProd}(\texttt{?x}_1, \ldots, \texttt{?x}_n) = (\texttt{?x}_1 * \cdots * \texttt{?x}_n),$$
$$\text{ProvDiff}(\texttt{?x}_1, \texttt{?x}_2) = \text{if}(\texttt{?x}_2 = 0, \texttt{?x}_1, 0).$$

If a structure different from $\mathbb{N}$ were used, we could still have redefined these operations based on custom functions supported by the SPARQL engine. Indeed, several SPARQL engines allow the definition of custom functions. Therefore, our query rewriting can be used as the base for an efficient application of homomorphisms. We plan the study of such an extension as future work.

As another future work we intend to work on lazy approaches for how-provenance computation, that is, approaches where provenance is computed for a user-specified set of solutions. This avoids the execution of expensive queries for results that are not of interest of the user. Finally, we argue that the field of query processing and provenance for SPARQL would benefit from the development of new benchmarks specifically designed for provenance evaluation. While existing benchmarks such as FedShop [17] provide query workloads executable under different federation configurations, they do not directly support the evaluation of provenance computation – e.g., they lack ground-truth provenance annotations or queries designed to stress-test provenance systems. New benchmarks addressing these gaps would enable a more systematic comparison of provenance-aware systems across both centralized and federated settings.

Supplementary Material Statement

The source code of NPCS and Fed-NPCS, along with scripts to recreate the experimental setup, all required libraries, queries, and results can be found at `https://github.com/ZubariaForthAcc/NPCS_Fed-NPCS`, and in a long-term preservation archive at the University of Stuttgart [10].

References

1 Maribel Acosta, Olaf Hartig, and Juan Sequeda. Federated RDF query processing. In Albert Zomaya, Javid Taheri, and Sherif Sakr, editors, *Encyclopedia of Big Data Technologies*, pages 1–8. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-319-63962-8_228-2.

2 Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. ANAPSID: an adaptive query processing engine for SPARQL endpoints. In *ISWC (1)*, Lecture Notes in Computer Science, pages 18–34. Springer, 2011. doi:10.1007/978-3-642-25073-6_2.

3 Julien Aimonier-Davat, Brice Nédelec, Minh Hoang Dang, Pascal Molli, and Hala Skaf-Molli. FedUP: Querying large-scale federations of SPARQL endpoints. In *WWW*, pages 2315–2324. ACM, 2024. doi:10.1145/3589334.3645704.

4 Güneş Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. Diversified stress testing of rdf data management systems. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble, editors, *The Semantic Web – ISWC 2014*, pages 197–212, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-11964-9_13.

5 Renzo Angles, Carlos Buil-Aranda, Aidan Hogan, Carlos Rojas, and Domagoj Vrgoc. Wdbench: A wikidata graph query benchmark. In Ulrike Sattler, Aidan Hogan, C. Maria Keet, Valentina Presutti, João Paulo A. Almeida, Hideaki Takeda, Pierre Monnin, Giuseppe Pirrò, and Claudia d'Amato, editors, *The Semantic Web - ISWC 2022 - 21st International Semantic Web Conference, Virtual Event, October 23-27, 2022, Proceedings*, volume 13489 of *Lecture Notes in Computer Science*, pages 714–731, Cham, 2022. Springer. doi:10.1007/978-3-031-19433-7_41.

6 Renzo Angles and Claudio Gutierrez. The expressive power of SPARQL. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, volume 5318 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2008. doi:10.1007/978-3-540-88564-1_8.

7 Renzo Angles and Claudio Gutierrez. The multiset semantics of SPARQL patterns. In Paul Groth, Elena Simperl, Alasdair J. G. Gray, Marta Sabou, Markus Krötzsch, Freddy Lécué, Fabian Flöck, and Yolanda Gil, editors, *The Semantic Web - ISWC 2016 - 15th International Semantic Web*

Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I, volume 9981 of *Lecture Notes in Computer Science*, pages 20–36, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-46523-4_2.

8 Bahareh Sadat Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng. Gprom - A swiss army knife for your provenance needs. *IEEE Data Eng. Bull.*, 41(1):51–62, 2018. URL: http://sites.computer.org/debull/A18mar/p51.pdf.

9 Zubaria Asma, Daniel Hernández, Luis Galárraga, Giorgos Flouris, Irini Fundulaki, and Katja Hose. NPCS: native provenance computation for SPARQL. In Tat-Seng Chua, Chong-Wah Ngo, Ravi Kumar, Hady W. Lauw, and Roy Ka-Wei Lee, editors, *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, pages 2085–2093. ACM, 2024. doi:10.1145/3589334.3645557.

