A Platform-independent Approach for Parallel Reasoning with OWL EL Ontologies using Graph Representation

Zhangquan Zhou, Guilin Qi, Zimian Wu, Jun Shi
School of Computer Science and Engineering, Southeast University, China
{zzq, gqi, wzm, sj}@seu.edu.cn

***Abstract*—OWL EL is a tractable fragment of OWL 2. Classification, which is the task of computing a subsumption hierarchy between concepts, plays an important role in many applications of OWL EL. The current classification methods are proposed based on the specific parallel computation platforms, thus can hardly adapt to other platforms and satisfy different needs of users or developers. In this paper, we propose a platform-independent approach to performing classification in OWL EL. We first give a novel and well defined graph formalism $\mathcal{GEL}$ for representing EL ontologies. Based on this formalism, we describe the classification on a platform-independent computation model, which also captures the lower bound of run-time of all parallel classification algorithms. We further give a refined algorithm based on the proposed model, and show the correctness of the algorithm. We implement a prototype system, which can easily switch between multi-core and a distributed cluster. Finally, we conduct experiments on several real-world OWL EL ontologies. The experimental results show that our system outperforms two state-of-the-art EL reasoning systems, and has a linear scalability on the extensions of GO ontologies.**

***Keywords*-OWL 2 EL; description logics; parallel classification; graph; platform-independence**

I. Introduction

The Web Ontology Language OWL has been designed as the major standard for ontology languages. The most recent version is OWL 2[1]. Among different profiles of OWL 2, OWL 2 EL (EL for short), which is essentially based on the description logic $\mathcal{EL}$ [2], stands out for its positive complexity results and the sufficient expressive power. Therefore, EL has been widely adopted in many domains of application, in particular in the bio-medical domains (see the work on Gene Ontology[2] and SNOMED CT [19]). Recently, EL has also been used for traffic congestions diagnosing [11]. In these applications, *classification*, which is the task of computing a subsumption hierarchy between concept descriptions, plays an important role. By means of it, developers and users can optimize query answering, ontology diagnosis and debugging.

The first efficient reasoner for classification in OWL EL is CEL [3]. Since CEL is essentially a serial reasoner, it dose not perform very well on large ontologies, like SNOMED CT. Recently, there is an increasing interest in exploiting parallel technologies to achieve high efficiency and scalability on large ontologies in OWL EL. ELK [10] is the first attempt to use multi-core techniques and other optimizations to enhance the efficiency of reasoning in OWL EL. It classifies the SNOMED CT ontology in less than half a minute (the classification time of CEL is 15 minutes). Since ELK can only run on a single machine, it is restricted to the main memories of the utilized machines. Another line of the works on parallel reasoning in OWL EL is based on the distributed computation platforms to gain more computing resources. These works are based on MapReduce [16], [26] or Redis [15]. The proposed experiments show that reasoners based on such platforms have a good performance in scalability. However, these reasoners are typically not efficient due to the inherent overhead of the platforms. In summary, the current systems on parallel reasoning in OWL EL are platform-dependent, i.e., they have to be implemented in a specific platform.

In this paper, we propose a platform-independent approach for parallel EL classification. A platform-independent approach is important due to the following reasons:

- For system developers, the algorithms and optimizations used to exploit the advantages of a specific platform can also be used on other platforms.
- Since a platform-independent approach allows hiding the specific platforms, it can achieve a tradeoff between high efficiency and sufficient scalability.
- For users who are working on the platform-independent systems, they have alternative choices of different platforms according to their own devices.

Motivated by the popularity of utilizing a graph as a platform-independent model for processing large scale datasets, we propose a novel graph representation for EL ontologies (Section III). After that, we combine our graph representation with a famous parallel computing model, based on which the classification can be implemented on different parallel computing platforms (Section IV). We further give a refined parallel classification algorithm and show its correctness (Section V). We implement a prototype system which can take advantage of both of multi-core and distributed computation techniques. That is, it can be run on different platforms such as multi-core machines and a cluster

[1]http://www.w3.org/TR/owl2-profiles/
[2]http://geneontology.org/

of personal computers by setting different configurations. We conduct experiments on several real-world OWL EL ontologies. The experimental results show that our system outperforms two state-of-the-art EL reasoning systems, and has a linear scalability on the extensions of GO ontologies (Section VI).

II. Preliminaries

In this section, we introduce some basic notions of $\mathcal{EL}^+$, which is a tractable description logic that underpins OWL EL. A (complex) concept in $\mathcal{EL}^+$ is formulated according to the syntax rule: $C ::= A|\top|\bot|C \sqcap D|\exists r.C$, where A ranges over atomic concepts, r ranges over role names, $\top$ denotes the *top concept* (or universal concept), while $\bot$ denotes the *bottom concept* (or empty concept), C, D denote complex concepts that are constructed inductively from primitive concepts (a primitive concept is either of an atomic concept, top concept or bottom concept). An $\mathcal{EL}^+$ ontology is a finite set of *general concept inclusion* (GCI) axioms of the form $C \sqsubseteq D$ and *role inclusion* (RI) axioms of the form $r_1 \circ \cdots \circ r_n \sqsubseteq r$ ($r_1 \circ \cdots \circ r_n$ is the composition of these roles). The semantics of $\mathcal{EL}^+$ follows the standard model theoretic semantics and can be found in [2]. A polynomial algorithm is proposed to perform classification of $\mathcal{EL}^+$ ontologies in [3]. This algorithm first transforms a given ontology into a normal form such that all GCI axioms are of the form $A \sqsubseteq B$, $A_1 \sqcap A_2 \sqsubseteq B$, $A \sqsubseteq \exists r.B$ or $\exists r.B \sqsubseteq A$. In these normalized axioms A and B are primitive concepts. All normalized RI axioms are in the form of $r_1 \circ r_2 \sqsubseteq s$ or $r \sqsubseteq s$. The normalization can be done in linear time. *In the following paper, we assume that an $\mathcal{EL}^+$ ontology is normalized.* Let $\mathcal{C}_O$ be the classification result for the given ontology $\mathcal{O}$ and initialized as an empty set. For each primitive concept A, the algorithm adds $A \sqsubseteq_{\mathcal{O}} A$ and $A \sqsubseteq_{\mathcal{O}} \top$ to $\mathcal{C}_O$, where '$\sqsubseteq_{\mathcal{O}}$' means that the axiom is newly derived. Then $\mathcal{C}_O$ is extended by applying the completion rules in Table 1 until no more rules can be applied.

Table 1: Completion rules for $\mathcal{EL}^+$

CR1	If $X \sqsubseteq_{\mathcal{O}} A_1 \in \mathcal{C}_O, ..., X \sqsubseteq_{\mathcal{O}} A_l \in \mathcal{C}_O$, $A_1 \sqcap ... \sqcap A_l \sqsubseteq B \in \mathcal{O}$ and $X \sqsubseteq_{\mathcal{O}} B \notin \mathcal{C}_O$ then $\mathcal{C}_O := \mathcal{C}_O \cup \{X \sqsubseteq_{\mathcal{O}} B\}$, where $l = 1$ or 2.
CR2	If $X \sqsubseteq_{\mathcal{O}} A \in \mathcal{C}_O, A \sqsubseteq \exists r.B \in \mathcal{O}$ and $X \sqsubseteq_{\mathcal{O}} \exists r.B \notin \mathcal{C}_O$ then $\mathcal{C}_O := \mathcal{C}_O \cup \{X \sqsubseteq_{\mathcal{O}} \exists r.B\}$
CR3	If $X \sqsubseteq_{\mathcal{O}} \exists r.Y \in \mathcal{C}_O, Y \sqsubseteq_{\mathcal{O}} A \in \mathcal{C}_O, \exists r.A \sqsubseteq B \in \mathcal{O}$, and $X \sqsubseteq_{\mathcal{O}} B \notin \mathcal{C}_O$ then $\mathcal{C}_O := \mathcal{C}_O \cup \{X \sqsubseteq_{\mathcal{O}} B\}$
CR4	If $X \sqsubseteq_{\mathcal{O}} \exists r.Y \in \mathcal{C}_O, r \sqsubseteq s \in \mathcal{O}$ and $X \sqsubseteq_{\mathcal{O}} \exists s.Y \notin \mathcal{C}_O$ then $\mathcal{C}_O := \mathcal{C}_O \cup \{X \sqsubseteq_{\mathcal{O}} \exists s.Y\}$
CR5	If $X \sqsubseteq_{\mathcal{O}} \exists r.Y \in \mathcal{C}_O, Y \sqsubseteq_{\mathcal{O}} \exists s.Z \in \mathcal{C}_O, r \circ s \sqsubseteq t \in \mathcal{O}$ and $X \sqsubseteq_{\mathcal{O}} \exists t.Z \notin \mathcal{C}_O$ then $\mathcal{C}_O := \mathcal{C}_O \cup \{X \sqsubseteq_{\mathcal{O}} \exists t.Z\}$

The following example will be used throughout the paper to illustrate several definitions and rules.

Example 1: Given an normalized $\mathcal{EL}^+$ ontology $\mathcal{O}$ which consists of the following axioms:

$$\alpha_1 : X \sqsubseteq \exists r.A \quad \alpha_2 : A \sqsubseteq B \quad \alpha_3 : \exists r.B \sqsubseteq Y$$
$$\alpha_4 : Y \sqsubseteq A_1 \quad \alpha_5 : X \sqsubseteq A_2 \quad \alpha_6 : A_1 \sqcap A_2 \sqsubseteq C$$

Clearly, we can add $X \sqsubseteq_{\mathcal{O}} \exists r.A$ (ax_1) and $A \sqsubseteq_{\mathcal{O}} B$ to $\mathcal{C}_O$ (ax_2). By applying CR3 on α_3, ax_1 and ax_2, we obtain $X \sqsubseteq_{\mathcal{O}} Y$ (ax_3). $X \sqsubseteq_{\mathcal{O}} C$ can also be obtained by applying CR1 twice.

III. A Graph-based Formalism for $\mathcal{EL}^+$ Ontologies

In order to give a platform-independent approach, we can find a model to represent $\mathcal{EL}^+$ ontology, based on which the classification can be implemented on different parallel computation platforms. We consider using graph as such model, motivated by that a simple structured graph has been used as a platform-independent model to describe data in several works on parallel data processing, like Neo4j[3] (a graph database) and Pregel [13] (a distributed platform for large scale data processing). Based on a graph model, parallel techniques can be applied. However, some challenges exist when transforming an EL ontology to a graph.

• **Handling complex concepts.** Since EL allows inductively constructing concepts, it is challenging to give a graph to represent a complex concept that is suitable for parallel processing. There have been some works using *conceptual graph* (CG) to represent the description logic fragments with high expressive power (covering the syntax of EL) [6], [4]. However, the transformed graphs are complex and the operations on them cannot lead to an efficient and even automatic procedure [5].

• **Handling subsumption relations.** For the subsumption relations between two primitive concepts of the form $A \sqsubseteq B$, they can be easily transformed to an edge of a graph. This method is adopted in [12]. However, for a subsumption relation involving three concepts or roles such as $A_1 \sqcap A_2 \sqsubseteq B$ or $A \sqsubseteq \exists r.B$, which can be considered as a ternary relation among these three concepts or roles, the techniques given in [12] cannot be applied.

