

AutoRML: Automated Declarative RDF Generation Leveraging Semantic Table Annotation

Ioannis Dasoulas ✉️🏠 

KU Leuven – Leuven.AI – Flanders Make@KULeuven, Belgium

Ali Elhalawati ✉️ 

KU Leuven – Leuven.AI – Flanders Make@KULeuven, Belgium

Anastasia Dimou ✉️ 

KU Leuven – Leuven.AI – Flanders Make@KULeuven, Belgium

Abstract

Knowledge graph (KG) construction from heterogeneous data is a complex process that requires good understanding of the data to semantically annotate entities and their relations, and define the terms of the KG based on these entities and relations. Due to this complexity, KG construction remains primarily manual. On the one hand, *semantic annotation systems* focus on entity and relation disambiguation, but ultimately they do not construct a KG. On the other hand, *declarative systems* which typically construct the KGs, assume that the semantic annotations are already available. However, the two types of systems have not yet been effectively integrated. In this paper, we propose a formal method that integrates semantic annotation and declarative systems for automating end-to-end

KG construction through automated declarative mappings generation. We validate our approach by creating AutoRML, a system that can leverage different semantic annotation frameworks to annotate tabular data and declaratively construct KGs. Evaluations demonstrate that AutoRML can construct KGs identical to manually constructed ones when the same target knowledge base is used as reference. AutoRML supports semi-automatic KG construction when semantic annotations are insufficient for full automation, generating human-friendly declarative mappings that can be refined by experts. We showcase AutoRML's performance with diverse datasets, showing its potential in automating data integration workflows.

2012 ACM Subject Classification Information systems → Extraction, transformation and loading

Keywords and phrases Knowledge Graphs, Mapping Languages, RML, Semantic Table Annotation, Entity Linking, Property Linking

Digital Object Identifier 10.4230/TGDK.4.3.2

Category Research

Supplementary Material *Software (Source Code)*: <https://github.com/dtai-kg/AutoRML> [17]

archived at [swh:1:dir:fa651da6d489f5e579e3a0b7819a8216854a1f76](https://swh.io/1/dir/fa651da6d489f5e579e3a0b7819a8216854a1f76)

Other (Research Data / Evaluation Results): <https://doi.org/10.5281/zenodo.14535918>

Funding This research was partially supported by Flanders Make (REXPEK project), the strategic research centre for the manufacturing industry, and the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” program.

Received 2025-11-06 **Accepted** 2026-08-21 **Published** 2026-09-30

Generative AI Declaration Generative AI tools were used to assist with language and stylistic editing of the manuscript and, where applicable, to generate templates for the presentation and formatting of tables. All AI-assisted material was critically reviewed, revised, and verified by the authors. The authors retain full responsibility for the content, accuracy, and interpretation of the work.

1 Introduction

Knowledge graphs (KGs) are often generated from heterogeneous (semi-)structured data sources. Declarative KG construction has prevailed [75], relying on declarative mappings, i.e. rule sets that define the standard definition of data transformation steps. Declarative mappings promote



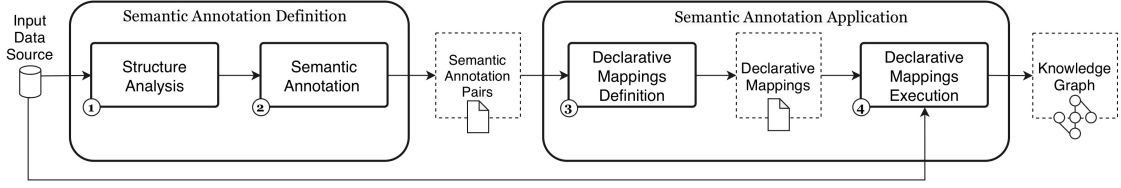
© Ioannis Dasoulas, Ali Elhalawati, and Anastasia Dimou;
licensed under Creative Commons License CC-BY 4.0

Transactions on Graph Data and Knowledge, Vol. 4, Issue 3, Article No. 2, pp. 2:1–2:34



Transactions on Graph Data and Knowledge

TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Overview of steps for declarative KG construction from (semi-)structured data.

the maintainability and reproducibility of the data transformation process [8], as well as the interoperability with KG construction systems [75]. However, despite the plethora of declarative mapping languages, human-friendly mapping serializations [31, 37] and mapping editing tools [37, 38, 45, 47], as well as the advancements in KG construction performance [74], defining declarative mappings remains primarily manual due to its complexity [9].

To declaratively construct KGs, a sequence of steps is performed (Figure 1). First, the structure and semantics of the data sources need to be analyzed (Figure 1: Steps 1,2) to perform essential semantic annotation tasks, such as named entity recognition and relation extraction [40]. Then, the declarative mappings need to be defined, indicating how the semantic annotations should be applied to the data sources relying on a declarative mapping language (Figure 1: Step 3). The declarative mappings are then executed to construct the KG (Figure 1: Step 4), transforming the original data source into a KG.

Methods for automating parts of the KG construction process exist but do not address all steps of the process. *Mapping bootstrappers* [79] generate declarative mappings (semi-)automatically for relational databases (RDBs) by leveraging the RDB schema to generate a target ontology (i.e a formal description of the domain) [5, 7, 20] or generate declarative mappings given an existing ontology [49, 50, 66, 70]. Although they ultimately combine the *Semantic Annotation* and *Declarative Mappings Definition* steps, they still present notable drawbacks that hinder their effectiveness. The generated mappings are often not immediately usable [79], encountering issues, such as label conflicts and multiple data representations [64]. They also do not generalize well for different types of RDB schemas [64, 70], let alone other data structures, lacking the effective semantic understanding and annotation task variety of modern semantic annotation systems.

Semantic table annotation (STA) systems [56] have made significant progress in understanding structured data sources by performing different entity and relation disambiguation tasks using external knowledge bases as reference or learning from already annotated sources. While STA systems focus on semantic annotation accuracy, their integration into automated declarative KG construction remains challenging. First, their usage for automating declarative mappings has been understudied [75], requiring the investigation of multiple scenarios that depend on the structure and semantics of the input data sources. Second, annotation completeness is inconsistent, as STA systems may fail to annotate parts of a data source, depending on its characteristics and the reference knowledge base. Third, different STA systems may represent their outputs in heterogeneous formats, each following its own conventions, which complicates their systematic use for declarative mapping generation. Finally, the use of information from more complex STA tasks, such as column-qualifier annotation, has not been broadly investigated.

To address these challenges, we propose a novel methodology that combines the strengths of STA and declarative mapping definition to enable automated KG construction through automated mappings generation, focusing on tabular data. First, we conduct a broad investigation of how STA results can be systematically exploited to automate the definition of declarative mappings, examining different scenarios of table structure and semantics to cover the range of cases en-

countered in practice. We formally define the automated generation of declarative mappings as a function of available semantic annotations. Second, we validate our method by creating **AutoRML**, a framework for automated declarative KG construction leveraging our methodology. Missing or incomplete annotations are addressed by inserting placeholders into the corresponding declarative mapping rules, which allows for semi-automated KG construction and provides experts with a clear starting point for refinement when annotations are insufficient. AutoRML is designed to work with multiple STA systems through an annotation formatting component, which maps heterogeneous STA outputs into a unified standard format, enabling their consistent use in mapping generation. We evaluate AutoRML conducting experiments with diverse dataset collections and STA systems. Our results showcase AutoRML’s ability to accurately perform automatic declarative KG construction when the semantic annotations provided are precise. Our approach builds on the vision paper by Chaves-Fraga and Dimou [9] that first discussed the potential of leveraging STA for automated declarative mappings generation.

Our contributions are the following:

- A novel formal methodology for automating declarative mappings generation for tabular data, leveraging semantic annotations, following an in-depth analysis of declarative mappings definition for tabular data.
- An open-source framework, AutoRML, that utilizes the proposed methodology to automatically generate YARRRML and RML rules, as well as construct KGs for tabular data.
- An evaluation of AutoRML’s ability to automate declarative KG construction, conducting experiments with different dataset collections and STA systems.

The paper is structured as follows: Section 2 covers background and related work. Section 3 analyzes declarative mappings definition for tabular data with RML and Section 4 introduces our approach for automating declarative mappings generation using STA. Section 5 introduces our framework, AutoRML, which implements our approach and Section 6 presents AutoRML’s evaluation settings and results. Section 7 discusses our approach’s strengths and limitations, and Section 8 our conclusions.

2 State of the Art

In this section we introduce the fundamental concepts of this paper, including KGs, tabular data and their properties, and STA: the process of semantically annotating tables. We also discuss related works on semantic annotation, declarative mapping definition, and KG construction.

2.1 Background

Knowledge Graph. Several representations for knowledge graphs (KG) have been proposed in the literature, with RDF graphs being one of the most widely adopted models [15]. Throughout this paper, we assume KGs to be RDF graphs. A KG is a knowledge base in which information is encoded as a set of triples of the form $\langle s, p, o \rangle$ where s is a subject, p is a predicate, and o is an object. Typically, a triple relates a node s to a node o via p . Each node of a triple belongs to one of the following sets: the set of **entities** E corresponding to graph nodes representing a semantic concept and are identified by IRIs, the set of nodes without global unique identifiers called **blank nodes** B , or the set of **literal values** L . Each predicate of a triple belongs to a set of IRI **predicates** P . Specifically, $\langle s, p, o \rangle \in (E \cup B) \times P \times (E \cup B \cup L)$. KGs are typically modeled based on ontologies. Ontologies are formal representations of the meaning of terms within a domain [41].

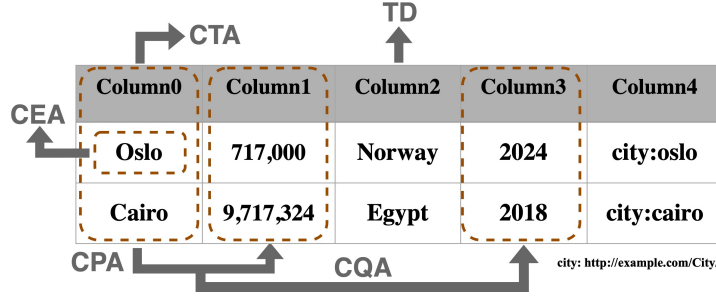
Tabular Data. A table T is a two-dimensional structure consisting of a finite set of rows R and a finite set of columns C . For a row $r_i \in R$ and a column $c_j \in C$, the entry T_{ij} denotes the table cell of r_i and c_j with a value v_{ij} . A table may or may not have a table name that provides context for the table contents. A table may or may not contain a header row, which is typically the first table row and provides context for the table columns [82]. Each column either contains entity mentions, in which case it can be characterized as a named entity column (*NE-column*), or literal entity attributes, in which case it can be characterized as a literal column (*L-column*) [82]. A table typically includes a subject column, which contains the set of entities the table is about, while the rest of the columns represent related entities (i.e. relationships) or attributes (i.e. properties) of those entities. The subject column is not necessarily a table key, as it may contain duplicate values [76]. A table may also have no subject column or have a combination of columns constitute the table’s subject [76]. The rest of the columns form binary or n-ary relationships with subject entities, or describe subject entities’ literal attributes.

Semantic Table Annotation. Semantic Table Annotation (STA) is the process of interpreting tabular data with the support of a knowledge base. The knowledge base is usually an RDF-based graph in the form of an ontology or a full-fledged KG , as these are semantically rich, by defining semantic concepts, their relationships, hierarchy, and literal attributes. To define the target knowledge base for STA, we refer to both as KG s, as is typically done in literature [41, 57, 60]. Let T be a table with a set of columns C and a set of rows R , and $KG = \{\langle s, p, o \rangle\}$, with v_{ij} denoting the value of the cell in row $r_i \in R$ and column $c_j \in C$, as previously defined. STA consists of a set of annotation tasks [1, 14, 34], with each task linking table elements from T to KG elements, producing a set of annotation result pairs $\mathcal{A}$. The most common STA tasks are the following:

- **Cell-Entity Annotation (CEA)** task: The task of annotating a cell value $v_{ij} \in T$ with an entity $e \in E$. The result of the task, $\mathcal{A}_{CEA}$ is a set of cell-entity pairs (v_{ij}, e) .
- **Column-Type Annotation (CTA)** task: The task of annotating a column $c_j \in T$ with an entity $e \in E$. The result of the task, $\mathcal{A}_{CTA}$ is a set of column-entity pairs (c_j, e) .
- **Column-Property Annotation (CPA)** task: The task of annotating a pair of columns $(c_a, c_b) \in T$ with a predicate $p \in P$. The result of the task, $\mathcal{A}_{CPA}$ is a set of column pairs matched with a predicate $((c_a, c_b), p)$.
- **Table Topic Detection (TD)** task: The task of annotating a table T with an entity $e \in E$. The result of the task, $\mathcal{A}_{TD}$ is a table-entity pair (T, e) .
- **Column-Qualifier Annotation (CQA)** task: The task of annotating n-ary column relationships expressed by a pair of sets of columns $(C_a, C_b) \subseteq C$ with a predicate $p \in P$. The result of the task, $\mathcal{A}_{CQA}$ can be defined as a pair of n-ary column sets matched with predicates $((C_a, C_b), p)$.

STA also includes a number of semantic annotation complementary tasks that reveal the table’s generic characteristics [19, 54, 43]. These are typically treated as part of the broader STA process since they provide structural annotations that facilitate the semantic annotation process. These complementary tasks annotate table T elements according to their structural significance, contents, datatype, or language producing again a set of annotation result pairs $\mathcal{A}$. The most common STA complementary tasks are the following:

- **Subject Column Detection (SCD)** task: The task of annotating a column $c_j \in C$ as a subject column (*sub_col*) or non-subject column (*sub_col'*). The result of the task, $\mathcal{A}_{SCD}$, is a set of column-subject column annotation pairs $(c_j, \text{sub_col})$ or $(c_j, \text{sub_col'})$.
- **Entity Column Detection (ECD)** task: The task of annotating a column c_j as an *NE-column* (*ne_col*) or *L-column* (*l_col*). The result of the task, $\mathcal{A}_{ECD}$, is a set of column-annotation pairs $(c_j, \text{ne_col})$ or $(c_j, \text{l_col})$.



■ **Figure 2** Example of primary STA tasks.

- **Column Datatype Annotation (CDA)** task: The task of annotating c_j with a datatype d , such as “Integer”, “Float”, “Date”, “String”, etc. The result of the task, $\mathcal{A}_{CDA}$, is a set of column-annotation pairs (c_j, d) .
- **Column Language Annotation (CLA)** task: The task of annotating a column which includes text values, c_j , with its primary language l , such as “English”, “German”. The result of the task, $\mathcal{A}_{CLA}$, is a set of column-annotation pairs (c_j, l) .

We showcase examples for these tasks through Figure 2. Assuming the Wikidata KG [77] is used as a knowledge base for extracting semantic annotations, the CEA task results in annotating the cell “Oslo” as Wikidata entity $Q585$ (“Oslo: capital city of Norway”) and the CTA task results in annotating column “Column0” as Wikidata entity class $Q515$ (“city: large human settlement”). The CPA task results in annotating the relationship between columns “Column0” and “Column1” as the Wikidata predicate $P1082$ (“population: number of people inhabiting the place”) and the CQA task results in identifying column “Column3” as a qualifier column, that expands predicate $P1082$ with the predicate $P585$ (“point in time: date something took place, existed or a statement was true”), referencing the date at which the population was measured. The TD task results in annotating the table’s topic with the Wikidata entity class $Q515$ (“city: large human settlement”) or $Q126680309$ (“City statistics”). Regarding complementary tasks, “Column0”, “Column2”, and “Column4” are *NE-columns* since they contain entity mentions, while “Column1” and “Column3” are *L-columns* since they contain literal attributes. “Column0” and “Column2” can be annotated with the “String” datatype and “English” language, “Column1” and “Column3” with the “Integer” datatype and “Column4” with the “String” or “URL” datatype, depending on the use case.

