IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

WEB REASONING AND ANSWER SET PROGRAMMING WITH SEMANTIC FINITE AND INFINITE SET PROGRAMMING

[1]Author- G Priyanka Jeeva Karunya, Scholar at CSE department and

[2]Author- Dr. Pankaj Kawadkar, Professor at CSE department both from Sri Satya Sai University of Technology and Medical Sciences-Sehore, MP

ABSTRACT

The Semantic Web is actually apt to demand knowledge and energy certain. This's pricey, and so it might well be restricted to certain domains on the internet which see a powerful edge in the use of its, though over time as the expertise gets much more commonplace it must get more inexpensive. Furthermore, the' network effect' could work as both an incentive as well as a screen. One of the primary benefits of supplying Semantic Web annotation is actually that's could be discussed and can easily get use to others, consequently when there's little details to share, then there's very little motivation to take the excess cost of sharing; however, after the ball begins to roll, there's an exponential benefit in merging the personal data of yours with others. The main aim of this study is to discuss the Semantic Web Reasoning and Answer Set Programming with Finite and Infinite Set Programming. It is concluded that the chance to exchange knowledge with outside energy sources in a completely declarative framework like ASP is particularly crucial in view of applications of the Semantic Web region.

Keywords *- Semantic Web, Answer Set Programming, finite, infinite, programming, Description Logics etc.*

1. INTRODUCTION

The present World Wide Web gives new open doors for teach called Knowledge Representation (KR). Web resources should be better embraced for programmed handling by PC operators. That is conceivable by more deliberate and formal portrayal instruments utilized for characterizing substance of the pages. In addition, Uniform Resource Identifiers (URIs), which are the naming plan for web resources, enables KR frameworks to enhance connecting between semantic archives by maintaining a strategic distance from the ambiguities of common dialect. That is the reason new potential outcomes and difficulties are opened for knowledge representation presumptions. A standout amongst the most ground-breaking highlights of the World Wide Web is the likelihood of interconnections between knowledge sources. Interconnections are given by connecting systems on pages. As it were, rather than duplicating information from outside sources made by other individuals to our page, we simply give connects to such sources. Semantic Web can be dealt with as an augmentation to current web and the successor of Web 2.0. In late Web 2.0 engineering, there

is a great deal of unstructured data. Information is spoken to in different ways, contradictory witch each other. What Semantic Web offers is characterized significance of information, machine-readable depiction of substance, rearrangements of information trade, order and deduction components,

data handling and coordination in machine-robotized way.

1.1 Semantic Web

The present World Wide Web has made an extraordinary progress. WWW is widely utilized all around the globe. A basic establishment, on which it is based, has given its quality and worldwide network. Relatively every client can make and distribute hypertext assets in a basic way. Fundamentally "web content comprises basically of appropriated hypertext, and is gotten to by means of a mix of watchword-based pursuit and connection route". Endless supply of information sum opens through the web, current web has experienced downsides. The enormous measure of substance required to be prepared with a specific end goal to discover wanted realities, has "featured some genuine weaknesses in the hypertext worldview":

- difficulties in finding right information through straightforward looking and perusing components,
- problems of discovering actualities, which has basic journalists,
- irrelevant comes about subsequent to giving of more mind-boggling inquiries in the perusing motors, absence of semantics,
- Lack of reasoning offices

Individuals are confronting bothers in web content perusing. It is self-evident, that substance preparing undertakings are considerably more muddled with regards to implement them into programming operators. It is on the grounds that advancements utilized for hypertext archives creation are outlined with the aim of depicting and showing information in the human way. The absence of formal, logic-based information structure is the enormous issue for machines, which require numerically based path for knowledge handling. Here are the difficulties, current web will confront.