In our work, we use a directed labeled graph to express EL ontologies. In order to transform a subsumption relation involving three concepts or roles to edges, we propose to encode one of the concepts or roles in the subsumption relation to the labels of the edges. For example, the axiom of the form $A_1 \sqcap A_2 \sqsubseteq B$ can be transformed to two edges $e^{\sqcap A_2}(A_1, B)$ and $e^{\sqcap A_1}(A_2, B)$, where $e^{\sqcap A_2}(A_1, B)$ denotes an edge with A_1 and B as its starting and ending vertices respectively, and '$^{\sqcap}A_2$' as its label ($e^{\sqcap A_1}(A_2, B)$ can be explained similarly). In this way, we transform ternary relations among three concepts or roles to binary relations. Formally, we define the transformation function τ as follows:

[3]http://neo4j.com/

$$
\begin{array}{lcll}
\tau(A \sqsubseteq B) & = & \{e^{\sqsubseteq}(A,B)\} & (1)\\
\tau(A_1 \sqcap A_2 \sqsubseteq B) & = & \{e^{\sqcap A_2}(A_1,B), e^{\sqcap A_1}(A_2,B)\} & (2)\\
\tau(A \sqsubseteq \exists r.B) & = & \{e^{+r}(A,B)\} & (3)\\
\tau(\exists r.A \sqsubseteq B) & = & \{e^{-r}(A,B)\} & (4)\\
\tau(r \sqsubseteq s) & = & \{e^{\sqsubseteq}(r,s)\} & (5)\\
\tau(r \circ s \sqsubseteq t) & = & \{e^{\circ s}(r,t)\} & (6)
\end{array}
$$

where the label '$\sqsubseteq$' denotes the standard subsumption relation. $e^{+r}(A,B)$ (resp. $e^{-r}(A,B)$) means that an edge with A, B as its starting and ending vertices, and '$+r$' (resp. '$-r$') as its label with $\exists r$ appearing on the right-hand (resp. left-hand). The edges of the form $e^{\circ s}(r,t)$ can be similarly explained. Based on the above method, one can transform an normalized ontology to a graph according to (1)-(6). In Example 2, a graph (see solid lines) is given by transforming the ontology in Example 1 based on τ. It is easy to check that the transformation terminates in linear time and the size of the constructed graph is also linearly bounded.

A graph-based Syntax. Based on the above analysis, the definition of our graph representation for EL is given in the following.

Definition 1: (**$\mathcal{GEL}$ graph**) Let LV and LE denote all the vertex labels and edge labels as shown in (1)-(6). A $\mathcal{GEL}$ graph is a labeled graph and defined as a pair $\mathcal{G} ::= (\langle V, f\rangle, \langle E, g\rangle)$ where V and E are the vertex set and edge set respectively, and f and g are the functions from V to LV and E to LE respectively.

The semantics of $\mathcal{GEL}$. We define the interpretation $\mathcal{I}_{\mathcal{G}}$ for a $\mathcal{GEL}$ graph $\mathcal{G}$ by a pair $\mathcal{I}_{\mathcal{G}} ::= \langle \triangle_{\mathcal{G}}, \cdot_{\mathcal{G}}\rangle$, where $\triangle_{\mathcal{G}}$ covers the domain, and $\cdot_{\mathcal{G}}$ is a function that maps each concept (resp. role) to a subset of $\triangle_{\mathcal{G}}$ (resp. a binary relation on $\triangle_{\mathcal{G}}$). The satisfaction relation between an interpretation $\mathcal{I}_{\mathcal{G}}$ and a labeled edge can be mapped to the corresponding axiom in $\mathcal{EL}^+$ interpretation. We then say that $\mathcal{I}_{\mathcal{G}}$ is a model of $\mathcal{G}$ if, for each labeled edge in $\mathcal{G}$, $\mathcal{I}_{\mathcal{G}}$ satisfies it. Furthermore, $\mathcal{G} \models e^{\sqsubseteq}(A,B)$ if and only if all models of $\mathcal{G}$ satisfy $e^{\sqsubseteq}(A,B)$. The following theorem shows that $\mathcal{GEL}$ is semantically equal to $\mathcal{EL}^+$.

Theorem 1: [4] Given a normalized $\mathcal{EL}^+$ ontology $\mathcal{O}$, a $\mathcal{GEL}$ graph $\mathcal{G}_O$ is constructed from $\mathcal{O}$ by τ. For any primitive concepts A and B, we have $\mathcal{O} \models A \sqsubseteq B$ iff $\mathcal{G}_O \models e^{\sqsubseteq}(A,B)$.

proof idea. To show the only if direction, we construct a canonical interpretation mod as follows:

$$
\begin{array}{lcl}
\triangle^{mod} & := & LV\\
D^{mod} & := & \{C \in \triangle^{mod} | \mathcal{G}_O \models e^{\sqsubseteq}(C,D)\}\\
r^{mod} & := & \{(C,D) \in \triangle^{mod} \times \triangle^{mod} | \mathcal{G}_O \models e^{+r}(C,D)\}
\end{array}
$$

We can verify that mod is a model of $\mathcal{O}$ based on function τ. Since for any $A \in LV$, $\mathcal{G}_O \models e^{\sqsubseteq}(A,A)$ trivially holds, by the definition of mod, A is in A^{mod}. If $mod \models A \sqsubseteq B$, A is also in B^{mod}. $\mathcal{G}_O \models e^{\sqsubseteq}(A,B)$ follows by the definition of mod. The if direction can be proved in a similar way. □

[4]The proofs of this theorem and other theorems can be found in the technical report.

Classification on $\mathcal{GEL}$ Graph. We now consider the problem of classification on a $\mathcal{GEL}$ graph. A naive method to reason with a graph is to compute the transitive closure of the graph. This method is adopted in [12] and can be easily implemented. However, this simple method does not work in our case, because the rules in Table 1 cannot be transformed to the computation of transitive closure. In order to achieve the sound and complete classification, we can adapt the graph operations to the completion rules for $\mathcal{EL}^+$ given in Table 1. To adapt CR1, when l = 2, we use R2, otherwise we use R1. We then propose R3 (resp. R4, R6 and R7) to adapt CR2 (resp. CR3, CR4 and CR5). R5 is set to check the unsatisfiable concepts. All the rules can be found in Table 2. Since the meaning of the rules in Table 2 is quite intuitive by considering the transformation function τ and the completion rules given in Table 1, we do not explain them in detail. In the following paper, we use $\mathcal{G}^*$ to denote the classification result of $\mathcal{G}$, which is achieved by applying the rules in Table 2 until a fix-point is reached.

Example 2: We perform classification on the graph (see Figure 1) constructed by transforming the ontology given in Example 1, and show some new edges (distinguished by dashed lines) as examples. First, by applying R4, X is connected to Y by an '$\sqsubseteq$' labeled edge. Then R1 is applied and a new edge $e^{\sqsubseteq}(X, A_1)$ is inserted. Next R2 is applied, and $e^{\sqsubseteq}(X, C)$ is inserted.

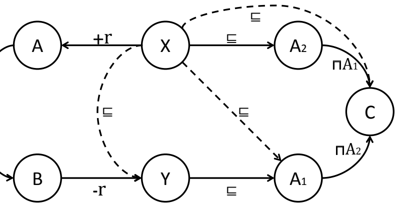

Figure 1. A partial classification result, where solid lines denote the original edges and dashed lines denote the newly generated edges.

Table 2: The completion rules for $\mathcal{GEL}$

R1.	If $e^{\sqsubseteq}(A,B) \in \mathcal{G}^*, e^{\sqsubseteq}(B,C) \in \mathcal{G}$ and $e^{\sqsubseteq}(A,C) \notin \mathcal{G}^*$, then $\mathcal{G}^* = \mathcal{G}^* \cup \{e^{\sqsubseteq}(A,C)\}$.
R2.	If $e^{\sqsubseteq}(X,A_1) \in \mathcal{G}^*, e^{\sqsubseteq}(X,A_2) \in \mathcal{G}^*, e^{\sqcap A_2}(A_1,B) \in \mathcal{G}$ and $e^{\sqsubseteq}(X,B) \notin \mathcal{G}^*$, then $\mathcal{G}^* = \mathcal{G}^* \cup \{e^{\sqsubseteq}(X,B)\}$.
R3.	If $e^{\sqsubseteq}(A,B) \in \mathcal{G}^*, e^{+r}(B,C) \in \mathcal{G}$ and $e^{+r}(A,C) \notin \mathcal{G}^*$, then $\mathcal{G}^* = \mathcal{G}^* \cup \{e^{+r}(A,C)\}$.
R4.	If $e^{+r}(A,B) \in \mathcal{G}^*, e^{\sqsubseteq}(B,C) \in \mathcal{G}^*, e^{-r}(C,D) \in \mathcal{G}$ and $e^{\sqsubseteq}(A,D) \notin \mathcal{G}^*$, then $\mathcal{G}^* = \mathcal{G}^* \cup \{e^{\sqsubseteq}(A,D)\}$.
R5.	If $e^{+r}(A,B) \in \mathcal{G}^*, e^{\sqsubseteq}(B,\bot) \in \mathcal{G}^*$ and $e^{\sqsubseteq}(A,\bot) \notin \mathcal{G}^*$, then $\mathcal{G}^* = \mathcal{G}^* \cup \{e^{\sqsubseteq}(A,\bot)\}$.
R6.	If $e^{+r}(A,B) \in \mathcal{G}^*, e^{\sqsubseteq}(r,s) \in \mathcal{G}$ and $e^{+s}(A,B) \notin \mathcal{G}^*$, then $\mathcal{G}^* = \mathcal{G}^* \cup \{e^{+s}(A,B)\}$.
R7.	If $e^{+r}(A,B) \in \mathcal{G}^*, e^{+s}(B,C) \in \mathcal{G}^*, e^{\circ s}(r,t) \in \mathcal{G}$ and $e^{+t}(A,C) \notin \mathcal{G}^*$, then $\mathcal{G}^* = \mathcal{G}^* \cup \{e^{+t}(A,C)\}$.

The soundness and completeness of the completion rules given in Table 2 can be shown by Theorem 2. Theorem 3 gives the complexity of the classification on a $\mathcal{GEL}$ graph, which is the same as that of $\mathcal{EL}^+$.

Theorem 2: (Soundness and completeness) Given a $\mathcal{GEL}$ graph $\mathcal{G}$, $\mathcal{G}^*$ is the classification result of $\mathcal{G}$. For any edge $e^{\sqsubseteq}(A, B)$, $\mathcal{G} \models e^{\sqsubseteq}(A, B)$ iff $e^{\sqsubseteq}(A, B) \in \mathcal{G}^*$ or $e^{\sqsubseteq}(A, \bot) \in \mathcal{G}^*$.
proof idea. The proof is based on constructing a canonical model as in the proof of Theorem 1 and an induction on the steps of classification. □

Theorem 3: (Complexity) The classification on a given $\mathcal{GEL}$ graph $\mathcal{G}$ is P-complete.
proof idea. This theorem can be proved by reducing the well-known propositional HORNSAT problem to the classification problem. □

IV. The Parallel Classification Abstraction

In this section, we aim to give a platform-independent approach for parallel classification on a $\mathcal{GEL}$ graph. Our approach is inspired by the work given in [7], in which the notion *graph-parallel abstraction* is given to describe general parallel computations by hiding the specific platforms. A graph-parallel abstraction is defined as a pair $\langle \mathcal{G}, Q\rangle$ where $\mathcal{G} := \langle V, E\rangle$ is a directed graph with V and E as its vertex set and edge set respectively, and Q is a *vertex-program* which is executed in parallel on each vertex $v \in V$.

Based on the idea of graph-parallel abstraction, we formalize a parallel classification on a $\mathcal{GEL}$ graph as follows. A *parallel classification abstraction* (PCA) on a $\mathcal{GEL}$ graph is a triple $\langle \mathcal{G}, Q, P\rangle$, where $\mathcal{G}$ is a $\mathcal{GEL}$ graph, Q is a *vertex-program*, and P is a *monitor-program* which collects the global states of $\mathcal{G}$ and checks whether the classification is finished. Since the classification based on the rules in Table 2 is finished on a fix-point, the monitor-program is needed here for checking the termination condition of whether the fix-point is reached. We use $Q(X)$ to denote the vertex-program that is running on the vertex X, and further require that $Q(X)$ and $Q(Y)$ can interact when X and Y are adjacent vertices. Given a PCA $u := \langle \mathcal{G}, Q_u, P_u\rangle$, we define an output function σ such that $\sigma(u) \subseteq \mathcal{G}^*$. We then say that a PCA u is *well defined* if it satisfies the following two conditions:

C1:	For any $e^{\sqsubseteq}(A, B) \in \sigma(u)$ *iff* $e^{\sqsubseteq}(A, B) \in \mathcal{G}^*$, where $u = \langle \mathcal{G}, Q_u, P_u\rangle$, $\mathcal{G}^*$ is the classification result of $\mathcal{G}$.
C2:	Q_u and P_u terminate.

