Round-Trippable RDF 1.2 Interoperability

Transportation Research Record
0000, Vol. XX(X) 1–15
©National Academy of Sciences:
Transportation Research Board 0000
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/ToBeAssigned
journals.sagepub.com/home/trr
SAGE

Dominik Tomaszuk[1], Pierre-Antoine Champin[2,3]

Abstract

RDF 1.2 introduces *triple terms*, enabling an RDF triple to occur in the object position of another triple. This extension yields two conformance profiles: *RDF 1.2 Full*, allowing triple terms, and *RDF 1.2 Basic*, excluding them. Consequently, interoperability between Basic and Full implementations becomes non-trivial, as Basic processors cannot directly consume graphs containing triple terms. In response, the W3C Group Note Draft *RDF 1.2 Interoperability*, published by the RDF & SPARQL Working Group, describes a non-normative pair of transformations: *basic encoding* (Full → Basic) and a corresponding decoding (Basic → Full). This paper provides a mathematically precise account of that proposal, grounded in the RDF 1.2 abstract syntax and RDF 1.2 model-theoretic semantics. We formalize the syntax of triple terms, define recursive "appearance" of terms, specify encoding/decoding as functions on graphs and datasets, establish invertibility and idempotence results under explicit admissibility conditions, and briefly clarify their semantic status under standard RDF semantics.

Keywords: RDF 1.2, triple terms, interoperability, basic encoding, model-theoretic semantics, entailment

1 Introduction

The Resource Description Framework (RDF) [*1*] is a foundational data model for representing and exchanging graph-structured information on the Web. In its classical form, RDF expresses knowledge as a set of subject–predicate–object triples, where each triple asserts a directed relationship between two resources. This minimal unit of description has proven remarkably effective for interoperability: it enables a broad ecosystem of tools for parsing, querying, reasoning, validation, and integration across domains. At the same time, real-world knowledge graphs increasingly require more than plain binary assertions. In practice, many applications must attach provenance, evidence, confidence, qualifiers, attribution, temporal validity, and change tracking to *individual* assertions, not only to datasets or graphs as a whole [*2–4*]. These requirements arise naturally in settings such as data integration pipelines, scientific knowledge management, and enterprise governance, where traceability and accountability are first-class concerns [*5*].

For example, an employment graph may assert `:alice :worksFor :acme`, while a registry record `:claim1` supplies the source for precisely that assertion. The intended statement-level links can be visualized by writing $tt =$ $(\texttt{:alice}, \texttt{:worksFor}, \texttt{:acme})$ and:

$$\begin{aligned} \texttt{:claim1} &\xrightarrow{\texttt{rdf:reifies}} tt, \\ \texttt{:claim1} &\xrightarrow{\texttt{:source}} \texttt{:registry}. \end{aligned}$$

The triple-shaped object on the first line is a triple term; metadata attached to `:claim1` therefore concerns that proposition rather than the graph as a whole. We reuse this scenario as a running example in Example 3.8.

In RDF 1.1, statement-level annotation is typically achieved through modeling idioms that encode an assertion as a resource-like object that can be described. The most widely known technique is standard reification, in which a separate node is introduced to represent a triple and is linked to its constituent subject, predicate, and object. While reification is conceptually simple and fully compatible with RDF 1.1 syntax, it is often considered impractical at scale: it substantially increases graph size, complicates queries due to additional join structure, and typically imposes a representational burden on publishers and consumers [*6, 7*]. More general idioms, such as n-ary relation patterns, provide flexibility for qualifiers but similarly shift the representation

[1]Faculty of Computer Science, University of Bialystok, Białystok, Poland
[2]W3C, Sophia Antipolis, France
[3]University Côte d'Azur, Inria, CNRS, I3S UMR 7271, France

Corresponding author:
Dominik Tomaszuk, d.tomaszuk@uwb.edu.pl

away from the original binary predicate and often require a redesign of query and constraint logic around introduced intermediate nodes [8]. Named graphs, in turn, provide an elegant mechanism for graph-level metadata [9], yet they remain comparatively coarse when distinct metadata must be attached to individual statements. As a result, production knowledge graphs often rely on ad hoc conventions, system-specific extensions, or hybrid patterns that fragment data models and reduce portability across tools and platforms.

RDF 1.2 addresses these limitations by introducing *triple terms* (also informally known as quoted or embedded triples) as first-class citizens of the abstract syntax. Intuitively, a triple term allows a triple itself to be used as a term that can be the object of another triple. This extension enables a direct representation of statement-level identifiers: a resource can be used to denote a particular statement-level entity associated with an assertion, and the proposition expressed by that assertion can be referenced explicitly as a structured term. Crucially, a triple term is not asserted merely by occurring as a term: it can be *referred to* without being stated as true. This design makes it possible to use a *reifier*, i.e., a resource that refers to such a proposition, to describe suggested, disputed, unknown, or context-dependent claims while keeping their status separate from asserted facts [10]. Moreover, multiple independent reifiers can refer to the same proposition, enabling distinct contextualizations (e.g., different sources, time spans, locations, or confidence assessments) without duplicating the underlying statement.

From a conformance perspective, the introduction of triple terms yields two complementary RDF 1.2 profiles. *RDF 1.2 Full* supports triple terms and therefore supports statement-level referencing directly at the level of abstract syntax. *RDF 1.2 Basic* excludes triple terms, preserving a conservative profile aligned with processors that only handle ordinary RDF triples. This split is pragmatically motivated: it enables incremental adoption by allowing Basic implementations to remain valid RDF 1.2 processors, while Full implementations can exploit the additional expressivity. However, the split also creates an interoperability challenge. A Basic-only consumer cannot directly parse or process Full graphs containing triple terms, and data exchange across heterogeneous toolchains may require explicit transformations that bridge these profiles.

The W3C Group Note Draft *RDF 1.2 Interoperability*, published by the RDF & SPARQL Working Group, [11] proposes such a bridge by describing a non-normative pair of transformations: a *basic encoding* that maps an RDF 1.2 Full graph into a Basic graph, and a corresponding decoding procedure that reconstructs triple terms. Note that these algorithms should be understood as one possible interoperability strategy rather than the unique or authoritative mapping between RDF 1.2 Full and RDF 1.2 Basic. Different application contexts and optimization goals may therefore motivate alternative mappings. The key idea of the proposal studied here is structural materialization: each occurring triple term is replaced by a fresh blank node, and a small auxiliary Basic subgraph records the subject, predicate, and object components of the original triple term using a dedicated interoperability vocabulary. Decoding reverses this process by detecting these auxiliary patterns and substituting the reconstructed triple terms back into the graph. Importantly, the goal of these transformations is not to enforce semantic equivalence under entailment regimes; rather, it is to provide a robust *syntax-level* mechanism for profile-compatible exchange and round-trippable serialization in environments where Basic and Full processors must coexist.

This paper provides a mathematically precise account of that interoperability proposal, grounded in the RDF 1.2 abstract syntax and its model-theoretic semantics, confirming that the Working Group's proposal satisfies the design goal set in the Note Draft. We formalize recursive appearance of terms and triple terms, specify encoding and decoding as functions over graphs and datasets, and state explicit admissibility conditions required for invertibility. We then establish information-preservation properties (invertibility and idempotence) under these conditions, providing proof sketches for the main results, and clarify, from a semantic perspective, why entailment preservation is not guaranteed under the standard semantics considered here: the encoding introduces auxiliary blank nodes that are existentially quantified at graph scope, whereas the auxiliary Basic structure is not given an interpretation that recovers the Full truth conditions of triple terms.

Contributions. This paper contributes: (i) a compact formalization of RDF 1.2 Basic and Full syntax with recursively nested triple terms; (ii) a graph-theoretic definition of term "appearance" aligned with the recursive nature of triple terms; (iii) a precise specification of basic encoding/decoding over graphs and datasets, including memoization and freshness; (iv) correctness results (invertibility and idempotence) under explicit admissibility conditions, with proof sketches for the main results; and (v) brief semantic clarifications about the scope and limitations of the proposed transformations under standard RDF semantics.

Organization. Section 2 recalls the RDF 1.2 abstract syntax and the semantic layers needed in the sequel. Section 3 formalizes the interoperability setting and specifies the encoding and decoding procedures. Section 4 establishes correctness properties under admissibility constraints. Section 5 gives exact structural accounting and reports runtime experiments. Section 6 discusses related work, and Section 7 concludes with directions for further work.

2 Preliminaries: RDF 1.2 Syntax and Semantics

This section recalls the core RDF 1.2 abstract syntax and the model-theoretic semantics needed for our formal development [1, 10].

2.1 RDF Terms, Triple Terms, Triples, Graphs, and Datasets

Definition 2.1. Atomic symbols. Let $\mathcal{I}$ be the set of IRIs, $\mathcal{B}$ the set of blank nodes, and $\mathcal{L}$ the set of literals. Assume $\mathcal{I}$, $\mathcal{B}$, and $\mathcal{L}$ are pairwise disjoint.

Definition 2.2. RDF triples (Full) and triple terms. Let $\mathsf{Triple}_{\mathrm{Full}}$ be the least set such that:

(a) if $s \in (\mathcal{I} \cup \mathcal{B})$, $p \in \mathcal{I}$, and $o \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$, then $(s, p, o) \in \mathsf{Triple}_{\mathrm{Full}}$;

(b) if $s \in (\mathcal{I} \cup \mathcal{B})$, $p \in \mathcal{I}$, and $o \in \mathsf{Triple}_{\mathrm{Full}}$, then $(s, p, o) \in \mathsf{Triple}_{\mathrm{Full}}$.

