




Article

A Semi-Automatic Semantic Consistency-Checking Method for Learning Ontology from Relational Database

Chuangtao Ma [†], Bálint Molnár ^{*,†} and András Benczúr [†]

Department of Information Systems, Faculty of Informatics, Eötvös Loránd University (ELTE), 1117 Budapest, Hungary; machuangtao@caesar.elte.hu (C.M.); abenczur@inf.elte.hu (A.B.)

* Correspondence: molnarba@inf.elte.hu; Tel.: +36-(1)-372-2500/8042

† Current address: Pázmány Péter sétány 1/C, 1117 Budapest, Hungary.

Abstract: To tackle the issues of semantic collision and inconsistencies between ontologies and the original data model while learning ontology from relational database (RDB), a semi-automatic semantic consistency checking method based on graph intermediate representation and model checking is presented. Initially, the W-Graph, as an intermediate model between databases and ontologies, was utilized to formalize the semantic correspondences between databases and ontologies, which were then transformed into the Kripke structure and eventually encoded with the SMV program. Meanwhile, description logics (DLs) were employed to formalize the semantic specifications of the learned ontologies, since the OWL DL showed good semantic compatibility and the DLs presented an excellent expressivity. Thereafter, the specifications were converted into a computer tree logic (CTL) formula to improve machine readability. Furthermore, the task of checking semantic consistency could be converted into a global model checking problem that could be solved automatically by the symbolic model checker. Moreover, an example is given to demonstrate the specific process of formalizing and checking the semantic consistency between learned ontologies and RDB, and a verification experiment was conducted to verify the feasibility of the presented method. The results showed that the presented method could correctly check and identify the different kinds of inconsistencies between learned ontologies and its original data model.

Keywords: consistency checking; ontology learning; model checking; graph intermediate representation; relational database



Citation: Ma, C.; Bálint, M.; Benczúr, A. A Semi-Automatic Semantic Consistency-Checking Method for Learning Ontology from Relational Database. *Information* **2021**, *12*, 188. <https://doi.org/10.3390/info12050188>

Academic Editors: Giuseppe Psaila and Paolo Fosci

Received: 24 March 2021

Accepted: 18 April 2021

Published: 26 April 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Knowledge-based integration was regarded as one of the efficient integration methods due to the excellent semantic interoperability of knowledge bases (\mathcal{KB}). Typically, the mainstream methods for constructing knowledge bases are based on manual transformation and mapping, hence it is a costly and tedious task to construct and maintain knowledge bases by using the traditional manual mapping and transformation [1]. Ontology, one of the representatives and formalized knowledge bases, provides a rich semantic reference for the schema mapping and data integration due to its semantic interoperability and rigorous mathematical foundation [2]. Similarly, the traditional methods for constructing ontology from RDB are mainly based on predefined rules, in which a lot of effort and domain expertise are required.

Ontology learning (OL) is a kind of knowledge representation learning method, aiming to (semi-)automatically construct ontologies from various data, in which the entities and relationships are usually identified and extracted based on semantic computation and knowledge inference. The prevailing techniques of learning ontology could be classified into four categories: association rule mining (ARM), formal concept analysis (FCA), inductive logic programming (ILP), neural networks (NN), and machine learning [3]. To some extent, ontology learning not only improves the efficiency of ontology construction but also eliminates the biases and limitations of human knowledge [4].

There is no doubt that ontology learning could free humans from the tedious mapping and transformation and minimize the negative influence of human knowledge biases while manually constructing ontologies. However, it is a common phenomenon that some inconsistencies and semantic conflicts will inevitably occur during (semi-)automatic ontology learning [5]. Due to the various naming conventions of entities and attributes, and the different semantic contexts that exist in different databases, it is unavoidable that semantic collisions and inconsistencies will occur between learned ontologies and their original databases. Consequently, these inconsistencies will weaken the capabilities of the semantic interoperability of learned ontologies. Therefore, the issues of inconsistencies and redundancies are becoming the bottleneck in the scenario of (semi-)automatic ontology learning from a relational database.

In general, ontology consistency could be manually checked in the ontology evaluation, in which several criteria are defined to evaluate the quality of ontology, e.g., consistency, completeness, conciseness, etc. [6]. Semantic consistency is a fundamental criterion for evaluating the quality of ontology during ontology construction and merging. The semantic consistency checking not only checks for consistency at the syntactic level, but also at the semantic level. However, the ontology evaluation is a time-consuming and laborious work [7]. There are even more inconsistencies that need to be checked and evaluated when it comes to learning ontology from RDB, since the current ontology learning algorithm is immature. In particular, the issues of inconsistencies and redundancies will get much worse when learning ontology from multiple data sources. Hence, how to efficiently identify the inconsistencies between learned ontologies and their original databases is one of the critical tasks in the (semi-)automatic ontology learning from RDB.

To address the above issues, this article presents a semi-automatic semantic consistency checking method based on the graph intermediate representation and model checking. We formalized the semantic correspondences between databases and ontologies based on labeled transition systems and Kripke structure by introducing the W-Graph, which is a graph-based intermediate representation model. We initially encoded the specifications of learned ontologies with DLs to leverage the excellent expressivity of description logics (DLs) [8]. We translated the aforementioned semantic specifications of learned ontologies from DLs to the CTL formula to improve machine readability. The remainder of this article is organized as follows. The related work on the topic of semantic consistency checking of ontologies is summarized in Section 2. The problem of the inconsistencies during learning ontology from relational database are described and the preliminary definitions are given in Section 3. The specific processes of the presented semantic consistency checking method are given, and the corresponding verification experiment is conducted to verify the feasibility and effectiveness of the presented method in Section 4. The conclusion is summarized, and the future work is given in Section 5.

2. Related Work

Ontology consistency checking is a critical task in ontology construction, ontology alignment, and ontology evolution, by which the inconsistencies could be identified and eliminated. The prevailing methods for checking ontology consistency could be classified into two categories: logical reasoning and graph intermediate representation.

2.1. Consistency Checking Based on Logical Reasoning

Logical reasoning is a classic method to check the consistency of knowledge bases, in which the hypotheses, arguments, and consequences could be inferred and justified. To ensure the consistency of ontologies, Baclawski et al. [9] designed a tool (ConsVISor) for checking the consistency of ontology. ConsVISor could check whether a given ontology is consistent by verifying the axioms based on inference of logic programming engine. However, it simply identifies the inconsistency at the syntactic level, which may lead to some errors when it meets some synonyms or abbreviations. As a result that the OWL-DL reasoning mechanism provides automatic detection of inconsistencies by formal

description logic, Neumann et al. [10] designed a reference architecture and prototype of the ontological XML database system (OXDBS). In this system, the consistency validation module was designed, in which the consistency is checked based on the formal validation of the OWL-DL reasoner. Apart from using the First-Order Logic (FOL) and DLs, the Semantic Web Rule Language (SWRL) was also utilized to formalize semantics. To resolve semantic conflicts in the anti-fraud rule-based expert systems, del Mar et al. [11] proposed a semantic conflict resolution method to check the consistency of rules in rule-based expert systems. More specifically, different kinds of rules were formulated in SWRL, and then the inconsistent, overlapping, and duplicate rules were detected in the rule-based expert system by introducing the ontology reasoning mechanism.

In addition to checking ontology consistency based on logic reasoning, there are some works focused on the issues of inconsistency prediction and elimination. To eliminate the inconsistencies at an earlier time, Bayoudhi [12] predicated the logical inconsistencies during updating OWL2-DL based on predefined rules, where the potential inconsistencies could be detected and resolved at an earlier time. To tackle the issue of inconsistent ontologies, Rosati et al. [13] presented a reasoner-based Quonto Inconsistent Data handler (QuID). In this method, the semantic inconsistencies of ontologies in ABox could be repaired automatically based on the data manipulation and query rewriting.

It is noteworthy that some unexpected results, e.g., unsatisfied class, erroneous correspondences, etc., will occur while checking the ontology consistency based on reasoning. In particular, it is ineffective for the reasoner to check the inconsistencies if there exists no standard logic inconsistency [12]. In this case, the domain experts are required to manually determine whether these unexpected results are acceptable [14]. To minimize the human intervention in checking consistency during automatic ontology mapping and merging system, Fahad et al. [15] proposed a method to identify semantic inconsistencies from initial mappings by leveraging the subsumption analysis to analyze the elements of ontologies, i.e., concepts and properties. Aimed at semi-automatically constructing ontologies from the Web corpus, Bai et al. [16] proposed a domain ontology learning approach, in which a two-stage clustering approach and SOM neural network are utilized to extract and build domain ontology from Chinese-document corpus. Accordingly, a consistency checking method based on racer reasoner is presented to check the consistency of learned ontologies. Additionally, to address the inconsistency between ABox and TBox during ontology reasoning, Paulheim et al. [17] proposed a ABox consistency checking method based on machine learning. In this method, an approximate reasoner was introduced to check the ABox consistency, and it is eventually considered as a binary classification problem, by which the ABox is translated to feature vectors for training the decision trees.