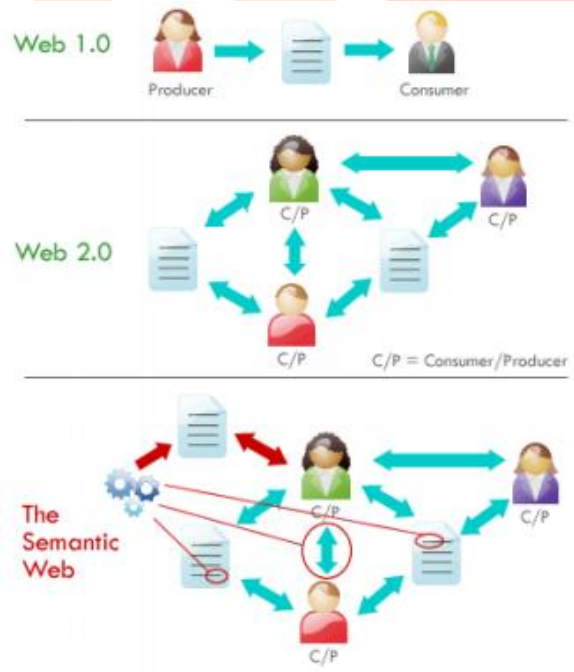

Figure 1: Web evolution from old Web 1.0 up to Semantic Web

1.2 The Future of the Semantic Web

We've observed in this specific study that there's been enthusiastic and significant attempt during the last couple of years to check out as well as create the technology, shared vocabularies as well as ideas that are turning Tim BernersLee's vision into a reality. There's a great deal of method to visit till it's a regular component of the Web infrastructure but, nonetheless, there's been startling improvement within the last several years, with analysis organizations amongst the leaders.

- Barriers to adoption
- Summary of effect areas
- Timescales
- Final word

2. REVIEW OF LITERATURE

Faber, Wolfgang. (2020) Answer Set Programming (ASP) is actually a rule-based language rooted in conventional Logic Programming, Knowledge Representation, Databases, and Nonmonotonic Reasoning. It has a flexible language for declarative problem solving, with assistance of effective general-purpose solvers & reasoners. The bigger part of this article offers an introduction to ASP, with a historical viewpoint, a definition of the primary language, a guideline to knowledge representation, as well as an overview of current ASP solvers. One component focus on one widely used feature: aggregates as well as generalized atoms.

Erdem et al., (2016) ASP has been put on fruitfully to a broad range of places in AI and also in some other fields, both in academia and in industry, because of the expressive representation languages of ASP as well as the constant enhancement of ASP solvers. We show several of these ASP programs, particularly, in knowledge representation as well as reasoning, robotics, computational biology and bioinformatics in addition to a bit of industrial uses. We talk about the problems resolved by ASP in these apps and highlight the strengths of ASP like a helpful AI paradigm.

Nickles, Mileo and Matthias, Alessandra. (2014) We suggest a framework for reasoning regarding powerful Web data, based on probabilistic Answer Set Programming (ASP). The strategy of ours, which is proto typically implemented, provides for the annotation of first order formulas along with ASP rules as well as facts with probabilities, and also for learning of such weights from examples (parameter estimation).

Gelfond, Kahl and Michael, Y.. (2013) Knowledge representation as well as reasoning is actually the basis of artificial intelligence, declarative programming, as well as the look of knowledge intensive application systems capable of performing smart things. Employing probabilistic and logical formalisms based on answer set programming (ASP) as well as action languages, this publication shows how knowledge intensive methods could be provided knowledge about the planet and just how it may be utilized to solve non trivial computational issues.

Corapi et al., (2011) In this particular paper we talk about the look of an Inductive Logic Programming (ILP) system in Answer Set Programming (ASP) and much more in common the issue of integrating the 2. We show the way to formalize the learning issue as an ASP system as well as give information on the way the optimization features of contemporary solvers can be adapted to derive ideal hypotheses.

3. ANSWER SET PROGRAMMING

Answer Set Programming (ASP) is actually a program paradigm which may be utilized to represent knowledge and also to solve information-intensive and combinatorial issues. Combinatorial search concerns are the ones thought to not have an effective method to minimize the area of fixes (NP hard), a number of examples of this kind are provided in this specific document. The idea of knowledge is quite wide and extremely determined by the domain name. It's inclusive of beliefs to different degrees, facts as well as rules which may be used for enabling reasoning, and commonly, which commonsense understanding we've about the planet.

The puzzles given here are resolved to a programming application I made, freely inspired by Clingo as well as Datalog, the ASP produced for Faculty of Potsdam. Differently from Datalog, the scripts noted here are derived from a domain name particular language, and they leverage the clearly typed compiler of Scala. The goal of this particular dissertation isn't in order to make a far more effective way to complete ASP but only to construct the resources for moving on with the project I am conducting, and then to check out the possibility of this particular paradigm in other parts like data management as well as principal methods. I would like to convey the understanding this software program paradigm creates benefits in completeness and terms suppleness for real life scenarios. ASP has been used for:

- Choice support system to assist the Space Shuttle controllers to cope with vital scenarios due to several failures.
- Effective workforce management of GioiaTauro seaport subjected to enormous quantities of constraints.
- Classify real requirements as well as profiles of Telecom Italia users (1M calls/day) for producing personalized experience of customer service.