Declarative Mappings Definition with RML. Declarative mappings can be expressed as rules with declarative mapping languages, such as the W3C recommended Relational to RDF Mapping Language (R2RML) [16] and its well-known extension RDF Mapping Language (RML), which generalizes to data sources beyond relational databases. In this paper, we use RML as a mapping language for declarative mappings definition, following its widespread adoption. We use the latest version of RML¹.

RML is a super-set of R2RML [16]. It declaratively defines the construction of RDF KGs from heterogeneous data. An RML mapping is a set of Triples Maps (`rml:TriplesMap`), each defining how the KG triple terms (subject, predicate, object) are generated from a set of sources (Listing 1). Each Triples Map contains one Logical Source (`rml:LogicalSource`), one Subject Map (`rml:SubjectMap`) and zero or more Predicate Object Maps (`rml:PredicateObjectMap`). The Logical Source specifies the

¹ <http://w3id.org/rml/core>, last accessed on November 5th, 2025

input source from which data are retrieved (lines 2-3). The Subject Map defines how the triples' subjects are created (lines 4-8) and can indicate the subject's class, with the predicate `rml:class` (line 6). The Predicate Object Map (lines 9-13) consists of Predicate Maps (`rml:PredicateMap`) that define the triples' predicate (lines 10-11) and Object Maps (`rml:ObjectMap`) that define the triples' object (lines 12-13). Zero or more Graph Maps (`rml:GraphMaps`) can be specified to indicate the named graph that the generated triple belongs to, forming a quad (lines 7-8).

We adopt existing R2RML formalizations to describe Triples Maps formally [27, 28]. Given that the data source we examine is tabular, the formalization is also applicable for RML. Formally, a Triples Map TM is defined as a tuple $(\mathcal{LS}, S_{map}, \mathcal{POM})$, where $\mathcal{LS}$ denotes the Logical Source, in this case, the physical path to the input table T , $path_T$, S_{map} denotes the Subject Map, and $\mathcal{POM}$ denotes a finite set of Predicate-Object Maps ($\mathcal{PO}_{map}$). The Subject Map is defined as $S_{map} = (ST_{map}, CL, G_{map})$ where ST_{map} is a term map dedicated to the subject, CL is a set of zero or more class IRIs, and G_{map} is a Graph Map. Each $\mathcal{PO}_{map} \in \mathcal{POM}$ is a tuple $(\mathcal{PM}, \mathcal{OM}, G_{map})$ where $\mathcal{PM}$ is a set of Predicate Maps ($\mathcal{P}_{map}$), $\mathcal{OM}$ is a set of Object Maps ($\mathcal{O}_{map}$), and G_{map} is a Graph Map.

Each Subject Term Map (ST_{map}), Predicate Map ($\mathcal{P}_{map}$), Object Map ($\mathcal{O}_{map}$), and Graph Map (G_{map}) is a Term Map (`rml:TermMap`), producing an RDF term (either an IRI, a Blank Node or a Literal). Formally, a Term Map is defined as a tuple $TM = (\mathcal{VT}, type, datatype, langtag)$, where $\mathcal{VT}$ is the value-term that defines how RDF terms are generated, $type$ is the term type that denotes the type of the generated RDF term, $datatype$ is an IRI representing datatype of generated Literals (for Literal Object Maps) and $langtag$ is the language tag for Literal strings [27, 28].

The $\mathcal{VT}$ of Term Maps can be (i) *constant-valued* (*con*), generating the same RDF term, (ii) *reference-valued* (*ref*), generating a term directly from a data field of a data source, or (iii) *template-valued* (*temp*), generating a term from the combination of constant strings and data fields of a data source. The term type, $type$, of a *reference-valued* or *template-valued* Term Map determines the kind of generated RDF term. If a Term Map is a Subject Map, it can either have an IRI (`rml:IRI`) or a Blank Node (`rml:BlankNode`) term type. If a Term Map is a Predicate Map, it can only have an IRI (`rml:IRI`) term type. If a Term Map is an Object Map, it can either have an IRI (`rml:IRI`), Blank Node (`rml:BlankNode`) or a Literal (`rml:Literal`) term type. The term type is optionally defined with the `rml:termType` predicate inside the Term Map. Listing 2 showcases the constructed triples of a KG declaratively constructed using the first row of the running example and the RML mappings of Listing 1.

■ **Listing 1** RML Triples Map example, containing a Logical Source, a Graph Map, a Subject Map, and a Predicate Object Map to specify RDF construction from a referenced data source.

```

1 <#RMLTriplesMap> a rml:TriplesMap;
2   rml:logicalSource [a rml:LogicalSource;
3     rml:source "/path/cities.csv";];
4   rml:subjectMap [a rml:SubjectMap;
5     rml:template "http://example.com/{Column0}";
6     rml:class http://example.com/City;
7     rml:graphMap [a rml:GraphMap;
8       rml:constant http://example.com/Graph;];
9   rml:predicateObjectMap [a rml:PredicateObjectMap;
10     rml:predicateMap [a rml:PredicateMap;
11       rml:constant http://example.com/capitalOf];
12     rml:objectMap [a rml:ObjectMap;
13       rml:template "http://example.com/{Column2}";];].

```

RML also enables the declarative evaluation of data transformations that are defined with the Function Ontology (FnO) [21], through the RML-FNML module [46]. Function Execution (`rml:FunctionExecution`) evaluates a function defined with the Function Map (`rml:FunctionMap`). Inputs (`rml:Input`) include Parameter Maps (`rml:ParameterMap`) to reference function parameters

■ **Listing 2** Declaratively constructed triples using the Listing 1 RML mappings for the first row of Table 2.

```
1 <http://example.com/Oslo> a <http://example.com/City> <http://example.com/Graph>.
2 <http://example.com/Oslo> <http://example.com/capitalOf>
3 <http://example.com/Norway> <http://example.com/Graph>.
```

required by FnO. The function’s output is declared using a Return Map (rml:ReturnMap). Listing 3 showcases the usage of declarative functions to transform the objects within the Object Map to URLs with lowercase characters using the `ex:toLowerCaseURL`² function. The Function Execution describes which function is used and specifies its input: the data reference to “Column2”. Listing 4 showcases the constructed triples of a KG declaratively constructed using the first row of the running example and RML mappings of Listing 3.

■ **Listing 3** Declarative function definition with RML-FNML that transforms Column2 to lowercase text.

```
1 <#RMLTriplesMap> a rml:TriplesMap;
2   rml:logicalSource <#LogicalSourceMap>;
3   rml:subjectMap [a rml:SubjectMap;
4     rml:template "http://example.com/{Column0}";
5     rml:class http://example.com/City];
6   rml:predicateObjectMap [a rml:PredicateObjectMap;
7     rml:predicate http://example.com/capitalOf;
8     rml:objectMap [a rml:ObjectMap;
9       rml:functionExecution <#ToLower_Execution>;
10      rml:return http://example.com/countryOut];].
11 <#ToLower_Execution> a rml:FunctionExecution;
12   rml:function http://example.com/toLowerCaseURL;
13   rml:input [a rml:Input;
14     rml:parameter http://example.com/valueParam;
15     rml:inputValueMap [rml:reference "Column2"]; ].
```

■ **Listing 4** Declaratively constructed triples using the Listing 3 RML mappings for the first row of Table 2.

```
1 <http://example.com/Oslo> a <http://example.com/City>.
2 <http://example.com/Oslo> <http://example.com/capitalOf> <http://example.com/norway>.
```

The RML-Star [22, 46] extension enables the declarative construction of RDF-star [33] graphs. RDF-star introduces the *quoted triple term*, to further describe triples (i.e., make statements about statements) in the form of quoted triples. Quoted triples may be asserted (i.e., the triple that is quoted is also included in the graph) or non-asserted. RML-star constructs quoted triples with the Star Map (rml:StarMap), defined either in a Subject or Object Map. The Triples Map that constructs the quoted triples connects to a Star Map through the `rml:quotedTriplesMap` predicate. Quoted Triples Map can be asserted (rml:AssertedTriplesMap) or non-asserted (rml:NonAssertedTriplesMap). Listing 5 uses a Non-Asserted Triples Map to construct triples that showcase population counts from our running example, expanding it with a Triples Map that specifies the year that the population was counted. Listing 6 showcases the constructed triples of a KG declaratively constructed using the first row of the running example and RML mappings of Listing 5.

² “ex:” is a prefix for: “http://example.com/”

■ **Listing 5** Quoted triples construction with RML-Star that further describe the “ex:population” predicate between “Column0” and “Column1”.

```

1 <#RMLQuotedTriplesMap> a rml:NonAssertedTriplesMap;
2   rml:logicalSource <#LogicalSource>;
3   rml:subjectMap [a rml:SubjectMap;
4     rml:template "http://example.com/{Column0}"];
5   rml:predicateObjectMap [a rml:PredicateObjectMap;
6     rml:predicate http://example.com/population;
7     rml:objectMap [a rml:ObjectMap;
8       rml:reference "Column1"];].
9 <#RMLTriplesMap> a rml:TriplesMap;
10  rml:logicalSource <#LogicalSource>;
11  rml:subjectMap [a rml:StarMap;
12    rml:quotedTriplesMap <#RMLQuotedTriplesMap>];
13  rml:predicateObjectMap [a rml:PredicateObjectMap;
14    rml:predicate http://example.com/yearCollected;
15    rml:objectMap [a rml:ObjectMap;
16      rml:reference "Column3"];].

```

■ **Listing 6** Declaratively constructed quoted triple using the Listing 5 RML mappings for the first row of Table 2.

```

1 << <http://example.com/Oslo> <http://example.com/population> "717000"
2 >> <http://example.com/yearCollected> "2024".

```

2.2 Related Work

Different approaches have been proposed to automate parts of the declarative KG construction process (Figure 1) from (semi-)structured data. STA approaches focus on automating the semantic annotation of tabular sources (Semantic Annotation Definition step) by leveraging external knowledge bases and representation learning techniques. STA methods can be broadly categorized into KG lookup methods, deep learning-based methods, and hybrid methods.

KG lookup methods query large KGs to identify and rank candidate KG entities and properties for different annotation tasks. To rank retrieved candidates, approaches consider *heuristic score-based* algorithms (e.g., TorchicTab-Heuristic [19], DAGOBAB 2022 [43], KGCODE-Tab [54], LinkingPark [55], MTab 2021 [61]), *heuristic graph-based* algorithms (e.g., CSV2KG [72], Mantis-Table [13]), *embedding techniques* (e.g., DAGOBAB 2019 [57], Magic [71]), and *probabilistic graphical models* (e.g., Mulwad et al. [58], MTab 2019 [60]).

Deep learning-based methods focus on increasing the scalability of STA, by producing column representation vectors, leveraging annotated tabular data as a training set. Sherlock [42] and Sato [81] apply neural networks to classify column values. Latest works leverage *encoder-based pre-trained language models* (e.g., BERT [24]) to retrieve tabular data representations for different downstream table understanding tasks (e.g., HYTREL [12], TURL [23], DODUO [73], TorchicTab-Classification [19], TaBERT [80]), using the “semantic knowledge” that pre-trained language models provide. *Decoder-based general-purpose large language models* (e.g., ChatGPT [63]) have also demonstrated very promising results in table understanding and KG construction tasks for structured data (e.g., ArcheType [30], Korini et al. [51, 52], Table-GPT [53], Narayan et al. [59]).

Hybrid methods combine KG information with deep learning models to address issues such as type granularity and lack of context in the table (e.g., ColNet [11], KGLink [78]). ColNet [11] applied a convolutional neural network for type prediction on embeddings of entities obtained from KG lookup. KGLink [78] combined Wikidata information with language models to alleviate the problems caused by type granularity and missing table context.

STA systems continuously increase their semantic annotation accuracy, adapting to new types of tables, and are able to provide semantic annotations for more complex tasks, such as CEA and CQA [1, 34]. However, *STA systems* are not designed to generate declarative mappings (Declarative Mappings Definition step), focusing primarily on annotation accuracy. Current works that attempt to automatically construct KGs (e.g., Dorodnykh et al. [26] leveraging STA PheKnowLator [6] leveraging life science domain-specific annotations) do not also address the definition of declarative mappings, which help ensure reusability of produced semantic annotations and compatibility with *Declarative KG construction systems*.

Declarative KG construction systems generate KG triples from heterogeneous data sources, streamlining the Declarative Mappings Execution step [75]. They leverage declarative mappings that define the data and schema transformations. These systems have shown great progress in declarative mappings coverage, processing declarative mappings for different use cases, as well as in KG construction speed and memory utilization. However, the automation of the declarative mappings definition that these systems utilize (Declarative Mappings Definition step) is not broadly investigated, as recent survey indicates [75].

Mapping bootstrappers focus on (semi-)automatically generating declarative mappings for RDBs. They map the RDB schema to KGs, often by leveraging the W3C Direct Mapping specification [69, 79] (Semantic Annotation Definition and Declarative Mapping Definition step for RDBs). Some works first generate a putative ontology from the RDB schema (Direct Mapping approach), and use it to infer declarative mappings (e.g., D2RQ [5], MIRROR [20], Ontop [7]). Others also leverage additional information such as existing ontologies (e.g., BootOx [49], Karma [50], IncMap [65, 66], AutoMap4OBDA [70]), improving the generated mappings by employing ontology matching (Ontology Matching approach). However, *Mapping bootstrappers* cannot be utilized for tabular data without an RDB schema and their results depend on the database modeling approach, which can vary. Different RDB schema design patterns have shown to result in different mappings [66]. Also, the generated mappings are not immediately usable. The Direct Mapping approaches produce mappings that contain a large number of unorganized properties and classes which are data source-specific [79]. The Ontology Matching approaches are still far from generating a full list of usable mappings [70], requiring post-processing, validation and extension by experts [49, 64]. Moreover, these works do not take advantage of existing knowledge bases to semantically annotate data with high accuracy as recent *STA systems* do [12, 34, 43, 73].

Large language models have been tested in the direct generation of declarative mappings (Semantic Annotation Definition and Declarative Mappings Definition step) for KG construction, given an ontology and structured data as input [39]. While the results are promising, their syntactic correctness and accuracy is still not consistent.

Besides tabular data, some works have also experimented with the (semi-)automated KG construction of other types of structured data, such as JSON (e.g., J2RML [68]) and XML (e.g., Bedini et al. [4], Hannah et al. [32]). These solutions are not designed for declarative KG construction, hence their framework does not include the definition of declarative mappings.

The investigation of related work demonstrates that *mapping bootstrappers* are focused primarily on RDBs and lack the semantic annotation accuracy and adaptability to different annotation tasks and domains [70]. At the same time, *STA systems* have made significant progress in semantically understanding tables, but are not designed to address the definition of declarative mappings that would enable declarative KG construction automation. Our approach aims to combine the

strengths of *STA systems* and *declarative KG construction systems* to propose an end-to-end automated declarative KG construction framework that is able to accurately construct KGs without the need of an input data schema. This ensures maintainability and explainability through the definition of declarative mappings (Semantic Annotation Definition, Declarative Mappings Definition and Declarative Mappings Execution).

3 Analysis: Declarative Mappings Definition for Tabular Data

In this section we conduct an analysis of declarative mapping definition for transforming tabular data into KGs, using RML. We examine different usage scenarios for tabular data, and how they can be transformed to KGs declaratively. We refer to Figure 2 as an example.