2.2. Consistency Checking Based on Graph Intermediate Representation

Considering that both ontology and RDB could be formalized as a directed graph, the correspondences between them could be specified and formalized based on graph mapping. In consequence, the consistency checking based on graph intermediate representation has aroused scholar's attention. To address the incapability of logical reasoning to provide an explanation for the user to resolve the identified inconsistencies, Lam et al. [18] presented a graph-based ontology inconsistency checking method. More specifically, the ontology was formalized as a directed graph, in which the consistency between concepts was checked by analyzing the paths of a graph. In order to identify the inconsistencies in building ontology, Yang et al. [19] proposed a semantic consistency method based on a graph-based intermediate model. In this method, the task of semantic consistency checking was referred to as a problem of a semantic equivalent query over the isomorphism graph, which was eventually solved by the model checker. However, an intermediate graph was constructed based on the RDB schema and instance, which ignores the subtle difference between the intermediate graph model and ontology. Strictly speaking, we could not conclude that the constructed ontologies are consistent with the original database when the query result of the sub-graph is consistent with the intermediate graph model.

In addition to checking the ontology consistency during the construction of ontologies, some works address the consistency checking based on graph-intermediate representation during the ontology evolution and mapping. To tackle the inconsistency issue in updating the ontologies, Mahfoudh et al. [20] proposed an a priori approach to resolve the ontology inconsistencies based on the Simple PushOut (SPO) graph transformation. In this approach, the ontology changes and inconsistencies were formalized as a typed attributed graph, which provided a mechanism to avoid the inconsistent ontologies by controlling graph transformations and rewriting rules with SPO graph transformations. Similarly, to avoid the inconsistency in transforming RDB to Resource Description Framework (RDF) graph, Jun et al. [21] focused on the semantics-preserving mapping method based on rules. In this work, several rules were defined based on predicate logic to avoid semantic loss during mapping of multi-column keys constraints into RDF constraints. However, the definition of rules requires domain expertise, which is a tedious task as well.

2.3. Brief Summary

To recap, the existing works on the topic of ontology consistency checking could be classified into two categories: logical reasoning [10,11,13,15–17], and graph-based intermediate representation [18–21]. Even if the consistency could be checked directly by using the ontology reasoning mechanism, it is incapable of explaining to the user to resolve the identified inconsistencies [18]. In particular, the performance of the consistency checking heavily depends on the consistency and integrity of ontologies, therefore, the reasoning of inconsistent ontologies may lead to some erroneous axioms and conclusions [9]. It is worth mentioning that the ontology consistency checking based on reasoning could only check the consistency between ABox and TBox of ontology [17], and it is incapable of checking the consistency between ontologies and their original knowledge in the process of building ontology.

Currently, the main knowledge still resides in relational databases, thus, it is a trivial task to check the semantic consistency during learning ontologies from RDB. Nevertheless, the existing methods of consistency checking mainly focus on checking the consistency of learning ontology from the Web and document corpus [16], while there is less work focus on checking the consistency of learning ontology from RDB. Therefore, how to efficiently check the semantic consistency between ontology and its original RDB in the early phase of ontology learning from the database remains an open question.

To address this question, we present a semi-automatic semantic consistency checking method based on the graph intermediate representation and model checking. The current work is similar to the [19] proposed method because both of us employ W-Graph as an intermediate model to formalize semantic correspondence and check the ontology consistency. As we previously mentioned, the equivalent sub-graph query over the intermediate graph may lead to some incorrect results when RDB schema significantly differentiates ontologies. To overcome this limitation, we directly refer to the task of consistency checking as a global model checking rather than the equivalent sub-graph query over an intermediate graph model. More specifically, we encode the specifications of learned ontologies with DLs, and then we translate these specifications to the CTL formula. Hereafter, the ontology consistency can be checked by the model checker by verifying whether the formalized specifications are consistent with the graph-based intermediate model—Kripke structure.

3. Problem Statement and Preliminaries

In this section, we describe the problem of semantic consistency between ontologies and databases when learning ontology from RDB, and we briefly introduce the formal definitions of two intermediate graph models and logic languages.

3.1. Problem Statement

The semantics of an ontology \mathcal{O} is formally defined by using the axioms of DLs. As we all know, DLs are decidable fragments of FOL [22], thus the semantics of ontology could

be directly defined by FOL formulae, denoted as $\Sigma_{\mathcal{O}}$. For each triple of an ontology, $t \in \mathcal{O}$, the semantics over the predicate triple are defined as φ_t . Accordingly, the semantics of ontology \mathcal{O} that encode with FOL formula are denoted as $\{\varphi_t \mid t \in \mathcal{O}\}$.

Traditionally, the basic elements of relational database, i.e., database schema \mathbf{R} , constraints Σ over \mathbf{R} , instance I of \mathbf{R} , could be mapped into RDF graph and OWL by using direct mapping \mathcal{M} . As we mentioned earlier, the \mathcal{M} is defined based on the set of datalog predicates and rules, in which a lot of experience and effort from domain experts are required. We could say a direct mapping \mathcal{M} is semantic preserving mapping [23], namely, there is no semantic inconsistency between RDB and OWL, if for every database schema \mathbf{R} , set of constraints Σ (PKs and FKs), instance I of \mathbf{R} , and semantics of the ontology $\Sigma_{\mathcal{O}}$, which satisfies: $I \models \Sigma$ iff $\mathcal{M}(\mathbf{R}, \Sigma, I)$.

It is worth mentioning that semantic collision and inconsistency will occur with high probability while (semi-)automatically construct ontology from multiple databases. The predominant semantic inconsistencies and redundancies are relationship redundancy, concept or property redundancy, and fact inconsistency. The reasons behind these inconsistencies and redundancies are various naming conventions among different databases. For instance, the machine could recognize that Prof is the abbreviation of Professor, Title is synonymous with Position in academic ranks, while it is quite difficult to identify that the Assistant Professor is synonymous with Lecturer. Therefore, it is a crucial task to detect and resolve these inconsistencies while (semi-)automatically learning ontology from a relational database at an earlier stage.

3.2. Preliminaries

This subsection introduces the formal definitions of two kinds of directed graph models, and the fundamental syntax and formula of logic language.

3.2.1. W-Graph

W-Graph is a graph-based formal language, which provides an intermediate and semantic equivalent graph model between RDB and ontologies [24]. Essentially, it is a directed labeled graph [25], and could be formalized as a triple $W_G = \langle N, E, \ell \rangle$, where:

- $N = \{N_a, N_c\}$ is a finite set of nodes, N_a is a finite set of atomic nodes that is depicted as ellipses, and N_c is a finite set of composite nodes that is depicted as rectangles.
- $E \subseteq N_c \times (\ell \times \mathcal{L}) \times N$ is a set of labeled edges in the form of triple, ℓ is a function $\ell : N \rightarrow \mathcal{C} \times (\mathcal{L} \cup \{\perp\})$, $\mathcal{C} = \{solid, dashed\}$ and \mathcal{L} is a set of labels.

Similar to RDB and RDF, there are two kinds of models of W-Graph: W-Schema and W-Instance. Conventionally, W-Schema represents the patterns of the knowledge, while W-Instance represents the concrete contents of the knowledge. Formally, W_I denotes the instance of W_G , for each edge e of the instance W_I meets $\ell_{\mathcal{G}}(e) = solid$, and for each node n of the instance W_I meets $\ell_{\mathcal{G}}(n) \neq \perp$, where \perp represents the dummy nodes.

Based on the definition of W-Graph, the bi-simulation semantics could be defined. Accordingly, the semantics-preserving mapping between RDB and ontologies could be formalized by W-Graph.

3.2.2. Kripke Structure

Kripke structures are finite directed graphs that represent the transition of states [26]. In the state-transition graph, vertices are labeled with sets of atomic propositions (AP). Formally, Kripke structure over a set of APs could be represented as a triple $\mathcal{K} = \langle S, I, R, L \rangle$, where:

- S is a finite set of states, $I \subseteq S$ is a set of initial states, $R \subseteq S \times S$ is a set of transitions.
- L is a labeling function: $L : S \rightarrow 2^{AP}$, which associates each state with a set of AP.
- A sequence of states S and their transitions R is viewed as a path $\pi = s_0, s_1, s_2, \dots, s_n$ in Kripke structure.
- Given an infinite path π , $L(\pi) = L(s_0), L(s_1), L(s_2), \dots, L(s_n)$ is an infinite sequence set of atomic propositions.