- Setup of services and products are put through advanced rules in railways security controls, tourism, auto, computer hardware, software programs.

ASP is actually a kind of logic programming where rules (or maybe arguments) could be thought of as executable specs. An ASP method doesn't listen for queries, rather it applies all conditions recursively until the produced knowledge doesn't evolve any longer. When presently there are actually no more satisfiable factors to run, the device returns the summary of results, the so called stable versions, or maybe answer sets. Let us see it with a good example about key figures.

In ASP there are at least two main stages that denote the execution of the program:

- Grounding- The space of possible solutions is fully expanded.
- Solving- The unmatching solutions are filtered out according to the defined constraints.

4. PROGRAMS AND ANSWER SETS

We begin the ASP debate of ours with the propositional environment. The building blocks for programs are actually rules, literals, and atoms. Atoms are actually elementary propositions (factual statements) which could be false or true ; literals are actually atoms a and the negations of theirs not really a. Rules are expressions of the form

$$a \leftarrow b1,\ldots,bm,not\ c1,\ldots,not\ cn \qquad (1)$$

where a and all bi's and cj's are atoms. Intuitively, a rule (1) is a justification to "establish" or "derive" that a (the so- called head) is true, if all literals to the

right of ← (the so-called body) are true in the following sense: a non-negated literal bi is true if it has a derivation, a negated one, not cj, is true if the atom cj does not have one. For instance, the rule

light_on ← power_on, not broken

informally means we can assert that the light is on, if we established the power is on and there is no reason to think the lamp is broken. Rules may have no body. For instance, we may have a rule:

$power\_on \leftarrow .$

Such rules are called facts, as the head is unconditionally true, and the arrow ← is typically omitted.

Programs are actually finite collections of rules. They're thought of as justifications for sets of atoms that have precisely those atoms which could be started. It's essential to say that not isn't a regular negation operator. Instead, it's intended to stand for a modality non derivable. Taking a look at the tiny system with the 2 rules talked about here, power_on must be derived (as it's provided as a fact), while intuitively broken shouldn't (the system, which describes what we know, doesn't have rule to derive) that is broken. This in turn enables us to de rive light_on.

Formalizing these intuitions posed a challenge to the expertise representation as well as logic programming communities for a long time. Ultimately, solution sets offered an answer which received acceptance.

5. SEMANTIC WEB REASONING AND ANSWER SET PROGRAMMING

5.1 Answer Set Programming with Infinity

DLPs and answer sets are defined, and their definitions are extended to take into consideration infinite domains and their inverses. Individual names are designated as constants and are written in lowercase; variables, on the other hand, are written in capital letters. There is no difference between the phrase's variables and constants. Predicate $p_1(t_1)$, binary predicate $p_2(t_1,t_2)$, and term $p_2^-(t_1,t_2)$ are the definitions of atoms, which are defined as p_1 a unary predicate, and p_2 a binary predicate. In the case of binary (potentially inverted) predicates, we assume $p(t_1\text{-}t_2)^- = p^-(t_1\text{-}t_2)$ for binary predicates, and for unary predicates, and we assume p^{--} to be defined as an inverted atom a with the predicate and the arguments reversed in the case of atoms a. Inverting atoms does not make sense for predicates of higher arity, therefore we limit ourselves to unary and binary predicates. Atoms having no variables are referred to as ground atoms. The term "literal" refers to atoms that are either preceded by (e.g. l = a) or preceded by the prefix (l = ¬a). $(\neg a)^-$ is defined as $(\neg a)^-$ for the atom a. "With no variables, we get something called a "ground literal." Extendable literals are anything that is not a literal, such as something of the type $not(l)$. An extended literal that has no variables is known as a ground extended literal.$\neg X =$

$\{\rightarrow l|l\epsilon X\}$, where $\rightarrow\rightarrow$a is defined as a literal. If $X \cap \rightarrow X = \emptyset$, then a set of ground literals X is congruent. As an example, let us assume that an extended literal is defined in terms of the set of underlying literals, which is defined as $X^- = l\,|not(l) \in X$. Predicates can be expressed in several ways, but the most common is through the use of Greek letters. Attaching variables enables us to write e.g.α(X) for $\{a(X)|a \in \alpha\}, \beta(X,Y) for \{b(X,Y)|b\epsilon\beta\}$, or $not(\alpha)(X) for \{not(a)X))|a\epsilon\alpha\}$. This is because we can write e.g. There is also a binary predicate≠, which we presume has the normal meaning.