A *triple term* is an RDF triple viewed as an RDF term; we set $\mathcal{TT} := \mathsf{Triple}_{\mathrm{Full}}$. Thus, we identify triple terms with (possibly nested) Full triples; $\mathcal{TT}$ is not a separate syntactic sort.

Definition 2.3. RDF terms. Define the set of RDF terms as:

$$\mathcal{T} := \mathcal{I} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{TT}.$$

Elements of $\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ are called *basic RDF terms*.

Definition 2.4. Nesting depth. Define $\mathrm{depth} : \mathsf{Triple}_{\mathrm{Full}} \to \mathbb{N}$ by structural recursion:

$$\mathrm{depth}(s, p, o) = \begin{cases} 0 & \text{if } o \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}, \\ 1 + \mathrm{depth}(o) & \text{if } o \in \mathcal{TT}. \end{cases}$$

Lemma 2.5. Well-foundedness / no cycles. *Every $t \in \mathsf{Triple}_{\mathrm{Full}}$ has a finite nesting depth $\mathrm{depth}(t)$. In particular, cyclic triple constructions are excluded, and proofs by induction on depth are sound.*

Proof. By the least-set construction of $\mathsf{Triple}_{\mathrm{Full}}$, any $t \in \mathsf{Triple}_{\mathrm{Full}}$ is obtained from a base case (clause (a)) by finitely many applications of clause (b). Each application of clause (b) introduces exactly one additional layer of nesting, hence strictly increases depth by 1. Therefore $\mathrm{depth}(t)$ is finite for every t, and cyclic constructions are excluded.

Definition 2.6. RDF triples (Basic). An *RDF 1.2 Basic triple* is a triple $(s, p, o) \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$. We write $\mathsf{Triple}_{\mathrm{Basic}}$ for the set of all Basic triples.

Definition 2.7. RDF graphs. An RDF 1.2 Full graph is a finite set of Full triples: $\mathsf{Graph}_{\mathrm{Full}} := \mathcal{P}_{\mathrm{fin}}(\mathsf{Triple}_{\mathrm{Full}})$. An RDF 1.2 Basic graph is a finite set of Basic triples: $\mathsf{Graph}_{\mathrm{Basic}} := \mathcal{P}_{\mathrm{fin}}(\mathsf{Triple}_{\mathrm{Basic}})$. Triples in a graph are *asserted* in that graph.

Remark 2.8. Finiteness restriction. Our graphs and datasets are explicitly finite, whereas RDF 1.2 Concepts defines RDF graphs as arbitrary sets of triples. This deliberate restriction narrows the generality of the specification, but it is appropriate for the algorithmic transformations studied here and covers the practically relevant cases considered in our experiments.

Definition 2.9. Appearance of terms and triple terms. Define the set of terms occurring in a triple $t = (s, p, o)$ recursively:

$$\mathrm{Occ}(t) := \{s, p\} \cup \mathrm{Occ}_{\mathrm{obj}}(o),$$

$$\text{where} \quad \mathrm{Occ}_{\mathrm{obj}}(x) = \begin{cases} \{x\} & x \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}, \\ \{x\} \cup \mathrm{Occ}(x) & x \in \mathcal{TT}. \end{cases}$$

For a graph G, let $\mathrm{Occ}(G) := \bigcup_{t \in G} \mathrm{Occ}(t)$. The triple terms *appearing* in G are $\mathrm{AppearingTT}(G) := \mathrm{Occ}(G) \cap \mathcal{TT}$.

Definition 2.10. RDF datasets. An RDF dataset is a pair $D = (G_0, \{(n_i, G_i)\}_{i=1}^{k})$ where G_0 is the default graph and each (n_i, G_i) is a named graph with name $n_i \in \mathcal{I} \cup \mathcal{B}$ and G_i an RDF graph. Graph names are unique within a dataset.

2.2 Model-Theoretic Semantics: Interpretations and Entailment

We recall the minimal *simple interpretation* layer and its entailment, and then note how RDF and RDFS entailment strengthen it via semantic extensions. Ground evaluation supplies the recursive semantic base for triple terms. Non-ground graphs are then evaluated by assigning their blank nodes and reducing the result to that ground case. This distinction is directly relevant to the transformations studied here: Full triple terms receive constructor-based denotations, whereas the fresh blank nodes introduced by basic encoding are existentially assigned resources, which is why syntactic invertibility does not by itself imply semantic equivalence.

Definition 2.11. Simple interpretation. A simple interpretation I is a structure consisting of:

(1) a non-empty set IR (the domain of resources),

(2) a set IP (the set of properties),

(3) a mapping $IS : \mathcal{I} \to (IR \cup IP)$ assigning denotations to IRIs,

(4) a partial mapping $IL : \mathcal{L} \to IR$ assigning referents to literals when defined,

(5) an extension mapping $IEXT : IP \to \mathcal{P}(IR \times IR)$,

(6) an *injective* constructor $IT : IR \times IP \times IR \to IR$ used to interpret ground triple terms.

Definition 2.12. Ground RDF expressions. A basic RDF term is *ground* if it is not a blank node. Thus, IRIs and literals are ground, blank nodes are not. An RDF triple (s, p, o) is ground iff s, p, and o are ground (recursively applying this definition if o is a triple term). An RDF graph is ground iff each of its triples is ground.

Definition 2.13. Truth of ground triples and ground triple terms. Let E be a ground RDF expression. Its interpretation is defined as:

$$\begin{aligned} I(\ell) &= IL(\ell) \quad \text{for literals } \ell, \\ I(u) &= IS(u) \quad \text{for IRIs } u, \\ I((s,p,o)) &= IT(I(s), I(p), I(o)) \\ &\qquad \text{for ground triple terms } (s,p,o). \end{aligned}$$

and a ground triple (s, p, o) is true in I iff $I(p) \in IP$ and $\langle I(s), I(o)\rangle \in IEXT(I(p))$. A ground graph is true iff all its triples are true.

Definition 2.14. Blank nodes as existential variables. Let $A : \mathcal{B} \to IR$ be a mapping from blank nodes in a graph G into the domain of I. Write $[I{+}A]$ for the extension of I that maps blank nodes via A and evaluates triples and graphs accordingly. Then I satisfies G iff $[I{+}A](G)$ is true for *some* such mapping A.

Definition 2.15. Simple entailment. A graph G *simply entails* a graph H (written $G \models_s H$) iff every simple interpretation that satisfies G also satisfies H.

Remark 2.16. Semantic extensions. RDF entailment and RDFS entailment refine simple entailment by imposing additional semantic conditions on interpretations (e.g., for datatype IRIs and the RDF/RDFS vocabulary). We will write $G \models_{\text{RDF}} H$ and $G \models_{\text{RDFS}} H$ to distinguish these regimes.

3 Interoperability Between RDF 1.2 Basic and Full

This section formalizes interoperability between RDF 1.2 Basic and RDF 1.2 Full, i.e., the controlled exchange between graphs that exclude triple terms and graphs that permit them. We present the basic encoding (Full → Basic), which replaces each occurring triple term with a fresh blank node and records its components using a reserved Basic vocabulary, and the corresponding basic decoding (Basic → Full), which reconstructs triple terms by reversing this representation. To guarantee determinism and invertibility, both directions are stated together with explicit admissibility conditions that rule out collisions and ill-formed encodings.

3.1 RDF 1.2 Basic vs Full and the Interoperability Goal

Definition 3.1. Basic and Full profiles. An RDF 1.2 graph conforms to the *Basic* profile iff it contains no triple term; equivalently, all objects are basic RDF terms. Every Basic graph also conforms to the Full profile; a Full graph is non-Basic iff it contains at least one triple term.

Basic processors can consume only Basic graphs, while Full processors can represent and process triple terms. RDF 1.2 Interoperability proposes transformations to allow exchange between the two classes in a controlled manner. The central idea is to replace each occurring triple term with a fresh blank node and attach a small Basic subgraph that stores the triple components explicitly.

3.2 Basic Encoding (Full → Basic)

The basic encoding translates an RDF 1.2 Full graph into a Basic graph by repeatedly eliminating triple terms. Concretely, while a triple term tt appears in the current graph, the procedure mints a fresh blank node b, replaces all occurrences of tt appearing in the graph with b, and adds the PropositionForm quadruple that records the components of tt using the reserved interoperability vocabulary. This is repeated until no triple term appears, yielding a Basic graph. We now present the reserved vocabulary, admissibility conditions, and a deterministic memoizing formulation of the algorithm.

3.2.1 Reserved vocabulary The basic encoding uses a fixed vocabulary:

`rdf:type`,
`rdf:PropositionForm`,
`rdf:propositionFormSubject`,
`rdf:propositionFormPredicate`,
`rdf:propositionFormObject`,

Here the prefix `rdf:` abbreviates the namespace IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#`.

An encoded triple term is represented by a blank node b with:

$(b, \texttt{rdf:type}, \texttt{rdf:PropositionForm})$,
$(b, \texttt{rdf:propositionFormSubject}, s)$,
$(b, \texttt{rdf:propositionFormPredicate}, p)$,
$(b, \texttt{rdf:propositionFormObject}, o)$.

The dedicated PropositionForm vocabulary is used instead of the RDF 1.1 reification vocabulary (`rdf:Statement`, `rdf:subject`, `rdf:predicate`, and `rdf:object`) so that transformation artefacts can be recognized unambiguously. The older vocabulary already occurs in widely deployed datasets and therefore cannot be reserved for this transformation without risking collisions with independently published reification data. By contrast, `rdf:PropositionForm` was introduced specifically for the interoperability transformation and can be treated as reserved for that purpose [*11*].