3.2.3. Computation Tree Logic

Computation tree logic (CTL) is a logic language for describing the properties of the computation tree, in which many executions can be reasoned by formulas at once. Formally, given a transition system $\mathcal{T} = \langle S, \rightarrow, s_0 \rangle$, the computation tree of \mathcal{T} is the acyclic unfolding graph. Let AP be a set of atomic propositions, accordingly, the set of CTL formulas over AP are defined as follows:

- In essence, every AP is a CTL formula, formally, if φ_1, φ_2 are CTL formulas and a is an element of AP, the CTL formulas are denoted as a binary function:

$$\text{CTL formula} = \begin{cases} a, a \in AP \\ \varphi_1, \varphi_2, \text{else} \end{cases}$$

- When φ_1, φ_2 are CTL formulas, the following syntaxes are defined:

$$\neg\varphi_1, \varphi_1 \vee \varphi_2, \text{EX}\varphi_1, \text{EF}\varphi_1, \text{EG}\varphi_1, \varphi_1 \text{EU}\varphi_2, \dots$$

Given a Kripke structure $\mathcal{K} = \langle S, I, R, L \rangle$, the semantics of CTL formula are denoted as a set of states $\llbracket \varphi \rrbracket_{\mathcal{K}}$. There are two kinds of CTL formulas, state formula and path formula [27], which are employed to verify whether the given states and paths satisfy the predefined specifications, respectively. Given an initial state s_0 of Kripke structure \mathcal{K} , when it meets $s_0 \in \llbracket \varphi \rrbracket_{\mathcal{K}}$, the corresponding Kripke structure \mathcal{K} satisfies the CTL formula φ , denoted as $\mathcal{K} \models \varphi$.

3.2.4. Description Logics

Description logics (DLs) is a formal logic language mainly used to specify knowledge base \mathcal{KB} , namely, DLs provide a mechanism to formalize and reason the knowledge from various data [22]. In DLs, knowledge is usually represented as knowledge base $\mathcal{KB} = \langle \mathcal{A}, \mathcal{T} \rangle$, while: \mathcal{A} is a set of assertions of individuals (ABox), and \mathcal{T} is a set of terminologies (TBox). The basic elements of DLs could be categorized into three types: *individuals*, *concepts*, and *roles*. Given an atomic concept A, common concepts C, D, and a role R, the following syntax, assertions and axioms are defined:

- Syntax definitions: C, D \rightarrow A is an atomic concept, \top denotes top concept, \perp denotes bottom concept, $\neg C$ denotes negative concept, $C \sqcup D$ denotes union of two concepts, $C \sqcap D$ denotes intersection of two concepts, $\exists R.C$ denotes existential restriction, and $\forall R.C$ denotes universal restriction.
- Assertions and axioms: C(a) and R(a,b) denote concept assertions and role assertions for individuals a,b, concepts C, and roles R in ABox; $C \sqsubseteq D$ denotes the axioms between concepts C and D in TBox.

The DL reasoner provides a concept consistency checking mechanism to verify if ABox is consistent with respect to TBox [28]. Given a terminology \mathcal{T} and concept C, DLs could verify if there exists a non-empty interpretation C^I of C that satisfies each inclusion dependency in \mathcal{T} , which could be denoted as $\mathcal{T} \models C^I$ [29].

4. Consistency Checking Based on Model Checking

This section introduces the specific process of a semantic consistency checking method based on graph intermediate representation and model checking. To begin with, a general framework of this method is presented, and the modeling process of the semantics between ontology and RDB is described. In addition, an example is given to verify the feasibility and effectiveness of the presented method.

4.1. General Description of Proposed Method

The reasoning of ontologies provides a mechanism for checking consistency, by which only the consistency between ABox and TBox can be checked. In this work, we address the consistency checking between ontologies and their original databases during learn-

ing ontology from RDB, rather than the consistency checking between ABox and TBox of ontologies.

Consistency between ontologies and databases is a state that satisfies the given consistency criteria. Usually, these consistency criteria are defined based on rules, hence the consistency is checked by verifying whether the constructed ontologies satisfy a given consistency criterion. The basic consistency criteria of ontologies are completeness and non-redundancy [6], however, it is a contradiction between these two criteria. Therefore, a trade-off between ontology completeness and non-redundancy should be made.

Semantic consistency checking is a process of identifying semantic collisions, i.e., detect inconsistencies and duplicates, which is an essential step in constructing and maintaining knowledge bases. Model checking is a computer-assisted method for analyzing and verifying the dynamical systems by modeling them as state-transition systems [30]. Specifically, model checking could be utilized to verify whether a given model M satisfies predefined specification φ , formally, it could be denoted as $M \models \varphi$.

To ensure the completeness and non-redundancy of ontologies when learning ontology from relational database, a semantic consistency method based on model checking is presented. This method could be employed to check the ontology consistency at an earlier stage of ontology learning from RDB. To make it easier to understand, the main workflow of the presented semantic consistency checking method is depicted in Figure 1.

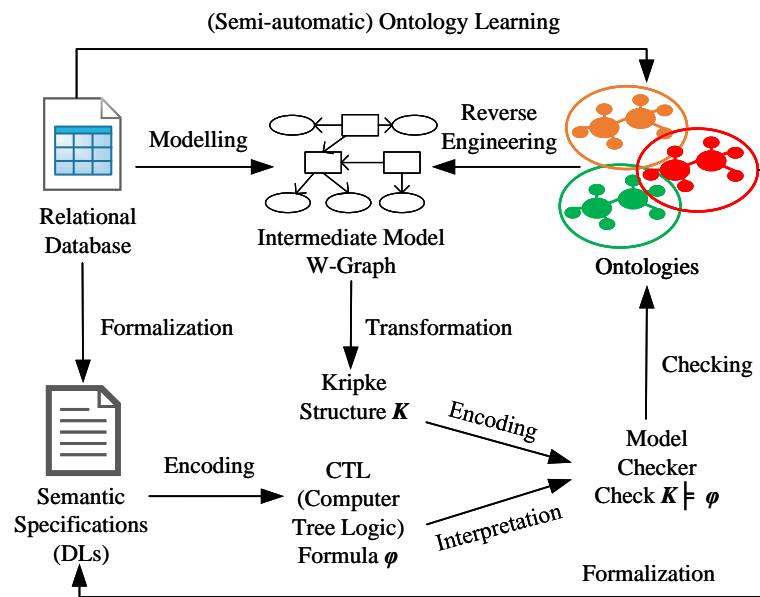


Figure 1. Semantic consistency checking for learning ontology from relational database.

As we can see in Figure 1, an intermediate model, W-Graph, is utilized to formalize the semantic correspondences between ontologies and its original relational database, which are transformed to the Kripke structure. Considering the semantic compatibility and the available inverse roles of \mathcal{ALCC} DL, \mathcal{ALCC} DL are employed to formalize the semantic specifications, which are eventually encoded by the CTL formula. Thereafter, the Kripke structure is encoded by the SMV program, which along with the CTL formula is interpreted by the symbolic model checker. Thereby, the semantic consistency between learned ontologies and their original database could be automatically checked by a model checker.

4.2. Introduction to Mini University Data Model

Considering that the current algorithm for learning ontology from RDB is still in its exploratory stage, the Mini University ontology [31], and its corresponding database, is selected as an example to describe the specific steps of semantic consistency checking

based on model checking. Despite that the Mini University is not a complex example, it is quite representative of a relational database because it almost contains the common database elements. This is one of the reasons why the Mini University data model and its corresponding ontology are selected to demonstrate the specific steps of this method.

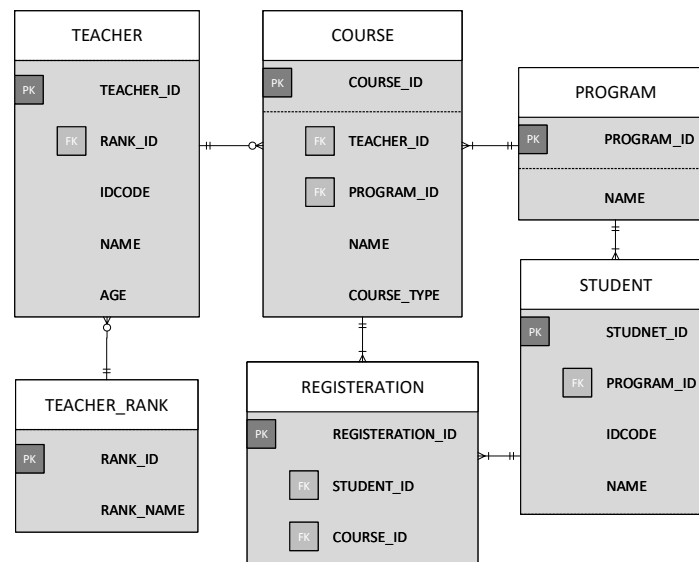


Figure 2. Database schema of Mini University.