In a disjunctive logic programme (DLP), there are a finite number of rules $\alpha \leftarrow \beta$ that must be followed in order for the programme to work. It's common practise to refer to programmes that do not provide the negation of any of their rules $\beta^- \cup \alpha^- = \emptyset$ as "failing" (naf). The term "simple" refers to programmes that have no naf, i.e., no disjunction in the head, for all rules β. Variable-free programmes are thrown out. We call P_H the anchored programme produced from a programme P by replacing each variable in P by every conceivable constant in H, which is a non-empty collection of constants H. Keep in mind that the number of regulations in P_H is virtually limitless (if H is infinite). There must be a grounded variant of an infinite DLP in order for it to work. The (potentially infinite) collection of non-empty constants $\mathcal{H}_{P_\mathcal{H}}$ occurring in P_H is the universe of a grounded programme. $\mathcal{H}_{P_\mathcal{H}}$Is equal to$\mathcal{H}$, as can be shown. In order to generate the (potentially endless) $B_{P_\mathcal{H}}$set of ground atoms, one must use the predicates in P_H and their inverses, along with the constants in H. This is the foundation of every P_H-based programme. A grounded DLP P interpretation I is any consistent set of literals that is a subset of $B_P \cup \rightarrow B_P$. It is possible to satisfy the rule $\alpha \leftarrow \beta$ if $\alpha \sqcap I \neq \emptyset$whenever I interpret the DLP P without the null hypothesis. Or, if the disjunction of literals in the head is true, then the conjunction of literals in the body of a rule must also be true, according to intuition. When $p(t_1,t_2) \in I \Leftrightarrow p^-(t_2,t_1) \in I$, it is a model of a grounded DLP without naf if it meets every rule in P and all literals in $B_P \cup \rightarrow B_P$ are included in I. Furthermore, if J⊂I of P does not exist, the model is minimum. The Gelfond-Lifschitz transformation yields the programme P^I by eliminating in P given a grounded DLP P and an interpretation I.

- Rules having $not(l)$ in their bodies are marked with a l ∈I,
- If a rule contains an $not(l)$ anywhere in its body or head, use l∉ I.
- If a rule contains an $not(l)$ anywhere in its body or head.

A tuple is an explanation of a non-grounded DLP P in the sense that the interpretation of the non-grounded $P_{\mathcal{H}_I}$ in I is an interpretation of I. $P^I_{H_I}$ is an answer set to a DLP P if I is a minimum model of P (I, H_I) and H_I is an answer set to P If a DLP P has an answer set, it is consistent. According to P, a unary p (potentially negated) can be said to be satisfied if it is found in P. The answer set (I, H_I) of P is said to be finitely satisfiable if it contains an answer set such that p(a) ∈ I for all a ∈ H_I. Satisfiability testing is the process of ensuring that a (potentially negated) unary predicate is satisfiable. There may be limitless domains, but a minimum model of a simple programme may be used to explain why literals are necessary. The operator T, which calculates the closure of a collection of literals with respect to a programme P, allows us to articulate the rationale behind a literal in a more formal way. The operator $P^M_{H_M}$ is defined for a DLP P and an answer set (M, H_M) of P that is a convenient feature.

$$T_{P^M_{\mathcal{H}_M}} : \mathcal{B}_{P^M_{\mathcal{H}_M}} \cup \neg\mathcal{B}_{P^M_{\mathcal{H}_M}} \rightarrow \mathcal{B}_{P^M_{\mathcal{H}_M}} \cup \neg\overline{\mathcal{B}}_{P^M_{\mathcal{H}_M}}{}^2 \quad (2)$$

As follows:

$$T_{P^M_{\mathcal{H}_M}}(B) = B \cup \{a, a^-|a \leftarrow \beta \in P^M_{\mathcal{H}_M} \wedge \beta \subseteq B\} \quad (3)$$

B is defined as$T^0(B) as B, and\ T^{n+1}(B) as\ T^n(T(B))$, respectively. Operator $P^M_{H_M}$offers the severe impacts of a set B as per $P^M_{H_M}$operator.