3.2.2 Admissibility conditions

Because the encoding introduces these triples syntactically, we require input graphs to avoid collisions that would compromise invertibility.

Definition 3.2. Encoding-admissible graphs. A Full graph G is *encoding-admissible* iff it contains no triple of the form $(b, \texttt{rdf:type}, \texttt{rdf:PropositionForm})$ for any blank node b.

Remark 3.3. Why the marker condition is sufficient. The condition need not separately prohibit pre-existing triples with predicates `rdf:propositionFormSubject`, `rdf:propositionFormPredicate`, or `rdf:propositionFormObject`. Let $H = \text{basic_encode}(G)$ and suppose $(b, \texttt{rdf:type}, \texttt{rdf:PropositionForm}) \in H$. The marker $(b, \texttt{rdf:type}, \texttt{rdf:PropositionForm})$ cannot originate in G by encoding admissibility, so b must have been minted by the encoder. Freshness gives $b \notin \text{Occ}(G)$; consequently, no input component triple can have b as its subject. Conversely, a pre-existing blank node used with a component predicate but without the marker never becomes a PropositionForm node. Thus such triples cannot violate component uniqueness after encoding. This is also the marker-only collision condition stated by the interoperability transformation [*11*].

3.2.3 Encoding function

Definition 3.4. PropositionForm quadruple. For a blank node b and a (possibly nested) triple term $tt = (s, p, o)$ define:

$$\begin{aligned}\text{PF}(b, tt) := \{&(b, \texttt{rdf:type}, \texttt{rdf:PropositionForm}),\\ &(b, \texttt{rdf:propositionFormSubject}, s),\\ &(b, \texttt{rdf:propositionFormPredicate}, p),\\ &(b, \texttt{rdf:propositionFormObject}, o)\}.\end{aligned}$$

Definition 3.5. Fresh blank nodes. Let $\mathsf{fresh}(G)$ return a blank node $b \in \mathcal{B}$ such that $b \notin \text{Occ}(G)$.

Definition 3.6. Encoding of triple terms. Let $M : \mathcal{TT} \rightharpoonup \mathcal{B}$ be a partial map (memoization). Define a recursive procedure $\text{encTT}(tt, M, G_i, G)$ returning $(b, \Delta M, E)$ such that b is the blank node replacing tt, ΔM is the finite map of memoization entries newly produced during this call, and E is the Basic fragment added.

If $tt \in \text{dom}(M)$, return $(M(tt), \emptyset, \emptyset)$. Otherwise, let $tt = (s, p, o)$. If $o \in \mathcal{TT}$, first encode o producing $(b_o, \Delta M_o, E_o) = \text{encTT}(o, M, G_i, G)$ and set $o' := b_o$; else $o' := o$ and $(\Delta M_o, E_o) = (\emptyset, \emptyset)$. Let $b := \mathsf{fresh}(G_i \cup G \cup E_o)$, set:

$$E := E_o \cup \text{PF}(b, (s, p, o')), \quad \Delta M := \Delta M_o \cup \{tt \mapsto b\}.$$

Return $(b, \Delta M, E)$.

Definition 3.7. Basic encoding of graphs. Let $\mathcal{D}_{\text{enc}}$ be the union of the set of all Basic graphs and the set of all encoding-admissible Full graphs. Define $\text{basic_encode} : \mathcal{D}_{\text{enc}} \to \mathsf{Graph}_{\text{Basic}}$ as follows. Given G_i, initialize $G_o := \emptyset$ and $M := \emptyset$. For each triple $(s, p, o) \in G_i$:

- If $o \in \mathcal{TT}$, compute $(b, \Delta M, E) = \text{encTT}(o, M, G_i, G_o)$ and update $M := M \cup \Delta M$ and $G_o := G_o \cup E$, then set $o := b$.
- Update $G_o := G_o \cup \{(s, p, o)\}$.

Return G_o.

Example 3.8. Running example. Let $tt = (\texttt{:alice}, \texttt{:worksFor}, \texttt{:acme})$ and let

$$\begin{aligned}G_{\text{ex}} = \{\, &tt,\\ &(\texttt{:claim1}, \texttt{rdf:reifies}, tt),\\ &(\texttt{:claim1}, \texttt{:source}, \texttt{:registry})\,\}.\end{aligned}$$

The occurrence of tt as the object of the `rdf:reifies` triple causes the encoder to mint, say, `_:pf1`. It replaces that term occurrence by `_:pf1` and adds $\text{PF}(\texttt{_:pf1}, tt)$, the four triples recording the type, subject, predicate, and object; the asserted triple $tt \in G_{\text{ex}}$ remains asserted and unchanged. If the same term occurs again as an object, memoization reuses the blank node. Decoding removes exactly the four auxiliary triples and substitutes tt for `_:pf1`, thereby reconstructing G_{ex}.

Remark 3.9. Basicness of the encoding result. For every $G \in \mathcal{D}_{\text{enc}}$, we have $\text{basic_encode}(G) \in \mathsf{Graph}_{\text{Basic}}$. Indeed, every triple term occurring in object position is replaced by a blank node before insertion into G_o, and the auxiliary PropositionForm triples introduced by the encoding also satisfy the Basic-triple shape.

Definition 3.10. Basic encoding of datasets. For a dataset $D = (G_0, \{(n_i, G_i)\})$ whose component graphs lie in $\mathcal{D}_{\text{enc}}$, define:

$$\begin{aligned}\text{basic_encode}(D) := (&\text{basic_encode}(G_0),\\ &\{(n_i, \text{basic_encode}(G_i))\}).\end{aligned}$$

with the global constraint that all minted blank nodes are fresh w.r.t. all RDF terms occurring in any graph of D and w.r.t. all blank-node graph names n_i in D.

3.3 Basic Decoding (Basic → Full)

The basic decoding reverses the encoding by identifying proposition-form blank nodes and reconstructing the corresponding triple terms from their recorded components. It is defined only for unambiguous encodings, i.e., when each proposition-form node has exactly one subject, predicate, and object component, and the resulting reconstruction is well-founded.

3.3.1 Decoding-admissible graphs

Definition 3.11. PropositionForm nodes. For a Basic graph G, define:

$$\mathrm{PFNodes}(G) := \{\, b \in \mathcal{B} \mid (b, \texttt{rdf:type}, \texttt{rdf:PropositionForm}) \in G \,\}.$$

Definition 3.12. Unique component extraction. For $b \in \mathrm{PFNodes}(G)$ define partial functions $\mathrm{subj}_G(b)$, $\mathrm{pred}_G(b)$, $\mathrm{obj}_G(b)$ as follows. Let

$$S_b := \{\, s \in \mathcal{T}_{\mathrm{Basic}} \mid (b, \texttt{rdf:propositionFormSubject}, s) \in G \,\},$$

$$P_b := \{\, p \in \mathcal{T}_{\mathrm{Basic}} \mid (b, \texttt{rdf:propositionFormPredicate}, p) \in G \,\},$$

$$O_b := \{\, o \in \mathcal{T}_{\mathrm{Basic}} \mid (b, \texttt{rdf:propositionFormObject}, o) \in G \,\}.$$

where $\mathcal{T}_{\mathrm{Basic}} := \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$. Then $\mathrm{subj}_G(b)$ is defined iff $S_b = \{s\}$ for some s and we set $\mathrm{subj}_G(b) := s$; similarly $\mathrm{pred}_G(b)$ is defined iff $P_b = \{p\}$ and we set $\mathrm{pred}_G(b) := p$, and $\mathrm{obj}_G(b)$ is defined iff $O_b = \{o\}$ and we set $\mathrm{obj}_G(b) := o$.

Definition 3.13. Typed reconstruction. If $\mathrm{subj}_G(b)$, $\mathrm{pred}_G(b)$, and $\mathrm{obj}_G(b)$ are defined, define the *candidate reconstruction*:

$$\mathrm{tt}_G(b) := (\mathrm{subj}_G(b), \mathrm{pred}_G(b), \mathrm{obj}_G(b)).$$

This reconstruction is *well-typed* iff

$$\mathrm{subj}_G(b) \in \mathcal{I} \cup \mathcal{B} \quad \text{and} \quad \mathrm{pred}_G(b) \in \mathcal{I}.$$

(In particular, predicates are required to be IRIs, and subjects cannot be literals.)

Definition 3.14. Encoding quadruple. For $b \in \mathrm{PFNodes}(G)$ with defined components, let:

$$\begin{aligned}\mathrm{PFQuad}_G(b) := \{\, & (b, \texttt{rdf:type}, \texttt{rdf:PropositionForm}),\\ & (b, \texttt{rdf:propositionFormSubject}, \mathrm{subj}_G(b)),\\ & (b, \texttt{rdf:propositionFormPredicate}, \mathrm{pred}_G(b)),\\ & (b, \texttt{rdf:propositionFormObject}, \mathrm{obj}_G(b)) \,\}.\end{aligned}$$

Definition 3.15. Decoding dependency and acyclicity. Let G be a Basic graph and suppose $\mathrm{tt}_G(b)$ is defined for all $b \in \mathrm{PFNodes}(G)$. Define a directed graph of dependencies $\mathcal{D}_G$ with vertex set $\mathrm{PFNodes}(G)$ and an edge $b \to b'$ iff $b' \in \mathrm{Occ}_{\mathrm{obj}}(\mathrm{obj}_G(b))$ (equivalently, since $\mathrm{obj}_G(b) \in \mathcal{T}_{\mathrm{Basic}}$ and $\mathrm{Occ}_{\mathrm{obj}}(x) = \{x\}$ for basic terms, iff $\mathrm{obj}_G(b) = b'$), i.e., iff the object component of b is itself a proposition-form blank node. We say that G is *decoding-acyclic* iff $\mathcal{D}_G$ is acyclic.