Figure 2 shows the database schema of Mini University, we can see that there are five regular entities, one associative entity (REGISTRATION). Meanwhile, there are different kinds of associations, i.e., one-to-one, one-to-many, and many-to-many. In particular, it also contains database constraints, i.e., primary key, foreign key, which represent the unique identifier and associations, respectively.

4.3. Formalization of Relational Data Model

As we mentioned in Section 4.1, model checking could be utilized to check whether the model satisfies the given specifications. To leverage the model checking to check the semantic consistency of learning ontology from RDB, an intermediate model, W-Graph, is utilized to model the semantics of Mini University data model. Furthermore, DLs are employed to formalize the specification of semantic correspondence between initial databases and target ontologies. Inspired by the model checking based on DLs [29], we converted the problem of semantic consistency into a global model checking, which could be automatically checked by the model checker.

4.3.1. Constructing Kripke Structure from Database

The formal definitions of the Kripke structure and W-Graph are given in Section 3.2. Based on these definitions, we could construct the Kripke structure from the relational data model by introducing the graph-based intermediate model. Accordingly, the semantics of the data model could be modeled by W-Graph before constructing the Kripke structure from W-Graph. The specific steps of constructing the Kripke structure from a relational database could be split into two phases:

Modeling Semantics of Database by W-Graph. A relational database (RDB) is a table-based data model, while W-Graph is a graph-based model. There are several differences between these models, e.g., storage format, structure, and constraints, etc. This work aims to check the semantic consistency of learning ontology from a relational database, we only transform the semantics of the database into a W-Graph model. We map the entities and relationships in RDB into nodes and edges of W-Graph, respectively, thereby, the W-Schema

and W-Instance are built based on the definitions of W-Graph. Accordingly, the schema and semantics of the Mini University database could be transformed into W-Instance, which is shown in Figure 3.

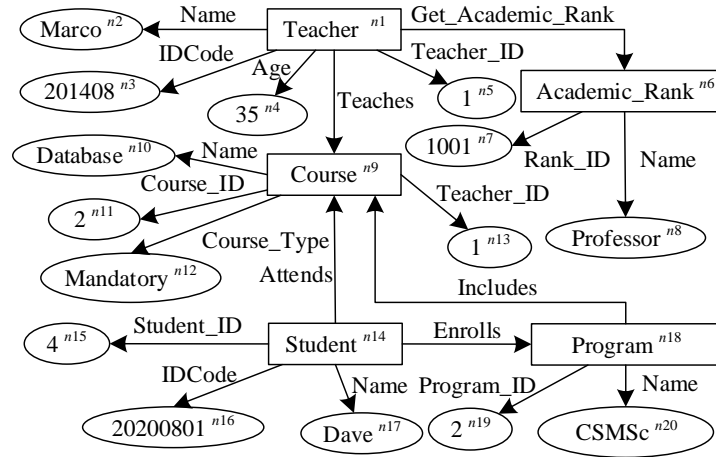


Figure 3. W-Instance of Mini University data model.

As shown in Figure 3, the W-Instance of Mini University database model could be formally depicted as the following triple: $W_{Mini\ University} = \langle N, E, \ell \rangle$, where:

- $N = \langle \{n_1, n_6, n_9, n_{14}, n_{18}\}, \{n_2, n_3, n_4, n_5, n_7, n_8, n_{11}, n_{12}, n_{13}, n_{15}, n_{16}, n_{17}, n_{19}, n_{20}\} \rangle$.
- $E = \langle (n_1, (solid, Teaches), n_9), (n_1, (solid, Name), n_2), (n_1, (solid, IDCode), n_3), (n_1, (solid, Age), n_4), (n_1, (solid, Teacher_ID), n_5), (n_1, (solid, Get_Academic_Rank), n_6), (n_6, (solid, Level_Code), n_7), (n_6, (solid, Name), n_8), (n_9, (solid, Name), n_{10}), (n_9, (solid, Course_ID), n_{11}), (n_9, (solid, Course_Type), n_{12}), (n_9, (solid, Teacher_ID), n_{14}), (n_{14}, (solid, Student_ID), n_{15}), (n_{14}, (solid, IDCode), n_{16}), (n_{14}, (solid, Name), n_{17}), (n_{14}, (solid, Enrolls), n_{18}), (n_{18}, (solid, Attends), n_9), (n_{18}, (solid, Program_ID), n_{19}), (n_{18}, (solid, Name), n_{20}) \rangle$.
- $\ell(n_1) = (solid, Teacher), \ell(n_2) = \ell(n_8) = \ell(n_{10}) = \ell(n_{17}) = \ell(n_{20}) = (solid, Name), \ell(n_3) = \ell(n_{16}) = (solid, IDCode), \ell(n_4) = (solid, Age), \ell(n_5) = \ell(n_{13}) = (solid, Teacher_ID), \ell(n_6) = (solid, Get_Academic_Rank), \ell(n_7) = (solid, Rank_ID), \ell(n_9) = (solid, Course), \ell(n_{11}) = (solid, Course_ID), \ell(n_{12}) = (solid, Course_Type), \ell(n_{14}) = (solid, Student), \ell(n_{15}) = (solid, Student_ID), \ell(n_{18}) = (solid, Program), \ell(n_{19}) = (solid, Program_ID)$.

Constructing Kripke Structure from W-Graph. As we mentioned in Section 3.2, Kripke structure is a finite directed graph where the vertices are labeled with sets of AP. Hence, given a W-Graph W_G , the corresponding Kripke structure over atomic propositions AP_G are defined as $\mathcal{K}_G = \langle S_G, I_G, R_G, L_G \rangle$ based on the following transformation rules:

- An atomic proposition AP_G is compound of a set of atomic nodes N_a in W_G , formally, $AP_G = \{n \mid \forall n \in N_a, n = \ell(n)\}$.
- A finite set of composite nodes N_c in W_G are regarded as an initial state I_G in Kripke structure.
- The edge E between two nodes $n_i, n_j, \{n_i, n_j \mid \forall n_i \in N, \forall n_j \in N\}$ is mapped into a transition R_G in Kripke structure.
- The edge labels, e.g., edge label, negative label (\bar{l}), and inverse label (l^{-1}) are transformed into the labeling function L_G in Kripke structure.

Based on the above transformation rules, the corresponding Kripke structure could be constructed and depicted in Figure 4.

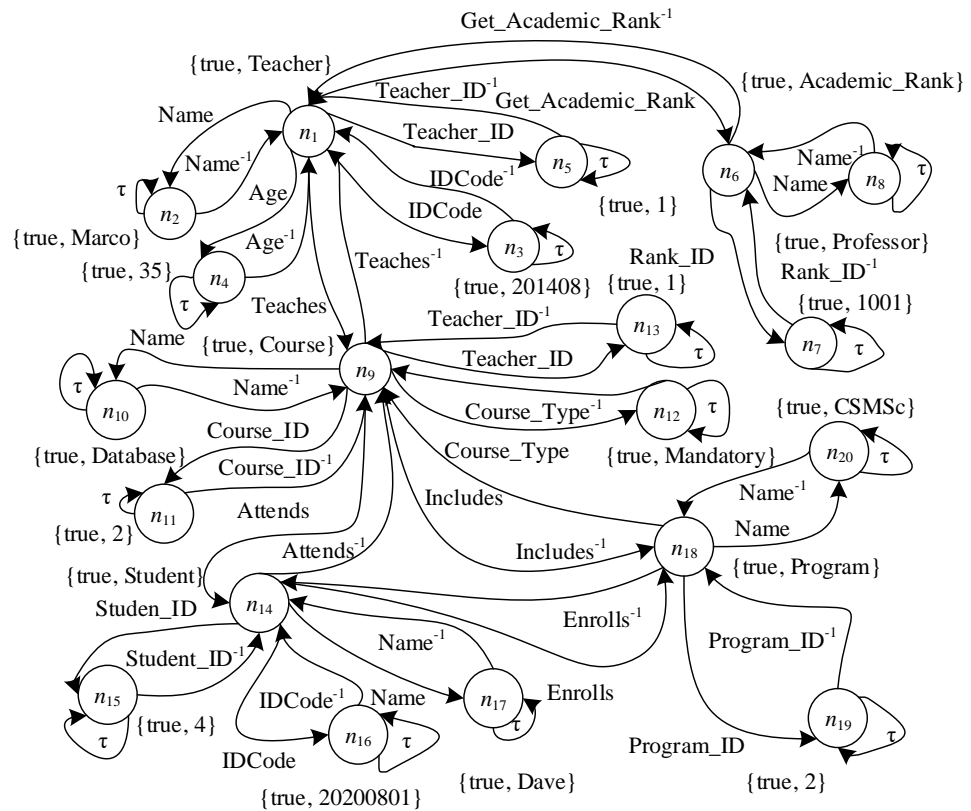


Figure 4. Kripke structure of Mini University data model.