Theorem 1. For simplicity, we'll assume P is a DLP and that (M, H_M) is a subset of P's answer set. Then ∀a ∈ M · ∃n < ∞ · a ∈ $T^n(\emptyset)$

A preliminary sketch for your consideration. Consider a ∃a ∈ M · ∀n < ∞ · a ∈ T^n(∅) assumption. It's necessary to write down each and every one of the following sets of values for each of the variables: r: a'← $\beta \in P_{H_M}^M$. We can continue this approach for all b indefinitely because r can always be selected. As a result, we may create a tight subset M', which includes all of M, except for a, a-, and all b. (intuitively, we throw away all the literals that are causing a to be not finitely deducible). $P_{H_M}^M$ Is a model that can be proved to exist in M'. An inconsistency with M's minimalism. An answer set's finite foundation may be discovered using the preceding theorem (M, H_M). Satisfiability verification and DLs modeling can benefit from the decidability proof.

5.2 Conceptual Logic Programs

Unsolvable: Satisfiability testing in the aforementioned context of answer set programming with infinite infinity is impossible to determine. Our goal is to reclaim the decidability of satisfiability testing while being cautious to preserve the expressiveness of our DLPs. Restricting DLPs to notional logic programmes based on modal logics (and DLs in particular), we end up with DLPs that may be fulfilled by an answer set that is in the form of a tree structure, which has the tree-model characteristic. In modal logic, it is believed that the robust decidability is due to this tree-model characteristic. This is confirmed by the tree-model property, which is critical for CLP satisfiability testing decidability. Regulations that express the fact that it doesn't matter whether or not a literal is present in the answer set, as long as no other laws forbid or enforce its presence, make up a CLP. Tree constraints and tree rules, both of which are general rules, are used to ensure the tree-model property, which means that they possess a tree structure, are included in a CLP. It is formal to say that a (finite) tree (T) is a (finite) subset of the set$\mathbb{N}_0^{*4}$, such that we obtain the result that$x.c \in T$, where$x \in \mathbb{N}_0^{*4}$. The root of T is the word '', which is referred to as a node. The successors of a node x are referred to as$x \in T\ we\ say\ x.c, c \in \mathbb{N}_0$. As a rule, $x \bullet 0 = x\ and\ (x \bullet c) \bullet -1$ are equal to each other, but x • 1 is undefined. A tree is said to be k-ary if every node x has k successors. The rank of a tree is the number of successors that a node can have. Σ Is a tuple of (T, V) where the tree is T and the function that labels T's nodes with letters from the alphabet is V:T→ Σ . Allowing for more generic inequalities, as well as extending the definition of free tree DLPs to include rules with binary literals in the header, rather to simply unary literals in the head.

5.3 Simulating Description Logics and Finite Answer Set Programming

For example, CLPs can be used to imitate expressive DLs, answer set programming with a finite (Herbrand) universe, and datalogprogrammes where $not()$ −literals may appear in the body of a sentence. The Herbrand universe of the programme q(X) ← f(c, b, c) is {b, c}. In contrast to the answer set programming mentioned, where the universe is a superset (potentially infinite) of {c, b}, this universe is limited (assuming a DLP has a finite list of regulations). You may, however, convert the finite DLP obtained by substituting the Herbrand universe for the Herbrand universe's Herbrand universe in order to create a CLP that can be associated with a variable. In the previous example, this results in the sentences q(b)(X) ← f(c, b, c)(X) andq(c)(X) ← f(c, b, c)(X), with the grounded literals now regarded as unary predicates. The mathematical model is the result of this investigation.

Theorem 2. M is an answer set of a logic program $P\ iff\ (M', \{a\})$, with 'a' a constant, is an answer set of the $CLP\ P'$ where $M' = \{l(a) | l \in M\}\ and\ P' = \{r(X) | r \in P_{H_M}^M\}, with\ r(X)$ defined such that every literal l in r is replaced by $l(X)$.