Definition 3.16. Decoding-admissible graphs. A Basic graph G is *decoding-admissible* iff:

(D1) for every $b \in \mathrm{PFNodes}(G)$, the three component functions in Definition 3.12 are defined (exist and are unique);

(D2) for every $b \in \mathrm{PFNodes}(G)$, the candidate reconstruction $\mathrm{tt}_G(b)$ is well-typed in the sense of Definition 3.13;

(D3) G is decoding-acyclic in the sense of Definition 3.15;

(D4) for every $b \in \mathrm{PFNodes}(G)$ and every triple $(b, p, o) \in G$, we have $(b, p, o) \in \mathrm{PFQuad}_G(b)$.

Since predicates in Basic triples are IRIs, condition (D4) implies that PF-nodes may occur outside their PropositionForm quadruples only in object position. This prevents decoding from producing asserted triples with triple terms in subject position.

Lemma 3.17. Encoded graphs are decoding-admissible. *For every encoding-admissible graph G, the graph* $\mathrm{basic_encode}(G)$ *is decoding-admissible.*

Proof. Let $H := \mathrm{basic_encode}(G)$. By the argument in Remark 3.3, every proposition-form node $b \in \mathrm{PFNodes}(H)$ is minted by some call to encTT; a node occurring only in an input component triple does not become a PropositionForm node. Each minted b contributes exactly one quadruple $\mathrm{PF}(b, (s, p, o'))$ to H. Freshness guarantees that b did not occur in the input and is new when minted, so the only triples of H having subject b are the four triples of $\mathrm{PFQuad}_H(b)$; hence both the uniqueness condition (D1) and the subject-position restriction (D4) hold. Since $s \in \mathcal{I} \cup \mathcal{B}$, $p \in \mathcal{I}$, and $o' \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ by the encoding definition, the candidate reconstruction $\mathrm{tt}_H(b)$ is well-typed, so (D2) holds. Finally, nested triple terms are encoded from the inside out: if b' occurs as the object component of b, then b' encodes a proper subterm of the triple term encoded by b. Dependencies therefore follow strictly decreasing nesting depth, so $\mathcal{D}_H$ is acyclic and (D3) holds.

Definition 3.18. Substitution induced by decoding. Let G be decoding-admissible. Define $\sigma_G : \mathcal{T} \to \mathcal{T}$ by structural

recursion:

$$\begin{aligned} \sigma_G(b) &:= \sigma_G(\mathrm{tt}_G(b)) && \text{for } b \in \mathrm{PFNodes}(G), \\ \sigma_G((s,p,o)) &:= (s,p,\sigma_G(o)), \\ \sigma_G(x) &:= x && \text{otherwise.} \end{aligned}$$

Extend σ_G to graphs pointwise:

$$\sigma_G(H) := \{\, \sigma_G(t) \mid t \in H \,\}.$$

Since σ_G is applied after removing $\bigcup_{b \in \mathrm{PFNodes}(G)} \mathrm{PFQuad}_G(b)$, condition (D4) ensures that no proposition-form blank node remains in subject position. Recursive calls on proposition-form blank nodes therefore proceed only through object-position dependencies, so Definition 3.15 guarantees that this definition is well-founded.

Definition 3.19. Basic decoding. Define $\mathrm{basic_decode}$ as a partial function on $\mathsf{Graph}_{\mathrm{Basic}}$ with domain equal to the set of decoding-admissible graphs:

$$\mathrm{basic_decode}(G) := \sigma_G\left(G \setminus \bigcup_{b \in \mathrm{PFNodes}(G)} \mathrm{PFQuad}_G(b)\right),$$

where σ_G is the substitution from Definition 3.18. If G is not decoding-admissible, then $\mathrm{basic_decode}(G)$ is undefined.

3.3.2 Basic decoding of datasets (Basic → Full) The graph-level decoding defined above extends to datasets, but blank nodes have *dataset scope*. Accordingly, dataset-level decoding is defined only under a dataset-level admissibility condition that prevents cross-component collisions of proposition-form artefacts.

Definition 3.20. Dataset-level occurrence. Let $D = (G_0, \{(n_i, G_i)\}_{i=1}^k)$ be an RDF dataset. Define the set of RDF terms occurring in the dataset (including graph names) as:

$$\mathrm{Occ}_{\mathrm{ds}}(D) := \mathrm{Occ}(G_0) \cup \bigcup_{i=1}^{k} \big(\mathrm{Occ}(G_i) \cup \{n_i\}\big).$$

Definition 3.21. Dataset-level PropositionForm nodes. Define the set of proposition-form blank nodes occurring in the dataset as:

$$\mathrm{PFNodes}(D) := \mathrm{PFNodes}(G_0) \cup \bigcup_{i=1}^{k} \mathrm{PFNodes}(G_i).$$

Definition 3.22. Decoding-admissible datasets. A dataset $D = (G_0, \{(n_i, G_i)\}_{i=1}^k)$ is *decoding-admissible* iff:

(DS1) each component graph is decoding-admissible, i.e., $\mathrm{basic_decode}(G_0)$ and $\mathrm{basic_decode}(G_i)$ are defined for all $i \in \{1, \ldots, k\}$ in the sense of Definition 3.19;

(DS2) for all distinct indices $i \neq j$ (including 0 for the default graph) and all $b \in \mathcal{B}$, if $b \in \mathrm{PFNodes}(G_i)$ then $b \notin \mathrm{Occ}(G_j)$;

(DS3) for all $i \in \{1, \ldots, k\}$ and all $b \in \mathcal{B}$, if $b \in \mathrm{PFNodes}(D)$ then $b \neq n_i$.

Definition 3.23. Basic decoding of datasets. Define $\mathrm{basic_decode}$ as a partial function on RDF datasets with domain equal to the set of decoding-admissible datasets:

$$\mathrm{basic_decode}(D) := \big(\mathrm{basic_decode}(G_0), \{(n_i, \mathrm{basic_decode}(G_i))\}_{i=1}^k\big),$$

where $\mathrm{basic_decode}(G_i)$ is the graph-level decoding of Definition 3.19. If D is not decoding-admissible, then $\mathrm{basic_decode}(D)$ is undefined.

Remark 3.24. Incremental and streaming execution. Basic encoding can process asserted triples incrementally, provided that the input is known to be in $\mathcal{D}_{\mathrm{enc}}$ (or partial output can be discarded if a later triple shows otherwise), and that it retains the memoization map from triple terms to blank nodes and a dataset-wide fresh-node allocator for the entire blank-node scope. For an input graph G, the memoization state uses $O(|\mathrm{AppearingTT}(G)|)$ entries. To guarantee freshness against blank-node labels not yet encountered in a stream, input blank nodes must be relabelled into an internal namespace disjoint from generated nodes, or the input must first be scanned. If membership in $\mathcal{D}_{\mathrm{enc}}$ must be guaranteed before any output is exposed, a validation pass or equivalent buffering is required. Encoding is therefore stateful rather than a stateless tuple-at-a-time mapping. For an arbitrary input order, decoding cannot in general be completed in one bounded-memory pass: a use of a PropositionForm node may precede its component triples, and duplicate components or cyclic dependencies may be discovered only later. A practical decoder consequently needs a two-pass/external index or buffering; a bounded-memory one-pass decoder additionally requires an ordering contract that presents complete, acyclic PropositionForm descriptions before their uses. The current reference implementation materializes each graph, while implementing and evaluating these streaming variants remains future work.

4 Correctness Properties

This section states the key correctness guarantees of the interoperability transformations introduced in Section 3. We focus on (i) information preservation via invertibility of decoding after encoding on admissible inputs, and (ii) stability properties capturing that the procedures do not introduce further changes when applied repeatedly or when no encoding artefacts are present. The results are stated at the level of graph syntax, i.e., they concern the exact reconstruction of triples and triple terms rather than preservation of entailment under RDF semantics.

4.1 Information preservation and invertibility

Theorem 4.1. Invertibility on admissible graphs. *Let* $G \in \mathsf{Graph}_{\mathrm{Full}}$ *be encoding-admissible. Then* $\mathrm{basic_decode}(\mathrm{basic_encode}(G)) = G$.

Proof sketch. Proceed by induction on the maximum nesting depth of triple terms appearing in G, using Definition 2.4 and lemma 2.5 implicitly. The encoding recursively normalizes nested objects first, then replaces each triple term by a fresh blank node and records its components in exactly one PropositionForm quadruple. Memoization ensures that repeated occurrences of the same triple term reuse the same blank node. Conversely, distinct triple terms cannot be assigned the same blank node: when a new node is minted, $\mathsf{fresh}(G_i \cup G \cup E_o)$ chooses it outside the occurrences already present in the input graph and in all previously accumulated encoding fragments, which include every earlier PropositionForm node. Hence the encoding induces an injective assignment from encoded triple terms to proposition-form blank nodes. Decoding then identifies each such blank node, reconstructs the corresponding triple term uniquely, removes the quadruple, and substitutes all remaining occurrences back, yielding the original graph.

4.2 Idempotence

Proposition 4.2. Encoding idempotence. *For every Basic graph* $G \in \mathsf{Graph}_{\mathrm{Basic}}$, $\mathrm{basic_encode}(G) = G$. *In particular, for every encoding-admissible* $G \in \mathsf{Graph}_{\mathrm{Full}}$,

$$\mathrm{basic_encode}(\mathrm{basic_encode}(G)) = \mathrm{basic_encode}(G).$$

Proposition 4.3. Decoding stability. *If a Basic graph* G *contains no PropositionForm node, then* $\mathrm{basic_decode}(G) = G$.