As shown in Figure 4, each node in W -Instance is transformed as a state, while the label functions between two adjacent nodes in W -Instance are mapped as transitions. In particular, the inverse transition, e.g., Age^{-1} , $Name^{-1}$, between two states is introduced for modeling the possible state transitions. Formally, the atomic propositions, initial states, transitions, and labeling functions could be defined as follows:

- $AP_G = \{Teacher, Marco, 201408, 35, 1, Academic_Rank, 1001, Professor, Course, Database, 2, Mandatory, 1, Student, 4, 20200801, Dave, Program, 2, CSMSc\}$.
- $S_G = \{n_1, n_2, n_3, \dots, n_{20}\}$.
- $I_G = \{n_1\}$.
- $R_G = \{(n_1, Name, n_2), (n_2, Name^{-1}, n_1), (n_1, IDCode, n_3), (n_3, IDCode^{-1}, n_1), (n_1, Age, n_4), (n_4, Age^{-1}, n_1), (n_1, Teacher_ID, n_5), (n_5, Teacher_ID^{-1}, n_1), (n_1, Get_Academic_Rank, n_6), (n_6, Get_Academic_Rank^{-1}, n_6), (n_6, Rank_ID, n_7), (n_7, Rank_ID^{-1}, n_6), (n_6, Teaches, n_7), (n_7, Teaches^{-1}, n_6), (n_6, Name, n_8), (n_8, Name^{-1}, n_6), (n_9, Name, n_{10}), (n_{10}, Name^{-1}, n_9), (n_9, Course_ID, n_{11}), (n_{11}, Course_ID^{-1}, n_9), (n_9, Course_Type, n_{12}), (n_{12}, Course_Type^{-1}, n_9), (n_9, Teacher_ID, n_{13}), (n_{13}, Teacher_ID^{-1}, n_9), (n_9, Attends, n_{14}), (n_{14}, Attends^{-1}, n_9), (n_{14}, Student_ID, n_{15}), (n_{15}, Student_ID^{-1}, n_{14}), (n_{14}, IDCode, n_{16}), (n_{16}, IDCode^{-1}, n_{14}), (n_{14}, Name, n_{17}), (n_{17}, Name^{-1}, n_{14}), (n_{14}, Enrolls, n_{18}), (n_{18}, Enrolls^{-1}, n_{14}), (n_{18}, Program_ID, n_{19}), (n_{19}, Program_ID^{-1}, n_{18}), (n_{18}, Name, n_{20}), (n_{20}, Name^{-1}, n_{18}), (n_{18}, Includes, n_9), (n_9, Includes^{-1}, n_{18})\}$.
- $L_G = \{L_G(n_1) = (true, Teacher), L_G(n_2) = (true, Marco), L_G(n_3) = (true, 201408), L_G(n_4) = (true, 35), L_G(n_5) = (true, 1), L_G(n_6) = (true, Academic_Rank), L_G(n_7) = (true, 1001), L_G(n_8) = (true, Professor), L_G(n_9) = (true, Course), L_G(n_{10}) = (true, Database), L_G(n_{11}) = (true, 2), L_G(n_{12}) = (true, Mandatory), L_G(n_{13}) = (true, 1), L_G(n_{15}) = (true, Student), L_G(n_{14}) = (true, 4), L_G(n_{15}) = (true, 20200801), L_G(n_{16}) = (true, Dave), L_G(n_{17}) = (true, Program), L_G(n_{18}) = (true, 2), L_G(n_{19}) = (true, 2), L_G(n_{20}) = (true, CSMSc)\}$.

Based on the above definitions, the W-Instance was transformed into a Kripke structure. Essentially, it is a kind of finite-state model that can be easily encoded by the SMV program.

4.3.2. Formalization of Specification

To check the consistency by using model checking, the semantic specification of learned ontologies should be formalized by a logic formula. We suppose that the outputs of the ontology learning from Mini University database are depicted in Figure 5.

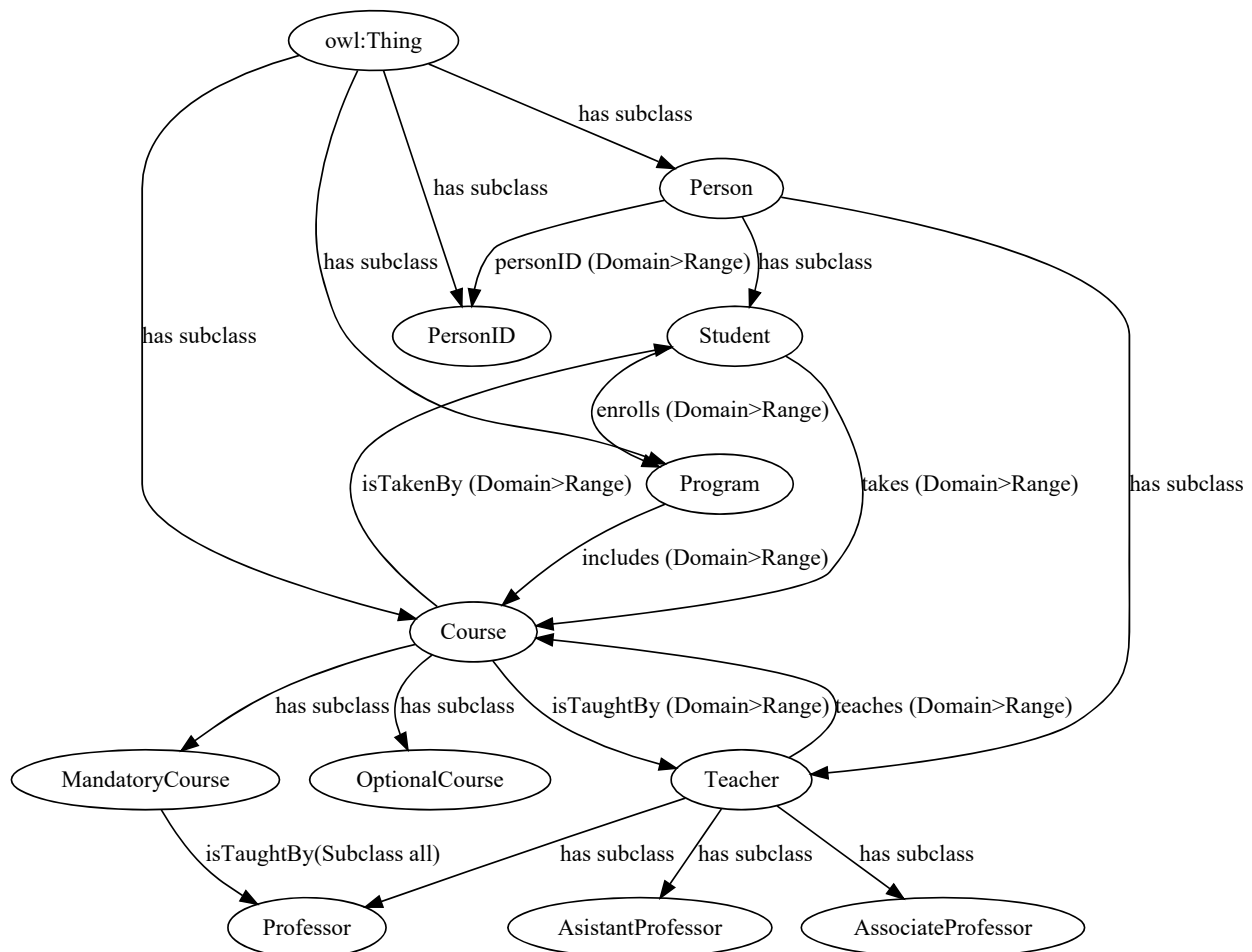


Figure 5. Example ontology of Mini University data model.