3.1 Subject Map Definition Analysis

We examine the different ways a **Subject Map** can be constructed, depending on the existence of a column or a combination of columns that represent KG subjects.

Table-based Subject definition. If the table contains a column or a combination of columns that can be used to define KG subjects, the **Subject Map** can be defined by using the columns' contents. A column can be directly referenced if it already contains named entities in the form of IRIs, with a *reference-valued* IRI Subject Map. For example, “Column4” contains city website IRIs. These can be used directly as KG subjects with the **Subject Map** being defined using a *reference-valued* IRI Term Map (Listing 7).

■ **Listing 7** Reference-valued IRI Subject Map.

```
1 <#TM1> rml:subjectMap <#SM1>.
2 <#SM1> rml:reference "Column4";
3       rml:termType rml:IRI.
```

■ **Listing 8** Reference-valued Blank Node Subject Map.

```
1 <#TM2> rml:subjectMap <#SM2>.
2 <#SM2> rml:reference "Column0";
3       rml:termType rml:BlankNode.
```

Otherwise, resources may be represented by non-IRI cells, such as plain text or numbers. For example, the contents of “Column0” can be used to form the subject IRIs of KG triples. A Blank Node (Listing 8) or a *template-valued* IRI (Listing 9) **Term Map** (combining the data fields with an IRI template) can be used to define the **Subject Map**.

■ **Listing 9** Template-valued IRI Subject Map.

```
1 <#TM3> rml:subjectMap <#SM3>.
2 <#SM3> rml:template "http://example.com/
3           City/{Column0}"/>.
```

■ **Listing 10** Blank Node Subject Map.

```
1 <#TM4> rml:subjectMap <#SM4>.
2 <#SM4> rml:termType rml:BlankNode.
```

If the column combination first requires pre-processing before being referenced, an extra custom mapping function can be defined to process their contents as required. This function can use a combination of *reference-valued* and *constant-valued* maps as inputs, similarly to the example showcased in (Listing 3).

Non-table-based Subject definition. A **Subject Map** can also be defined from a single or multiple entities which are not part of the table, e.g., a person that has traveled in the listed cities of our running example. A Blank Node can be used to reference unknown entities that are not part of the table and define them as subjects (Listing 10). Alternatively, some table columns may describe a single subject entity that is not part of the table. Then, a *constant-valued* **Term Map** can be used to define the single subject that some columns refer to (Listing 11). If this subject's IRI is known, a *constant-valued* IRI **Term Map** can be used, otherwise a *constant-valued* Blank Node **Term Map** that refers to a single unknown subject (Listing 12).

Listing 11 Constant-valued IRI Subject Map.

```
1 <#TM5> rml:subjectMap <#SM5>.
2 <#SM5> rml:constant "http://example.com/
3           Person/Jack".
```

Listing 12 Constant-valued Blank Node Subject Map.

```
1 <#TM6> rml:subject _:Person.
```

Subject Class definition. A Subject Map may have one or more class IRIs, represented by the `rml:class` predicate, demonstrating the subject’s class. For each RDF term generated by the Subject Map, RDF triples with predicate `rdf:type` and the class IRI as object will be generated. The class is defined with a constant IRI that symbolizes the semantic type of the subjects generated by the Subject Map. For example, if it is known that the contents of “Column0” are represented by the class “`ex:City`”², its cells can be annotated with this class (Listing 13) within the Subject Map.

Listing 13 Subject Map Class.

```
1 <#TM7> rml:subjectMap <#SM7>.
2 <#SM7> rml:template "http://example.com/City/{Column0}"
3         rml:class "http://example.com/City".
```

3.2 Predicate Map Definition Analysis

Predicate Maps’ value is defined with an IRI value that symbolizes the relationship between two terms. For example, since “Column0” contains country capitals and “Column2” contains corresponding countries, their terms can be related using the predicate “`ex:capitalOf`”² (Listing 14).

Alternatively, the Predicate Map may be defined by a column whose contents correspond to RDF predicates or can be used to create valid RDF predicates.

Listing 14 Constant-valued Predicate Map.

```
1 <#POM1> rml:predicateMap <#PM1>.
2 <#PM1> rml:constant "http://example.com/capitalOf".
```

3.3 Object Map Definition Analysis

An Object Map can be constructed with different methods depending on factors such as the existence of a column or a combination of columns that represent KG objects, as well as the columns’ semantic context (NE-column or L-column).

Table-based Object definition. A column can be directly referenced, if it already contains named entities in the form of IRIs with a *reference-valued* Object Map (e.g., Listing 15 for “Column4”).

Listing 15 Reference-valued IRI Object Map.

```
1 <#POM1> rml:objectMap <#OM1>.
2 <#OM1> rml:reference "Column4";
3         rml:termType rml:IRI.
```

Listing 16 Reference-valued Literal Object Map.

```
1 <#POM2> rml:objectMap <#OM2>.
2 <#OM2> rml:reference "Column1";
3         rml:datatype xsd:integer;
4         rml:termType rml:Literal.
```

Otherwise, the Object Map can be defined from a column or combination of columns that consists of non-IRI cells, i.e. plain text or numerals, representing resources or literal values, depending on the columns’ semantics. If a combination of columns represents literal values (L-column), e.g., “Column1” which represents population numbers, a *reference* (Listing 16) or *template-valued* (Listing 17) literal Term Map can be used to define the Object Map.

■ **Listing 17** Template-valued Literal Object Map.

```

1 <#POM3> rml:objectMap <#OM3>.
2 <#OM3> rml:template "{Column1} people";
3 rml:datatype xsd:string;
4 rml:termType rml:Literal.

```

■ **Listing 18** Template-valued IRI Object Map.

```

1 <#POM4> rml:objectMap <#OM4>.
2 <#OM4> rml:template "http://example.com
3 /Country/{Column2}";
4 rml:termType rml:IRI.

```

If a combination of columns represents resources (NE-column), e.g., “Column2” which represents countries, a *template-valued* IRI (referencing named entities, Listing 18), or a *template-valued* or *reference-valued* Blank Node Object Map can be defined (Listing 19).

■ **Listing 19** Reference-valued Blank Node Object Map.

```

1 <#POM5> rml:objectMap <#OM5>.
2 <#OM5> rml:reference "Column2";
3 rml:termType rml:BlankNode.

```

■ **Listing 20** Constant-valued IRI Object Map.

```

1 <#POM6> rml:objectMap <#OM6>.
2 <#OM6> rml:constant "http://example.com/
3 City";
4 rml:termType rml:IRI.

```

Non-table-based Object Map definition. An Object Map can also be defined from a single or multiple resources or literals which are not part of the table. For example, “Column0” cells may be described by a single resource or literal value that is not part of the table. Then, a *constant-valued* Term Map can be used to define a single object that describes “Column0” cells. If the object is known, a *constant-valued* IRI or Literal Term Map can be used (Listings 20 and 21, respectively), otherwise a *constant-valued* Blank Node Term Map referring to a single unknown object (Listing 22).

■ **Listing 21** Constant-valued Literal Object Map.

```

1 <#POM7> rml:objectMap <#OM7>.
2 <#OM7> rml:constant "City";
3 rml:termType rml:Literal.

```

■ **Listing 22** Constant-valued Blank Node Object Map.

```

1 <#POM8> rml:object _:Mayor.

```

Referencing Object Map definition. RML allows the definition of Object Maps by referencing the subjects of another Triples Map, using the Referencing Object Map (rml:RefObjectMap). A join between the Logical Sources is required, if the two Triples Maps reference different sources. The join condition (rml:joinCondition) performs the join of the sources similarly to SQL joins, consisting of a reference to a column name that is part of the Logical Source that defines the Referencing Object Map (rml:ChildMap) and a reference to a column name that is part of the Referencing Object Map's Parent Map (rml:ParentMap) [25, 46].

4 Approach: Declarative Mappings Generation using Semantic Annotations

In this section, we introduce our approach which leverages *STA* to automate declarative KG construction through automated declarative mappings generation. We investigate how different annotation tasks can be used in automating declarative mappings generation. This section references the same running example as Section 3.

4.1 Approach Formalization

Task Definition. We formalize the task of automating declarative KG construction through the automated generation of declarative mapping rules by leveraging semantic annotations for tabular data. Let $KG_{M,D}$ be the target *KG* constructed given an input data source *D* and a set of declarative mappings *M* that define the transformation of *D* [27, 28]. Let *D* be a table *T*, as

defined in Section 2.1 and let KG_{ref} be an existing reference KG that can be used to semantically annotate T . In this paper, our task is to automatically generate M using semantic annotation results $\mathcal{A}$ derived from KG_{ref} . The KG derived from M and T , $KG_{M,T}$, will then be a set of RDF triples populated with data from T , semantically typed and linked according to the schema and entities defined in KG_{ref} .

Automated Generation of Declarative Mappings. We propose the automated generation of M through semantic annotations $\mathcal{A}$ derived from the STA tasks introduced in Section 2.1. In RML, M is formally defined as a finite non-empty set of RML Triples Maps TM , as defined in Section 2.1. In what follows, we introduce our method for automatically generating each TM leveraging semantic annotation results $\mathcal{A}$, building on the formal definition of each in literature [27, 28]:

- **Logical Source ($\mathcal{LS}$).** A Logical Source denotes the path of input data source T , $path_T$. However, derived versions of T , that include additional semantic information can also be used to define $\mathcal{LS}$. We detail the automatic generation of $\mathcal{LS}$ in Section 4.3.1, using semantic annotations to decide on which version of T is referenced.
- **Subject Map (S_{map}).** The Subject Map is formally defined as $S_{map} = (ST_{map}, CL, G_{map})$ (see Section 2.1). We argue that S_{map} can be inferred by automating the generation of ST_{map} and CL , leveraging $\mathcal{A}$. G_{map} is by default undefined, since all generated triples are assumed to belong to the default named graph, unless otherwise specified. We detail the automatic generation of S_{map} through ST_{map} and CL in Section 4.3.2, using semantic annotations.
- **Predicate-Object Map (PO_{map}).** Each $PO_{map} \in POM$ is a tuple $(\mathcal{PM}, \mathcal{OM}, G_{map})$ (see Section 2.1). We argue that PO_{map} can be inferred by automating the generation of each $P_{map} \in \mathcal{PM}$ and each $O_{map} \in \mathcal{OM}$, leveraging $\mathcal{A}$. G_{map} is defined undefined, unless specified otherwise, as mentioned earlier. We detail the automatic generation of PO_{map} through P_{maps} and O_{maps} in Section 4.3.3, using semantic annotations.

We analyze how STA complementary annotations lay the foundations for automated generation of declarative mappings, and how semantic annotations fully enable it in Sections 4.2 and 4.3 respectively, according to our approach. We show how the Logical Source, Subject Map, and Predicate-Object Map are generated automatically using $\mathcal{A}$ in Sections 4.3.1, 4.3.2, and 4.3.3, respectively. Finally, we discuss the edge case of tables containing n-ary column relationships, in which the results of $\mathcal{A}_{CPA}$ can be leveraged, in Section 4.3.4.

Table 1 summarizes the correspondence between the STA annotations and the formal components of the generated RML mappings. Following Equations. 1–5 the table identifies the RML component or field affected by each annotation task and specifies the fallback applied when the corresponding annotation is unavailable.

4.2 Structural Table Annotations: Mappings Generation Foundations

The STA complementary annotation results $(\mathcal{A}_{SCD}, \mathcal{A}_{ECD}, \mathcal{A}_{CDA}, \mathcal{A}_{CLA})$ (Section 2.1) that *STA systems* employ provide crucial initial information for declarative mapping definition. Our approach leverages STA complementary annotations as the starting point for declarative mappings definition automation:

- $\mathcal{A}_{SCD}$ informs the creation of a Subject Map, establishing a column as the one that contains resources that the rest of the columns describe. The identification of no subject column reveals that the table describes a single or multiple entities that are not part of the table.

■ **Table 1** Role of STAs in Formal RML Mapping Generation. The table summarizes how each STA task contributes to the formal equations of the mapping generation procedure, which RML component is affected, and the corresponding fallback case of each equation.

STA annotation task	Equations	Affected RML component	Fallback of the equation
Subject Column Detection (SCD)	Eq. 2	Subject Term Map ST_{map}	$(\mathcal{VT}, type)$ in $ST_{map} = (ref, \mathbf{BlankNode})$; A Blank Node Subject Map is generated.
Entity Column Detection (ECD)	Eqs. 1 and 5	Logical Source path $\mathcal{S}_{path}$ in $\mathcal{LS}$ and term type $type$ of an object map O_{map} .	In Eq. 1: the path of the original input table $path_T$ is used. In Eq. 5, $(\mathcal{VT}, type, datatype, lantag)$ in $O_{map} = (ref con, \mathbf{BlankNode}, \perp, \perp)$: the object is represented as a Blank Node.
Cell-Entity Annotation (CEA)	Eqs. 2 and 5	The value term $\mathcal{VT}$ of a subject term map ST_{map} or an Object Map O_{map}	$\mathcal{VT} = temp_{c_j}$; the value term is generated from the raw cell values of c_j using a <i>template-valued</i> RML Term Map
Column-Type Annotation (CTA)	Eq. 3	Class set CL of a subject map S_{map}	$CL = \emptyset$; no Class assertion is generated for S_{map}
Column-Property Annotation (CPA)	Eq. 4	Predicate Map P_{map} and its value term $\mathcal{VT}$	$\mathcal{VT} = \perp$; P_{map} is not generated
Column-Data-type Annotation (CDA)	Eq. 5	Object Map O_{map} and its datatype field $datatype$	$datatype = \perp$; the Object Map generates a literal without an explicit datatype.
Column-Language Annotation (CLA)	Eq. 5	Object Map O_{map} and its language-tag field $lantag$	$lantag = \perp$; the Object Map generates a literal without a language tag.

- $\mathcal{A}_{ECD}$ informs the creation of **Predicate-Object Maps**, indicating whether each column represents RDF resources (*NE-columns*) or literal values (*L-columns*). Also, the identification of *NE-columns* leads to the creation of additional **Subject Maps** for these columns, if more semantic information is available about them during the annotation process, such as the entity class that a column's contents belong in.
- $\mathcal{A}_{CDA}$ informs the **Literal Object Maps'** datatype.
- $\mathcal{A}_{CLA}$ informs the **Literal Object Maps'** language tag, for textual columns.

► **Example 1.** Assuming that “Column0” and “Column2” are annotated as *NE-columns*, with the former identified as the subject column, while “Column1”, “Column3”, and “Column4” are *L-columns*, with custom datatypes “Integer”, “Integer”, and “URL” respectively. This information already provides the foundation for generating the corresponding RML mappings. An example initial foundation of RML mappings using structural annotations is showcased in Listing 23.

■ **Listing 23** RML Template example generated from structural annotations.

```

1 <#PersonTriplesMap> a rml:TriplesMap;
2   rml:logicalSource [ a rml:LogicalSource;
3     rml:source <FILLIN_SOURCE>;
4     rml:referenceFormulation ql:CSV];
5   #Column0 Subject Map
6   rml:subjectMap [ a rml:SubjectMap;
7     <FILLIN_SUBJECT>
8     rml:class <FILLIN_COLUMN_TYPE>]
9   #Column1 Predicate Object Map
10  rml:predicateObjectMap [
11    rml:predicate <FILLIN_PREDICATE>;
12    rml:objectMap [ a rml:ObjectMap;
13      rml:reference "Column1";
14      rml:datatype xsd:integer];
15
16
17  #Column2 Predicate Object Map
18  rml:predicateObjectMap [
19    rml:predicate <FILLIN_PREDICATE>;
20    rml:objectMap [ a rml:ObjectMap;
21      <FILLIN_OBJECT>];
22  #Column3 Predicate Object Map
23  rml:predicateObjectMap [
24    rml:predicate <FILLIN_PREDICATE>;
25    rml:objectMap [ a rml:ObjectMap;
26      rml:reference "Column3";
27      rml:datatype xsd:integer];
28  #Column4 Predicate Object Map
29  rml:predicateObjectMap [
30    rml:predicate <FILLIN_PREDICATE>;
31    rml:objectMap [ a rml:ObjectMap;
32      rml:reference "Column4";
33      rml:datatype xsd:anyURI];

```

These initial mappings can then be filled in by semantic annotations inferred to enable fully automated declarative KG construction or be manually filled in by users, enabling a semi-automatic process in describing declarative rules. Their definition varies depending on available semantic information such as information about the existence of qualifier columns. This will be explored further in the remainder of the paper.