4.3 Dataset-level correctness

Theorem 4.4. Invertibility on admissible datasets. *Let* $D = (G_0, \{(n_i, G_i)\}_{i=1}^{k})$ *be an RDF 1.2 dataset such that each component graph* G_i *(including* G_0*) is encoding-admissible. Assume that* $\mathrm{basic_encode}(D)$ *mints blank nodes that are fresh w.r.t.* all *RDF terms occurring anywhere in* D *and w.r.t. all blank-node graph names* n_i *(as required by the dataset-level encoding definition). Then* $\mathrm{basic_decode}(\mathrm{basic_encode}(D)) = D$.

Proof sketch. By the definition of $\mathrm{basic_encode}(D)$ and $\mathrm{basic_decode}(D)$, both transformations act componentwise on each graph while sharing a dataset-wide freshness constraint on minted blank nodes. Fix any component graph G_i. Since G_i is encoding-admissible, the graph-level invertibility theorem applies and yields $\mathrm{basic_decode}(\mathrm{basic_encode}(G_i)) = G_i$. Because decoding does not alter graph names and because the dataset-wide freshness constraint prevents collisions across components (including with blank-node graph names), the componentwise equalities assemble to equality of datasets, i.e., default graph and all named graphs match exactly.

Proposition 4.5. Encoding idempotence on datasets. *For every RDF dataset* D *whose component graphs are Basic graphs (i.e., contain no triple terms),* $\mathrm{basic_encode}(D) = D$. *In particular, for every RDF dataset* D *whose component graphs lie in* $\mathcal{D}_{\mathrm{enc}}$,

$$\mathrm{basic_encode}(\mathrm{basic_encode}(D)) = \mathrm{basic_encode}(D).$$

Proof sketch. Immediate from the graph-level encoding idempotence proposition applied to each component graph and the componentwise definition of dataset encoding.

Proposition 4.6. Decoding stability on datasets. *If, for every component graph* G_i *of a dataset* D*, the set* $\mathrm{PFNodes}(G_i)$ *is empty, then* $\mathrm{basic_decode}(D) = D$.

Proof sketch. If $\mathrm{PFNodes}(G_i) = \emptyset$, then graph-level decoding is the identity on G_i by the decoding stability proposition. Apply this to all components and use the componentwise definition of $\mathrm{basic_decode}(D)$.

5 Experimental Study

This section evaluates the practical behaviour of the RDF 1.2 interoperability transformations introduced earlier, focusing on their engineering cost rather than on semantic preservation under entailment. Using a self-contained reference implementation, we account for structural expansion and measure runtime overhead across controlled synthetic workloads and targeted edge-case instances. The study complements the formal results in Section 4 by validating that the implementation realizes the specified transformations and by assessing scalability and robustness under stress conditions.

5.1 Goals and Evaluation Questions

The RDF 1.2 interoperability transformations are primarily intended as *syntax-level* mechanisms enabling exchange between RDF 1.2 Full processors (supporting triple terms) and RDF 1.2 Basic processors (forbidding triple terms). Consequently, the experimental study does not aim to establish semantic equivalence under entailment regimes, but rather to quantify the practical cost of round-tripping and the structural overhead of a profile-compatible representation.

More precisely, we evaluate the implementation against the following questions:

- **EQ1 (Implementation conformance):** Does the implementation realize the formally specified round-trip property on encoding-admissible RDF 1.2 Full graphs, including graphs with nested triple terms?

- **EQ2 (Structural accounting):** Given the exact identity $|\text{basic_encode}(G)| = |G| + 4|\text{AppearingTT}(G)|$, how do (i) the ratio of asserted triples whose object is a triple term, (ii) triple term nesting depth, and (iii) triple term reuse determine $|\text{AppearingTT}(G)|$, the number of unique triple terms appearing in G, and hence the expansion ratio?

- **EQ3 (Runtime scalability):** How do encoding and decoding runtimes scale with the input size N and with triple term complexity (depth and multiplicity)?

- **EQ4 (Robustness validation):** How does the implementation behave on malformed or ambiguous Basic encodings, such as missing proposition-form components, duplicated components, cyclic reconstructions, or hybrid inputs?

5.2 Implementation and Experimental Setup

All experiments were performed with our self-contained Python reference implementation of the RDF 1.2 Interoperability transformations. The benchmark runner parses N-Triples 1.2/N-Quads 1.2 inputs [12, 13], applies `basic_encode` (Full → Basic) and `basic_decode` (Basic → Full) at the graph or dataset level as appropriate, and validates round-trip implementation conformance by checking structural equality between the original and reconstructed RDF structure. All software used to generate datasets, run benchmarks, and reproduce the reported measurements is publicly available as an archived release on Zenodo [14]. All datasets used in this study (synthetic and real-world) are publicly available on Figshare [15].

The execution host was an Intel NUC13ANKi7 with a 13th-generation Intel Core i7-1360P processor (12 cores, 16 hardware threads, up to 5.0 GHz), 62.36 GiB of reported RAM, and a Samsung SSD 990 PRO 1 TB NVMe drive. It ran 64-bit Linux Mint 21.2 Victoria with kernel 6.8.0-107-generic and Python 3.10.12. The drive used a Btrfs file system. Storage measurements reported by `hdparm` were 1,734.04 MB/s for buffered disk reads and 16,029.40 MB/s for cached reads.

The runner reports parsing time measured once per input and median encoding and decoding times over ten runs. To reduce the influence of OS jitter, each configuration is executed ten times and we report the median of measurements. The output additionally records the number of input triples, output triples, the expansion ratio $|G_{\text{enc}}|/|G|$, and statistics that characterize triple term complexity, including the number of unique triple terms and the number of generated proposition-form nodes. Because these timings are not normalized across hardware platforms, the absolute values should be read as measurements within one fixed execution environment; the empirical timing claims concern relative scaling trends rather than machine-independent performance constants, while structural overhead is derived analytically.

5.3 Datasets

To systematically explore the design space, we generate synthetic RDF graphs in which triple terms appear only in object position, consistently with the RDF 1.2 Full profile. Each graph is parameterized by: (i) the number of asserted triples N, (ii) the ratio $r \in [0, 1]$ of asserted triples whose object is a triple term, (iii) the nesting depth d of triple terms (with $d = 0$ producing Basic graphs), and (iv) a reuse factor $u \in (0, 1]$ controlling the pool from which triple term objects are sampled.

More precisely, let the number of top-level triple term occurrences be

$$K = \begin{cases} 0, & d = 0, \\ \text{round}(rN), & d > 0, \end{cases}$$

where round denotes Python's nearest-integer rule with ties to even. For $K > 0$, the generator creates a pool of $P = \max\{1, \text{round}(uK)\}$ distinct root triple terms and draws each of the K occurrences uniformly from that pool with replacement. If X pool elements are actually drawn, then the number of unique triple terms counted recursively is exactly $U = dX$ because each root is a chain of d distinct terms; in particular, $X \leq P$ and

$$\mathbb{E}[U] = dP \left(1 - \left(1 - \frac{1}{P}\right)^K\right).$$

For $K = 0$, $U = 0$. Thus u fixes the candidate-pool size, while the realized U also reflects sampling with replacement; this accounts, for example, for observed counts just below the pool maximum.

Intuitively, for fixed N and $d > 0$, increasing r increases how frequently triple terms occur, increasing d increases the number of nested triple terms introduced per occurrence, and decreasing u increases reuse (i.e., fewer unique triple terms are shared by more occurrences). In our benchmark suite we instantiate a grid of values for (N, r, d, u) in order to obtain controlled stress tests ranging from Basic-only graphs ($d = 0$) to deeply nested graphs with frequent triple term occurrences.

5.4 Metrics

We evaluate the following families of metrics.

Structural overhead. Let G be the input Full graph and $G_{\text{enc}} = \text{basic_encode}(G)$ be the encoded Basic graph. We report (i) the absolute increase in the number of triples,

$$\Delta_T = |G_{\text{enc}}| - |G|,$$

and (ii) the expansion ratio,

$$\exp_T = \frac{|G_{\text{enc}}|}{|G|}.$$

Let $U = |\mathrm{AppearingTT}(G)|$. For an encoding-admissible input, the injective replacement induced by memoization and freshness maps the $|G|$ asserted input triples to exactly $|G|$ distinct Basic triples. In addition, each unique triple term yields one distinct PropositionForm node and exactly four auxiliary triples. Therefore the structural overhead is the exact identity

$$|G_{\mathrm{enc}}| = |G| + 4U, \qquad \Delta_T = 4U, \qquad \exp_T = 1 + \frac{4U}{|G|}.$$

The structural counts below are deterministic accounting and implementation checks of this identity, not empirical estimates of it.

Runtime. We report encoding time t_{encode} and decoding time t_{decode} (in seconds), measured using a high-resolution timer. Each configuration is executed ten times and we report the median runtime.

Triple term statistics. We record (i) U, the number of unique triple terms appearing in the input (including nested ones), and (ii) the number of generated proposition-form nodes, which should match U under a memoizing implementation that emits one proposition-form node per unique triple term.

5.5 Experimental Protocol

For each generated file, the runner executes the following pipeline: (i) parsing of the input into the internal graph representation, (ii) encoding into a Basic graph, (iii) decoding back into a Full graph, and (iv) equality check between the original and decoded graphs. Parsing is measured once per input file, whereas encoding and decoding are each repeated ten times to obtain robust median measurements. Each benchmark row is annotated with the generation parameters (N, r, d, u), the measured runtimes, and the structural statistics.