As shown in Figure 5, there are 12 concepts or properties, i.e., Person, Student, Teacher, Professor, AssociateProfessor, AssistantProfessor, Program, Course, MandatoryCourse, OptionalCourse, and their relationships or roles, including taxonomies, associative relationships, and inverse relationships. The main taxonomies in this ontology are subsumption, e.g., Student and Teacher are subsumption of Person. The representative associative relationship is Enrolls, Takes, Teaches, and Includes. Moreover, there exist many inverse relationships, e.g., Teaches and Takes are inverse of isTaughtBy and isTakenBy respectively. In addition, there are some constraint axioms, e.g., MandatoryCourse can only be TaughtBy Professor, and cardinality, Student is a person who enrolls in at least one Program. Accordingly, the semantic specification of learned ontologies could be formalized by using *ALCI* DL syntax as follows:

Person $\sqsubseteq \top$, Person $\sqcap \exists$ PersonID.String, Student \sqsubseteq Person, Teacher \sqsubseteq Person, Professor \sqsubseteq Teacher, AssociateProfessor \sqsubseteq Teacher, AssistantProfessor \sqsubseteq Teacher.

$\text{Student} \equiv \text{Person} \sqcap \geq 1 \text{ Enrolls.Program}$, $\text{Student} \equiv \text{Person} \sqcap \exists \text{ Takes.Course}$, $\text{Teacher} \equiv \text{Person} \sqcap \exists \text{ Teaches.Course}$, $\text{Professor} \equiv \text{Teacher} \sqcap \exists \text{ Teaches.MandatoryCourse}$,
 $\forall \text{ Teaches.Course} \sqsubseteq \text{Professor} \sqcup \text{AssistantProfessor} \sqcup \text{AssociateProfessor} \sqcup \neg \text{Student}$,
 $\text{Teaches} \equiv \text{isTaughtBy}^- . \text{Course} \sqsubseteq \top$, $\text{MandatoryCourse} \sqsubseteq \text{Course}$, $\text{OptionalCourse} \sqsubseteq \text{Course}$,
 $\text{MandatoryCourse} \equiv \text{Course} \sqcap \forall \text{ isTaughtBy.Professor}$, $\text{OptionalCourse} \equiv \text{Course} \sqcap \forall \text{ isTaughtBy.Teacher}$.

$\text{Program} \sqsubseteq \top$, $\text{Program} \sqsubseteq \exists \text{ Includes.Course}$, $\text{Student} \sqsubseteq \exists \text{ Enrolls.Program}$, $\forall \text{ Enrolls.Program} \sqsubseteq \text{CSBSc} \sqcup \text{CSMSc} \sqcup \text{CSPHD}$.

Based on \mathcal{ALCI} DL syntax, the semantic specifications of ontologies are formally described by using the TBox syntax of DL. Thus, the following specifications of the corresponding ontologies are formally defined by ABox of DL syntax as follows:

Marco: $\text{Person} \sqcap \text{Professor}$, Dave: $\text{Person} \sqcap \text{Student}$, Database: $\text{Course} \sqcap \text{MandatoryCourse}$,
 CSMSc: Program

$(\text{Marco}, \text{Database}): \text{Teaches}$, $(\text{Dave}, \text{CSMSc}): \text{Enrolls}$, $(\text{CSMSc}, \text{Database}): \text{Includes}$.

Thereby, the Professor whose name is Marco, he teaches Database that Includes in CSMSc Program, EnrolledBy a Student, whose name is Dave could be formalized by using DLs formula:

$\text{Professor}(\text{Marco}) \sqcap \exists \text{ Teaches.Course}(\text{Database}) \sqcap \exists \text{ IncludedBy.Program}(\text{CSMSc}) \sqcap \exists \text{ EnrolledBy.Student}(\text{Dave})$.

4.4. Consistency Checking Algorithm

As previously mentioned, we referred to the task of checking the consistency of ontologies as a global model checking. Accordingly, the symbolic model checker is employed by considering that model checking could return the result of whether a finite-state model (Kripke model) satisfies the given specification [32].

In the CTL model checking, the specifications encoded in the CTL formula are checked by NuSMV to verify whether it satisfies with the given finite-state model [33], the model checker returns a counterexample when it is unsatisfied. When it comes to semantic consistency checking, the specifications of the ontology encoded by the CTL formula can be checked. Accordingly, the model checker will return the results indicating whether the specifications of the learned ontology satisfy the relational data model that is represented and formalized by W-Graph and Kripke structure. Considering that nuXmv [34] is an extended version of NuSMV, which provides a strong verification based on advanced SAT-based algorithms, thereby the nuXmv is employed to verify the presented method. The specific process of consistency checking based on graph intermediate representation and model checking is shown in Algorithm 1.

Algorithm 1 Semantic Consistency Checking based on nuXmv Model Checker

Input:

\mathcal{K}_G : Kripke structure of original RDB represented based on W-Graph.

\mathcal{O} : Ontologies generated from RDB schema and instance \mathcal{R} .

Output:

\mathcal{R}_C : The result of consistency checking: **true** or **false**.

- 1: **procedure** CONSISTENCY CHECKING
 - 2: Encoding Kripke structure \mathcal{K}_G with SMV program.
 - 3: Translating semantics of ontologies to CTL formula $\varphi(\mathcal{O}, \mathcal{R})$.
 - 4: Verifying if the $\varphi(\mathcal{O}, \mathcal{R})$ satisfies the \mathcal{K}_G .
 - 5: **if** $\mathcal{K}_G \models \varphi(\mathcal{O}, \mathcal{R})$ **then**
 - 6: $\mathcal{R}_C = \text{true}$.
 - 7: **else if** $\mathcal{K}_G \not\models \varphi(\mathcal{O}, \mathcal{R})$ **then**
 - 8: $\mathcal{R}_C = \text{false}$.
 - 9: **end if**
 - 10: return \mathcal{R}_C .
 - 11: **end procedure**
-

As we can see in Algorithm 1, the inputs of consistency checking based on model checking are the Kripke model and CTL logic formula, while the output is the result of whether the input Kripke model satisfies the given semantic specification. In the following subsection, we will verify the feasibility and effectiveness of the presented method in checking the semantic consistency of learning ontology from RDB.

4.5. Verification

In the preceding subsection, we transformed W-Graph into Kripke structure and formalized semantic specifications with logic formula. Thereby, in this subsection, we verify the feasibility and effectiveness of the presented consistency checking method by encoding the Kripke structure with SMV and running the nuXmv model checker. Figure 6 summarizes the main steps of the verification.

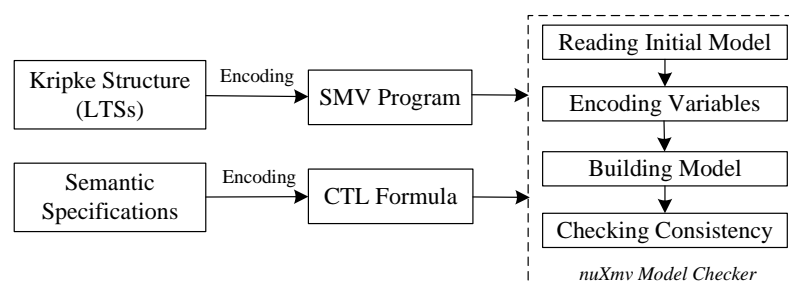


Figure 6. The main steps of the verification.

4.5.1. Encoding the Kripke Structure and LTSs with SMV Program

The nuXmv model checker is a kind of symbolic model verifier, which is usually utilized to check the consistency of the state transition system. Considering that both the Kripke structure and labeled transition system (LTS) are semantic models, which have equivalent expressive [35], hence we can view the Kripke structure as a labeled transition system. As a result that the state transition system encoded by the SMV program is only readable by the nuXmv model checker, we encode the Kripke structure with the SMV program.

In our case, the label in the Kripke structure of Mini University data model is regarded as a state in a finite-state transition system, which is encoded by the SMV program. During this encoding, the inverse labels (l^{-1}) are encoded as a state variable with the prefix of Inv_, e.g., Age^{-1} are encoded as Inv_Age, Attends^{-1} are encoded as Inv_Attends, etc. In order to decrease the complexity of the model, we ignore the self-transition of the leaf state in the Kripke model. Namely, we only define and assign the state, values, and the transitions between these states and labels.