To parallelize classification on a $\mathcal{GEL}$ graph, we give a specific PCA that is instantiated based on a popular parallel model: the *Bulk Synchronous Parallel* (BSP) computing model [23]. One can efficiently design and optimize the parallel algorithms on a BSP model. A computation task based on BSP consists of a sequence of steps, called *superstep*, with synchronization points between each two supersteps. In each superstep, parallel computations and communications among processors are performed.

We now define the PCA based on BSP by $u_{\langle bsp\rangle} := \langle \mathcal{G}, Q_{\langle bsp\rangle}, P_{\langle bsp\rangle}\rangle$. In each superstep of $u_{\langle bsp\rangle}$ the vertex-program $Q_{\langle bsp\rangle}(X)$ on each vertex X exhaustively applies the rules in Table 2 *without handling the new generated edges* in this superstep. After all the vertex-programs terminate, the monitor-program $P_{\langle bsp\rangle}$ starts up a new superstep and notifies all vertices to restart the vertex-programs to handle the new edges generated in the last superstep. If we use $\sigma^i(u_{\langle bsp\rangle})$ to denote the classification results obtained in the i^{th} superstep, the classification procedure can be formulated as follows:

$$\sigma^0(u_{\langle bsp\rangle}) = \mathcal{G},$$
$$\sigma^{i+1}(u_{\langle bsp\rangle}) = \eta(\sigma^i(u_{\langle bsp\rangle})) \cup \sigma^i(u_{\langle bsp\rangle}).$$

where the function η accepts the classification results in the last superstep as the input, and returns the new generated edges after all the vertex-programs terminate. When $\sigma^n(u_{\langle bsp\rangle}) = \sigma^{n+1}(u_{\langle bsp\rangle})$ holds for some n (which also means there is no more new edge generated in any vertex), the monitor-program $P_{\langle bsp\rangle}$ terminates the whole computation. As to the correctness of $u_{\langle bsp\rangle}$, we refer the readers to Theorem 5 in the next section.

As shown above the classification on $\mathcal{GEL}$ graph is P-complete, which means in the worst case it is inherently a serial procedure. We observe from the experiments that, for most real OWL EL ontologies, the efficiency of classification can be significantly improved by parallelism based on the implementation of $u_{\langle bsp\rangle}$. This can be explained by the view of Amdahl's Law[5] that indicates the speedup of a computation task only depends on the fraction of sequential computation part with *the assumption of infinite processors being allocated* (IPA). The work mechanism of $u_{\langle bsp\rangle}$ shows the rule applications are independent of other ones in the same superstep, but rely on the results obtained in previous supersteps. Thus according to the Amdahl's Law, with IPA, the run-time of each superstep is $O(1)$ in theory, while the whole run-time depends on the number of supersteps[6]. Since the number of supersteps is shown to be an important metric to bound the run-time of parallel classification on $\mathcal{GEL}$ graph, we formalize it as follows:

Definition 2: (Classification depth) Given a $\mathcal{GEL}$ graph $\mathcal{G}$, and its corresponding BSP-based PCA $u_{\langle bsp\rangle} := \langle \mathcal{G}, Q_{\langle bsp\rangle}, P_{\langle bsp\rangle}\rangle$, suppose n is the least integer such that $\sigma^n(u_{\langle bsp\rangle}) = \sigma^{n+1}(u_{\langle bsp\rangle})$, we say n is the classification depth of $\mathcal{G}$, denoted by $\mathtt{cd}(\mathcal{G})$.

We can further prove that the run-time of any parallel classification algorithm is lower-bounded by the classification depth (see Theorem 4). This can be proved by showing that for any correct parallel classification algorithm Π, a corresponding BSP-based PCA can always be constructed from Π.

[5]Amdahl argues in [1] that the speedup s of a computation task depends on the fraction of sequential computation part, i.e., $s \leq 1/(f + \frac{1-f}{p})$, where s is the speedup, f is the fraction of sequential computation part and p is the number of processors.

[6]This conclusion also holds with a polynomial number of processors, since the classification on $\mathcal{GEL}$ graph is in P.

Theorem 4: Given a $\mathcal{GEL}$ graph $\mathcal{G}$, and its classification depth $\mathtt{cd}(\mathcal{G})$, for any correct parallel classification algorithm Π, the run-time of Π is lower-bounded by $\mathtt{cd}(\mathcal{G})$.

Based on the notion of classification depth, we say the classification on a $\mathcal{GEL}$ graph $\mathcal{G}$ is solvable in parallel when $\mathcal{G}$ belongs to the following class:

Definition 3: ($\mathcal{GEL}$ parallelly solvable class (PSC)) PSC is a class of $\mathcal{GEL}$ graphs, where it holds for each graph $\mathcal{G}$ $\in$PSC that $\mathtt{cd}(\mathcal{G})$ is poly-logarithmic in $|\mathcal{G}|$. We say the classification on $\mathcal{G}$ $\in$PSC is solvable in parallel.

The classification of a $\mathcal{GEL}$ graph in PSC can be proved to be in NC[7], which is studied by theorists as a complexity class where each problem can be efficiently solved in parallel [18]. In Section VII, we can see that the classification depths of all the evaluated real-world $\mathcal{GEL}$ graphs are far less than the original graph sizes, which indicates that these graphs can be seen as the members in PSC. This in turn motivates us to use parallel technologies to improve the performance of classification.

V. A Refined PCA algorithm

The parallel classification on $\mathcal{GEL}$ graph can be implemented based on $u_{\langle bsp \rangle}$ proposed in the previous section. However a naive $u_{\langle bsp \rangle}$ would cause some problems such as *amount of redundant computations* and *frequent interactions* (see [7]). Thus we give a variant of $u_{\langle bsp \rangle}$ by introducing several optimizations to deal with these issues:

• **Reducing rule applications.** If we allow each vertex-program applying the rules in Table 2 on all edges in one superstep, this would lead to a large amount of redundant rule applications which have been computed in previous supersteps. Thus we restrict that all rule applications can only be triggered by newly inserted edges. For example, when an edge of the form $e^{\sqsubseteq}(A, B)$ is newly inserted, it triggers the corresponding vertex-program to apply R1 - R5. Since the rule R4 involves two newly inserted edges (see Table 2, $e^{+r}(A, B), e^{\sqsubseteq}(B, C) \in \mathcal{G}^*$) as the preconditions, R4 should be applied when any of these two edges is inserted. R7 is processed analogously. This ensures the completeness of final results.

• **Accessing successors only.** As shown in R1 (see Table 2), the second precondition involves an original edge ($e^{\sqsubseteq}(B, C) \in \mathcal{G}$), the edge involved in the first precondition ($e^{\sqsubseteq}(A, B) \in \mathcal{G}^*$) and the newly inserted edge ($e^{\sqsubseteq}(A, C)$) are both the outgoing edges from A. Thus if R1 is applied by the vertex-program $Q(A)$, all the newly inserted edges are generated from the successors of A. This is analogous to R3. Consider Example 2 again, the vertex-program $Q(X)$ for X only needs to access the successors of X, e.g., A_2 (resp. Y), to apply R2 (resp. R1). Thus we optimize these rules by restricting that a vertex can only access its successors. According to [13], [7], this optimization leads to a low frequency of interactions.

Algorithm 1 The algorithm for the monitor-program

Input: $\mathcal{G}$: a $\mathcal{GEL}$ graph
Output:$\mathcal{G}^*$: a global parament storing the classification result of $\mathcal{G}$

```
1: For $\forall A \in LV$ do
2:     $(A.tq).add(e^{\sqsubseteq}(A, A))$;
3:     $Q(A).execute()$;
4: While $\exists A \in LV$ s.t. $A.tq$ is not empty do
5:     start a new superstep;
6: For $\forall A \in LV$ do
7:     $Q(A).terminate()$;
```

Algorithm 2 The algorithm for the vertex-program

Input: $\mathcal{G}$: a $\mathcal{GEL}$ graph
Output:$\mathcal{G}^*$: a global parament storing the classification result of $\mathcal{G}$

```
1: While $A.tq$ is not empty do
2:     $e \leftarrow (A.tq).pop()$;
3:     visit-new-nbr($A, e$);
```

• **Parallelizing role inclusions.** We have observed that the set of edges of the form $e^{\sqsubseteq}(r, s)$ and $e^{\circ s}(r, t)$ is small enough to fit in memory and keeps its size during classification. Thus, for R6 and R7, we assume that every vertex-program $Q(v)$ can access such edges. In a specific implementation, such edges can be parallelized to all processors.

Based on the above optimizations, we give our $PCA := \langle \mathcal{G}, Q, P \rangle$, where P corresponds to Algorithm 1, Q corresponds to Algorithm 2. Algorithm 1 maintains for each vertex (w.l.o.g. A) a tracing queue (denoted by $A.tq$), which records the newly inserted edges w.r.t. A. Initially, Algorithm 1 preprocesses all vertices and pushes all self-pointing edges in their own tracing queues and launches the vertex-program $Q(A)$ (line 3). After that, a 'While' iteration (corresponding to a superstep) is used to check the termination condition that whether no more edge is inserted in all vertices (line 4). If the termination condition is satisfied, the algorithm will terminate all vertex-programs (line 7) and the classification finishes.

In Algorithm 2, the corresponding vertex-program iteratively checks whether a new edge is added in $A.tq$ in the last superstep. For each new edge (w.l.o.g. e), `visit-new-nbr(A,e)` (see Algorithm 3) is called to apply all the rules (line 2 - 28). The function `visit-new-nbr(A,e)` first checks the type of label of e. Then, different rules are applied for different conditions. For example, consider the first 'For' loop, R1 is applied by checking all the outgoing '$\sqsubseteq$' labeled edges of B. When a rule is triggered and a new edge $e^{\varphi}(X, Y)$ for some label φ is generated, the function `insert` will first check $\mathcal{G}^*$, if $e^{\varphi}(X, Y)$ does not appear in $\mathcal{G}^*$, it will insert this edge to $\mathcal{G}^*$ and add it in $X.tq$. For R5, since its preconditions are

[7]This can be proved by constructing a Boolean Circuit with a polynomial-bounded size and a poly-logarithmic depth.

part of those of R4, it can be applied together with R4 (see line 12 and line 19 of Algorithm 3).