4.3 Semantic Table Annotations for Automated Declarative Mappings Definition

Semantic annotations including (i) the identification of *entities* contained within the table (CEA / entity disambiguation), (ii) their *semantic types* (CTA / entity type disambiguation) and (iii) the *relationships between columns* (CPA-CQA / relation extraction) are crucial for semantically understanding tables and transforming them to KGs [40]. Our approach leverages the results of these tasks to automatically generate declarative mappings, building on initial foundations set using structural annotations, ultimately enabling automated declarative KG construction.

Specifically, the **CEA task** identifies the resources that named entities reference, which we use to define Subject Maps (e.g., in Listing 23, line 7) and Object Maps (e.g., line 19) for *NE-columns*.

► **Example 2.** If the cell value “Oslo” in “Column0” is annotated by CEA with the entity “ex:Oslo”², the IRI that represents the subject resource can be referenced by the Subject Map. Similarly, if the cell value “Norway” in “Column2” is annotated with the Wikidata entity “ex:Norway”², the Object Map can reference the corresponding IRI.

The **CTA task** determines the resource types associated with columns, which we use to define the classes of Subject Maps (e.g., line 8).

► **Example 3.** If “Column0” is annotated by CTA with the entity “ex:City”², this IRI can be referenced by the Subject Map to define its class. As a result, each subject generated from the column is explicitly typed as an instance of “ex:City”.

The **CPA task** identifies the relationships between columns, which we use to define Predicate Maps (e.g., lines 11, 17, 22, 28).

► **Example 4.** If “Column0” is related to “Column2” through the predicate “ex:capitalOf”² according to CPA, this IRI can be referenced by the Predicate Map to define the predicate between the two columns. The CQA task identifies n-ary relationships between columns. N-ary relationships require different strategy in defining declarative mappings, which is investigated in Section 4.3.4.

In the remainder of this section, we break down our approach to leverage semantic information, derived from STA systems, for synthesizing declarative mappings for KG construction. We explore different scenarios and show how semantic annotations can either be pre-computed by STA systems or inferred during declarative mapping processing by leveraging custom mapping functions.

4.3.1 Logical Source

If the table contains *NE-columns* represented by text, these columns cannot be referenced directly for KG construction since they do not contain the corresponding named entity IRIs. Let $\mathcal{A}_{ECD}$ be the set of structural column annotations, where a pair $(c_j, ne_col) \in \mathcal{A}_{ECD}$ indicates that a column c_j contains named entities. We automatically generate $\mathcal{LS}$'s path, S_{path} , from $\mathcal{A}_{ECD}$ such that:

$$S_{path} = \begin{cases} path_{T^*}, & \text{if } (c_j, ne_col) \in \mathcal{A}_{ECD} \\ path_T, & \text{otherwise.} \end{cases} \quad (1)$$

where T is the original input table, T^* represents a derived, semantically enhanced, view of T that includes an updated column for every *NE-column* c_j , populated with IRIs from the semantic annotation set $\mathcal{A}_{CEA}$, and $path_D$ represents the physical path to an input source D . ne_col is the ECD annotation label that identifies column c_j as a *NE-Column* as shown in Section 2.1. If an *NE-column* is already represented with IRIs, it can directly be used.

► **Example 5.** In the running example, “Column0” contains the city name “Oslo” as plain text. Since “Column0” is identified as an *NE-column*, and the CEA annotation links “Oslo” to the entity IRI <http://www.wikidata.org/entity/Q585>, the original table does not directly provide the identifier required by the mapping. The annotation can therefore be materialized in a semantically enhanced table T^* , whose updated column contains the corresponding entity IRI. Consequently, Equation 1 selects $path_{T^*}$ as the logical-source path. If no textual *NE-column* requires semantic enhancement, the equation selects $path_T$ instead, as illustrated in Figure 3.

Column0_IRI	Column1	Column2_IRI	Column3	Column4
wd:Q585	717,000	wd:Q20	2024	city:oslo
wd:Q85	9,717,324	wd:Q79	2018	city:cairo

wd: <http://www.wikidata.org/entity/>
city: <http://example.com/City/>

■ **Figure 3** Semantically enhanced table T^* with cell entity annotations corresponding to T .

4.3.2 Subject Map

We reference the analysis conducted in Section 3.1 for automatically defining the **Subject Map** leveraging semantic annotations. Due to the heterogeneity of tabular data, the automatic generation of the S_{map} adapts based on the existence of a subject column and the availability of semantic annotations (Figure 4). In our approach, the generation of S_{map} is automatically inferred from the generation of ST_{map} and CL .

Subject Term Map. For a Subject Term Map $ST_{map} = (\mathcal{VT}, type, datatype, langtag)$, we set $ST_{map} = (\mathcal{VT}, type, \perp, \perp)$, since $datatype$ and $langtag$ are undefined for Subject Maps (see Section 2.1). Hence, ST_{map} can be automatically inferred by inferring the value of its $\mathcal{VT}$ and $type$. We argue that $\mathcal{VT}$ and $type$ are automatically generated based on $\mathcal{A}_{CEA}$ and $\mathcal{A}_{SCD}$ such that:

$$(\mathcal{VT}, type) = \begin{cases} (ref_{c_j^*}, IRI), & \text{if } (c_j, sub_col) \in \mathcal{A}_{SCD} \ \& \ (v_{ij}, e) \in \mathcal{A}_{CEA} \\ (temp_{c_j}, IRI) & \text{if } (c_j, sub_col) \in \mathcal{A}_{SCD} \ \& \ (v_{ij}, e) \notin \mathcal{A}_{CEA} \\ (ref, BlankNode) & \text{otherwise} \end{cases} \quad (2)$$

If c_j is identified as the subject column sub_col through $\mathcal{A}_{SCD}$ and CEA results ($\mathcal{A}_{CEA}$) exist for c_j cells, we reference the enriched IRI column c_j^* from T^* with a *reference-valued* RML Term Map ($ref_{c_j^*}$) to define $\mathcal{VT}$ using the retrieved CEA labels. Listing 7 demonstrates an example. If no CEA results exist for c_j , the value-term can be generated using the raw cell values of c_j with a *template-valued* RML Term Map ($temp_{c_j}$). Listing 9 demonstrates an example. If no subject column is identified, the table is assumed to describe a single entity or a collection of resources not explicitly listed in a column. A Blank Node map is used to represent the anonymous subject, optionally with a *reference-valued* RML Term Map which can reference one or more columns. Listing 8 demonstrates an example. For every additional *NE-column* ne_col , a secondary Subject Map can be generated following the same logic, allowing the description of other table entities.

► **Example 6.** For the first row, “Column0” contains “Oslo”, and $\mathcal{A}_{SCD}$ identifies “Column0” as the subject column. If CEA contains

$$(\text{“Oslo”}, http://www.wikidata.org/entity/Q585) \in \mathcal{A}_{CEA},$$

then Equation 2 selects its first case:

$$(\mathcal{VT}, type) = (ref_{c_0^*}, IRI).$$

Thus, the Subject Map references the IRI stored in the enhanced column “Column0*”, and the subject generated for the first row is <http://www.wikidata.org/entity/Q585>. Listing 24 shows this reference-valued Subject Map.

■ **Listing 24** Subject Map definition with CEA, CTA annotations.

```
1 <#RMLTriplesMap> a rml:TriplesMap;
2   rml:logicalSource <#SemanticSource>;
3   rml:subjectMap [a rml:SubjectMap;
4     rml:reference "Column0*"]].
```

If CEA annotations are not provided, they can also be retrieved through a custom RML-FNML mapping function, referencing the original data source. A mapping function that retrieves CEA labels can leverage different types of input to do so, such as the content of the cells, prior knowledge or other available semantic annotations, depending on the use case.

► **Example 7.** If no explicit CEA result is available for “Column0”, Equation 2 selects the template-valued case $(\mathcal{VT}, type) = (temp_{c_0}, IRI)$, which constructs an IRI from the raw value “Oslo”. Alternatively, if “Column0” is known to contain cities, for example through the CTA annotation $(c_0, Q515) \in \mathcal{A}_{CTA}$, a custom RML-FNML function can use the cell value and this type information to retrieve the CEA IRI. The resulting function execution provides an alternative to the template-valued fallback and can be used to define the Subject Map, as shown in Listing 25.

■ **Listing 25** Subject Map definition referencing custom function that infers CEA labels using column cells and CTA label.

```

1 <#RMLTriplesMap> a rml:TriplesMap;
2   rml:logicalSource <#OriginalSource>;
3   rml:subjectMap [a rml:SubjectMap;
4     rml:functionExecution
5     <#Get_CEA_Execution>;
6     rml:return ex:ceaIri;].

```

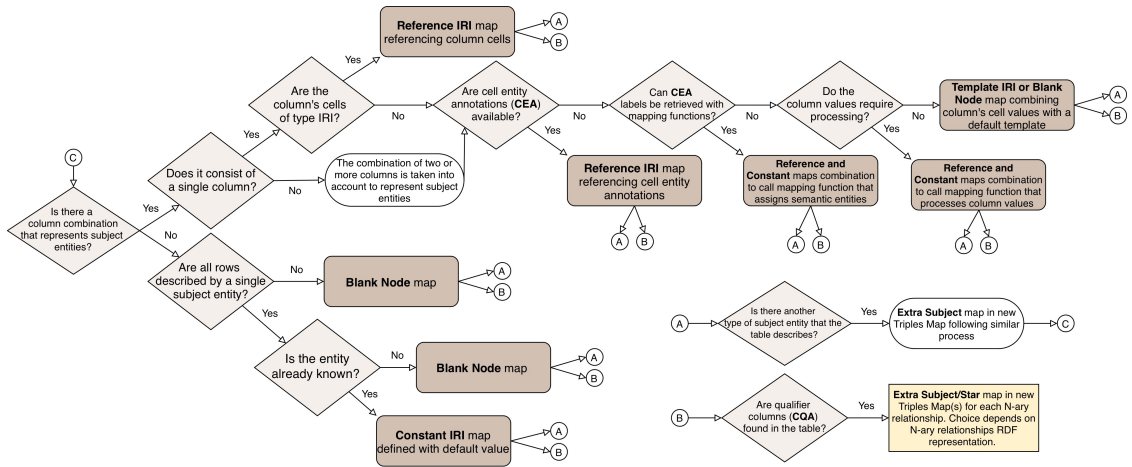
```

7 <#Get_CEA_Execution> a rml:FunctionExecution;
8   rml:function ex:getCEA;
9   rml:input [a rml:Input;
10     rml:parameter ex:valueParam;
11     rml:inputValue [ rml:reference
12       "Column0"],
13     [a rml:Input;
14       rml:parameter ex:typeParam;
15       rml:inputValue [rml:constant
16         www.wikidata.org/entity/Q515] ].

```

An overview of the analysis for automating the definition of Subject Maps by leveraging available structural and semantic information can be seen in Figure 4. The possibility of qualifier columns being present is investigated in Section 4.3.4.

■ **Figure 4** Subject Map automated definition flowchart for tabular data with semantic annotations.



Subject Class. The set of classes, CL can be automatically defined using CTA results ($\mathcal{A}_{CTA}$) such that:

$$CL = \{e \mid (c_j, e) \in \mathcal{A}_{CTA}\} \quad (3)$$

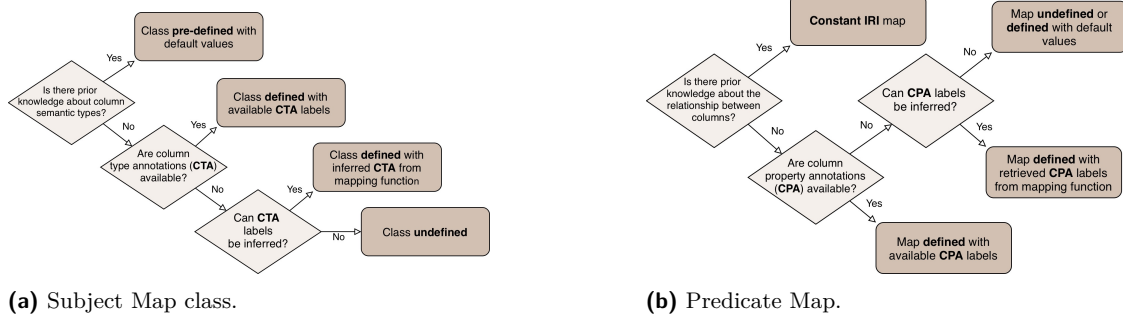
Every entity e retrieved as an annotation of a column c_j , can be used to define the class of the respective Subject Map. If no annotation is retrieved, CL is an empty set, hence no class is defined.

► **Example 8.** Suppose that CTA annotates “Column0” with the class $ex:City$. Then $(c_0, ex:City) \in \mathcal{A}_{CTA}$. Equation 3 consequently gives $CL = \{e \mid (c_0, e) \in \mathcal{A}_{CTA}\} = \{ex:City\}$. The generated subject for “Oslo” can therefore be assigned the class $ex:Oslo$ `rdf:type` $ex:City$.

If CTA annotations are not provided, they can also be retrieved through a custom RML-FNML mapping function, referencing the original data source. A mapping function that retrieves CTA labels can leverage different types of input to do so, such as the content of the cells, prior knowledge or other available semantic annotations, depending on the use case.

► **Example 9.** If no CTA result is available, a custom function can use the contents of “Column0” to infer its semantic class. For example, a lookup function can return $ex:City$ for the column based on its values. Formally, the function supplies an inferred annotation $(c_0, ex:City) \in \mathcal{A}_{CTA}^*$, where $\mathcal{A}_{CTA}^*$ denotes the set of annotations inferred during mapping generation. Equation 3 can

Figure 5 Subject Map class (left) and Predicate Map (right) automated definition flowchart for tabular data with semantic annotations.



then be applied to $\mathcal{A}_{CTA}^*$, resulting in $CL = \{ex:City\}$. Listing 26 illustrates this alternative implementation using an additional Predicate-Object Map with the `rdf:type` predicate.

Listing 26 Subject Map class definition referencing custom function that infers CTA labels using column cells.

```

1 <#RMLTriplesMap> a rml:TriplesMap;
2   rml:logicalSource <#OriginalSource>;
3   rml:subjectMap <#SubjectMap>;
4   rml:predicateObjectMap [
5     rml:predicateMap rdf:type;
6     rml:objectMap [a rml:ObjectMap;
7       rml:functionExecution
8         <#Get_CTA_Execution>;
9       rml:return ex:ctaIri];].
10 <#Get_CTA_Execution> a rml:FunctionExecution;
11   rml:function ex:getCTA;
12   rml:input [a rml:Input;
13     rml:parameter ex:valueParam;
14     rml:inputValue [ rml:reference
15       "Column0"]].