5.6 Real-world case study: YAGO4

To complement the controlled synthetic benchmarks, we evaluate the transformations on a large real-world RDF 1.2 graph derived from the YAGO4 dataset [*16*]. We focus on a single graph file that already uses RDF 1.2 quoted triples and a reifier-style pattern via `rdf:reifies`. This graph file is representative for statement-level qualifiers such as temporal validity, where an asserted triple is referenced by a separate node carrying metadata (e.g., start/end dates).

Input characteristics. The input graph contains $|G| = 2{,}695{,}942$ triples (set semantics). Parsing the file took $t_{\mathrm{parse}} = 203.906$ seconds with a peak RSS of 3,870.9 MB after parsing. Quoted triples (triple terms) occur only in object position, consistently with the RDF 1.2 Full profile.

Triple terms and reifiers. We observed $N_{\mathrm{tt}} = 925{,}022$ occurrences of triple terms as objects. All of them are distinct in this file ($U_{\mathrm{tt}} = 925{,}022$), yielding a reuse ratio $N_{\mathrm{tt}}/U_{\mathrm{tt}} = 1.0$. The maximum quoted-triple nesting depth is 1, i.e., the dataset uses non-nested quoted triples.

The dataset also contains $N_{\mathrm{reifies}} = 925{,}022$ triples using `rdf:reifies`, with $R = 925{,}022$ distinct reifier nodes, i.e., essentially one reifier per quoted triple. The metadata degree of reifier nodes (counting outgoing metadata statements) has median 2.0 and maximum 28. Moreover, the median number of reifiers per quoted triple is 1.0 and the maximum is 1, confirming a one-to-one pattern.

Basic encoding overhead (Full → Basic). Applying $\mathrm{basic_encode}$ yields an encoded Basic graph of size $|G_{\mathrm{enc}}| = 6{,}396{,}030$, corresponding to an absolute increase of

$$\Delta_T = |G_{\mathrm{enc}}| - |G| = 3{,}700{,}088$$

and an expansion ratio

$$\exp_T = \frac{|G_{\mathrm{enc}}|}{|G|} = 2.372466.$$

As required by the encoding definition, the increase satisfies the exact identity:

$$\Delta_T - 4U_{\mathrm{tt}} = 0,$$

i.e., the encoding introduces precisely four triples per *unique* quoted triple term. In the benchmark runner, the transformation-only encoding time was $t_{\mathrm{encode}} = 24.867$ seconds.

End-to-end CLI timings. Since the standalone converter performs parsing and serialisation in addition to the in-memory transformation, we also report end-to-end timings measured with `/usr/bin/time -v`. For Full → Basic encoding, the elapsed wall-clock time was $t^{\mathrm{e2e}}_{\mathrm{encode}} = 260.78$ seconds, with maximum resident set size 6.10 GiB. For Basic → Full decoding (starting from the encoded file), the elapsed wall-clock time was $t^{\mathrm{e2e}}_{\mathrm{decode}} = 330.56$ seconds, with maximum resident set size 5.85 GiB. These end-to-end measurements highlight that, at multi-million-triple scale, I/O and parsing dominate total runtime, while the structural overhead is fully explained by the $4U_{\mathrm{tt}}$ proposition-form rule.

Round-trip implementation validation (EQ1). To complete the real-world case study, we additionally ran Basic → Full decoding on the encoded output and checked $\mathrm{basic_decode}(\mathrm{basic_encode}(G)) = G$ by structural equality, using the same equality criterion as in the synthetic benchmark suite. For YAGO4, decoding completed successfully and the equality check returned `true`. This is a conformance check of the implementation against the already proved invertibility theorem, not independent empirical evidence for that theorem.

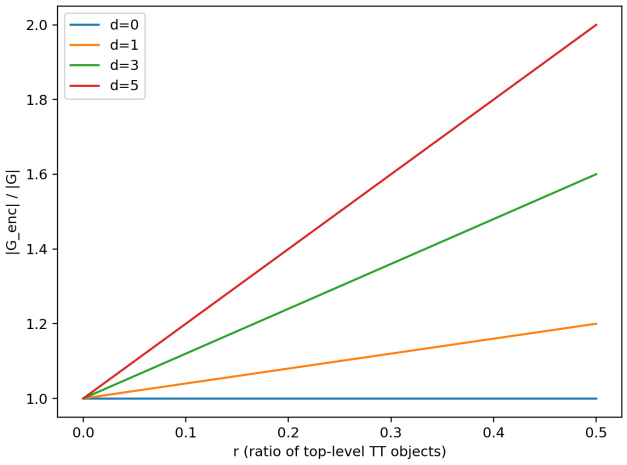

Figure 1. Expansion ratio $|G_{enc}|/|G|$ as a function of r for multiple nesting depths d.

Interpretation. Unlike many synthetic configurations where reuse can substantially mitigate overhead, YAGO4 exhibits *no* reuse of quoted triples ($N_{\mathrm{tt}} = U_{\mathrm{tt}}$). Consequently, the Basic encoding overhead is maximal with respect to U_{tt} and results in a $2.37\times$ increase in triple count. This highlights a practical worst case for profile-based interoperability: when real-world data encodes metadata by introducing a fresh quoted triple for each annotated fact, the Basic encoding cost becomes a linear function of the number of annotated facts.

The YAGO4 case also illustrates why the Working Group note keeps this interoperability mapping non-normative. When each quoted triple is already associated with a unique reifier, an implementation could reuse that existing node instead of minting a fresh proposition-form node, thereby reducing both structural overhead and implementation cost. Our reference implementation does not pursue such dataset-specific optimizations, because its goal is to realize the generic mapping formalized in this paper.

5.7 Results and Discussion

Round-trip implementation validation (EQ1). The invertibility theorem in Section 4 establishes round-trip correctness mathematically. The purpose of EQ1 is narrower: to test whether the reference code realizes the formal functions. Across the entire benchmark grid of encoding-admissible inputs, every structural equality check returned `true`. For Basic graphs ($d = 0$), the tests also verified that encoding and decoding act as the identity when no PropositionForm nodes are present. These observations validate the implementation, not the theorem itself.

Structural accounting and expansion patterns (EQ2). Figure 1 summarizes the expansion ratio $|G_{\mathrm{enc}}|/|G|$ as a function of the asserted-triple ratio r for multiple nesting depths d. As expected, for $d = 0$ the expansion is exactly 1.0 for all r, since the graphs are Basic and contain no triple terms. For $d > 0$, the expansion grows monotonically with r and increases with the nesting depth.

Table 1. Structural overhead of RDF 1.2 basic encoding on synthetic graphs (slice: $N = 100000$, $u = 0.1$). For each configuration (r, d), the table reports the deterministic input size ($|G|$), size after encoding ($|G_{\mathrm{enc}}|$), and expansion ratio $|G_{\mathrm{enc}}|/|G|$.

r_param	d_param	in_triples	enc_triples	exp_triples
0	0	100000	100000	1.000
0.01	0	100000	100000	1.000
0.1	0	100000	100000	1.000
0.5	0	100000	100000	1.000
0	1	100000	100000	1.000
0.01	1	100000	100400	1.004
0.1	1	100000	104000	1.040
0.5	1	100000	119996	1.200
0	3	100000	100000	1.000
0.01	3	100000	101200	1.012
0.1	3	100000	112000	1.120
0.5	3	100000	160000	1.600
0	5	100000	100000	1.000
0.01	5	100000	102000	1.020
0.1	5	100000	120000	1.200
0.5	5	100000	199980	2.000

The detailed structural statistics for the largest synthetic slice ($N = 100000$, $u = 0.1$) are reported in Table 1. These counts instantiate the exact identity $\Delta_T = 4U$. For $d = 1$ and $r = 0.1$, $U = 1000$, so the encoded graph contains exactly $100000 + 4 \cdot 1000 = 104000$ triples: $\Delta_T = 4000 = 4 \cdot 1000$, and the expansion is 1.040. For $d = 3$ and $r = 0.5$, $U = 15000$, giving $\Delta_T = 60000 = 4 \cdot 15000$, exactly 160000 triples in total, and expansion 1.6. In the most demanding shown configuration ($d = 5$, $r = 0.5$), $U = 24995$, so the encoded graph contains exactly 199980 triples; its expansion is 1.9998, displayed as 2.000 after rounding.

Thus higher reuse (smaller u) produces a smaller candidate pool and generally reduces U, whereas deeper nesting multiplies the number of distinct terms contributed by each selected root chain. The table illustrates how the generator parameters determine U and checks that the implementation emits the structurally required $4U$ triples; the identity itself follows from the encoding definition.

Runtime scalability (EQ3). Figure 2 reports median encoding and decoding times as a function of N for a representative fixed configuration (r, d, u), showing near-linear scaling in input size. The per-configuration breakdown in Table 2 further illustrates that runtime grows with both the number of input triples and the number of unique triple terms.