It is a family of logical formalisms that are helpful for knowledge representation, such as the representation of taxonomies in particular application domains. DLs are based on frame based systems. Concepts and roles are two of the most fundamental aspects of the language. It is thus possible to identify different DLs by the collection of constructors that may be used to create sophisticated ideas or roles. An axiom that states that a notion is swallowed by another can be represented using terminology. If a kid is popular, he or she will have at least three other popular friends, as seen in the following phrase:

$$PopChild \sqsubseteq Child \sqcap (\geq 3\ friend.PopChild) \quad (4)$$

PopChild, Child, and Friend are all concept names. All things for which there are at least n separate Q-successors that belong to D are represented by a notion expression (≥n Q.D). Furthermore, there are n distinct y_i that are related to (x, Q.D) and to (x, y_i). This means that (x, y_i) is connected to (x, D). A member of the intersection function ⊔indicates that an item is a member of both the A and the B operands. It's not uncommon to see ⊔ (union) and (negativity) being used as constructors, both of which have set-theoretic implications. Existence restriction ∃R.C and value restriction ∀R.C are two more often used DLs constructors. They can use the word 'father' to refer to things like a wealthy ∃father. RichPerson. All the x's that have at least one y, related through the role of father (i.e. they have at least one father) and that y belongs to the concept Rich Person are represented by this concept expression The value limitation ∀friends. Rich Person.' can be used to describe people with only rich friends. For example, if x is an ancestor of y and y is an ancestor of z, then the ancestor role might be designated as transitive, in which case x becomes an ancestor of z. As an alternative to P^-, there is P^-, which constructs roles by taking the opposite of a role.

It is not possible to simulate finite answer set programming for some DLs because they do not have the finite model property, such as knowledge bases that contain only infinite models. SHIQ is an example of a DL that provides the formal semantics of the ontology language OIL with transitive closure of roles instead of transitivity of roles. The following is a definition of the syntax for SHIQ concept expressions.

$$D_1, D_2 \rightarrow A|\neg D_1|D_1 \sqcap D_2|D_1 \sqcup D_2|\exists R.D_1|\forall R.D_1|(\leq n\ Q.D_1)|(\geq n\ Q.D_1) \quad (5)$$

$$Q \rightarrow P|P^-$$

$$R \rightarrow Q|Q\ *$$

For DL, the interpretation $I = (\Delta^I, .^I)$consists of a non-empty (potentially infinite) domain I and an interpretation variable Δ^I formulated as having for the SHIQ* DL translation.

$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for concept names A

$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for role names P

$P^{-\mathcal{I}} = \{(y, x)|(x, y) \in P^{\mathcal{I}}\}$ for role names P

$(\neg D_1)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus D_1^{\mathcal{I}}$

$(D_1 \sqcap D_2)^{\mathcal{I}} = D_1^{\mathcal{I}} \cap D_2^{\mathcal{I}}$

$(D_1 \sqcup D_2)^{\mathcal{I}} = D_1^{\mathcal{I}} \cup D_2^{\mathcal{I}}$

$(\exists R.D_1)^{\mathcal{I}} = \{x|\exists y : (x, y) \in R^{\mathcal{I}} \wedge y \in D_1^{\mathcal{I}}\}$

$(\forall R.D_1)^{\mathcal{I}} = \{x|\forall y : (x, y) \in R^{\mathcal{I}} \Rightarrow y \in D_1^{\mathcal{I}}\}$

$(\leq n\ Q.D_1)^{\mathcal{I}} = \{x|\#\{y|(x, y) \in Q^{\mathcal{I}} \wedge y \in D_1^{\mathcal{I}}\} \leq n\}$

$(\geq n\ Q.D_1)^{\mathcal{I}} = \{x|\#\{y|(x, y) \in Q^{\mathcal{I}} \wedge y \in D_1^{\mathcal{I}}\} \geq n\}$

$(R^*)^{\mathcal{I}} = R^{\mathcal{I}^*}$ i.e. the reflexive transitive closure of $R^{\mathcal{I}}$ (6)