Algorithm 3 visit-new-nbr

Input: A: a vertex; e: a new edge inserted to A
Output:$\mathcal{G}^*$: a global parameter storing the classification result

```
1:  If $e = e^{\sqsubseteq}(A,B)$ then
2:      For each $C$, s.t. $e^{\sqsubseteq}(B,C) \in \mathcal{G}$ do      //R1
3:          insert $e^{\sqsubseteq}(A,C)$ to $\mathcal{G}^*$; $A.tq$.push($e^{\sqsubseteq}(A,C)$);
4:      For each $C$ and $A_1$, s.t. $e^{\sqcap A_1}(B,C) \in \mathcal{G}$ and
5:                  $e^{\sqsubseteq}(A,A_1) \in \mathcal{G}^*$ do      //R2
6:          insert $e^{\sqsubseteq}(A,C)$ to $\mathcal{G}^*$; $A.tq$.push($e^{\sqsubseteq}(A,C)$);
7:      For each $C$ and $r$, s.t. $e^{+r}(B,C) \in \mathcal{G}$ do      //R3
8:          insert $e^{+r}(A,C)$ to $\mathcal{G}^*$; $A.tq$.push($e^{+r}(A,C)$);
9:      For each $D$, s.t. $e^{+r}(D,A) \in \mathcal{G}^*$ do
10:         For each $C$ and $r$, s.t. $e^{-r}(B,C) \in \mathcal{G}$ do      //R4
11:             insert $e^{\sqsubseteq}(D,C)$ to $\mathcal{G}^*$; $D.tq$.push($e^{\sqsubseteq}(D,C)$);
12:         If $B$ is $\bot$ then      //R5
13:             insert $e^{\sqsubseteq}(D,\bot)$ to $\mathcal{G}^*$; $D.tq$.push($e^{\sqsubseteq}(D,\bot)$);
14:
15: If $e = e^{+r}(A,B)$ then
16:     For each $C$, s.t. $e^{\sqsubseteq}(B,C) \in \mathcal{G}^*$ do
17:         For each $D$ and $r$ s.t. $e^{-r}(C,D) \in \mathcal{G}$ do      //R4
18:             insert $e^{\sqsubseteq}(A,D)$ to $\mathcal{G}^*$; $A.tq$.push($e^{\sqsubseteq}(A,D)$);
19:         If $C$ is $\bot$ then      //R5
20:             insert $e^{\sqsubseteq}(A,\bot)$ to $\mathcal{G}^*$; $A.tq$.push($e^{\sqsubseteq}(A,\bot)$);
21:     For each $s$ s.t. $e^{\sqsubseteq}(r,s) \in \mathcal{G}$ do      //R6
22:         insert $e^{+s}(A,B)$ to $\mathcal{G}^*$; $A.tq$.push($e^{+s}(A,B)$);
23:     For each $C, s$ and $t$, s.t. $e^{+s}(B,C) \in \mathcal{G}^*$ and
24:                 $e^{\circ s}(r,t) \in \mathcal{G}$ do      //R7
25:         insert $e^{+t}(A,C)$ to $\mathcal{G}^*$; $A.tq$.push($e^{+t}(A,C)$);
26:     For each $C, s$ and $t$, s.t. $e^{+s}(C,A) \in \mathcal{G}^*$ and
27:                 $e^{\circ r}(s,t) \in \mathcal{G}$ do      //R7
28:         insert $e^{+t}(C,B)$ to $\mathcal{G}^*$; $C.tq$.push($e^{+t}(C,B)$);
```

As shown in Algorithm 3, in each 'For' loop, vertex-programs running on different vertices interact with each other in two ways: *accessing the edges of neighbors*, and *inserting new edges*. We have proved that these interactions do not influence the correctness of classification. Formally, Theorem 5 is given to show that our PCA satisfies the two conditions C1 and C2.

Theorem 5: Given a $\mathcal{GEL}$ graph $\mathcal{G}$, and a PCA u based on Algorithm 1 and Algorithm 2, u satisfies both of the conditions **C1** and **C2**.

proof idea. (C2) can be proved easily. (C1) is proved by reduction to absurdity. Assume that when Algorithms 1 and 2 terminate, some rules can still be applied to derive new edges of the form $e^{\sqsubseteq}(A,B)$, then (C1) is not satisfied. Thus this assumption is invalid. Consider the applications of R2 in Example 2, suppose $e^{\sqsubseteq}(X,A_1)$ and $e^{\sqsubseteq}(X,A_2)$ can be inserted by some rules in any order. Our proof shows that no matter in what order, either of $e^{\sqcap A_2}(A_1,C)$ and $e^{\sqcap A_1}(A_2,C)$ will be checked, and R2 will always be applied to insert $e^{\sqsubseteq}(X,C)$. For the application of R4, there are two parts in Algorithm 3 to handle different orders. The case of R7 is similar. □

Table 4: The statistics of ontologies and graphs

ontology	♯concept	♯role	♯axiom	♯vertex	♯edge
GO	49,375	9	116,523	49,384	125,336
Galen	35,989	950	81,493	36,939	83,948
NCI	27,933	70	46,904	28,003	47,360
SCT[a]	464,601	62	805,429	464,663	887,846
SGG	511,511	1,013	921,696	512,524	982,237

[a] abbr. of SNOMED CT.

VI. Implementation and Evaluation

Implementation. We implemented a prototype system, called GEL, based on Java Concurrent and OWL APIs. Our system hides the specific platforms and can run on a multi-core system or a distributed platform by easily changing its configuration. Thus we set two groups of experiments to evaluate the performance of our system on different parallel computation platforms.

Evaluation. In the first group of experiments which is run on a multi-core machine, we use some famous real-world medical ontologies: GO, Galen, NCI, SNOMED CT[8] and the integration of GO, Galen and SNOMED CT (SGG)[9]. All of these ontologies are normalized. The statistics of these ontologies is given in Table 4. We then perform comparison experiments on jCEL[10], ELK and our system (called GEL). The running environment for this experiment is a SuperCloud server with a 128 Gigabyte memory and 12 physical cores, in each core three logic threads can be allocated.

To fairly compare our system with ELK, we set the same number of threads to them. Since jCEL is a serial implementation, it can only run on a single thread. All experimental results are collected in Table 5. The loading time and the transforming time are ignored since these two phases can be quickly. From Table 5, we can see that for small ontologies like NCI, all three systems perform equally well. For large ontologies, our system outperforms jCEL and ELK when more threads are allocated.

Table 5: The runtime of classification (seconds)

	♯thread	GO	Galen	NCI	SCT	SGG
jCEL	1	20.22	33.25	1.43	676.75	776.93
ELK	1	3.96	3.27	2.07	25.42	33.81
	5	4.16	2.42	1.92	15.93	18.83
	10	3.55	3.01	2.19	14.99	18.47
	15	3.94	2.98	2.08	13.86	18.57
	20	3.27	2.87	2.14	15.27	18.79
GEL	1	8.69	5.87	2.58	34.57	43.32
	5	2.84	2.13	1.69	16.55	17.93
	10	2.44	1.88	1.31	15.46	16.42
	15	1.87	1.74	1.30	14.33	14.66
	20	1.53	1.74	1.27	14.58	13.75

[8]All these ontologies are in EL version and available at http://wwwtcs.inf.tu-dresden.de/ meng/toyont.html.

[9]The integration of ontologies is achieved using LogMap, which is available in http://www.cs.ox.ac.uk/isg/projects/LogMap/.

[10]We choose jCEL instead of other serial EL reasoners because jCEL implements CR1-CR5 and its efficiency is comparable with others.

In the second group of experiments, we evaluate the scalability of our system on a distributed platform. To this end, we build a small distributed cluster which consists of 5 nodes and each one of them is an economic Lenovo ThinkCentre machine with a 2 Gigabyte RAM and two physical cores. The experiments are performed on GO and its different extensions[11] which import other medical datasets. We allocate different number of machines to conduct experiments. Before classifying the graph transformed from GO or its extensions, GEL first partitions it into several parts and distributes these parts to different machines based on the strategy called *edge-cut*, which has been studied to distribute large scale graphs [25], [21]. Specifically, GEL uniquely assigns vertices to machines while allowing edges to span across different machines, such that the numbers of edges in different machines are balanced.

Table 6: The run-time of scalability tests (seconds)

			GO	EXT1	EXT2	EXT3
♯vertex			49,375	76,989	109,271	109,393
♯init edge			125,336	206,594	267,547	268,061
♯final edge			1,623,824	7,476,971	8,364,192	8,407,674
	♯	1	12.15/0.0	-/-	-/-	-/-
G	n	2	6.43/18.17	21.46/226.61	38.25/311.64	39.23/323.93
E	o	3	4.74/18.20	22.36/220.96	25.77/318.21	22.34/312.87
L	d	4	2.97/18.53	13.93/169.02	14.46/207.27	11.25/214.26
	e	5	2.37/16.89	6.83/132.80	10.22/145.24	10.43/145.82

The experimental results are collected in Table 6, where each cell contains a pair of numbers whose left (resp. right) part represents the maximum reasoning time (resp. network communication time) among all the machines. This group of experiments actually includes two dimensions: in the first one, the number of machines is fixed and the test ontologies are changed in size; in the other one, the test ontology is unchanged, and the cluster is extended by adding more machines. As wee can see, all the extensions of GO leads to exhausting memory on a single machine. This also happens for the other two in-memory reasoners (jCEL and ELK) in the same machine. When more than one nodes are allocated, our system can handle all the extensions. The experimental results show an approximately linear time trend in both of the first dimension (when running on 5 machines) and the second dimension (on the largest ontology EXT3). On the other hand, the network communication time is shown to dominate the whole runtime, but is trending downward when adding more machines. This indicates that the performance of our system can be further optimized by reducing the network communications, for example utilizing compression and caching techniques.

[11]All the other extension ontologies can be found under ftp://ftp.geneontology.org/go/ontology/extensions/. In our experiment, EXT1 is the integration of GO, anatomy, bfo, EXT2 the integration of EXT1, cell and chemical, EXT3 the integration of EXT2, metazoan and plant.

Classification depth. From Theorem 4, we know that the run-time of any correct parallel classification algorithm with respect to a $\mathcal{GEL}$ graph $\mathcal{G}$ is lower-bounded by the classification depth of $\mathcal{G}$. We investigate the classification depths of the test ontologies: GO, Galen, NCI, SNOMED CT and SGG. For each ontology, we give a statistic diagram as shown in Figure 2, where the abscissa records the number of supersteps, the histograms denote the amount of new inserted edges in each superstep (corresponding to the left ordinate), the curves of different colors denote the in-memory classification time with different numbers of threads allocated (corresponding to the right ordinate). As we can see, the maximum classification depth among these ontologies is 35 (see the diagram of SGG) that is far less than the original graph size (that can be estimated by the number of edges and vertices). This also means that the classification on each of the graph transformed from the test ontologies is solvable in parallel according to Definition 3 (See Section IV). In each superstep, a positive correlation exists between the amount of new inserted edges and classification time, which also shows that the performance can be improved with more threads allocated.

VII. Related Work

There exist several works on parallel reasoning in ontology languages which are different from OWL EL. The authors of [14] propose an approach for parallelizing materialization of Datalog programs. It covers general Datalog programs and is further adapted to OWL languages, such as OWL 2 RL. Deslog [24] is a parallel tableau-based description logic reasoner for $\mathcal{ALC}$, designed for thread-level parallelism. WebPIE [22] is a distributed forward-chaining, and bottom-up reasoner that uses the MapReduce programming model to distribute and parallelize the computation. Marvin [17] is a distributed system for massive processing of RDF data based on a peer-to-peer platform. In [20], the authors present a method to perform scalable nonmonotonic reasoning based on MapReduce. A scalable RDFS reasoning engine is presented in [9] based on distributed hash tables (DHTs), a popular case of peer-to-peer networks. The authors of [8] propose some optimizations over the semantics of RDFS, pD* (OWL Horst) and OWL 2 RL, and evaluate them on a distributed OWL reasoner (SAOR). The above works are all based on specific platforms and cannot be applied to OWL EL.