10 Zubaria Asma, Daniel Hernández, Luis Galárraga, Giorgos Flouris, Irini Fundulaki, and Katja Hose. Code for Native Provenance Computation for Federated and Non-Federated SPARQL Queries, 2026. Software (Source Code and Dataset), swhId: swh:1:dir:c3726715a2159edc556ef0df042df5b515a906d6, URL: https://github.com/ZubariaForthAcc/NPCS_Fed-NPCS.

11 Carlos Buil-Aranda. *Federated query processing for the semantic web*. PhD thesis, Technical University of Madrid, Spain, 2014. doi:10.3233/978-1-61499-350-6-i.

12 Carlos Buil-Aranda, Marcelo Arenas, Óscar Corcho, and Axel Polleres. Federating queries in SPARQL 1.1: Syntax, semantics and evaluation. *J. Web Semant.*, 18(1):1–17, 2013. doi:10.1016/j.websem.2012.10.001.

13 Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-44503-X_20.

14 Angelos Charalambidis, Antonis Troumpoukis, and Stasinos Konstantopoulos. SemaGrow: optimizing federated SPARQL queries. In *SEMANTiCS*, pages 121–128. ACM, 2015. doi:10.1145/2814864.2814886.

15 Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, June 2000. doi:10.1145/357775.357777.

16 Carlos Viegas Damásio, Anastasia Analyti, and Grigoris Antoniou. Provenance for SPARQL queries. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, volume 7649 of *Lecture Notes in Computer Science*, pages 625–640, Cham, 2012. Springer. doi:10.1007/978-3-642-35176-1_39.

17 Minh-Hoang Dang, Julien Aimonier-Davat, Pascal Molli, Olaf Hartig, Hala Skaf-Molli, and Yotlan Le Crom. Fedshop: A benchmark for testing the scalability of sparql federation engines. In Terry R. Payne, Valentina Presutti, Guilin Qi, María Poveda-Villalón, Giorgos Stoilos, Laura Hollink, Zoi Kaoudi, Gong Cheng, and Juanzi Li, editors, *The Semantic Web – ISWC 2023: 22nd International Semantic Web Conference*, volume 14266 of *Lecture Notes in Computer Science*, pages 285–301, Cham, 2023. Springer. doi:10.1007/978-3-031-47243-5_16.

18 Renata Queiroz Dividino, Sergej Sizov, Steffen Staab, and Bernhard Schueler. Querying for provenance, trust, uncertainty and other meta knowledge in RDF. *J. Web Semant.*, 7(3):204–219, 2009. doi:10.1016/j.websem.2009.07.004.

19 Luis Galárraga, Kim Ahlstrøm Jakobsen, Katja Hose, and Torben Bach Pedersen. Answering provenance-aware queries on RDF data cubes under memory budgets. In Denny Vrandecic, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I*, volume 11136 of *Lecture Notes in Computer Science*, pages 547–565. Springer, 2018. doi:10.1007/978-3-030-00671-6_32.

20 Garima Gaur, Arnab Bhattacharya, and Srikanta Bedathur. How and why is an answer (still) correct? maintaining provenance in dynamic knowledge graphs. In Mathieu d'Aquin, Stefan Dietze, Claudia Hauff, Edward Curry, and Philippe Cudré-Mauroux, editors, *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 405–414. ACM, ACM, 2020. doi:10.1145/3340531.3411958.

21 Floris Geerts and Antonella Poggi. On database query languages for k-relations. *J. Appl. Log.*, 8(2):173–185, 2010. doi:10.1016/j.jal.2009.09.001.

22 Floris Geerts, Thomas Unger, Grigoris Karvounarakis, Irini Fundulaki, and Vassilis Christophides. Algebraic structures for capturing the provenance of SPARQL queries. *J. ACM*, 63(1):7:1–7:63, 2016. doi:10.1145/2810037.

23 Boris Glavic and Gustavo Alonso. Perm: Processing provenance and data on the same data model through query rewriting. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 174–185, Washington, DC, USA, 2009. IEEE Computer Society. doi:10.1109/ICDE.2009.15.

24 Olaf Görlitz and Steffen Staab. SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In *COLD*, CEUR Workshop Proceedings. CEUR-WS.org, 2011. URL: https://ceur-ws.org/Vol-782/GoerlitzAndStaab_COLD2011.pdf.

25 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In Leonid Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007,*

Beijing, China, pages 31–40, New York, NY, USA, 2007. ACM. doi:10.1145/1265530.1265535.

26 Zhenzhen Gu, Francesco Corcoglioniti, Davide Lanti, Alessandro Mosca, Guohui Xiao, Jing Xiong, and Diego Calvanese. A systematic overview of data federation systems. *Semantic Web*, 15(1):107–165, 2024. doi:10.3233/SW-223201.

27 Olaf Hartig. Querying trust in RDF data with tsparql. In Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Simperl, editors, *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings*, volume 5554 of *Lecture Notes in Computer Science*, pages 5–20, Cham, 2009. Springer. doi:10.1007/978-3-642-02121-3_5.