```

An overview of the analysis for automating the definition of Subject Map Classes by leveraging available structural and semantic information can be seen in Figure 5a.

4.3.3 Predicate-Object Map

Each $PO_{map} \in \mathcal{POM}$ can be inferred by automating the generation of P_{map} and O_{map} , leveraging $\mathcal{A}$. We detail the generation of P_{map} and O_{map} in what follows.

Predicate Map. We refer to the analysis conducted in Section 3.2 for automatically defining the Predicate Map leveraging semantic annotations. For a Predicate Map, $P_{map} = (\mathcal{VT}, type, datatype, langtag)$, we set $P_{map} = (\mathcal{VT}, IRI, \perp, \perp)$, since P_{map} always has an IRI *type* and *datatype* and *langtag* are undefined for Predicate Maps (see Section 2.1). Hence, P_{map} can be automatically inferred by inferring the value of its $\mathcal{VT}$. We argue that $\mathcal{VT}$ for a P_{map} between two columns c_a and c_b is automatically generated based on $\mathcal{A}_{CPA}$ such that:

$$\mathcal{VT} = \begin{cases} con_p, & \text{if } ((c_a, c_b), p) \in \mathcal{A}_{CPA}, \\ \perp, & \text{otherwise.} \end{cases} \quad (4)$$

If CPA results exist in $\mathcal{A}_{CPA}$ linking the column c_a to c_b , the predicate is defined as a constant IRI p (e.g., `ex:capitalOf`) with a *constant-valued* RML Term Map (con_p). If there is no identified relation between c_a and c_b , no Predicate Map is defined between these columns.

► **Example 10.** In the running example, CPA identifies a relationship between “Column0” and “Column2”: $((c_0, c_2), ex:capitalOf) \in \mathcal{A}_{CPA}$. Therefore, Equation 4 selects its first case and sets $\mathcal{VT} = con_p$ and $con_p = ex:capitalOf$. The resulting constant-valued Predicate Map connects the

subject generated from “Column0” to the object generated from “Column2”. If no CPA annotation exists for (c_0, c_2) , the equation sets $\mathcal{VT} = \perp$, and no corresponding Predicate Map is generated unless a predicate is supplied through a custom function or manual domain knowledge.

If CPA annotations are not provided, they can also be retrieved through a custom RML-FNML mapping function, referencing the original data source. A mapping function that retrieves CPA labels can leverage different types of input to do so, such as the content of the cells, prior knowledge or other available semantic annotations, depending on the use case.

► **Example 11.** A function can process the values in “Column0” and “Column2” and return *ex:capitalOf*. Formally, the function supplies an inferred predicate $p^* = \Phi_{CPA}(c_0, c_2) = ex:capitalOf$. The predicate map can then be generated with

$$\mathcal{VT} = con_{p^*},$$

even though the original CPA set does not contain $((c_0, c_2), p)$. Listing 27 shows how this function execution can be referenced from the Predicate Map.

■ **Listing 27** Predicate Map definition referencing custom function that infers CPA labels using column cells.

```

1  <#RMLTriplesMap> a rml:TriplesMap;
2    rml:logicalSource <#OriginalSource>;
3    rml:subjectMap [a rml:SubjectMap;
4      rml:reference "Column0"];
5    rml:predicateObjectMap [
6      rml:predicateMap [
7        rml:functionExecution
8          <#Get_CPA_Execution>;
9        rml:return ex:capitalOf;
10     rml:objectMap [a rml:ObjectMap;
11       rml:reference "Column2";];];].
12
13 <#Get_CPA_Execution> a rml:FunctionExecution;
14   rml:function ex:getCPA;
15   rml:input [a rml:Input;
16     rml:parameter ex:valueParam1;
17     rml:inputValue [ rml:reference
18       "Column0"],
19     [a rml:Input;
20       rml:parameter ex:valueParam2;
21       rml:inputValue [ rml:reference
22         "Column2"] ] ].

```

An overview of the analysis for automating the definition of Predicate Maps by leveraging available structural and semantic information can be seen in Figure 5b.

Object Map. We reference the analysis conducted in Section 3.3 for automatically defining the Object Map leveraging semantic annotations. Our approach unifies the handling of *L-columns* and *NE-columns* into a single definition for a column c_j based on its object column type and the availability of semantic annotations. Every Object Map $O_{map} \in PO_{map}$, where $O_{map} = (\mathcal{VT}, type, datatype, lantag)$, is automatically generated by inferring $\mathcal{VT}$, *type*, *datatype*, and *lantag*. For each column c_j that represents object entities according to $\mathcal{A}_{CPA}$ we define the components of O_{map} as follows:

$$(\mathcal{VT}, type, datatype, lantag) = \begin{cases} (ref_{c_j} | temp_{c_j}, Literal, \perp, l), & \text{if } (c_j, l_col) \in \mathcal{A}_{ECD} \text{ \& } (c_j, l) \in \mathcal{A}_{CLA} \\ (ref_{c_j} | temp_{c_j}, Literal, d, \perp), & \text{if } (c_j, l_col) \in \mathcal{A}_{ECD} \text{ \& } (c_j, d) \in \mathcal{A}_{CDA} \\ (ref_{c_j} | temp_{c_j}, Literal, \perp, \perp), & \text{if } (c_j, l_col) \in \mathcal{A}_{ECD} \\ (ref_{c_j}^*, IRI, \perp, \perp), & \text{if } (c_j, ne_col) \in \mathcal{A}_{ECD} \text{ \& } (v_{ij}, e_{ij}) \in \mathcal{A}_{CEA} \\ (temp_{c_j}, IRI, \perp, \perp), & \text{if } (c_j, ne_col) \in \mathcal{A}_{ECD} \text{ \& } (v_{ij}, e_{ij}) \notin \mathcal{A}_{CEA} \\ (ref | con, BlankNode, \perp, \perp), & \text{otherwise} \end{cases} \quad (5)$$

If c_j is identified as an *L-column* through $\mathcal{A}_{ECD}$, then a *reference-valued* or *template-valued* RML Literal Term Map is generated $(ref_{c_j} | temp_{c_j})$, as demonstrated in Listings 16 and 22, respectively. Depending on the existence of datatype and language annotation results ($\mathcal{A}_{CDA}$, $\mathcal{A}_{CLA}$), the Object Map can also have a *datatype* or *lantag*. Note that both cannot co-exist in RML [27]. If c_j is identified as an *NE-column* and CEA annotation results ($\mathcal{A}_{CEA}$) are available for c_j , a

reference-valued RML Term Map is generated ($ref_{c_j^*}$) referencing the enriched IRI column c_j^* from T^* . Listing 15 demonstrates an example. If $\mathcal{A}_{CEA}$ are not available, a *template-valued* RML Term Map ($temp_{c_j}$) is generated using the raw cell values of c_j , as demonstrated in Listing 18. If no explicit annotation is provided as part of $\mathcal{A}_{ECD}$, a Blank Node map is used to represent the anonymous object, optionally with a *reference-valued* RML Term Map which can reference one or more columns or a *constant-valued* RML Term Map referencing a single unknown object. Listings 14 and 19 demonstrate examples.

► **Example 12.** For the first row, “Column2” contains “Norway”. Since “Column2” is identified as an *NE-column* and CEA contains (“Norway”, $ex:Norway$) $\in \mathcal{A}_{CEA}$, Equation 5 selects the CEA-based case:

$$(\mathcal{VT}, type, datatype, langtag) = (ref_{c_2^*}, IRI, \perp, \perp).$$

Thus, the generated object is $ex:Norway$. Listing 28 shows the corresponding Object Map, which references the enriched column “Column2*”.

■ **Listing 28** Object Map definition referencing inferred CEA labels.

```
1 <#RMLTriplesMap> a rml:TriplesMap;
2   rml:logicalSource <#SemanticSource>;
```

```
3   rml:subjectMap <#SubjectMap>;
4   rml:predicateObjectMap [
5     rml:predicateMap <#PredicateMap>;
6     rml:objectMap [a rml:ObjectMap;
7       rml:reference "Column2"]].
```

If CEA annotations are not provided, they can also be retrieved through a custom RML-FNML mapping function, referencing the original data source, as previously showcased.

► **Example 13.** If no CEA result is available for “Column2”, Equation 5 selects the template-valued IRI case: $(\mathcal{VT}, type, datatype, langtag) = (temp_{c_2}, IRI, \perp, \perp)$. Alternatively, a custom function can use the cell values together with available semantic information, such as the CTA annotation $(c_2, Q6256) \in \mathcal{A}_{CTA}$, to infer the CEA IRI for “Norway”. In that case, the function retrieves a CEA IRI, which is then used to define an IRI-valued Object Map, replacing the template-valued fallback. This process is illustrated in Listing 29.

■ **Listing 29** Object Map definition referencing custom function that infers CEA labels using column cells and CTA labels.

```
1 <#RMLTriplesMap> a rml:TriplesMap;
2   rml:logicalSource <#OriginalSource>;
3   rml:subjectMap <#SubjectMap>;
4   rml:predicateObjectMap [
5     rml:predicateMap <#PredicateMap>;
6     rml:objectMap [a rml:ObjectMap;
7       rml:functionExecution
8         <#Get_CEA_Execution>;
9       rml:return ex:ceaIri];].
```

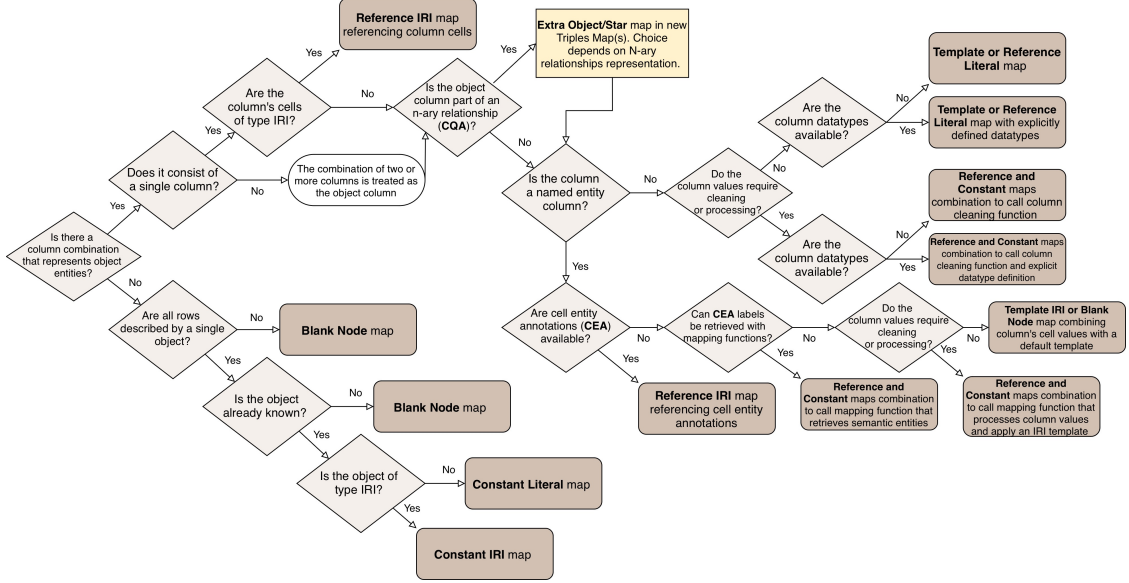
```
10 <#Get_CEA_Execution> a rml:FunctionExecution;
11   rml:function ex:getCEA;
12   rml:input [a rml:Input;
13     rml:parameter ex:valueParam;
14     rml:inputValue [
15       rml:reference "Column2"],
16     [a rml:Input;
17       rml:parameter ex:typeParam;
18       rml:inputValue [rml:constant
19         www.wikidata.org/entity/Q6256] ].
```

An overview of the analysis for automating the definition of Object Maps by leveraging available semantic annotations can be seen in Figure 6. The possibility of qualifier columns being present is investigated in Section 4.3.4.

4.3.4 Triples Maps for N-ary Column Relationships

We examine the edge case of n-ary column relationships in tables. As seen in Section 2.1, n-ary column relationships can also be expressed by tabular columns, with some columns (qualifier columns) further describing a relationship between a set of columns. For our running example, it was shown that “Column3” may not directly provide information about another column of the table, but provide additional information for the relationship between “Column0” and “Column1”, thus forming an 3-ary relationship between these columns.

■ **Figure 6** Object Map automated definition flowchart for tabular data with semantic annotations.



Different methods exist in representing statements about statements in RDF [29, 33, 62]. We explore the representation of n-ary column relationships with RDF-Star [33] and reification (i.e., Blank Nodes) [29], due to their wide adoption from the RDF community, to then investigate the automatic definition of their declarative mappings.

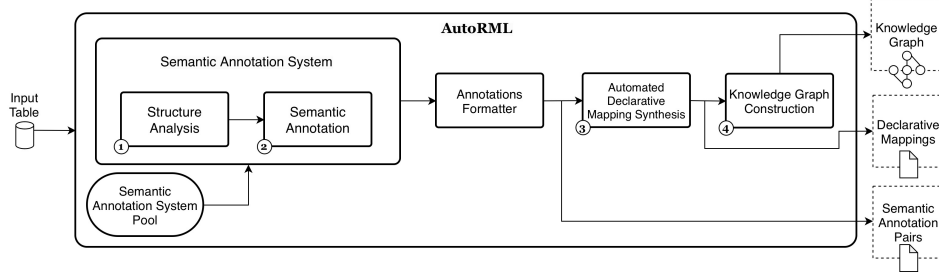
► **Example 14.** In the running example, “Column3” qualifies the population relationship between “Column0” and “Column1”. This information can be represented formally by a CQA annotation such as: $((\{c_0\}, \{c_1, c_3\}), ex:population) \in \mathcal{A}_{CQA}$, where “Column3” provides the qualifier for the population value in “Column1”. Unlike the binary CPA case, this annotation requires additional connected Triples Maps or an RDF-star Star Map. For example, the first row can generate the quoted triple: $\langle\langle ex:Oslo, ex:population, “717000” \rangle\rangle$ and attach the qualifier $ex:yearCollected$ “2024” to the quoted triple. Thus, CQA results guide the generation of the additional mapping structure needed for n-ary relationships, as shown in Listings 5 and 31. For n-ary relationships with $n > 3$, an additional Predicate Object Map can be added in the Triples Map for each qualifier column.

Alternatively, the usage of reification to describe the existing 3-ary column relationship can result in the RDF triples seen in Listing 30. Listing 31 shows how RML can be leveraged for declaratively constructing the corresponding triples. A Triples Map is used to define the reified triple ($\langle\#RMLTriplesMap1\rangle$) and a second Triples Map ($\langle\#RMLTriplesMap2\rangle$) is used to describe statements about it. Similarly with the previous approach, for n-ary relationships with $n > 3$, an additional Predicate Object Map can be added (in the $\langle\#RMLTriplesMap2\rangle$) for each additional qualifier column. The choice of approach depends on use case-specific requirements, such as querying complexity, RDF store capabilities and expressivity [29].

■ **Listing 30** Columns n-ary relationship with reification.

```
1 <http://example.com/Oslo> ex:populationMetric _:OsloPopulation.
2 _:OsloPopulation ex:population "717000"; ex:yearCollected "2024".
```

■ **Figure 7** Automated Declarative KG Construction from Tabular Data with AutoRML.