VIII. Conclusions and Future Work

In this paper, we took the first effort to study a platform-independent approach for parallel reasoning with OWL EL ontologies. To this end, we proposed a novel graph representation for EL ontologies. We then defined the semantics for our graph formalism and gave a group of completion rules for classification. After that, we designed a general parallel classification algorithm which can be implemented on

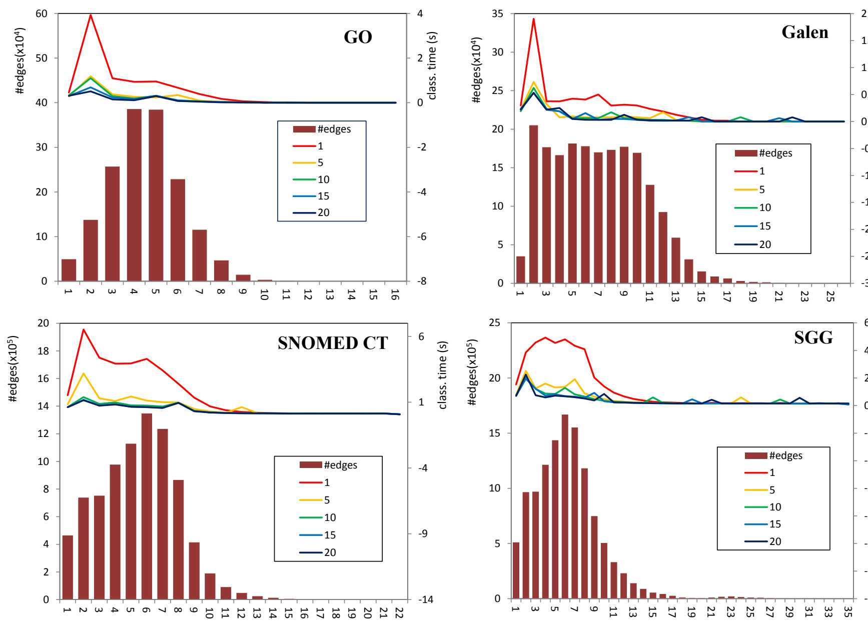

Figure 2: Each ontology of GO, Galen, NCI, SNOMED CT and SGG corresponds to a diagram that consists of two kinds of statis -tics. The histograms denote the amount of new inserted edges in each superstep. The curves denote the in-memory classification time.

different parallel computation platforms. We implemented a prototype system which can take advantage of both of multi-core and distributed computation techniques. The experimental results showed that our prototype system can achieve a tradeoff between efficiency and scalability.

Our approach can be extended to handle ABoxes. We can represent an individual a by a vertex labeled with $\{a\}$, and transform an axiom of the form $A \rightsquigarrow B$ in $\mathcal{EL}^{++}$ to the edge $e^{\rightsquigarrow}(A, B)$. However, it is challenging to design a PCA which can lead to efficient and scalable reasoning on large ABoxes. We also consider verifying our approach on different platforms. Since the refined PCA proposed in Section V is based on BSP, we can implement it on the famous BSP platforms, like Pregel and MapReduce.

References

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proc. of AFIPS*, pages 483–485, 1967.

[2] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ Envelope. In *Proc. of IJCAI*, pages 364–369, 2005.

[3] F. Baader, C. Lutz, and B. Suntisrivaraporn. Efficient reasoning in $\mathcal{EL}^{+}$. In *Proc. of DL*, 2006.

[4] F. Baader, R. Molitor, and S. Tobies. Tractable and decidable fragments of conceptual graphs. In *Proc. of ICCS*, pages 480–493, 1999.

[5] M. Chein and M. Mugnier. *Graph-based Knowledge Representation - Computational Foundations of Conceptual Graphs*. Advanced Information and Knowledge Processing. Springer, 2009.

[6] P. Coupey and C. Faron. Towards correspondence between conceptual graphs and description logics. In *Proc. of ICCS*, pages 165–178, 1998.

[7] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proc. of OSDI*, pages 17–30, 2012.

[8] A. Hogan, J. Z. Pan, A. Polleres, and S. Decker. SAOR: template rule optimisations for distributed reasoning over 1 billion linked data triples. In *Proc. of ISWC*, pages 337–353, 2010.

[9] Z. Kaoudi, I. Miliaraki, and M. Koubarakis. RDFS reasoning and query answering on top of dhts. In *Proc. of ISWC*, pages 499–516, 2008.

[10] Y. Kazakov, M. Krötzsch, and F. Simancik. The incredible ELK - from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. *J. Autom. Reasoning*, pages 1–61, 2014.

[11] F. Lécué, S. Tallevi-Diotallevi, J. Hayes, R. Tucker, V. Bicer, M. L. Sbodio, and P. Tommasi. Smart traffic analytics in the semantic web with STAR-CITY: scenarios, system and lessons learned in dublin city. *J. Web Sem.*, pages 26–33, 2014.

[12] D. Lembo, V. Santarelli, and D. F. Savo. A graph-based approach for classifying OWL 2 QL ontologies. In *Proc. of DL*, pages 747–759, 2013.

[13] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proc. of SIGMOD*, pages 135–146, 2010.

[14] B. Motik, Y. Nenov, R. Piro, and I. Horrocks. Parallel Materialisation of Datalog Programs in Main-Memory RDF Databases. In *Proc. of AAAI*, pages 129–137, 2014.

[15] R. Mutharaju, P. Hitzler, and P. Mateti. Distel: A distributed $\mathcal{EL}^+$ ontology classifier. In *Proc. of SSWS*, pages 17–32, 2013.

[16] R. Mutharaju, F. Maier, and P. Hitzler. A mapreduce algorithm for $\mathcal{EL}^+$. In *Proc. of DL*, 2010.

[17] E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, A. ten Teije, and F. van Harmelen. Marvin: Distributed reasoning over large-scale Semantic Web data. *J. Web Sem.*, 2009.

[18] W. L. R. Raymond Greenlaw, H. James Hoover. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, New York, 1995.

[19] K. A. Spackman, K. E. Campbell, and R. A. Côté. SNOMED RT: a reference terminology for health care. In *Proc. of AMIA*, 1997.

[20] I. Tachmazidis, G. Antoniou, G. Flouris, S. Kotoulas, and L. McCluskey. Large-scale Parallel Stratified Defeasible Reasoning. In *Proc. of ECAI*, pages 738–743, 2012.

[21] C. E. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic. FENNEL: streaming graph partitioning for massive scale graphs. In *Proc. of WSDM*, pages 333–342, 2014.

[22] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, and H. E. Bal. OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples. In *Proc. of ESWC*, pages 213–227, 2010.

[23] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, pages 103–111, 1990.

[24] K. Wu and V. Haarslev. A parallel reasoner for the description logic ALC. In *Proc. of DL*, 2012.

[25] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. Graphx: a resilient distributed graph system on spark. In *Proc. of GRADES*, page 2, 2013.

[26] Z. Zhou, G. Qi, C. Liu, P. Hitzler, and R. Mutharaju. Scale reasoning with fuzzy-$\mathcal{EL}^+$ ontologies based on MapReduce. In *Proc. of WL4AI*, pages 87–93, 2013.

APPENDIX

The Proof of Theorem 1.

Proof. ($\Rightarrow$) To show the only if direction, we construct a canonical interpretation mod as follows::

$$\begin{array}{lll} \triangle^{mod} & := & \{C|C \in V\} \\ D^{mod} & := & \{C \in \triangle^{mod}|\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(C,D)\} \\ r^{mod} & := & \{(C,D) \in \triangle^{mod} \times \triangle^{mod}|\mathcal{G}_\mathcal{O} \models e^{+r}(C,D)\} \end{array}$$

Note that we allow primitive concepts appearing in the domain of the interpretation.

Since for any $A \in LV$, $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(A,A)$ trivially holds, by the definition of mod, A is in A^{mod}. If $mod \models A \sqsubseteq B$, A is also in B^{mod}. $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(A,B)$ follows by the definition of mod. In the following, we show that mod is a model of $\mathcal{O}$, i.e., $mod \models \alpha$ for any $\alpha \in \mathcal{O}$. We organize our proof by distinguishing the axiom forms in $\mathcal{O}$.

(1) $\boldsymbol{A \sqsubseteq B}$. We show that $\forall X \in A^{mod}$, $X \in B^{mod}$ holds. Since $A \sqsubseteq B \in \mathcal{O}$, we have $e^{\sqsubseteq}(A,B) \in E$ by the definition of τ. According to the definition of mod, $A \in B^{mod}$. For any $X \in A^{mod}$, $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(X,A)$ holds. Since $e^{\sqsubseteq}(A,B) \in E$, it trivially follows that $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(A,B)$. By the semantics of $\mathcal{GEL}$, we have $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(X,B)$. Thus $X \in B^{mod}$.

(2) $\boldsymbol{A_1 \sqcap A_2 \sqsubseteq B}$. We show that $\forall X \in A_1^{mod} \cap A_2^{mod}$, $X \in B^{mod}$ holds. According to the definition of mod, for such X, we have $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(X,A_1)$ and $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(X,A_2)$. By τ, we also have $e^{\sqcap A_2}(A_1,B) \in E$, thus $\mathcal{G}_\mathcal{O} \models e^{\sqcap A_2}(A_1,B)$. According to the semantics of $\mathcal{GEL}$, we have $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(X,B)$. According to the definition of mod, we have $X \in B^{mod}$.

(3) $\boldsymbol{A \sqsubseteq \exists r.B}$. We show that $\forall X \in A^{mod}$, there exists a $Y \in B^{mod}$ such that $(X,Y) \in r^{mod}$. If such X exists, according to the definition of mod, we have $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(X,A)$. Since $e^{+r}(A,B)$ is in E by τ, $\mathcal{G}_\mathcal{O} \models e^{+r}(A,B)$ holds. Thus we have $\mathcal{G}_\mathcal{O} \models e^{+r}(X,B)$. Let $Y = B$, we have $Y \in B^{mod}$ such that $(X,Y) \in r^{mod}$. This finishes our proof.

(4) $\boldsymbol{\exists r.A \sqsubseteq B}$. We show that for any $X \in \triangle^{mod}$, if there exists a $Y \in A^{mod}$ such that $(X,Y) \in r^{mod}$, then X has to be in B^{mod}. If such X and Y exist, we have, from the definition of mod, $\mathcal{G}_\mathcal{O} \models E^{\sqsubseteq}(Y,A)$ and $\mathcal{G}_\mathcal{O} \models e^{+r}(X,Y)$. Since $e^{-r}(A,B)$ is in E by τ, $\mathcal{G}_\mathcal{O} \models e^{-r}(A,B)$ holds. By the semantics of $\mathcal{GEL}$, we have $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(X,B)$, and $X \in B^{mod}$ follows.

(5) $\boldsymbol{r \sqsubseteq s}$. We have to prove that for any $(X,Y) \in r^{mod}$, $(X,Y) \in s^{mod}$ holds. If such pair $(X,Y) \in r^{mod}$ exists, according to the definition of mod, we have $\mathcal{G}_\mathcal{O} \models e^{+r}(X,Y)$. Since $e^{\sqsubseteq}(r,s)$ is in E by τ, $\mathcal{G}_\mathcal{O} \models e^{\sqsubseteq}(r,s)$ holds. Thus we have $\mathcal{G}_\mathcal{O} \models e^{+s}(X,Y)$, and $(X,Y) \in s^{mod}$ has also been proved.

(6) $\boldsymbol{r \circ s \sqsubseteq t}$. We have to prove that for any $(X,Y) \in r^{mod}$ and $(Y,Z) \in s^{mod}$, $(X,Z) \in t^{mod}$ holds. If such pairs $(X,Y) \in r^{mod}$ and $(Y,Z) \in s^{mod}$ exist, according to the definition of mod, we have $\mathcal{G}_\mathcal{O} \models e^{+r}(X,Y)$ and $\mathcal{G}_\mathcal{O} \models e^{+s}(Y,Z)$. Since $e^{\circ s}(r,t)$ is in e by τ, $\mathcal{G}_\mathcal{O} \models e^{\circ s}(r,t)$ holds. Thus, by the semantics of $\mathcal{GEL}$, we have $\mathcal{G}_\mathcal{O} \models e^{+t}(X,Z)$, and $(X,Z) \in t^{mod}$ has also been proved.