28 Olaf Hartig, Pierre-Antoine Champin, Gregg Kellogg, and Andy Seaborne. RDF-star and SPARQL-star. Technical report, W3C Community Group Draft Report, December 2021. URL: https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html.

29 Lars Heling and Maribel Acosta. Federated SPARQL query processing over heterogeneous linked data fragments. In Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, WWW '22, pages 1047–1057, New York, NY, USA, 2022. ACM. doi:10.1145/3485447.3511947.

30 Daniel Hernández, Luis Galárraga, and Katja Hose. Computing how-provenance for SPARQL queries via query rewriting. *Proc. VLDB Endow.*, 14(13):3389–3401, 2021. doi:10.14778/3484224.3484235.

31 Dilshod Ibragimov, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi. Processing aggregate queries in a federation of SPARQL endpoints. In *ESWC*, Lecture Notes in Computer Science, pages 269–285. Springer, 2015. doi:10.1007/978-3-319-18818-8_17.

32 Dilshod Ibragimov, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi. Optimizing aggregate SPARQL queries using materialized RDF views. In Paul Groth, Elena Simperl, Alasdair J. G. Gray, Marta Sabou, Markus Krötzsch, Freddy Lécué, Fabian Flöck, and Yolanda Gil, editors, *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, volume 9981 of *Lecture Notes in Computer Science*, pages 341–359, 2016. doi:10.1007/978-3-319-46523-4_21.

33 Kim Ahlstrøm Jakobsen, Katja Hose, and Torben Bach Pedersen. Towards answering provenance-enabled SPARQL queries over RDF data cubes. In *JIST*, Lecture Notes in Computer Science, pages 186–203. Springer, 2016. doi:10.1007/978-3-319-50112-3_14.

34 Roman Kontchakov and Egor V. Kostylev. On expressibility of non-monotone operators in SPARQL. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, pages 369–379. AAAI Press, 2016. URL: https://aaai.org/papers/38-12889-on-expressibility-of-non-monotone-operators-in-sparql/.

35 Matteo Lissandrini, Katja Hose, and Torben Bach Pedersen. Example-driven exploratory analytics over knowledge graphs. In Julia Stoyanovich, Jens Teubner, Nikos Mamoulis, Evaggelia Pitoura, Jan Mühlig, Katja Hose, Sourav S. Bhowmick, and Matteo Lissandrini, editors, *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*, pages 105–117. OpenProceedings.org, 2023. doi:10.48786/edbt.2023.09.

36 Gabriela Montoya, Hala Skaf-Molli, and Katja Hose. The Odyssey Approach for Optimizing Federated SPARQL Queries. In *ISWC (1)*, Lecture Notes in Computer Science, pages 471–489. Springer, 2017. doi:10.1007/978-3-319-68288-4_28.

37 Bofeng Pan, Natalia Stakhanova, and Suprio Ray. Data provenance in security and privacy. *ACM Comput. Surv.*, 55(14s), July 2023. doi:10.1145/3593294.

38 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009. doi:10.1145/1567274.1567278.

39 Bastian Quilitz and Ulf Leser. Querying distributed RDF data sources with SPARQL. In *ESWC*, Lecture Notes in Computer Science, pages 524–538. Springer, 2008. doi:10.1007/978-3-540-68234-9_39.

40 Pingcheng Ruan, Gang Chen, Tien Tuan Anh Dinh, Qian Lin, Beng Chin Ooi, and Meihui Zhang. Fine-grained, secure and efficient data provenance for blockchain. *Proc. VLDB Endow.*, 12(9):975–988, May 2019. doi:10.14778/3329772.3329775.

41 Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation. In *ESWC*, Lecture Notes in Computer Science, pages 176–191. Springer, 2014. doi:10.1007/978-3-319-07443-6_13.

42 Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *ISWC (1)*, volume 7031 of *Lecture Notes in Computer Science*, pages 601–616. Springer, 2011. doi:10.1007/978-3-642-25073-6_38.

43 Aisling Third and John Domingue. Ethics and executability: Tracing decency in decentralised knowledge graph applications. In *ESWC 2023 Workshops and Tutorials. Semantic Methods for Events and Stories (SEMMES)*, volume 3443. CEUR Workshop Proceedings (CEUR-WS.org), July 2023. URL: https://ceur-ws.org/Vol-3443/ESWC%5f2023%5fTrusDeKW%5fpaper%5f3033.pdf.

44 Marcin Wylot, Philippe Cudré-Mauroux, and Paul Groth. Tripleprov: efficient processing of lineage queries in a native RDF store. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 455–466, New York, NY, USA, 2014. ACM. doi:10.1145/2566486.2568014.