4.5.2. Translating Semantic Specifications from DLs Formula to CTL Formula

Considering that each pre-state can be succeeded by more than one post-state in CTL, we translate the formal specifications of the ontologies that are encoded by *ALCC* DL syntax into the CTL formula. Accordingly, the specification of a Professor whose name is Marco, he teaches Database that Includes in MSC Program, EnrolledBy a Student, whose name is Dave, which could be translated into CTL formula as follows:

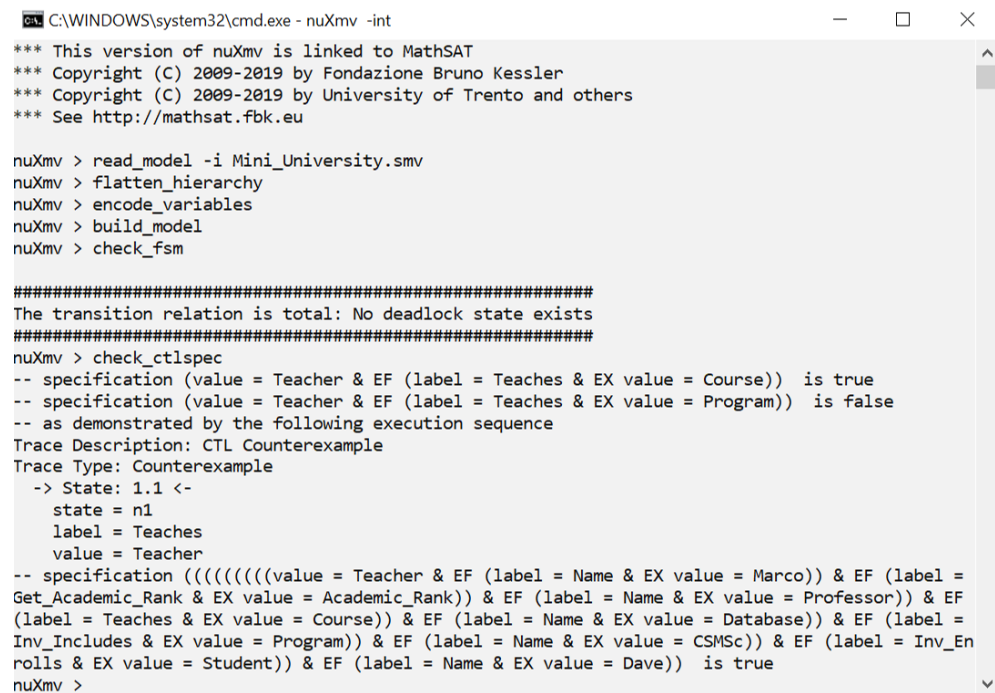
```

(value=Teacher & EF(label=Name & EX value=Marco) & EF(label=Get_Academic_Rank & EX value=Academic_Rank) & EF (label=Name & EX value=Professor) & EF (label=Teaches & EX value=Course) & EF(label=Name & EX value=Database) & EF (label=Inv_Includes & EX value=Program) & EF(label=Name & EX value=CSMSc) & EF(label=Inv_Enrolls & EX value=Student) & EF(label=Name & EX value=Dave))
  
```

Based on the above translations, semantic consistency checking could be converted into a global model checking problem that can be solved by a model checker.

4.5.3. Checking Ontology Consistency Based on Model Checker

To verify the feasibility of this method, we ran the nuXmv model checker on the Windows 10 machine with the Intel (R) Core (TM) i5 64-bit processor and 8GB RAM. Before verifying whether the given specification satisfies the Kripke structure, it is necessary to check if there exist deadlock states by using `check_fsm` command [36]. To verify the effectiveness of the presented method in checking different kinds of inconsistencies, i.e., property inconsistency, relationship inconsistency, etc., we checked not only the specifications that are consistent with the original database but also the specifications that are inconsistent with the original database. Accordingly, the result of the consistency checking is shown in Figure 7.



```

C:\WINDOWS\system32\cmd.exe - nuXmv -int
*** This version of nuXmv is linked to MathSAT
*** Copyright (C) 2009-2019 by Fondazione Bruno Kessler
*** Copyright (C) 2009-2019 by University of Trento and others
*** See http://mathsat.fbk.eu

nuXmv > read_model -i Mini_University.smv
nuXmv > flatten_hierarchy
nuXmv > encode_variables
nuXmv > build_model
nuXmv > check_fsm

#####
The transition relation is total: No deadlock state exists
#####
nuXmv > check_ctlspec
-- specification (value = Teacher & EF (label = Teaches & EX value = Course)) is true
-- specification (value = Teacher & EF (label = Teaches & EX value = Program)) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  state = n1
  label = Teaches
  value = Teacher
-- specification (((((((value = Teacher & EF (label = Name & EX value = Marco)) & EF (label =
Get_Academic_Rank & EX value = Academic_Rank)) & EF (label = Name & EX value = Professor)) & EF
(label = Teaches & EX value = Course)) & EF (label = Name & EX value = Database)) & EF (label =
Inv_Includes & EX value = Program)) & EF (label = Name & EX value = CSMSC)) & EF (label = Inv_En
rolls & EX value = Student)) & EF (label = Name & EX value = Dave)) is true
nuXmv >

```

Figure 7. Results of consistency checking based on nuXmv model checker.

In Figure 6, we can observe that there is no deadlock state in the current Kripke model. The results given by nuXmv indicate whether the given specifications satisfy the specific Kripke model. To begin with, we simply verified the relationship consistency. In the first case, the semantic correspondence of whether there exists a Teacher who Teaches a Course is checked, while the semantic correspondence of whether there exists a Teacher who Teaches a Program is checked in the second case. The nuXmv model checker returns the true in first case, while the false and a counterexample are given in the second case. These results indicate that the first specification is consistent with the original RDB, while the second one is inconsistent with the original RDB. Namely, the relationship of Teacher Teaches a Course is consistent with the original database, while the relationship of Teacher Teaches the Program is inconsistent. Hereafter, we verified the specification that formalizes from ontology and *ALCC* DL, the model checker returns the true, which indicates that there are no inconsistencies in the current case.

5. Conclusions

In this work, we presented a (semi-)automatic semantic consistency checking method based on the graph intermediate representation and model checking to check the semantic consistency while learning ontology from RDB. We formalized the semantic correspondence between ontologies and its original data model. We converted the problem of semantic consistency checking into the global model checking problem that was eventually solved automatically by the nuXmv model checker. In addition, we gave an example to

demonstrate the specific process of the formalization and conducted a verification experiment to verify the feasibility of the presented method. The results showed that this method could correctly check and return the results of whether the given semantic specification of the learned ontology satisfies the original RDB. Currently, this method is only verified in the scenario of learning ontology from a single database, thus how to check the semantic consistency of learning ontology from multiple databases could be investigated in the future. Meanwhile, considering that the complexity of the model checking will increase along with the volume and complexity of the data model, how to evaluate and optimize the complexity of the transformations and model checking are worthy to be investigated further. In addition to checking the semantic consistency, the more crucial work is to eliminate the identified inconsistencies and redundancies, thus how to eliminate these inconsistencies is meaningful work as well.

Author Contributions: Conceptualization, C.M. and B.M.; methodology, C.M. and B.M.; formal analysis, C.M.; validation, C.M.; investigation, C.M. and B.M.; resources, B.M. and A.B.; writing—original draft preparation, C.M.; writing—review and editing, B.M. and A.B.; supervision, B.M. and A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by grants of the European Union co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002), “Application Domain Specific Highly Reliable IT Solutions” project that has been implemented with the support provided from the National Research, Development, and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme, and by the grant of China Scholarship Council (201808610145).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data and code presented in this study are available on request from the author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Fauqueur, J.; Thillaisundaram, A.; Togia, T. Constructing large scale biomedical knowledge bases from scratch with rapid annotation of interpretable patterns. In Proceedings of the 18th BioNLP Workshop and Shared Task, Florence, Italy, 1 August 2019; pp. 142–151.
2. Giacomo, G.D.; Lembo, D.; Lenzerini, M.; Poggi, A.; Rosati, R. Using Ontologies for Semantic Data Integration. In *Studies in Big Data*; Springer International Publishing: Berlin, Germany, 2017; pp. 187–202. [[CrossRef](#)]
3. Ozaki, A. Learning Description Logic Ontologies: Five Approaches. Where Do They Stand? *KI Künstliche Intelligenz* **2020**, *34*, 317–327. [[CrossRef](#)]
4. Zhou, L. Ontology learning: State of the art and open issues. *Inf. Technol. Manag.* **2007**, *8*, 241–252. [[CrossRef](#)]
5. Zhu, L.; Hua, G.; Zafar, S.; Pan, Y. Fundamental ideas and mathematical basis of ontology learning algorithm. *J. Intell. Fuzzy Syst.* **2018**, *35*, 4503–4516. [[CrossRef](#)]
6. Vrandečić, D. Ontology Evaluation. In *Handbook on Ontologies*; Staab, S., Studer, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 293–313. [[CrossRef](#)]
7. Khadir, A.C.; Aliane, H.; Guessoum, A. Ontology learning: Grand tour and challenges. *Comput. Sci. Rev.* **2021**, *39*, 100339. [[CrossRef](#)]
8. O’Regan, G. Overview of Formal Methods. In *Concise Guide to Formal Methods: Theory, Fundamentals and Industry Applications*; Springer International Publishing: Cham, Switzerland, 2017; pp. 41–63. [[CrossRef](#)]
9. Baclawski, K.; Kokar, M.M.; Waldinger, R.; Kogut, P.A. Consistency Checking of Semantic Web Ontologies. In Proceedings of the Semantic Web—ISWC 2002, Sardinia, Italy, 9–12 June 2002; Horrocks, I., Hendler, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; pp. 454–459. [[CrossRef](#)]
10. Neumann, C.P.; Fischer, T.; Lenz, R. OXDBS: Extension of a Native XML Database System with Validation by Consistency Checking of OWL-DL Ontologies. In Proceedings of the Fourteenth International Database Engineering & Applications Symposium, Montreal, QC, Canada, 16–18 August 2010; Association for Computing Machinery: New York, NY, USA, 2010; pp. 143–148. [[CrossRef](#)]
11. del Mar Roldán-García, M.; García-Nieto, J.; Aldana-Montes, J.F. Enhancing semantic consistency in anti-fraud rule-based expert systems. *Expert Syst. Appl.* **2017**, *90*, 332–343. [[CrossRef](#)]