($\Leftarrow$) We now prove if direction. Similarly, we construct a model $mod^{'}$ for $\mathcal{G}_\mathcal{O}$ as follows:

$$\begin{array}{lll} \triangle^{mod^{'}}_{\mathcal{G}_\mathcal{O}} & := & \{C|C \in CN\} \\ D^{mod^{'}} & := & \{C \in \triangle^{mod^{'}}_{\mathcal{G}_\mathcal{O}}|\mathcal{O} \models C \sqsubseteq D\} \\ r^{mod^{'}} & := & \{(C,D) \in \triangle^{mod^{'}}_{\mathcal{G}_\mathcal{O}} \times \triangle^{mod^{'}}_{\mathcal{G}_\mathcal{O}}| \\ & & \mathcal{O} \models C \sqsubseteq \exists r.D\} \end{array}$$

Since for any $A \in CN$, $\mathcal{O} \models A \sqsubseteq A$ holds. By the definition of $mod^{'}$, A is in $A^{mod^{'}}$. If $mod^{'} \models e^{\sqsubseteq}(A,B)$, A is also in $B^{mod^{'}}$. Then $\mathcal{O} \models A \sqsubseteq B$ follows by the definition of $mod^{'}$. Thus we are left to prove $mod^{'}$ is a model of $\mathcal{G}_\mathcal{O}$, i.e., $mod^{'} \models \mathcal{G}_\mathcal{O}$. We distinguish the following cases according to the different forms of edges appearing in $\mathcal{G}_{\mathcal{O}}$:

(1) $e^{\sqsubseteq}(A,B)$. We show that $\forall X \in A^{mod'}$, $X \in B^{mod'}$ holds. We have $A \sqsubseteq B \in \mathcal{O}$ since $e^{\sqsubseteq}(A,B)$ has to be got from it by τ. According to the definition of mod', $A \in B^{mod'}$. For any $X \in A^{mod'}$, $\mathcal{O} \models X \sqsubseteq A$ holds. Since $\mathcal{O} \models A \sqsubseteq B$, by the semantics of $\mathcal{EL}^+$, $\mathcal{O} \models X \sqsubseteq B$ also holds. Thus $X \in B^{mod'}$.

(2) $e^{+r}(A,B)$. We show that $\forall X \in A^{mod'}$, there exists a $Y \in B^{mod'}$ such that $(X,Y) \in r^{mod'}$. If such X exists, according to the definition of mod', we have $\mathcal{O} \models X \sqsubseteq A$. Since $e^{+r}(A,B)$ has to be transformed from $A \sqsubseteq \exists r.B$ by τ. Then $A \sqsubseteq \exists r.B \in \mathcal{O}$ holds, which also implies $\mathcal{O} \models A \sqsubseteq \exists r.B$. Thus we have $\mathcal{O} \models X \sqsubseteq \exists r.B$, which means such Y exists.

(3) $e^{-r}(A,B)$. We show that for any $X \in \triangle^{mod'}$, if there exists a $Y \in A^{mod'}$ such that $(X,Y) \in r^{mod'}$, then X has to be in $B^{mod'}$. If such X and Y exist, we have, from the definition of mod', $\mathcal{O} \models Y \sqsubseteq A$ and $\mathcal{O} \models X \sqsubseteq \exists r.Y$. Since $e^{-r}(A,B)$ has to be transformed from $\exists r.A \sqsubseteq B$ by τ, we have $r.A \sqsubseteq B \in \mathcal{O}$ and $\mathcal{O} \models r.A \sqsubseteq B$. By the semantics of $\mathcal{EL}^+$, we have $\mathcal{O} \models X \sqsubseteq B$, and $X \in B^{mod'}$ follows.

(4) $e^{\sqcap A_2}(A_1,B)$. We show that $\forall X \in A_1^{mod'} \cap A_2^{mod'}$, $X \in B^{mod'}$ holds. According to the definition of mod', for such X, we have $\mathcal{O} \models X \sqsubseteq A_1$ and $\mathcal{O} \models X \sqsubseteq A_2$. Since $e^{\sqcap A_2}(A_1,B)$ has to be transformed from $A_1 \sqcap A_2 \sqsubseteq B$ by τ, we have $A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{O}$ and $\mathcal{O} \models A_1 \sqcap A_2 \sqsubseteq B$. Further $\mathcal{O} \models X \sqsubseteq B$ also holds and $X \in B^{mod'}$ has been proved.

(5) $e^{\sqsubseteq}(r,s)$. We show that for any $(X,Y) \in r^{mod'}$, $(X,Y) \in s^{mod'}$ holds. If such pair $(X,Y) \in r^{mod'}$ exists, according to the definition of mod', we have $\mathcal{O} \models X \sqsubseteq \exists r.Y$. Since $e^{\sqsubseteq}(r,s)$ has to be transformed from $r \sqsubseteq s$ by τ, we have $r \sqsubseteq s \in \mathcal{O}$ and $\mathcal{O} \models r \sqsubseteq s$. Thus we have$\mathcal{O} \models X \sqsubseteq \exists s.Y$, and $(X,Y) \in s^{mod'}$ has also been proved.

(6) $e^{\circ s}(r,t)$. We show that for any $(X,Y) \in r^{mod'}$ and $(Y,Z) \in s^{mod'}$, $(X,Z) \in t^{mod'}$ holds. If such pairs $(X,Y) \in r^{mod'}$ and $(Y,Z) \in s^{mod'}$ exist, according to the definition of mod', we have $\mathcal{O} \models X \sqsubseteq \exists r.Y$ and $\mathcal{O} \models Y \sqsubseteq \exists s.Z$. Since $e^{\circ s}(r,t)$ has to be transformed from $r \circ s \sqsubseteq t$ by τ, we have $r \circ s \sqsubseteq t \in \mathcal{O}$ and $\mathcal{O} \models r \circ s \sqsubseteq t$. Thus, by the semantics of $\mathcal{EL}^+$, we have $\mathcal{O} \models X \sqsubseteq \exists t.Z$, and $(X,Z) \in t^{mod'}$ has also been proved. □

The Proof of Theorem 2.

We first give a recursive definition of the classification on a $\mathcal{GEL}$ graph. We introduce a function $\gamma(\mathcal{G})$ which maps $\mathcal{G}$ to another new graph such that $\gamma(\mathcal{G})$ consists of new edges generated by exhaustively applying R1 - R7 to axioms appearing in $\mathcal{G}$ only, that is, function γ does not consider applications of rules on newly generated edges. Based on $\gamma(\mathcal{G})$, the classification is formally defined as follows:

Definition 4: (Classification on $\mathcal{GEL}$) Given a $\mathcal{GEL}$ graph $\mathcal{G}$, a group of functions $\gamma^i(\mathcal{G})$ $(i \geq 0)$ are defined inductively as follows:

$$\gamma^0(\mathcal{G}) = \mathcal{G},$$
$$\gamma^{i+1}(\mathcal{G}) = \gamma(\gamma^i(\mathcal{G})) \cup \gamma^i(\mathcal{G}).$$

When $\gamma^{n+1}(\mathcal{G}) = \gamma^n(\mathcal{G})$ for some n, we get a fixed point and we use $\gamma_C(\mathcal{G})(= \gamma^n(\mathcal{G}))$ to denote the classification result of $\mathcal{G}$. The procedure to compute $\gamma_C(\mathcal{G})$ is called *classification*.

Based on γ, Theorem 2 can be proved as follows.

Proof.(**Soundness**) We prove by induction on $\gamma^n(\mathcal{G})$.

The basic step: If an edge $e \in \mathcal{G}$, obviously $\mathcal{G} \models e$, and $e \in \gamma^0(\mathcal{G})$ holds.

The inductive step: Suppose $\mathcal{G} \models e$, for all $e \in \gamma^k$. We show that $\mathcal{G} \models e$, for all $e \in \gamma^{k+1}$. Suppose e is an arbitrary edge in $\gamma^{k+1} \backslash \gamma^k$, we distinguish the following cases according the different rules applied to generate e:

R1. Suppose there exist two edges: $e^{\sqsubseteq}(A,B) \in \gamma^i(\mathcal{G})$ and $e^{\sqsubseteq}(B,C) \in \gamma^j(\mathcal{G})$, where $0 \leq i,j \leq k$. By induction hypothesis, we have $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ and $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, for any model $\mathcal{I}$ of $\mathcal{G}$. Therefore, $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, it follows that $\mathcal{G} \models e$.

R2. Suppose there exist three edges: $e^{\sqsubseteq}(X,A_1) \in \gamma^{i_1}(\mathcal{G})$, $e^{\sqsubseteq}(X,A_2) \in \gamma^{i_2}(\mathcal{G})$ and $e^{\sqcap A_2}(A_1,B) \in \gamma^j(\mathcal{G})$, where $0 \leq i_1, i_2, j \leq k$. By induction hypothesis, we have $X^{\mathcal{I}} \subseteq A_1^{\mathcal{I}}$, $X^{\mathcal{I}} \subseteq A_2^{\mathcal{I}}$ and $A_1^{\mathcal{I}} \cap A_2^{\mathcal{I}} \subseteq B^{\mathcal{I}}$, for any model $\mathcal{I}$ of $\mathcal{G}$. Therefore, we have $X^{\mathcal{I}} \subseteq B^{\mathcal{I}}$, it follows that $\mathcal{G} \models e$.

R3. Suppose there exist two edges: $e^{\sqsubseteq}(A,B) \in \gamma^i(\mathcal{G})$ and $e^{+r}(B,C) \in \gamma^j(\mathcal{G})$, where $0 \leq i,j \leq k$. By induction hypothesis, we have $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ and for any $x \in B^{\mathcal{I}}$, there exists a $y \in C^{\mathcal{I}}$ such that $(x,y) \in r^{\mathcal{I}}$, for any model $\mathcal{I}$ of $\mathcal{G}$. Since all $z \in A^{\mathcal{I}}$ also belongs to $B^{\mathcal{I}}$, there exists a $w \in C^{\mathcal{I}}$ such that $(z,w) \in r^{\mathcal{I}}$, it follows that $\mathcal{G} \models e$.

R4. Suppose there exist three edges: $e^{+r}(A,B) \in \gamma^{i_1}(\mathcal{G})$, $e^{\sqsubseteq}(B,C) \in \gamma^{i_2}(\mathcal{G})$ and $e^{-r}(C,D) \in \gamma^j(\mathcal{G})$, where $0 \leq i_1, i_2, j \leq k$. By induction hypothesis, we have $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ and $\forall x \in A^{\mathcal{I}}$, there exists a $y \in B^{\mathcal{I}}$ such that $(x,y) \in r^{\mathcal{I}}$, for any model $\mathcal{I}$ of $\mathcal{G}$. Since $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, such y also belongs to $C^{\mathcal{I}}$. $e^{-r}(C,D) \in \gamma^j(\mathcal{G})$ means that for any z, when there exists a t such that $(z,t) \in r^{\mathcal{I}}$, z should be in $D^{\mathcal{I}}$. Based on this analysis, $x \in D^{\mathcal{I}}$ holds. It follows that $\mathcal{G} \models e$.

R5. Suppose there exist two edges: $e^{+r}(A,B) \in \gamma^i(\mathcal{G})$ and $e^{\sqsubseteq}(B,\bot) \in \gamma^j(\mathcal{G})$, where $0 \leq i,j \leq k$. By induction hypothesis, for any model $\mathcal{I}$ of $\mathcal{G}$, we have 1) $B^{\mathcal{I}} \subseteq \bot^{\mathcal{I}}$ and 2) $\forall x \in A^{\mathcal{I}}$, there exists a $y \in B^{\mathcal{I}}$ such that $(x,y) \in r^{\mathcal{I}}$. Since $B^{\mathcal{I}} \subseteq \bot^{\mathcal{I}}$, $B^{\mathcal{I}}$ has to be $\emptyset$. Thus no such x exists in 2), and $A^{\mathcal{I}} = \emptyset$ follows, which means no model satisfies A. It follows that $\mathcal{G} \models e$.

R6. Suppose there exist two edges: $e^{+r}(A,B) \in \gamma^i(\mathcal{G})$ and $e^{\sqsubseteq}(r,s) \in \gamma^j(\mathcal{G})$, where $0 \leq i,j \leq k$ (clearly, here $j = 0$ since no more such edge of the form $e^{\sqsubseteq}(r,s)$ is generated). By induction hypothesis, for any model $\mathcal{I}$ of $\mathcal{G}$, we have 1)

$\forall x \in A^{\mathcal{I}}$, there exists a $y \in B^{\mathcal{I}}$ such that $(x,y) \in r^{\mathcal{I}}$, 2) $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$, specifically $\forall (v,w) \in r^{\mathcal{I}}, (v,w) \in s^{\mathcal{I}}$. Obviously, all the pairs (x,y) in 1) also belong to $s^{\mathcal{I}}$, it follows that $\mathcal{G} \models e$.