■ **Listing 31** Declarative construction of RDF triples that describe n-ary column relationships with reification.

```
1 <RMLTriplesMap1> a rml:TriplesMap;
2   rml:logicalSource <#LogicalSource>;
3   rml:subjectMap [rml:template
4     "http://ex.com/{Column0}"];
5   rml:predicateObjectMap[
6     rml:predicate ex:populationMetric;
7     rml:objectMap [a rml:ObjectMap;
8       rml:template "{city}Population";
9     rml:termType rml:BlankNode;];]
```

```
10 <RMLTriplesMap2> a rml:TriplesMap;
11   rml:logicalSource <#LogicalSource>;
12   rml:subjectMap [a rml:SubjectMap;
13     rml:template "{city}Population";
14     rml:termType rml:BlankNode];
15   rml:predicateObjectMap[
16     rml:predicate ex:population;
17     rml:objectMap [a rml:ObjectMap;
18       rml:reference "{Column1}"];];
19   rml:predicateObjectMap[
20     rml:predicate ex:yearCollected;
21     rml:objectMap [a rml:ObjectMap;
22       rml:reference "{Column3}"];].
```

Regardless of the approach, the generation of the corresponding declarative mappings can be automated with CQA information, by generating additional connected **Triples Maps** following the process described throughout this section. CQA provides information for both main properties and their qualifiers, if existing, thus informing the creation of additional **Triples Maps** to declaratively describe n-ary column relationships.

5 AutoRML

We validate our approach by implementing a novel system, AutoRML (Figure 7). AutoRML leverages *STA systems* to retrieve semantic and structural table annotations which are then used to synthesize declarative mappings. The declarative mappings are generated following the formal methodology developed (Section 4), currently supporting declarative mappings generation using structural annotations, CEA, CTA, and CPA labels. The syntactic correctness of the declarative mappings is validated through declarative mapping serialization and KG construction systems. Specifically, Yatter [44] is used to confirm the syntactic correctness of the generated YARRRML human-friendly mappings, also generating the corresponding RML mappings, and Morph-KGC [3, 18] is used to confirm the syntactic correctness of the generated RML mappings by declaratively constructing the corresponding KG automatically.

AutoRML is developed for use with different *STA systems* by including an annotation formatting component that pre-processes annotation outputs. After extracting annotations for an input table using one of the available *STA systems*, the annotations are transformed into a standard format, adopted from the SemTab challenge [34] through an annotation formatting module. We extend the SemTab annotations format by additionally including representations of subject columns, primary column annotations (*NE-column* or *L-column*) and column datatypes annotations, since the original format did not include a representation for structural annotations.

When not all semantic annotations are available, AutoRML produces indicative mappings, filling in the corresponding values with *template-valued* Term Maps, which can be processed further by the user. This allows the semi-automatic generation of declarative mappings, even when the capabilities of used *STA systems* are limited, providing a basis for declarative KG construction.

L-column datatypes are mapped to XSD Datatypes³. If no datatype annotations are inferred, *L-columns* are not assigned with a datatype when generating the declarative mappings.

AutoRML’s complexity highly depends on the STA system it employs. Its automated mapping generation and KG construction, after retrieving *STA* labels, has a linear time complexity, $O(n)$, with n being the total number of discrete annotations (semantic and structural) retrieved. AutoRML is available at: <https://github.com/dtai-kg/AutoRML>.

6 Experiments

We perform a diverse set of experiments to assess AutoRML’s performance using different tabular data collections and *STA systems*. We measure the amount of declarative rules generated, the accuracy of the automatically constructed KGs and the level of automation AutoRML achieves for different dataset collections and *STA systems*. All inferred structural and semantic annotations, generated YARRRML, RML, and RDF files can be found in AutoRML’s Zenodo repository⁴. We also present a case study for leveraging the interoperability of constructed KGs to infer novel knowledge for our data.

6.1 Datasets and STA Systems Evaluated

To evaluate AutoRML, we utilize five diverse tabular dataset collections (Table 2):

- **WikidataTables 2024R1** [36] contains a large number of artificially generated tables from diverse domains whose contents are derived from Wikidata. We use the collection’s validation set which is comprised of 500 tables, paired with labels for the Wikidata entities and properties that the tables’ contents correspond to.
- **HardTablesR1** [35] also consists of artificially generated tables whose contents are derived from Wikidata. It is comprised of tables that hindered the performance of past *STA systems*. We use the collection’s validation set which is comprised of 200 tables, paired with labels for the Wikidata entities and properties that the tables’ contents correspond to.
- **BiodivTab-DBP** [2] is a real-world dataset collection that includes data originating from biodiversity research. It is manually annotated with DBPedia entities.
- **Kaggle Diverse CSV Dataset Samples**⁵ is a collection of real-world data across different domains gathered to support diverse analytical challenges in data science. Tables that only contained numerical data were not included in our evaluation.
- **GTFS-Madrid**⁶ is a collections of tables used within the GTFS-Madrid-Benchmark [10] and it includes Madrid’s subway network tables.

The former three collections – **WikidataTables 2024R1**, **HardTablesR1**, and **BiodivTab-DBP** are labeled with named entities from large general-purpose knowledge bases. The remaining two – **Kaggle Diverse CSV** and **GTFS-Madrid**, represent real-world tabular collections which

³ https://www.w3.org/2011/rdf-wg/wiki/XSD_Datatypes

⁴ <https://doi.org/10.5281/zenodo.14535918>

⁵ <https://www.kaggle.com/datasets/samybaladram/multidisciplinary-csv-datasets-collection>

⁶ <https://github.com/oeg-upm/kgc-tutorial-iswc2020/tree/master/morph-csv/data/gtfs>

■ **Table 2** Dataset collections statistics.

Dataset Collection	Number of Tables	Avg Number of Rows	Avg Number of Columns
WikidataTables 2024R1	500	5.95	2.46
HardTablesR1	200	4.74	2.60
BiodivTab-DBP	50	259.06	23.96
Kaggle Diverse CSV DS	25	35.48	8.28
GTFS-Madrid	10	33.50	7.50

■ **Table 3** Automatically generated RML rules and KGs per dataset collection. TM stands for Term Maps and POM for Predicate-Object Maps.

Dataset Collection	Successful KG constructions		Avg Number of Triples per Table		Avg Number of TMs per Table		Avg Number of POMs per Table	
	AutoRML (TorchicTab)	AutoRML (MTab)	AutoRML (TorchicTab)	AutoRML (MTab)	AutoRML (TorchicTab)	AutoRML (MTab)	AutoRML (TorchicTab)	AutoRML (MTab)
WikidataTables 2024R1	500 (100%)	500 (100%)	14.63	14.56	1.53	1.47	3.00	2.93
HardTablesR1	200 (100%)	200 (100%)	13.78	13.74	1.52	1.50	3.11	3.09
BiodivTab-DBP	50 (100%)	17 (34%)	1309.68	2118.00	7.62	7.17	30.58	32.47
Kaggle Diverse CSV DS	24 (96%)	18 (72%)	187.46	123.72	2.33	5.88	9.67	14.56
GTFS-Madrid	10 (100%)	9 (90%)	22.80	70.89	2.90	3.22	9.40	9.55

also contain propriety data, not typically found in general-purpose knowledge bases. **Kaggle Diverse CSV** is selected because it includes tables from diverse fields, while **GTFS-Madrid** is selected because it includes human-curated RML mappings, enabling direct comparison between manually and automatically generated mappings.

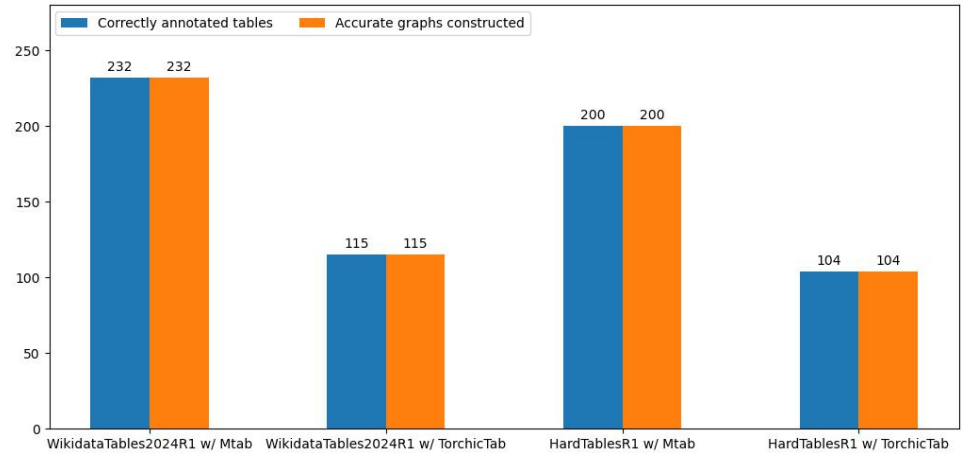
Our pool of *STA systems* for experiments included TorchicTab [19] and MTab [60], which were chosen as two of the most successful systems in the latest editions of the SemTab challenge [48, 34]. In our experiments, both systems annotated tables using Wikidata as their target KG.

6.2 Experiments Results

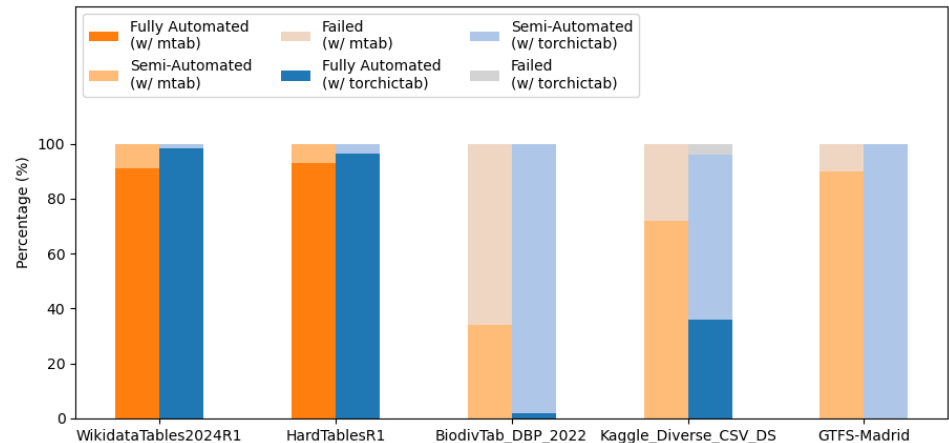
KG Construction Coverage. Table 3 highlights AutoRML’s success in declarative mapping generation and KG construction for different dataset collections. TorchicTab handles all tables but one, due to formatting errors it contained, while MTab exhibits difficulty in handling tables with unusual column header naming conventions. The number of **Term Maps** and **Predicate Object Maps** is on average proportional to the number of table columns, and the number of generated triples is on average proportional to the number of table rows, as expected. When the *STA system* successfully processes a table and produces any structural or semantic annotations, AutoRML produces syntactically correct mappings, filling in missing information with default **Term Maps**.

KG Construction Accuracy. To assess the accuracy of AutoRML, we use CEA, CPA and CTA ground-truth labels from dataset collections where such labels are available. For each table, we manually construct the KG corresponding to its ground-truth annotations, serving as the baseline. Our analysis focused on the two dataset collections with Wikidata ground truth annotations, WikidataTables 2024R1 and HardTablesR1, as both TorchicTab and MTab produce Wikidata IRI annotations, with Wikidata being their target KG. Our results (Figure 8) show that for all tables where entities, entity types, and relationships are correctly annotated, AutoRML successfully constructs KGs identical to the manually constructed ones, which are Wikidata subgraphs. The datatypes of literal objects are not taken into account in this metric, since such information is not provided in the ground truth annotations.

■ **Figure 8** Number of accurately annotated tables and corresponding accurate KGs automatically constructed by AutoRML.



■ **Figure 9** Percentages of automatic, semi-automatic and failed KG constructions.



KG Construction Automation. To quantify the degree of automation that AutoRML achieves, we define a KG construction from a table as *fully automated* when no default Term Maps need to be defined by AutoRML. Otherwise, we define a KG construction from a table as *semi-automated* when semantic annotations are incomplete or unavailable, in which case AutoRML still generates a mapping template that users can refine.

Our experiments reveal that for datasets associated with Wikidata concepts, AutoRML achieves a high degree of automation, since Wikidata is the target KG of the *STA systems* we tested (Figure 9). The completeness of the semantic annotations for these tables allows AutoRML to generate accurate KGs automatically. However, for the rest of the dataset collections which include proprietary or domain-specific data, not always linked with Wikidata concepts, the used *STA systems* retrieve less annotations. In addition, the larger the tables, the more likely it is for *STA systems* to not fully annotate all entities and relationships within the table. In these cases, AutoRML generates declarative mappings in a semi-automatic way, filling in missing annotations with default Term Maps.

6.3 Case Study: Knowledge Inference through Reasoning for Automatically Constructed Knowledge Graphs

To validate the semantic quality of constructed KGs, we conduct a case study demonstrating their semantic utility in knowledge inference through reasoning. A primary advantage of KGs over tabular data is the ability to use external resources for multi-hop reasoning and question-answering through interoperability. In our approach, since KG construction is based on a reference knowledge base KG_{ref} (e.g., Wikidata), the KG constructed by AutoRML, $KG_{M,T}$, is inherently interoperable with KG_{ref} . While T is an isolated file, $KG_{M,T}$ allows traversal of KG_{ref} for inferring knowledge that was not part of T . We verify this by validating that the generated IRIs are parts of KG_{ref} and by using these IRIs to infer new connections through KG_{ref} .

Following our running example (Table 2), AutoRML, with TorchicTab as its STA system, explicitly constructs the triple: $\langle \text{wd}^7:\text{Q585} \rangle$ (Oslo) a $\langle \text{wd}:\text{Q1549591} \rangle$ (Big City). Using this initial mapping of Oslo to a Big City we can utilize SPARQL queries to traverse the corresponding KG_{ref} , Wikidata, and infer more information about Oslo. For instance, through hierarchical reasoning, since Oslo is a Big City, we infer that Oslo is implicitly also a Human Settlement ($\langle \text{wd}:\text{Q486972} \rangle$) and a Populated Place ($\langle \text{wd}:\text{Q123964505} \rangle$). Also, in the original table, we only know of Oslo’s population and country, Norway. Through the mapping of the table cell Oslo to the Wikidata Oslo entity ($\langle \text{wd}:\text{Q585} \rangle$), we can link Oslo with more entities and attributes by traversing Wikidata, e.g., infer Oslo’s continent (Europe - $\langle \text{wd}:\text{Q46} \rangle$), time zone (UTC+01:00 - $\langle \text{wd}:\text{Q6655} \rangle$) and elevation above sea level (23 meters). Finally, we can infer textual descriptions and images for entities, if supported by KG_{ref} , particularly useful for retrieval and context enrichment. For instance, Oslo is described as “capital city of Norway” in Wikidata.