For Basic graphs ($d = 0$), encoding and decoding are fast and dominated by parsing and serialization overheads. For $d > 0$, encoding time increases with U because each new

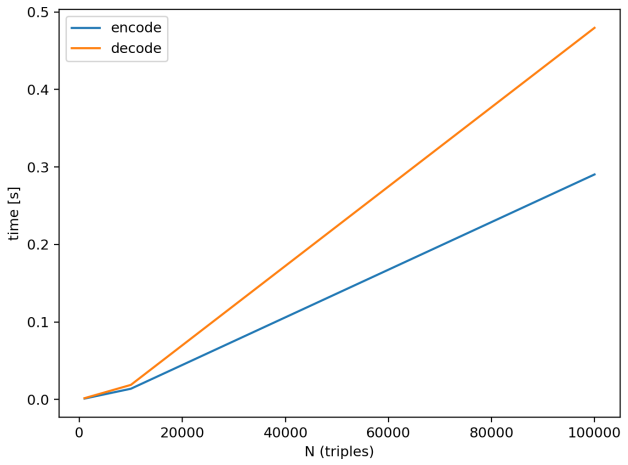

Figure 2. Median encoding/decoding runtime over ten repetitions as a function of N for fixed (r, d, u).

unique triple term triggers the generation of a proposition-form quadruple. Decoding time additionally includes the cost of (i) scanning proposition-form nodes, (ii) reconstructing triple terms, and (iii) substituting reconstructed terms back into the remaining triples. In the shown slice ($N = 100000$, $u = 0.1$), runtime growth across depths is consistent with the increase in unique triple terms: for instance, when $r = 0.1$ the number of unique triple terms grows from 1000 ($d = 1$) to 3000 ($d = 3$) and 5000 ($d = 5$), and both t_{encode} and t_{decode} increase accordingly. For the heaviest case shown ($r = 0.5$, $d = 5$), the benchmark reports $U = 24995$ and decoding exceeds one second, illustrating that round-tripping remains feasible but incurs measurable cost under intense nesting and frequent triple term usage.

A useful implementation consistency check is that the number of PropositionForm nodes equals the number of unique triple terms U, because memoization assigns exactly one such node to each unique term. Table 2 records this equality throughout the shown configurations (e.g., $U = 15000$ gives 15000 PropositionForm nodes). This checks the implementation's memoization; the resulting $4U$ overhead remains an exact consequence of the definition.

Robustness (EQ4). Beyond the well-formed benchmark suite, we additionally exercised the implementation on a set of *hand-crafted negative instances* intended to stress the decoder's admissibility checks. In this paper, we report these observations as *implementation-level sanity checks* rather than as a fully quantified experimental result: the goal of these tests was to confirm that the decoder fails *loudly* (i.e., returns "undefined" / raises an error) on representative classes of malformed or ambiguous Basic encodings, and does not silently produce an incorrect Full reconstruction.

Concretely, we validated that decoding is rejected for representative instances in the following classes: (i) missing proposition-form components (at least one of `rdf:propositionFormSubject`, `rdf:propositionFormPredicate`, `rdf:propositionFormObject` absent); (ii) duplicated components (multiple distinct subjects/predicates/objects for the same PF-node); (iii) ill-typed reconstructions (literal in subject position or non-IRI predicate); (iv) cyclic PF-dependencies (a cycle in the dependency graph induced by PF-nodes); (v) PF-nodes used as asserted subjects outside their PropositionForm quadruples, which would force decoded asserted triples with triple terms in subject position; and (vi) hybrid inputs that mix triple terms with proposition-form markers. These checks correspond directly to conditions (D1), (D2), and (D4) of Definition 3.16, together with the acyclicity criterion Definition 3.15.

Summary. In summary, equality tests validate that the implementation realizes the formally proved round trip on admissible inputs (EQ1). Structural accounting follows the exact identity $\Delta_T = 4U$ (EQ2), with the generator parameters determining U. Runtime scales approximately linearly with input size and with the number of unique triple terms (EQ3), while robustness checks validate rejection of representative ambiguous encodings (EQ4).

6 Related Work

The problem of attaching metadata to individual RDF assertions has been studied for many years and has led to multiple modeling idioms that differ in granularity, verbosity, and semantic status. RDF 1.2 triple terms, together with the Interoperability basic encoding/decoding, can be positioned as a profile-aware and round-trippable mechanism that complements these earlier approaches.

A classical RDF mechanism for making statements about statements is standard reification, defined via the vocabulary `rdf:Statement`, `rdf:subject`, `rdf:predicate`, and `rdf:object` [17]. Reification is purely representational: it introduces an additional node intended to describe a triple, but it does not, by itself, connect that description to the truth of the described triple. In practice, reification is often considered verbose, as it expands one statement into several triples and typically increases query complexity due to the need for additional joins. Closely related solutions employ n-ary relation patterns, where a relation instance is represented by an intermediate node that links the primary participants and carries qualifiers (e.g., time, provenance, confidence) as additional properties [8]. These patterns are highly general and work within standard RDF 1.1, but they move the representation away from the original binary predicate and often require redesigning queries and constraints around the introduced relation nodes.

Another widely used mechanism is named graphs, where statements are grouped into separately identified graphs to which provenance, trust, or other metadata can be attached [9]. This approach aligns naturally with RDF datasets

Table 2. Runtime and triple term statistics for RDF 1.2 interoperability transformations on synthetic graphs (slice: $N = 100000$, $u = 0.1$). For each configuration (r, d), the table reports median encoding time (t_{encode}) and decoding time (t_{decode}) over ten repetitions, the number of unique triple terms appearing in the input (tt_unique_total), and the number of generated proposition-form nodes (pf_nodes).

r_param	d_param	t_encode_s	t_decode_s	tt_unique_total	pf_nodes
0	0	0.2165	0.0098	0	0
0.01	0	0.2390	0.0149	0	0
0.1	0	0.2197	0.0111	0	0
0.5	0	0.2990	0.0157	0	0
0	1	0.1853	0.0096	0	0
0.01	1	0.1667	0.2124	100	100
0.1	1	0.2799	0.2677	1000	1000
0.5	1	0.4045	0.5963	4999	4999
0	3	0.2549	0.0145	0	0
0.01	3	0.2353	0.2232	300	300
0.1	3	0.2903	0.4795	3000	3000
0.5	3	0.6692	1.0127	15000	15000
0	5	0.2031	0.0111	0	0
0.01	5	0.2306	0.2782	500	500
0.1	5	0.4782	0.7436	5000	5000
0.5	5	1.0055	1.4172	24995	24995

and is effective for graph-level metadata. However, it is coarser than statement-level approaches: associating distinct metadata with individual triples typically requires isolating statements into separate named graphs, which may lead to fragmentation and operational overhead. Moreover, RDF does not prescribe a single semantic relationship between a graph name and the graph it names; incompatible application-specific readings, such as provenance, identity, version, or context, can therefore create data-integration problems [*1*]. The Singleton Property approach targets statement-level metadata by creating a fresh, unique property IRI for each assertion and relating it to a generic property via a dedicated link, thereby enabling qualifiers to be attached directly to that unique predicate occurrence [*7*]. This technique remains within standard RDF 1.1 syntax and comes with an explicit semantic account, but it can significantly increase the number of distinct properties and may impact reasoning and indexing due to vocabulary blow-up.

A more direct alternative is to extend RDF with embedded or quoted triples. Hartig's formalization of RDF* and SPARQL* provides foundations for treating triples as first-class terms that can be nested and queried concisely [*18*]. Subsequently, the RDF-star Community Group specification defines RDF-star and SPARQL-star, including concrete syntaxes and a formal semantics for quoted triples as an extension of the RDF model [*19*, *20*]. This work served as the starting point for the RDF & SPARQL Working Group effort that developed into RDF 1.2 triple terms and reifiers [*1*]. These approaches aim to mitigate the verbosity and practical shortcomings of standard reification, while highlighting subtle semantic issues such as the distinction between asserted and quoted triples. In comparison, the RDF 1.2 Interoperability basic encoding is structurally reminiscent of reification in that it represents statement-level structure explicitly as a small auxiliary subgraph, but it is profile-specific and designed for round-tripping between RDF 1.2 Full (with triple terms) and RDF 1.2 Basic (without them), subject to explicit admissibility conditions. Accordingly, its primary objective is syntax-level information preservation across heterogeneous implementations rather than semantic equivalence under arbitrary entailment regimes.

A direct structural comparison also clarifies the scope of the contribution. If the RDF 1.1 reification vocabulary were used as a drop-in surrogate under the same freshness and one-node-per-unique-term memoization policy, each unique triple term would again require one type triple and three component triples. It would therefore have the same exact overhead, $\Delta_T = 4U$, and the same asymptotic transformation work. PropositionForm is chosen for collision-safe recognition of transformation artefacts, not for a claimed size or runtime advantage [*11*]. The native Singleton Property pattern instead uses a fresh property IRI in a data triple and a link to its generic property, but targets a contextualized asserted relationship [*7*]. It does not prescribe a reversible surrogate for arbitrary RDF 1.2 triple terms, which may be unasserted, shared, or nested [*1*]. Defining such a baseline would first require conventions for recursive surrogates, IRI minting, reuse, collision handling, and decoding; those choices determine both size and runtime. Accordingly, we make no empirical superiority claim over these idioms. Existing comparisons address their native metadata workloads [*6*]; a semantically aligned transformation benchmark remains future work.

Recent RDF-star research covers several complementary stages of the data lifecycle: dictionary indexing and the query-performance effects of nested quoted triples [*21*];

declarative generation from heterogeneous data [22, 23]; RDF-star-specific graph embeddings [24]; transformation to property graphs [25]; and triplestore conformance and performance benchmarking [26]. These studies motivate processor-level evaluation, especially with respect to nesting depth, but address native RDF-star storage, generation, analytics, or querying rather than the Basic–Full round trip formalized here.

7 Conclusion

This paper provided a formal, implementation-oriented account of interoperability between the two RDF 1.2 conformance profiles enabled by triple terms: RDF 1.2 Full and RDF 1.2 Basic. The motivation is practical: statement-level identifiers and qualifiers are increasingly required in real-world knowledge graphs, while many deployed toolchains still operate under the Basic profile and cannot directly consume Full graphs.