12. Bayoudhi, L.; Sassi, N.; Jaziri, W. OWL 2 DL Ontology Inconsistencies Prediction. In Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics, WIMS '17, Amantea, Italy, 19–22 June 2017; Association for Computing Machinery: New York, NY, USA, 2017. [\[CrossRef\]](#)
13. Rosati, R.; Ruzzi, M.; Graziosi, M.; Masotti, G. Evaluation of Techniques for Inconsistency Handling in OWL 2 QL Ontologies. In Proceedings of the The Semantic Web—ISWC 2012, Newcastle, UK, 18–22 June 2012; Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., et al., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 337–349.
14. Pileggi, S.F.; Crain, H.; Yahia, S.B. An Ontological Approach to Knowledge Building by Data Integration. In Proceedings of the Computational Science—ICCS 2020, Amsterdam, The Netherlands, 3–5 June 2020; Krzhizhanovskaya, V.V., Závodszy, G., Lees, M.H., Dongarra, J.J., Sloot, P.M.A., Brissos, S., Teixeira, J., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 479–493.
15. Fahad, M.; Moalla, N.; Bouras, A. Detection and resolution of semantic inconsistency and redundancy in an automatic ontology merging system. *J. Intell. Inf. Syst.* **2012**, *39*, 535–557. [\[CrossRef\]](#)
16. Bai, X.; Sun, J.; Li, Z.; Lu, X. Domain Ontology Learning and Consistency Checking Based on TSC Approach and Racer. In Proceedings of the Web Reasoning and Rule Systems, Innsbruck, Austria, 7–8 June 2007; Marchiori, M., Pan, J.Z., Marie, C.d.S., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 148–162. [\[CrossRef\]](#)
17. Paulheim, H.; Stuckenschmidt, H. Fast Approximate A-Box Consistency Checking Using Machine Learning. In Proceedings of the The Semantic Web. Latest Advances and New Domains, Heraklion, Crete, Greece, 29 May–2 June 2016; Sack, H., Blomqvist, E., d’Aquin, M., Ghidini, C., Ponzetto, S.P., Lange, C., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 135–150. [\[CrossRef\]](#)
18. Lam, S.C.; Sleeman, D.; Vasconcelos, W. Graph-based ontology checking. In Proceedings of the Workshop Ontology Management: Searching, Selection, Ranking, and Segmentation in K-CAP 05, Banff, AB, Canada, 2 October 2005.
19. Yang, S.; Tan, H.; Wu, J. Semantic Consistency Checking in Building Ontology from Heterogeneous Sources. *J. Appl. Math.* **2014**, *2014*, 1–11. [\[CrossRef\]](#)
20. Mahfoudh, M.; Forestier, G.; Thiry, L.; Hassenforder, M. Algebraic graph transformations for formalizing ontology changes and evolving ontologies. *Knowl. Based Syst.* **2015**, *73*, 212–226. [\[CrossRef\]](#)
21. Jun, H.G.; Im, D.H.; Kim, H.J. Semantics-preserving optimisation of mapping multi-column key constraints for RDB to RDF transformation. *J. Inf. Sci.* **2020**, 1–15. [\[CrossRef\]](#)
22. Baader, F.; Horrocks, I.; Lutz, C.; Sattler, U. A Basic Description Logic. In *An Introduction to Description Logic*; Cambridge University Press: Cambridge, UK, 2017; pp. 10–49. [\[CrossRef\]](#)
23. Sequeda, J.F. Integrating Relational Databases with the Semantic Web: A Reflection. In Proceedings of the Reasoning Web. Semantic Interoperability on the Web: 13th International Summer School, Tutorial Lectures, London, UK, 7–11 July 2017; Ianni, G., Lembo, D., Bertossi, L., Faber, W., Glimm, B., Gottlob, G., Staab, S., Eds.; Springer: Cham, Switzerland, 2017; pp. 68–120. [\[CrossRef\]](#)
24. Yang, S.; Zheng, Y.; Yang, X. Semi-automatically building ontologies from relational databases. In Proceedings of the 2010 3rd International Conference on Computer Science and Information Technology, Chengdu, China, 9–11 July 2010; IEEE: New York, NY, USA, 2010; pp. 150–154. [\[CrossRef\]](#)
25. Dovier, A.; Quintarelli, E. Applying model-checking to solve queries on semistructured data. *Comput. Lang. Syst. Struct.* **2009**, *35*, 143–172. [\[CrossRef\]](#)
26. Ziller, R.; Schneider, K. Combining Supervisor Synthesis and Model Checking. *ACM Trans. Embed. Comput. Syst.* **2005**, *4*, 331–362. [\[CrossRef\]](#)
27. Kernberger, D.; Lange, M. Model Checking for the Full Hybrid Computation Tree Logic. In Proceedings of the 23rd International Symposium on Temporal Representation and Reasoning (TIME), Kongens Lyngby, Denmark, 17–19 October 2016; pp. 31–40. [\[CrossRef\]](#)
28. Nardi, D.; Brachman, R.J. An Introduction to Description Logics. In *The Description Logic Handbook: Theory, Implementation and Applications*, 2nd ed.; Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., Eds.; Cambridge University Press: Cambridge, UK, 2007; pp. 1–44. [\[CrossRef\]](#)
29. Ben-David, S.; Trefler, R.; Weddell, G. Model Checking Using Description Logic. *J. Log. Comput.* **2008**, *20*, 111–131. [\[CrossRef\]](#)
30. Clarke, E.M.; Henzinger, T.A.; Veith, H. Introduction to Model Checking. In *Handbook of Model Checking*; Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 1–26. [\[CrossRef\]](#)
31. Čerāns, K.; Būmans, G. RDB2OWL: A RDB-to-RDF/OWL Mapping Specification Language. In Proceedings of the 2011 Conference on Databases and Information Systems VI: Selected Papers from the Ninth International Baltic Conference, DB&IS 2010, Riga, Latvia, 5–7 July 2010; pp. 139–152. [\[CrossRef\]](#)
32. Ben-David, S. Applications of Description Logic and Causality in Model Checking. Ph.D. Thesis, University of Waterloo, Waterloo, ON, Canada, 2009.
33. Hassan, Z.; Bradley, A.R.; Somenzi, F. Incremental, Inductive CTL Model Checking. In *Computer Aided Verification*; Madhusudan, P., Seshia, S.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 532–547.

34. Cavada, R.; Cimatti, A.; Dorigatti, M.; Griggio, A.; Mariotti, A.; Micheli, A.; Mover, S.; Roveri, M.; Tonetta, S. The nuXmv Symbolic Model Checker. In *Computer Aided Verification*; Biere, A., Bloem, R., Eds.; Springer International Publishing: Cham, Switzerland, 2014; pp. 334–342.
35. Reniers, M.; Schoren, R.; Willemse, T. Results on embeddings between state-based and event-based systems. *Comput. J.* **2014**, *57*, 73–92. [[CrossRef](#)]
36. Souri, A.; Rahmani, A.M.; Navimipour, N.J.; Rezaei, R. A symbolic model checking approach in formal verification of distributed systems. *Hum. Centric Comput. Inf. Sci.* **2019**, *9*, 4. [[CrossRef](#)]