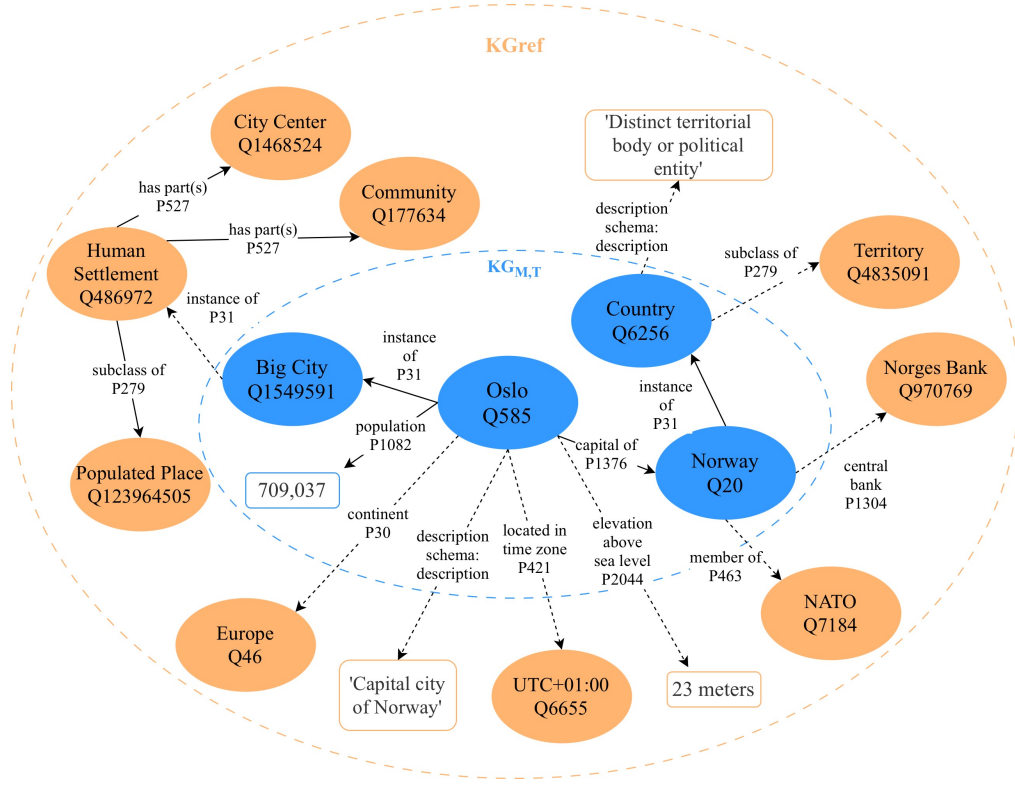
We present this case study for knowledge inference through KG reasoning for an example table in AutoRML’s repository at: <https://github.com/dtai-kg/AutoRML>. Figure 10 demonstrates an example of how knowledge can be inferred for $KG_{M,T}$ using the reference KG, KG_{ref} .

7 Discussion and Limitations

In this section, we discuss our approach’s evaluation results, its strengths and limitations, as well as the option of using declarative mapping functions to infer semantic information with custom methods and the limitations this may introduce.

Dependence on STA Systems and Robustness to Errors. The results shown in Section 6 highlight the dependency of our approach’s accuracy and automation degree on the performance of the *STA system* used, as expected. Consequently, any classification errors produced during the semantic annotation phase are inherently propagated to the generated declarative mappings and, ultimately, to the constructed KGs. To mitigate this and improve the robustness of the automated KG construction pipeline, future iterations of AutoRML can leverage the confidence scores that modern STA systems typically provide [19, 57]. Several mechanisms can be employed to manage annotation uncertainty: (i) **Confidence thresholding**: Annotations that fall below a predefined confidence threshold are rejected and replaced with fallback templates, prompting semi-automated, human-in-the-loop refinement; (ii) **Ensemble annotation**: Multiple STA systems are queried simultaneously and consensus or majority voting is utilized to filter out isolated system errors; (iii) **Probabilistic knowledge representation**: STA confidence scores are directly incorporated into the constructed KGs (e.g., using RDF-star quoted triples to annotate statements

⁷ “wd” is a prefix for: <http://www.wikidata.org/entity/>



■ **Figure 10** Knowledge inference through reasoning for $KG_{M,T}$, automatically constructed using the first row of Table 2 with AutoRML. $KG_{M,T}$'s IRIs are used to traverse the referencing KG, KG_{ref} to infer new knowledge about $KG_{M,T}$.

with their confidence scores), enabling probabilistic reasoning over the resulting KG. We consider the integration of such robustness mechanisms a primary direction for future work. Between these mechanisms, confidence thresholding can be immediately applied, since STA systems typically provide classification scores.

Scalability. As noted in Section 5, the overall scalability of our approach is heavily dominated by the performance of the underlying STA systems. While their complexity varies depending on the approach, the number of table rows and columns greatly affect performance. CEA typically requires an exhaustive, full-table scan to classify every individual cell, but performance bottlenecks can be mitigated when focusing on column-level and relation-level annotations, such as CTA and CPA. In these cases, scalability can be significantly improved by employing row sampling strategies [19]. By isolating a subset of k random rows across all columns and utilizing only this to infer column-level and relation-level annotations, the annotation time can be drastically reduced, without necessarily compromising annotation accuracy [19]. STA annotations without CEA estimation enable semi-automated declarative mappings generation using our method. However, due to the lack of CEA annotations, the generated mappings quality would be affected, with *NE-column* Subject and Object Maps falling back to template-valued Term Maps.

Target Knowledge Base Adaptability. The experiments demonstrate that when the used knowledge base aligns well with the table's content, AutoRML can generate highly accurate KGs with a significant degree of automation. However, in real-world scenarios, data sources often include

propriety concepts, such as in-house metrics, unique data object IDs, and localized terminologies, which pose difficulties for *STA systems* when targeting a general-purpose knowledge base to infer annotations.

This limitation highlights the need for *STA systems* capable of adapting to new target knowledge bases, especially domain specific ones. Recent advancements in LLMs offer promising avenues to address this issue, by leveraging fine-tuning or retrieval-augmented generation (RAG) [67] approaches. LLM-based *STA systems* [12, 23, 53, 80], which already produce promising results, can enable more flexible and accurate annotation across domains and knowledge bases.

Semantic Annotation Across Data Sources. In Section 6, we utilized the GTFS-Madrid CSV collection to evaluate AutoRML, because it includes human-generated RML mappings that can be compared with AutoRML’s results. The results demonstrate that AutoRML can successfully map named entities such as cities and regions to corresponding Wikidata entities. However, as explained, it falls short in identifying terms not present in Wikidata, such as public transportation stops and local route identifiers. Consequently, the produced RML mappings serve as an initial framework for experts to further refine.

The main limitation of our approach is that real-world data transformation tasks (in this case tabular data to KG transformations) often require the integration of information across multiple data sources. In such cases, data joins are often employed to combine data, as is also the case for the GTFS-Madrid declarative transformations. However, the *STA systems* we investigated, and subsequently AutoRML, do not support multi-table semantic annotation. Importantly, relationships can be inferred not only within a single table, but also across multiple tables, particularly when column pairs across different sources exhibit relational semantics. While this is a currently unsupported feature, extending AutoRML to identify and map these cross-table relational semantics is an important direction for future work.

This limitation highlights the need for advancements in semantic annotation methodologies that operate across data sources. Considering multiple data sources can potentially enhance the annotation accuracy of real-world data by providing more context to the *STA system*, and thus the quality of automated data transformation pipelines, such as AutoRML, making them more applicable to real use cases.

Usage of Declarative Mapping Functions. In Section 4.3 we showcased how RML-FNML enables the declarative execution of data transformations, which can be used to automate annotation tasks. However, RML-FNML demonstrates some limitations that may hinder the use of declarative transformations.

First, RML-FNML functions are executed at cell level. This often makes it challenging to perform several useful functions, such as column normalization and outlier removal that often require investigation of a full column or table. Also, this can pose a challenge for annotation functions, since cell level function execution cannot guarantee that all cells within a column will have same annotations when they are expected to (e.g., datatype identification).

Secondly, RML-FNML functions do not allow the reuse of results of other functions previously executed in another *Term Map*. For example, if there is a function that computes CTA labels and a second function that computes CEA labels, the latter does not have access to the results of the former; they must be calculated again if required. This is resource-consuming in semantic annotation methods, where different annotation tasks typically reuse the results of other annotation tasks to improve their accuracy [19, 54, 43].

8 Conclusion

In this paper, we investigated the automation of declarative KG construction through automated declarative mappings generation by leveraging semantic annotations, focusing on tabular data. We analyzed how structural and semantic annotations can contribute to automating data transformations, in a declarative way that ensures reproducibility, using RML. We implemented our approach in AutoRML, a system capable of automatically generating human-readable (YARRRML) and RML declarative mappings, and constructing KGs directly from tabular data. Through experiments, we showed that AutoRML can accurately construct KGs when semantic annotations are precise and discussed its limitations in scenarios involving domain-specific annotations. We argue that this approach can help data integration pipelines scale, facilitating automation, while emphasizing on reproducibility.

In the future, we plan to implement the automated construction of RDF-star graphs by leveraging CQA labels, enabling the identification and representation of more complex relationships. Also, we aim to integrate more *STA systems* in AutoRML's pool, including LLM-based systems, to evaluate their performance and adaptability to different domains. Finally, we intend to investigate semantic annotation across multiple tables, exploring its potential to provide richer context and more high-quality automatically generated declarative mappings.

References

- 1 Nora Abdelmageed, Jiaoyan Chen, Vincenzo Cutrona, Vasilis Efthymiou, Oktie Hassanzadeh, Madelon Hulsebos, Ernesto Jiménez-Ruiz, Juan Sequeda, and Kavitha Srinivas. Results of SemTab 2022. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 3320, pages 1–13. CEUR Workshop Proceedings, 2022. URL: <https://ceur-ws.org/Vol-3320/paper0.pdf>.
- 2 Nora Abdelmageed, Sirko Schindler, and Birgitta König-Ries. BiodivTab: semantic table annotation Benchmark Construction, Analysis, and New Additions. In Pavel Shvaiko, Jérôme Euzenat, Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, and Cássia Trojahn, editors, *Proceedings of the 17th International Workshop on Ontology Matching (OM 2022) co-located with the 21th International Semantic Web Conference (ISWC 2022), Hangzhou, China, held as a virtual conference, October 23, 2022*, volume 3324 of *CEUR Workshop Proceedings*, pages 37–48. CEUR-WS.org, 2022. URL: https://ceur-ws.org/Vol-3324/om2022_LTPaper4.pdf.
- 3 Julián Arenas-Guerrero, David Chaves-Fraga, Jhon Toledo, María S. Pérez, and Oscar Corcho. Morph-KGC: Scalable Knowledge Graph Materialization with Mapping Partitions. *Semantic Web*, 2022.
- 4 Ivan Bedini, Christopher Matheus, Peter F. Patel-Schneider, Aidan Boran, and Benjamin Nguyen. Transforming XML Schema to OWL Using Patterns. In *2011 IEEE Fifth International Conference on Semantic Computing*, pages 102–109, 2011. doi:10.1109/ICSC.2011.77.
- 5 Christian Bizer and Andy Seaborne. D2RQ—Treating non-RDF databases as virtual RDF graphs. *World Wide Web Internet and Web Information Systems*, January 2005.
- 6 Tiffany J. Callahan, Ignacio J. Tripodi, Adrienne L. Stefanski, Luca Cappelletti, Sanya B. Taneja, Jordan M. Wyrwa, Elena Casiraghi, Nicolas A. Matentzoglou, Justin Reese, Jonathan C. Silverstein, Charles Tapley Hoyt, Richard D. Boyce, Scott A. Malec, Deepak R. Unni, Marcin P. Joachimiak, Peter N. Robinson, Christopher J. Mungall, Emanuele Cavalleri, Tommaso Fontana, Giorgio Valentini, Marco Mesiti, Lucas A. Gillenwater, Brook Santangelo, Nicole A. Vasilevsky, Robert Hoehndorf, Tellen D. Bennett, Patrick B. Ryan, George Hripcsak, Michael G. Kahn, Michael Bada, William A. Baumgartner Jr, and Lawrence E. Hunter. An Open-Source Knowledge Graph Ecosystem for the Life Sciences, 2024. [arXiv:2307.05727](https://arxiv.org/abs/2307.05727).
- 7 Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL Queries over Relational Databases. *Semantic Web Journal*, 8(3):471–487, 2017. doi:10.3233/SW-160217.
- 8 David Chaves Fraga. *Knowledge Graph Construction from Heterogeneous Data Sources exploiting Declarative Mapping Rules*. PhD thesis, ETSI Informatica, June 2021.
- 9 David Chaves-Fraga and Anastasia Dimou. Declarative Description of Knowledge Graphs Construction Automation: Status & Challenges. In *Proceedings of the 3rd International Workshop on Knowledge Graph Construction (KGCW 2022)*, volume 3141, 2022. URL: <https://ceur-ws.org/Vol-3141/paper5.pdf>.
- 10 David Chaves-Fraga, Freddy Priyatna, Andrea Cimmino, Jhon Toledo, Edna Ruckhaus, and Oscar Corcho. GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport

- domain. *Journal of Web Semantics*, 65:100596, 2020. doi:10.1016/J.WEBSEM.2020.100596.
- 11 Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. ColNet: Embedding the Semantics of Web Tables for Column Type Prediction. In *AAAI Conference on Artificial Intelligence*, pages 29–36. AAAI Press, 2018. doi:10.1609/AAAI.V33I01.330129.
 - 12 Pei Chen, Soumajyoti Sarkar, Leonard Lausen, Balasubramaniam Srinivasan, Sheng Zha, Ruihong Huang, and George Karypis. HYTREL: Hypergraph-enhanced tabular data representation learning. In *NeurIPS 2023*, 2023. doi:10.48550/arXiv.2307.08623.
 - 13 Marco Cremaschi, Roberto Avogadro, Andrea Barazzetti, David Chierigato, and E Jiménez-Ruiz. MantisTable SE: an Efficient Approach for the Semantic Table Interpretation. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 2775, pages 75–85. CEUR Workshop Proceedings, 2020. URL: <https://ceur-ws.org/Vol-2775/paper8.pdf>.
 - 14 Vincenzo Cutrona, Jiaoyan Chen, Vasilis Efthymiou, Oktie Hassanzadeh, Ernesto Jiménez-Ruiz, Juan Sequeda, Kavitha Srinivas, Nora Abdelmageed, Madelon Hulsebos, Daniela Oliveira, et al. Results of semtab 2021. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 3103, pages 1–12. CEUR Workshop Proceedings, 2021. URL: <https://ceur-ws.org/Vol-3103/paper0.pdf>.
 - 15 Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. Recommendation, World Wide Web Consortium (W3C), February 2014. URL: <http://www.w3.org/TR/rdf11-concepts/>.
 - 16 Souripriya Das, Seema Sundara, and Richard Cyganiak. R2rml: Rdb to rdf mapping language, 2012.
 - 17 Ioannis Dasoulas. AutoRML. Software, version v0.1.6., swbId: swb:1:dir:fa651da6d489f8e579e3a0b7819a8216854a1f76 (visited on 2026-09-30). URL: <https://github.com/dtai-kg/AutoRML>, doi:10.4230/artifacts.28100.
 - 18 Ioannis Dasoulas, David Chaves-Fraga, Daniel Garijo, and Anastasia Dimou. Declarative RDF construction from in-memory data structures with RML. In *Proceedings of the 4th International Workshop on Knowledge Graph Construction co-located with 20th Extended Semantic Web Conference ESWC 2023*, volume 1613, page 0073, 2023.
 - 19 Ioannis Dasoulas, Duo Yang, Xuemin Duan, and Anastasia Dimou. TorchicTab: Semantic Table Annotation with Wikidata and Language Models. In *CEUR Workshop Proceedings*, volume 3557 of *CEUR Workshop Proceedings*, pages 21–37. CEUR Workshop Proceedings, CEUR-WS.org, 2023. URL: <https://ceur-ws.org/Vol-3557/paper2.pdf>.
 - 20 Luciano Frontino de Medeiros, Freddy Priyatna, and Oscar Corcho. MIRROR: Automatic R2RML Mapping Generation from Relational Databases. In Philipp Cimiano, Flavius Frasincar, Geert-Jan Houben, and Daniel Schwabe, editors, *Engineering the Web in the Big Data Era*, pages 326–343, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-19890-3_21.
 - 21 Ben De Meester, Tom Seymoens, Anastasia Dimou, and Ruben Verborgh. Implementation-independent Function Reuse. *Future Generation Computer Systems*, 110:946–959, September 2020. doi:10.1016/J.FUTURE.2019.10.006.
 - 22 Thomas Delva, Julian Arenas-Guerrero, Ana Iglesias-Molina, Oscar Corcho, David Chaves-Fraga, and Anastasia Dimou. RML-star: A declarative mapping language for RDF-star generation, 2021-10-14.
 - 23 Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. Turl: Table Understanding through Representation Learning. *SIGMOD Rec.*, 51:33–40, 2022. doi:10.1145/3542700.3542709.
 - 24 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 4171–4186. Association for Computational Linguistics, 2019. doi:10.18653/V1/N19-1423.
 - 25 Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *LDOW*, 2014.
 - 26 Nikita Dorodnykh and Aleksandr Yurin. Knowledge Graph Engineering Based on Semantic Annotation of Tables. *Computation*, 11(9), 2023. doi:10.3390/COMPUTATION11090175.
 - 27 Ali Elhalawati, Anastasia Dimou, and Jan Van den Bussche. A Declarative Formalization of R2RML Using Datalog and Its Efficient Execution. In *RuleChallenge@the 9th International Joint Conference on Rules and Reasoning (RuleML+RR 2025)*, 2025.
 - 28 Ali Elhalawati, Anastasia Dimou, and Jan Van den Bussche. A Declarative Formalization of R2RML using Datalog and its Efficient Execution. *Manuscript under review at Transactions on Graph Data and Knowledge (TGDK)*, 2026.
 - 29 Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. Introducing Wikidata to the Linked Data Web. 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 50–65, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-11964-9_4.
 - 30 Benjamin Feuer, Yurong Liu, Chinmay Hegde, and Juliana Freire. Archetype: A novel framework for open-source column type annotation using large language models. *Proceedings of the VLDB Endowment*, 17(9):2279–2292, May 2024. doi:10.14778/3665844.3665857.
 - 31 Herminio García-González, Iovka Boneva, Sławek Staworko, José Emilio Labra-Gayo, and Juan Manuel Cueva Lovelle. ShExML: improving the