R7. Suppose there exist three edges: $e^{+r}(A,B) \in \gamma^i(\mathcal{G})$, $e^{+s}(B,C) \in \gamma^j(\mathcal{G})$ and $e^{\circ s}(r,t) \in \gamma^k(\mathcal{G})$, where $0 \leq i,j,k \leq k$. By induction hypothesis, for any model $\mathcal{I}$ of $\mathcal{G}$, we have 1) $\forall x \in A^{\mathcal{I}}$, there exists a $y \in B^{\mathcal{I}}$ such that $(x,y) \in r^{\mathcal{I}}$, 2) $\forall y \in B^{\mathcal{I}}$, there exists a $z \in C^{\mathcal{I}}$ such that $(y,z) \in r^{\mathcal{I}}$, 3) $r^{\mathcal{I}} \circ s^{\mathcal{I}} \subseteq t^{\mathcal{I}}$, specifically, for any two pairs $(x,y) \in r^{\mathcal{I}}$ and $(y,z) \in s^{\mathcal{I}}$, $(x,z) \in t^{\mathcal{I}}$ holds. From 1) and 2), we have $\forall x \in A^{\mathcal{I}}$, there exist a $y \in B^{\mathcal{I}}$ and a $z \in C^{\mathcal{I}}$ such that $(x,y) \in r^{\mathcal{I}}$ and $(y,z) \in s^{\mathcal{I}}$ hold. With considering about 3), we further have $\forall x \in A^{\mathcal{I}}$, a z exists such that $(x,z) \in t^{\mathcal{I}}$. It follows that $\mathcal{G} \models e$.

(**Completeness**) We prove the completeness based on the canonical interpretation $citp$ of $\mathcal{G}$ which is defined as follows.

Definition 5: (Canonical Interpretation)
Given a $\mathcal{GEL}$ graph $\mathcal{G}$, $\gamma_C(\mathcal{G})$ is the classification result of $\mathcal{G}$. The canonical interpretation $citp$ is defined by

$$\begin{array}{lll} \triangle & := & \{C | C \in LV\} \\ D^{citp} & := & \{C \in \triangle | e^{\sqsubseteq}(C,D) \in \gamma_C(\mathcal{G})\} \\ & & \text{or } \emptyset \text{ when } e^{\sqsubseteq}(D,\perp) \in \gamma_C(\mathcal{G}) \\ r^{citp} & := & \{(C,D) \in \triangle \times \triangle | e^{\sqsubseteq}(C,E) \in \gamma_C(\mathcal{G}) \\ & & \wedge e^{+r}(E,D) \in \gamma_C(\mathcal{G}))\} \end{array}$$

If Lemma 1 and Lemma 2, which follow this part, are firstly proved, we simply have the following implications:

$$\mathcal{G} \models e^{\sqsubseteq}(A,B) \Rightarrow citp \models e^{\sqsubseteq}(A,B) \Rightarrow e^{\sqsubseteq}(A,B) \in \gamma_C(\mathcal{G}) \\ \text{or } e^{\sqsubseteq}(A,\perp) \in \gamma_C(\mathcal{G})$$

The theorem is then proved. Thus we are left to prove Lemma 1 and Lemma 2.

Lemma 1: Given a $\mathcal{GEL}$ graph $\mathcal{G}$, for any edge $e^{\sqsubseteq}(A,B)$, $citp \models e^{\sqsubseteq}(A,B)$ implies $e^{\sqsubseteq}(A,B) \in \gamma_C(\mathcal{G})$.

Proof. We investigate all the forms of edges as follows:

(1) $e^{\sqsubseteq}(A,A)$. From Algorithm 1, we have $e^{\sqsubseteq}(A,A) \in \gamma_C(\mathcal{G})$. By the definition of $citp$, $A \in A^{citp}$ if $e^{\sqsubseteq}(A,\perp) \notin \gamma_C(\mathcal{G})$, otherwise, $A^{citp} = \emptyset$.

(2) $e^{\sqsubseteq}(A,B)$. 1) If $A^{citp} = \emptyset$, then $e^{\sqsubseteq}(A,\perp) \in \gamma_C(\mathcal{G})$; 2) If $A^{citp} \neq \emptyset$, from (1), we have $A \in A^{citp}$, then $A \in B^{citp}$. By the definition of $citp$, $e^{\sqsubseteq}(A,B) \in \gamma_C(\mathcal{G})$.

(3) $e^{\sqsubseteq}(A,\perp)$. If $citp$ satisfies $e^{\sqsubseteq}(A,\perp)$, then $A^{citp} = \emptyset$. By the definition of $citp$, $e^{\sqsubseteq}(A,\perp) \in \gamma_C(\mathcal{G})$.

Lemma 2: Given a $\mathcal{GEL}$ graph $\mathcal{G}$, the canonical interpretation $citp$ is a model of $\mathcal{G}$, i.e., $citp \models \mathcal{G}$.

Proof. We distinguish the cases on different edges:

(1) $e^{\sqsubseteq}(v_1,v_2)$. In the case that v_1 and v_2 are concepts, we should prove $X \in v_2^{citp}$ for each $X \in v_1^{citp}$. $X \in v_1^{citp}$ implies $e^{\sqsubseteq}(X,v_1) \in \gamma_C(\mathcal{G})$ by the definition of $citp$. From $e^{\sqsubseteq}(X,v_1), e^{\sqsubseteq}(v_1,v_2)$ and R1, we have $e^{\sqsubseteq}(X,v_2) \in \gamma_C(\mathcal{G})$, by the definition of $citp$, $X \in v_2^{citp}$ holds. In another case that v_1 and v_2 are roles, we should prove $\forall (X,Y) \in v_1^{citp}$, $(X,Y) \in v_2^{citp}$ holds. From $(X,Y) \in v_1^{citp}$ and the definition of $citp$ and $e^{\sqsubseteq}(X,X)$, we have $e^{+v_1}(X,Y) \in \gamma_C(\mathcal{G})$. Further, based on R7 and $e^{\sqsubseteq}(v_1,v_2)$, we have $e^{+v_2}(X,Y) \in \gamma_C(\mathcal{G})$. By the definition of $citp$ and $e^{\sqsubseteq}(X,X)$, $(X,Y) \in v_2^{citp}$ holds.

(2) $e^{+r}(v_1,v_2)$. We should prove $\forall X \in v_1^{citp}$, these exists $Y \in v_2^{citp}$ such that $(X,Y) \in r^{citp}$. From $X \in v_1^{citp}$ and the definition of $citp$, we have $e^{\sqsubseteq}(X,v_1) \in \gamma_C(\mathcal{G})$. Further, based on R3 and $e^{+r}(v_1,v_2)$, $e^{+r}(X,v_2) \in \gamma_C(\mathcal{G})$ holds. By the definition of $citp$, $(X,v_2) \in r^{citp}$, since $v_2 \in v_2^{citp}$ holds from the proof of Lemma 1, this case is also proved.

(3) $e^{-r}(v_1,v_2)$. We should prove for any $X \in dom$, if there exists a $Y \in v_1^{citp}$ such that $(X,Y) \in r^{citp}$, then $X \in v_2^{citp}$. From $Y \in v_1^{citp}$, $(X,Y) \in r^{citp} \in \gamma_C(\mathcal{G})$ and the definition of $citp$, we have $e^{\sqsubseteq}(Y,v_1)$, $e^{+r}(X,Y) \in \gamma_C(\mathcal{G})$. Further, based on $e^{-r}(v_1,v_2)$, R4, we have $e^{\sqsubseteq}(X,v_2) \in \gamma_C(\mathcal{G})$. By the definition of $citp$, $X \in v_2^{citp}$ holds.

(4) $e^{\sqcap A}(v_1,v_2)$. We should prove $\forall X \in v_1^{citp} \cap A^{citp}$, $X \in v_2^{citp}$ holds. From $X \in v_1^{citp} \cap A^{citp}$ and the definition of $citp$, we have $e^{\sqsubseteq}(X,v_1), e^{\sqsubseteq}(X,A) \in \gamma_C(\mathcal{G})$. Further, based on $e^{\sqcap A}(v_1,v_2)$, R2, $e^{\sqsubseteq}(X,v_2)$ holds. By the definition of $citp$, $X \in v_2^{citp}$ holds too.

(5) $e^{\circ v_2}(v_1,v_3)$. We should prove $\forall (X,Y) \in v_1^{citp}$ and $(Y,Z) \in v_2^{citp}$, $(X,Z) \in v_3^{citp}$ holds. From $(X,Y) \in v_1^{citp}$ and $(Y,Z) \in v_2^{citp}$ and the definition of $citp$, we have $e^{+v_1}(X,Y)$, $e^{+v_2}(Y,Z) \in \gamma_C(\mathcal{G})$ (notice that $e^{\sqsubseteq}(X,X)$ and $e^{\sqsubseteq}(Y,Y)$ hold). Further, based on $e^{\circ v_2}(v_1,v_3)$ and R7, we have $e^{+v_3}(X,Z) \in \gamma_C(\mathcal{G})$. By the definition of $citp$, $(X,Z) \in v_3^{citp}$ holds. □

The Proof of Theorem 3.

Proof. **(Membership)** The classification is performed by applying all the rules in Table 3 to add new edges and terminates when no more edges can be added. Given a $\mathcal{GEL}$ graph $\mathcal{G}$, the maximum number of edges which may be added during classification is:

$$(|V_C| + |V_R|)^2 \cdot \sum_{\varphi}(\sharp e_{\varphi}), \text{ where } \varphi \in \{\sqsubseteq, +r, -r, {}^{\sqcap}A\}.$$

where $\sharp e_{\varphi}$ denotes the number of different types of edges labeled by φ. Specifically, since the label ${}^{\sqcap}A$ corresponds to the concept A, $\sharp e_{\sqcap A}$ is up to $|V_C|$, while $\sharp e_{+r}$, $\sharp e_{-r}$ are up to $|V_R|$. Thus we have the maximum number of added edges is up to $(|V_C| + |V_R|)^2 \cdot (|V_C| + 2|V_R| + 1)$. On the other hand, it is easy to check that performing each rule can be done in polynomial time. Thus, the $\mathcal{GEL}$ classification is in P.

(Hardness) We now reduce the propositional HORNSAT problem (check the satisfiability of a propositional horn formula), that is already proved to be P-complete, to the $\mathcal{GEL}$ classification problem, and in this way to show the $\mathcal{GEL}$ classification is P-complete.

Given a propositional horn formula ϕ, which is a conjunction of several horn clauses (each clause is a disjunction with at most one positive literal). We obtain a $\mathcal{GEL}$ graph $\mathcal{G}_\phi$ based on ϕ by performing the construction $f(\phi)$. $f(\phi)$ is defined as follows:

Initially, introduce a vertex V_σ and let $\mathcal{G}_\phi = \{e^{\sqsubseteq}(V_\sigma, V_\sigma)\}$. For each horn clause hc in ϕ, construction performs on the cases of different forms of hc:

(1) l_0 (a single positive literal): introduce a vertex V_{l_0} and let $\mathcal{G}_\phi = \mathcal{G}_\phi \cup \{e^{\sqsubseteq}(V_\sigma, V_{l_0})\}$.

(2) $l_0 \vee \neg l_1$ (one positive and one negative literals): introduce two vertexes V_{l_0} and V_{l_1}, let $\mathcal{G}_\phi = \mathcal{G}_\phi \cup \{e^{\sqsubseteq}(V_{l_1}, V_{l_0})\}$.

(3) $l_0 \vee \neg l_1 \vee \neg l_2$ (one positive and two negative literals): introduce three vertexes V_{l_0}, V_{l_1} and V_{l_2}, let $\mathcal{G}_\phi = \mathcal{G}_\phi \cup \{e^{\sqcap V_{l_2}}(V_{l_1}, V_{l_0}), e^{\sqcap V_{l_1}}(V_{l_2}, V_{l_0})\}$.