We gave a precise specification of the RDF 1.2 Interoperability transformations, basic encoding (Full $\rightarrow$ Basic) and basic decoding (Basic $\rightarrow$ Full), as functions on graphs and datasets. Our formalization makes explicit the key operational assumptions needed for predictable behavior, including freshness of minted blank nodes, memoization of repeated triple terms, and admissibility conditions that ensure unique reconstruction.

Under these assumptions, we established core correctness guarantees: decoding inverts encoding on encoding-admissible inputs, and encoding is idempotent (in particular, Basic graphs are fixed points). These results justify the use of the transformations as a round-trippable, information-preserving bridge between the two profiles.

At the semantic level, under the standard semantics considered here, the proposed transformations should be understood as syntax-level interoperability mechanisms rather than as fully semantics-preserving translations. The encoding replaces triple terms with existentially quantified blank nodes and introduces auxiliary structure whose interpretation is not fixed so as to recover the Full meaning of triple terms.

Our experimental study checked that the reference implementation realizes the formally proved round trip across a broad benchmark space including nested triple terms. Structural counts matched the exact identity $\Delta_T = 4U$, while runtime scaled near-linearly with input size and triple term complexity. Robustness checks additionally verified that representative malformed PropositionForm patterns are rejected rather than silently mis-decoded.

Several directions remain for future work. A first line concerns semantic strengthening: it would be useful to develop and analyze semantic extensions in which PropositionForm-style Basic encodings are given an explicit interpretation that more directly captures the intended relationship to Full triple terms, potentially enabling entailment-preserving fragments or well-characterized approximation bounds. A second line concerns integration with querying and entailment. We do not evaluate downstream query, validation, or reasoning workloads in SPARQL engines, SHACL validators, or rule engines. Their response to the auxiliary triples depends on physical indexing, optimization, and workload shape; a controlled processor-specific evaluation would help tool builders reason about query and constraint portability across Basic/Full boundaries. A third line is a semantically aligned comparison with alternative encodings: assertion status, nesting, reuse, identifier minting, collision policy, and inverse decoding should be fixed before structural and runtime results are compared. A fourth line concerns engineering aspects, including canonicalization strategies for decoding, the stateful streaming or incremental variants characterized in Remark 3.24, and validation techniques for detecting collisions with the reserved interoperability vocabulary in multi-source data integration settings.

References

1. Kellogg, G., O. Hartig, P.-A. Champin, and A. Seaborne. *RDF 1.2 Concepts and Abstract Data Model*. W3C working draft, W3C, 2026. Https://www.w3.org/TR/2026/WD-rdf12-concepts-20260303/.
2. Govindapillai, S., L.-K. Soon, and S.-C. Haw. An empirical study on Resource Description Framework reification for trustworthiness in knowledge graphs. *F1000Research*, Vol. 10, 2021, p. 881.
3. Missier, P. and L. Moreau. *PROV-DM: The PROV Data Model*. W3C recommendation, W3C, 2013. Https://www.w3.org/TR/2013/REC-prov-dm-20130430/.
4. Vrandečić, D. and M. Krötzsch. Wikidata: A Free Collaborative Knowledge Base. *Communications of the ACM*, Vol. 57, No. 10, 2014, pp. 78–85. doi:10.1145/2629489. URL https://dl.acm.org/doi/10.1145/2629489.
5. Team, O. What Is RDF-star, 2021. URL https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-star.
6. Orlandi, F., D. Graux, and D. O'Sullivan. Benchmarking RDF Metadata Representations: Reification, Singleton Property and RDF*. In *2021 IEEE International Conference on Semantic Computing (ICSC)*. 2021, pp. 233–240. doi:10.1109/ICSC50631.2021.00049. URL https://fabriziorlandi.net/pdf/2021/ICSC2021_REF-Benchmark.pdf.
7. Nguyen, V., O. Bodenreider, and A. P. Sheth. Don't Like RDF Reification? Making Statements about Statements Using Singleton Property. In *Proceedings of the 23rd International Conference on World Wide Web (WWW 2014)*. ACM, 2014, pp. 759–770. doi:10.1145/2566486.2567973.
8. Noy, N. and A. Rector. *Defining N-ary Relations on the Semantic Web*. W3C working group note,

W3C, 2006. Https://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/.
9. Carroll, J. J., C. Bizer, P. Hayes, and P. Stickler. Named Graphs, Provenance and Trust. In *Proceedings of the 14th International Conference on World Wide Web (WWW '05)*. ACM, 2005, pp. 613–622. doi:10.1145/1060745.1060835.
10. Patel-Schneider, P., D. Arndt, and E. Franconi. *RDF 1.2 Semantics*. W3C working draft, W3C, 2026. Https://www.w3.org/TR/2026/WD-rdf12-semantics-20260201/.
11. Champin, P.-A. *RDF 1.2 Interoperability*. W3C group note draft, W3C, 2026. Https://www.w3.org/TR/2026/DNOTE-rdf12-interop-20260309/.
12. Tomaszuk, D. and G. Kellogg. *RDF 1.2 N-Triples*. W3C working draft, W3C, 2026. Https://www.w3.org/TR/2026/WD-rdf12-n-triples-20260228/.
13. Tomaszuk, D. and G. Kellogg. *RDF 1.2 N-Quads*. W3C working draft, W3C, 2026. Https://www.w3.org/TR/2026/WD-rdf12-n-quads-20260304/.
14. Tomaszuk, D. domel/rdf-interop: v1.0.0. Zenodo software release, 2026. doi:10.5281/zenodo.18755743. URL https://doi.org/10.5281/zenodo.18755743.
15. Tomaszuk, D. RDF 1.2 Benchmark Datasets (Synthetic and Real). Figshare dataset, 2026. doi:10.6084/m9.figshare.31398489.v1. URL https://figshare.com/articles/dataset/RDF_1_2_Benchmark_Datasets_Synthetic_and_Real_/31398489.
16. Tanon, T. P., G. Weikum, and F. Suchanek. YAGO 4: A Reason-able Knowledge Base. In *The Semantic Web – 17th International Conference, ESWC 2020*, *Lecture Notes in Computer Science*, Vol. 12123. Springer, 2020, pp. 583–596. doi:10.1007/978-3-030-49461-2_34.
17. Cyganiak, R., D. Wood, and M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 2014. URL https://www.w3.org/TR/rdf11-concepts/. W3C Recommendation, 25 February 2014.
18. Hartig, O. Foundations of RDF* and SPARQL* (An Alternative Approach to Statement-Level Metadata in RDF). In *Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web (AMW 2017)*, *CEUR Workshop Proceedings*, Vol. 1912, 2017. URL https://ceur-ws.org/Vol-1912/paper12.pdf. Paper 12.
19. RDF-star and SPARQL-star Community Group. RDF-star and SPARQL-star (Editor's Draft). W3C Community Group Editor's Draft, 2023. URL https://w3c.github.io/rdf-star/cg-spec/editors_draft.html. Accessed 26 January 2026.
20. Hartig, O., P.-A. Champin, G. Kellogg, and A. Seaborne. RDF-star and SPARQL-star. W3C Community Group Final Report, 2021. URL https://www.w3.org/2021/12/rdf-star.html. Final Community Group Report, 17 December 2021.
21. Taelman, R. and R. Verborgh. In-Memory Dictionary-Based Indexing of Quoted RDF Triples. In *Joint Proceedings of the 7th Workshop on Storing, Querying and Benchmarking Knowledge Graphs and the 9th Workshop on Managing the Evolution and Preservation of the Data Web*, *CEUR Workshop Proceedings*, Vol. 3565. 2023, pp. 7–21. URL https://ceur-ws.org/Vol-3565/QuWeDa2023-paper1.pdf.
22. Delva, T., J. Arenas-Guerrero, A. Iglesias-Molina, O. Corcho, D. Chaves-Fraga, and A. Dimou. RML-star: A Declarative Mapping Language for RDF-star Generation. In *Proceedings of the ISWC 2021 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice*, *CEUR Workshop Proceedings*, Vol. 2980. 2021, pp. 1–5. URL https://ceur-ws.org/Vol-2980/paper374.pdf.
23. Arenas-Guerrero, J., A. Iglesias-Molina, D. Chaves-Fraga, D. Garijo, O. Corcho, and A. Dimou. Declarative Generation of RDF-star Graphs from Heterogeneous Data. *Semantic Web*, Vol. 16, No. 2, 2025, pp. 1–19. doi:10.3233/SW-243602. URL https://doi.org/10.3233/SW-243602.
24. Egami, S., T. Ugai, M. Oota, K. Matsushita, T. Kawamura, K. Kozaki, and K. Fukuda. RDF-star2Vec: RDF-star Graph Embeddings for Data Mining. *IEEE Access*, Vol. 11, 2023, pp. 142030–142042. doi:10.1109/ACCESS.2023.3341029. URL https://doi.org/10.1109/ACCESS.2023.3341029.
25. Abuoda, G., D. Dell'Aglio, A. Keen, and K. Hose. Transforming RDF-star to Property Graphs: A Preliminary Analysis of Transformation Approaches. In *Proceedings of the 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs*, *CEUR Workshop Proceedings*, Vol. 3279. 2022, pp. 17–32. URL https://ceur-ws.org/Vol-3279/paper2.pdf.
26. Abuoda, G., C. Aebeloe, D. Dell'Aglio, A. Keen, and K. Hose. StarBench: Benchmarking RDF-star Triplestores. In *Joint Proceedings of the 7th Workshop on Storing, Querying and Benchmarking Knowledge Graphs and the 9th Workshop on Managing the Evolution and Preservation of the Data Web*, *CEUR Workshop Proceedings*, Vol. 3565. 2023, pp. 34–49. doi:10.34726/5399. URL https://ceur-ws.org/Vol-3565/QuWeDa2023-paper4.pdf.