- usability of heterogeneous data mapping languages for first-time users. *PeerJ Computer Science*, 6:e318, 2020. doi:10.7717/PEERJ-CS.318.
- 32 George Hannah, Terry R Payne, Andrew Mitchell, Ellen Piercy, and Boris Konev. Towards a methodology for the semi-automatic generation of scientific knowledge graphs from XML documents. In *OM 2023: The 18th International Workshop on Ontology Matching collocated with the 22nd International Semantic Web*, 2023.
 - 33 Olaf Hartig. Foundations of RDF* and SPARQL* (An Alternative Approach to Statement-Level Metadata in RDF). In *Alberto Mendelzon Workshop on Foundations of Data Management*, 2017.
 - 34 Oktie Hassanzadeh, Nora Abdelmageed, Vasilis Efthymiou, Jiaoyan Chen, Vincenzo Cutrona, Madelon Hulsebos, Ernesto Jiménez-Ruiz, Aamod Khatiwada, Ketil Korini, Benno Kruit, et al. Results of SemTab 2023. In *CEUR Workshop Proceedings*, volume 3557, pages 1–14. CEUR-WS, 2023. URL: <https://ceur-ws.org/Vol-3557/paper0.pdf>.
 - 35 Oktie Hassanzadeh, Vasilis Efthymiou, and Jiaoyan Chen. SemTab 2022: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets - "Hard Tables" R1 and R2, 2022. doi:10.5281/zenodo.7416036.
 - 36 Oktie Hassanzadeh, Vasilis Efthymiou, and Jiaoyan Chen. SemTab 2024: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets - WikidataTables2024R1 and WikidataTables2024R2, 2024. doi:10.5281/zenodo.14207232.
 - 37 Pieter Heyvaert, Ben De Meester, Anastasia Dimou, and Ruben Verborgh. Declarative Rules for Linked Data Generation at Your Fingertips! In Aldo Gangemi, Anna Lisa Gentile, Andrea Giovanni Nuzzolese, Sebastian Rudolph, Maria Maleshkova, Heiko Paulheim, Jeff Z. Pan, and Mehwish Alam, editors, *The Semantic Web: ESWC 2018 Satellite Events*, pages 213–217, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-98192-5_40.
 - 38 Pieter Heyvaert, Anastasia Dimou, Aron-Levi Herregodts, Ruben Verborgh, Dimitri Schuurman, Erik Mannens, and Rik Van de Walle. Rmleditor: A graph-based mapping editor for linked data mappings. In Harald Sack, Eva Blomqvist, Mathieu d'Aquin, Chiara Ghidini, Simone Paolo Ponzetto, and Christoph Lange, editors, *The Semantic Web. Latest Advances and New Domains*, pages 709–723, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-34129-3_43.
 - 39 Marvin Hofer, Johannes Frey, and Erhard Rahm. Towards self-configuring Knowledge Graph Construction Pipelines using LLMs - A Case Study with RML. In *Fifth International Workshop on Knowledge Graph Construction@ ESWC2024*, 2024.
 - 40 Marvin Hofer, Daniel Obraczka, Alieh Saeedi, Hanna Köpcke, and Erhard Rahm. Construction of Knowledge Graphs: State and Challenges, 2023. doi:10.48550/arXiv.2302.11509.
 - 41 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge Graphs. *ACM Computing Surveys*, 54(4):1–37, July 2021. arXiv: 2003.02320 version: 6. doi:10.1145/3447772.
 - 42 Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 1500–1508, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3292500.3330993.
 - 43 Viet-Phi Huynh, Yoan Chabot, Thomas Labbé, Jixiong Liu, and Raphaël Troncy. From heuristics to language models: A journey through the universe of semantic table interpretation with DAGOBAB. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 3320, pages 45–58. CEUR Workshop Proceedings, 2022. URL: <https://ceur-ws.org/Vol-3320/paper6.pdf>.
 - 44 Ana Iglesias-Molina, David Chaves-Fraga, Ioannis Dasoulas, and Anastasia Dimou. Human-Friendly RDF Graph Construction: Which One Do You Chose? In Irene Garrigós, Juan Manuel Murillo Rodríguez, and Manuel Wimmer, editors, *Web Engineering*, pages 262–277, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-34444-2_19.
 - 45 Ana Iglesias-Molina, Luis Pozo-Gilo, Daniel Dona, Edna Ruckhaus, David Chaves-Fraga, and Oscar Corcho. Mapeathor: Simplifying the specification of declarative rules for knowledge graph construction. In *Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference (ISWC 2020)*, 2020.
 - 46 Ana Iglesias-Molina, Dylan Van Assche, Julián Arenas-Guerrero, Ben De Meester, Christophe Debruyne, Samaneh Jozashoori, Pano Maria, Franck Michel, David Chaves-Fraga, and Anastasia Dimou. The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF. In *International Semantic Web Conference*, pages 152–175. Springer, 2023. doi:10.1007/978-3-031-47243-5_9.
 - 47 Krzysztof Janowicz, Álvaro Sicilia, German Nemirovski, and Andreas Nolle. Map-On: A web-based editor for visual ontology mapping. *Semant. Web*, 8(6):969–980, January 2017. doi:10.3233/SW-160246.
 - 48 Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, Kavitha Srinivas, and Vincenzo Cutrona. Results of semtab 2020. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 2775, pages 1–8. CEUR Workshop Proceedings, 2020. URL: <https://ceur-ws.org/Vol-2775/paper0.pdf>.
 - 49 Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph

- Pinkel, Martin Skjæveland, Evgenij Thorstensen, and Jose Mora. BootOX: Practical Mapping of RDBs to OWL 2. In *The Semantic Web*, pages 113–132, October 2015. doi:10.1007/978-3-319-25010-6_7.
- 50 Craig A. Knoblock, Pedro Szekely, José Luis Ambite, Aman Goel, Shubham Gupta, Kristina Lerman, Maria Muslea, Mohsen Taheriyani, and Parag Mallick. Semi-automatically Mapping Structured Sources into the Semantic Web. In Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti, editors, *The Semantic Web: Research and Applications*, pages 375–390, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-30284-8_32.
 - 51 Ketii Korini and Christian Bizer. Column Type Annotation using ChatGPT, 2023. doi:10.48550/arXiv.2306.00745.
 - 52 Ketii Korini and Christian Bizer. Column Property Annotation using Large Language Models. In *The Semantic Web - Special Track: Large Language Models for Knowledge Engineering —Program*, 2024.
 - 53 Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. Tablegpt: Table fine-tuned gpt for diverse table tasks. *Proc. ACM Manag. Data*, 2(3):1–28, May 2024.
 - 54 Xinhe Li, Shuxin Wang, Wei Zhou, Gongrui Zhang, Chenghuan Jiang, Tianyu Hong, and Peng Wang. Kgcode-tab results for semtab 2022. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 3320, pages 37–44. CEUR Workshop Proceedings, 2022. URL: <https://ceur-ws.org/Vol-3320/paper5.pdf>.
 - 55 Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. LinkingPark: An automatic semantic table interpretation system. *Journal of Web Semantics*, 74:100733, 2020.
 - 56 Jixiong Liu, Yoan Chabot, Raphaël Troncy, Viet-Phi Huynh, Thomas Labbé, and Pierre Monnin. From tabular data to knowledge graphs: A survey of semantic table interpretation tasks and methods. *Journal of Web Semantics*, 76(C):100761, April 2022. doi:10.1016/J.WEBSEM.2022.100761.
 - 57 Jixiong Liu and Raphaël Troncy. DAGOBAB: an end-to-end context-free tabular data semantic annotation system. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 2553, pages 41–48. CEUR Workshop Proceedings, 2019. URL: <https://ceur-ws.org/Vol-2553/paper6.pdf>.
 - 58 Varish Mulwad, Tim Finin, and Anupam Joshi. Semantic message passing for generating linked data from tables. In *The Semantic Web – ISWC 2013*, volume 8218, pages 363–378. Springer International Publishing, 2013. doi:10.1007/978-3-642-41335-3_23.
 - 59 Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. Can Foundation Models Wrangle Your Data? *Proc. VLDB Endow.*, 16(4):738–746, December 2022. doi:10.14778/3574245.3574258.
 - 60 Phuc Nguyen, Natthawut Kertkeidkachorn, Ryutaro Ichise, and Hideaki Takeda. Mtab: Matching tabular data to knowledge graph using probability models. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 2553, pages 7–14. CEUR Workshop Proceedings, 2019. URL: <https://ceur-ws.org/Vol-2553/paper2.pdf>.
 - 61 Phuc Nguyen, Ikuya Yamada, Natthawut Kertkeidkachorn, Ryutaro Ichise, and Hideaki Takeda. SemTab 2021: Tabular Data Annotation with MTab Tool. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 3103, pages 92–101. CEUR Workshop Proceedings, 2021. URL: <https://ceur-ws.org/Vol-3103/paper8.pdf>.
 - 62 Vinh Nguyen, Olivier Bodenreider, and Amit 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 ’14*, pages 759–770, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2566486.2567973.
 - 63 OpenAI. ChatGPT, n.d. Accessed: 2024-08-06. URL: <https://openai.com/blog/chatgpt>.
 - 64 C. Pinkel, C. Binnig, E. Jimenez-Ruiz, E. Kharlamov, W. May, A. Nikolov, A. S. Bastinos, M. G. Skjaeveland, A. Solimando, M. Taheriyani, C. Heupel, and I. Horrocks. RODI: Benchmarking Relational-to-Ontology Mapping Generation Quality. *Semantic Web*, 9(1):25–52, January 2018. doi:10.3233/SW-170268.
 - 65 Christoph Pinkel, Carsten Binnig, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Andriy Nikolov, Andreas Schwarte, Christian Heupel, and Tim Kraska. IncMap: A Journey towards Ontology-based Data Integration. In *Datenbanksysteme für Business, Technologie und Web*, 2017.
 - 66 Christoph Pinkel, Carsten Binnig, Evgeny Kharlamov, and Peter Haase. IncMap: Pay as you go Matching of Relational Schemata to OWL Ontologies. In *International Semantic Web Conference*, volume 1111, January 2013. URL: https://ceur-ws.org/Vol-1111/cm2013_Tpaper4.pdf.
 - 67 Tyler Thomas Procko and Omar Ochoa. Graph Retrieval-Augmented Generation for Large Language Models: A Survey. In *2024 Conference on AI, Science, Engineering, and Technology (AISET)*, pages 166–169, 2024.
 - 68 Sergio Rodríguez Méndez, Armin Haller, Pouya Omran, Jesse Wright, and Kerry Taylor. J2RM: an Ontology-based JSON-to-RDF Mapping Tool. In *Proceedings of the 19th International Semantic Web Conference on Demos and Industry Tracks (ISWC)*, November 2020.
 - 69 Juan F. Sequeda, Syed Hamid Tirmizi, Oscar Corcho, and Daniel P. Miranker. Survey of directly mapping SQL databases to the Semantic Web. *The Knowledge Engineering Review*, 26(4):445–486, 2011. doi:10.1017/S0269888911000208.
 - 70 Álvaro Sicilia and German Nemirovski. AutoMap4OBDA: Automated Generation of R2RML Mappings for OBDA. In *International Conference Knowledge Engineering and Knowledge Management*, 2016. doi:10.1007/978-3-319-49004-5_37.
 - 71 Bram Steenwinckel, Filip De Turck, and Femke Ongenae. MAGIC: Mining an Augmented Graph

- using INK, starting from a CSV. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 3103, pages 68–78. CEUR Workshop Proceedings, 2021. URL: <https://ceur-ws.org/Vol-3103/paper6.pdf>.
- 72 Bram Steenwinckel, Gilles Vandewiele, Filip De Turck, and Femke Ongenae. Csv2kg: Transforming tabular data into semantic knowledge. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)*, volume 2553, pages 33–40. CEUR Workshop Proceedings, 2019. URL: <https://ceur-ws.org/Vol-2553/paper5.pdf>.
 - 73 Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data, SIGMOD '22*, pages 1493–1503. Association for Computing Machinery, 2022. doi:10.1145/3514221.3517906.
 - 74 Dylan Van Assche, David Chaves-Fraga, and Anastasia Dimou. KROWN: A Benchmark for RDF Graph Materialisation, 2024.
 - 75 Dylan Van Assche, Thomas Delva, Gerald Haesendonck, Pieter Heyvaert, Ben De Meester, and Anastasia Dimou. Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review. *Journal of Web Semantics*, 75:100753, 2023.
 - 76 Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proc. VLDB Endow.*, 4(9):528–538, June 2011. doi:10.14778/2002938.2002939.
 - 77 Denny Vrandečić and Markus Krötzsch. Wikidata: A Free Collaborative Knowledgebase. *Commun. ACM*, 57(10):78–85, September 2014. doi:10.1145/2629489.
 - 78 Yubo Wang, Hao Xin, and Lei Chen. KGLink: A column type annotation method that combines knowledge graph and pre-trained language model, 2024. doi:10.48550/arXiv.2406.00318.
 - 79 Guohui Xiao, Linfang Ding, Benjamin Cogrel, and Diego Calvanese. Virtual Knowledge Graphs: An Overview of Systems and Use Cases. *Data Intelligence*, 1(3):201–223, June 2019. doi:10.1162/DINT_A_00011.
 - 80 Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data, 2020. arXiv:2005.08314.
 - 81 Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. Sato: Contextual Semantic Type Detection in Tables. *Proc. VLDB Endow.*, 13(12):1835–1848, July 2020. URL: <http://www.vldb.org/pvldb/vol13/p1835-zhang.pdf>.
 - 82 Ziqi Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web*, 8(6):921–957, 2017. doi:10.3233/SW-160242.