(4) $\neg l_1$ (only one negative literal): introduce a vertex V_{l_1} and let $\mathcal{G}_\phi = \mathcal{G}_\phi \cup \{e^{\sqsubseteq}(V_{l_1}, V_\perp)\}$.

(5) $\neg l_1 \vee \neg l_2$ (only two negative literals): introduce two vertexes V_{l_1}, V_{l_2}, and let $\mathcal{G}_\phi = \mathcal{G}_\phi \cup \{e^{\sqcap V_{l_2}}(V_{l_1}, V_\perp), e^{\sqcap V_{l_1}}(V_{l_2}, V_\perp)\}$.

(6) $l_0 \vee \neg l_1 \vee ... \vee \neg l_j, j \geq 3$ (one positive and more than three negative literals): first, introduce j new literals $l_1^{'}, ..., l_j^{'}$ and transform hc to the following formula φ:

$$\varphi: \quad (\bigwedge_{k=1}^{j-1} l_k^{'} \vee \neg l_k \wedge \neg l_{k-1}^{'}) \wedge (l_1^{'} \vee \neg l_1) \wedge (l_0 \vee \neg l_j^{'})$$

If hc contains only negative literals (without l_0), we just change the rightmost clause $(l_0 \vee \neg l_j^{'})$ to $\neg l_j^{'}$, then for each sub horn clause in φ, (1)-(5) are accordingly performed.

Next, we show that based on the classification on $\mathcal{G}_\phi$, the satisfiability of ϕ can be checked. Suppose $\gamma^n(\mathcal{G}_\phi)$ is the classification result, we give a truth value assignment $\alpha(\phi)$ for ϕ based on the following condition:

- For any literal l in ϕ, if $e^{\sqsubseteq}(V_\sigma, V_l) \in \gamma^n(\mathcal{G}_\phi)$, assign TRUE to l; otherwise assign FALSE to l.

It is easy to check the correctness of $\alpha(\phi)$ in the case that ϕ is satisfiable. The unsatisfiability cases are justified as follows:

- For any literal l in ϕ, if $e^{\sqsubseteq}(V_\sigma, V_l) \in \gamma^n(\mathcal{G}_\phi)$ and $e^{\sqsubseteq}(V_l, V_\perp) \in \gamma^n(\mathcal{G}_\phi)$, ϕ is unsatisfiable.
- For any two literals l_1 and l_2 in ϕ, if $e^{\sqcap V_{l_2}}(V_{l_1}, V_\perp) \in \gamma^n(\mathcal{G}_\phi)$ $e^{\sqsubseteq}(V_\sigma, V_{l_1}) \in \gamma^n(\mathcal{G}_\phi)$ and $e^{\sqsubseteq}(V_\sigma, V_{l_2}) \in \gamma^n(\mathcal{G}_\phi)$, ϕ is unsatisfiable.

Furthermore, it is easy to check that $f(\phi)$ is a log-space construction. □

The Proof of Theorem 4.

proof. Let $\mathbb{U}_{\langle bsp \rangle}$ be the class of all BSP-based PCAs for $\mathcal{GEL}$. To build the proof of this theorem, we investigate any correct parallel classification algorithm, w.o.l.g. Π, and show that we can construct a corresponding $u_{\langle bsp \rangle} \in \mathbb{U}_{\langle bsp \rangle}$ from Π such that the run-time of Π is lower-bounded by the number of supersteps of $u_{\langle bsp \rangle}$ (result$_1$). We than show that the run-time of any member in $\mathbb{U}_{\langle bsp \rangle}$ is lower-bounded by $\mathtt{cd}(\mathcal{G})$ (result$_2$). Theorem 4 obviously follows with result$_1$ and result$_2$.

(result$_1$) For any any correct parallel classification algorithm Π, we attach time stamps to each edge generated during the classification. Let l_e be the time stamp of the edge e. Initially the original edges in $\mathcal{G}$ are attached with the integer 0, and for any new inserted edge e by applying some rule from e_1 and e_2, its time stamp is formulated as $l_e = max(l_{e_1}, l_{e_2}) + 1$. When a new stamp $l_e^{'}$ is prepared to attached to e which already has a stamp l_e, we select the minimal one: $l_e = min(l_e^{'}, l_e)$. After classification, each edge in $\mathcal{G}^*$ has a unique stamp. We choose an edge e^* that has the maximal stamp $l_{e^*} = max \bigcup_{e \in \mathcal{G}^*} \{l_e\}$. Then we construct a BSP-based PCA $u_{\langle bsp \rangle}$ from Π based on l_{e^*}. Specifically, $u_{\langle bsp \rangle}$ consists of l_{e^*} supersteps, where in the i^{th} superstep, the edges with i as their stamps are obtained. It is easy to check the correctness of $u_{\langle bsp \rangle}$. On the other hand, e^* is attached by l_{e^*}, which means there exist l_{e^*} rule applications $\beta_1, ..., \beta_{l_{e^*}}$ such that each two of them have a partial order (i.e., one depends on the other). Assume one rule application occupies one time unit (O(1)), e^* should be obtained after l_{e^*} time units. Thus the run-time of u is lower-bounded by l_{e^*}, formally, $T(\Pi) = \Omega(l_{e^*})$.

(result$_2$) According to the definition of BSP-based PCA, each $u_{\langle bsp \rangle} \in \mathbb{U}_{\langle bsp \rangle}$ is restricted to that in each step the vertex-program has to exhaustively apply the rules in Table 2 without handling the new generated edges. Thus the number of supersteps of different members in $\mathbb{U}_{\langle bsp \rangle}$ should be the same, i.e., the classification depth $\mathtt{cd}(\mathcal{G})$. Since any $u_{\langle bsp \rangle} \in \mathbb{U}_{\langle bsp \rangle}$ is also a PCA, based on result$_1$, it holds that the run-time of $u_{\langle bsp \rangle}$ is lower-bounded by $\mathtt{cd}(\mathcal{G})$. □

The Proof of Theorem 5.

Proof. **(C2)** In each part of `visit-new-nbr(A,e)`, when any new edge is inserted, the algorithm will first check whether this edge already exists. Since the number of inserted edges is finite, the algorithm would finally satisfy the termination condition.

(C1: $\Rightarrow$) Since only Algorithm 3 will generate new edges, we then investigate different 'For' loops in Algorithm 3.

loop 1: R1. The new inserted edge $e^{\sqsubseteq}(A, C)$ is derived from $e^{\sqsubseteq}(A, B)$ and $e^{\sqsubseteq}(B, C)$, where $e^{\sqsubseteq}(B, C)$ is an original edge. This matches R1.

loop 2: R2 The new inserted edge $e^{\sqsubseteq}(A, C)$ is derived from $e^{\sqsubseteq}(A, B)$, $e^{\sqsubseteq}(A, A_1)$ and $e^{\sqcap A_1}(B, C)$. By applying R3, $e^{\sqsubseteq}(A, C)$ can be obtained from $e^{\sqcap A_1}(A, C)$ and $e^{\sqsubseteq}(A, A_1)$.

loop 3: R4 The new inserted edge $e^{+r}(A, C)$ is derived from $e^{\sqsubseteq}(A, B)$ and $e^{+r}(B, C)$. This matches R4.

loop 4: R4 and R5 The new inserted edge $e^{\sqsubseteq}(D, C)$ is derived from $e^{\sqsubseteq}(A, B)$, $e^{-r}(B, C)$ and $e^{+r}(D, A)$, where $e^{-r}(B, C)$ is an original edge. By applying R4, $e^{\sqsubseteq}(D, C)$ can be obtained from $e^{+r}(D, A)$, $e^{\sqsubseteq}(A, B)$ and

and $e^{-r}(B, C)$. If B is $\perp$, the new inserted edge $e^{\sqsubseteq}(D, \perp)$ can also be achieved by applying R5.

loop 5: R4 and R5 The new inserted edge $e^{\sqsubseteq}(A, D)$ is derived from $e^{+r}(A, B)$, $e^{\sqsubseteq}(B, C)$ and $e^{-r}(C, D)$, where $e^{-r}(C, D)$ is an original edge. By applying R4, we can obtain $e^{\sqsubseteq}(A, D)$. If C is $\perp$, the new inserted edge $e^{\sqsubseteq}(A, \perp)$ can also be achieved by applying R5.

loop 6: R6 The new inserted edge $e^{+s}(A, B)$ is derived from $e^{+r}(A, B)$ and $e^{\sqsubseteq}(r, s)$. This matches R6.

loop 7: R7 The new inserted edge $e^{+t}(A, C)$ is derived from $e^{+r}(A, B)$, $e^{+s}(B, C)$ and $e^{\circ s}(r, t)$. This matches R7.

loop 8: R7 The new inserted edge $e^{+t}(C, A)$ is derived from $e^{+r}(B, A)$, $e^{+s}(C, B)$ and $e^{\circ r}(s, t)$. This also matches R7.

(C1: $\Leftarrow$) We assume that when Algorithm 1 and Algorithm 2 terminate, one of the rules in Table 2 can be applied to insert new edges labeled by "$\sqsubseteq$", and then show the contradiction. We distinguish on the different rules:

case1:R1. Suppose $e^{\sqsubseteq}(A, C)$ can be derived after termination from $e^{\sqsubseteq}(A, B)$ and $e^{\sqsubseteq}(B, C)$, where $e^{\sqsubseteq}(B, C)$ is an original edge. When $e^{\sqsubseteq}(A, B)$ is inserted to A, $Q(A)$ will trigger **loop 1** to apply R1. This is a contradiction to the assumption that the algorithm has terminated.

case2:R2. Suppose $e^{\sqsubseteq}(A, C)$ can be inserted after applying R2. It has to be inserted from $e^{\sqsubseteq}(A, A_1)$, $e^{\sqsubseteq}(A, A_2)$ and $e^{\sqcap}A_2(A_1, C)$. Then we investigate two different cases that $e^{\sqsubseteq}(A, A_1)$ and $e^{\sqsubseteq}(A, A_2)$ are inserted in different orders. When $e^{\sqsubseteq}(A, A_1)$ has been inserted, but $e^{\sqsubseteq}(A, A_2)$ has not been inserted, $e^{\sqsubseteq}(A, C)$ can also be achieved in loop 2 after $e^{\sqsubseteq}(A, A_2)$ is inserted, since $e^{\sqcap}A_1(A_2, C)$ and $e\sqsubseteq(A, A_1)$ have already existed. Another case is similar.

case3:R3. If R4 is applicable, then there exists $e^{\sqsubseteq}(A, B)$ and $e^{+r}(B, C)$, where $e^{+r}(B, C)$ is an original edge. $Q(A)$ would trigger the loop 3 in Algorithm 3 to apply R4.

case4:R4. If a new edge $e^{\sqsubseteq}(D, C)$ is inserted after applying R4, it has to be inserted from $e^{+r}(D, A)$, $e^{\sqsubseteq}(A, B)$ and $e^{-r}(B, C)$. However $e^{+r}(D, A)$, $e^{\sqsubseteq}(A, B)$ can be inserted in different orders. We can check that for each order, either of loop 3 (line 9) or loop 4 (loop 16) would handle it.

case5:R6. If R6 is applicable, then R4 is also applicable since the conclusions of R6 can only been used in R4 to drive new '$\sqsubseteq$' labeled edges. This contradicts case 4.

case6:R7. If R7 is applicable, then R4 is also applicable since the conclusions of R7 can only been used in R4 to drive new '$\sqsubseteq$' labeled edges. This contradicts case 4.

case7:R5. If R5 is applicable, then there exist $e^{+r}(A, B)$ and $e^{\sqsubseteq}(B, \perp)$. When either of these two edge is inserted, loop 3 (line 12) or loop 4 (loop 19) will be triggered to apply R5. This contradicts the assumption that the algorithm has terminated. □