Assuming A is an idea, and P is a role, we may begin. In the form $C_1 \sqsubseteq C_2$, a terminological axiom is defined by arbitrary concept expressions C_1 and C_2. A view of the world If $C_1^I \subseteq C_2^I$, I satisfy a terminological axiom $C_1 \sqsubseteq C_2$. If R_1 and R_2 are roles, then the role axiom is in the form of $R_1 \sqsubseteq R_2$ (possibly inverted or transitive closures). If $R_1^I \subseteq R_2^I$, then an interpretation I fulfils the role axiom $R_1 \sqsubseteq R_2$. A repository of information Σ is a collection of axioms pertaining to terminology and roles. A view of the world if I meet all of the axioms ofΣ, then I am a model of. SHIQ* concept expression C can be satisfied if there is a model of that has an interpretation that is not empty, i.e. $C^I \neq 0$. With CLPs, satisfiability checking in SHIQ* may be easily simulated. Ontologies such as the OWL DL ontology in Figure 2express that sales item are those objects for which there is at minimum one selling price. The DL

```
<owl:Class rdf:ID="SalesItem">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Item"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasPrice"/>
      <owl:minCardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:minCardinality>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Figure 2: An OWL DL example ontology

The axioms of the ontology form the basis of the knowledge base.

$$SalesItem \sqsubseteq Item \sqcap \exists hasPrice$$

$$Item \sqcap \exists hasPrice \sqsubseteq SalesItem$$

The respective CLP creates SalesItem, Item, hasPrice, and ¬hasPrice free:

$$SalesItem(X) \vee not(SalesItem(X)) \leftarrow$$

$$Item(X) \vee not(Item(X)) \leftarrow$$

$$hasPrice(X,Y) \vee not(hasPrice(X,Y)) \leftarrow$$

$$\neg hasPrice(X,Y) \vee not(\neg hasPrice(X,Y)) \leftarrow$$

Knowledge base standards determine how to negate concepts that emerge in the database.

$$\neg SalesItem(X) \leftarrow not(SalesItem(X))$$

$$\neg Item(X) \leftarrow not(Item(X))$$

$$\neg \exists hasPrice(X) \leftarrow not(\exists hasPrice(X))$$

$$\neg(Item\ u\ \exists hasPrice)(X) \leftarrow not((Item\ u\ \exists hasPrice)(X))$$

Along with the definitions of intersection and existence restrictions, there are also the rules of ∃hasPrice.

$$(Item \sqcap \exists hasPrice)(X) \leftarrow Item(X), \exists hasPrice(X)$$

$$\exists hasPrice(X) \leftarrow hasPrice(X,Y)$$

Our DL axioms are expressed in terms:

$$\leftarrow SalesItem(X), not((Item \sqcap \exists hasPrice)(X))$$

$$\leftarrow not(SalesItem(X)), (Item \sqcap \exists hasPrice)(X)$$

OWL's ontology may be expressed using just two rules: those that emulate DLs semantics and that can be automatically deduced. The SHIQ* knowledge base Σ "and the idea expression C are formally defined as $\Phi(C, \Sigma)$ as follows

clos(C,Σ)	$\Phi(C,\Sigma)$
concepts$A$	$A(X) \vee not(A(X)) \leftarrow$ (7)
rolenames$P$	$P(X,Y) \vee not(P(X,Y)) \leftarrow$ (8)
	$\neg P(X,Y) \vee not(\neg P(X,Y)) \leftarrow$ (9)
expressions $DD=\neg E$	$\neg E(X) \leftarrow not(E(X))$ (10)
$D=E \mathrm{H} F D=E \mathrm{H} F$	$E \mathrm{H} F(X) \leftarrow E(X), F(X)$ (11)
$D=\exists Q.E D=\exists Q^{*}.E$	$E \mathrm{H} F(X) \leftarrow E(X)$ (12)
$D=\forall R.E$	$E \mathrm{H} F(X) \leftarrow F(X)$ (13)
$D=(\leq n Q.E)$	$\exists Q.E(X) \leftarrow Q(X,Y), E(Y)$ (14)
$D=(\geq n Q.E)$	$\exists Q^{*}.E(X) \leftarrow E(X)$ (15)
$C1 \pm C2 \in \Sigma$	$\exists Q^{*}.E(X) \leftarrow Q(X,Y), \exists Q^{*}.E(Y)$ (16)
$R1 \pm R2 \in \Sigma$	$\forall R.E(X) \leftarrow \neg \exists R.\neg E(X)$ (17)
	$(\leq n Q.E)(X) \leftarrow \neg(\geq n+1 Q.E)(X)$ (18)
	$(\geq n Q.E)(X) \leftarrow Q(X,Y1),\ldots,Q(X,Y\mathrm{n}),$
	$E(Y1),\ldots,E(Y\mathrm{n}), Y1 \neq Y2,\ldots$ (19)
	$\leftarrow C1(X), not(C2(X))$ (20)
	$\leftarrow R1(X,Y), not(R2(X,Y))$ (21)

We have D ∈ clos(C, Σ)or every idea expression D in $\{C\} \cup \Sigma$.

We also have some of the possibilities for each D inclos(C, Σ):

$D = \neg D_1, D_1 \in clos(C, \Sigma)$

$D = D_1 \sqcup D_2, \{D_1, D_2\} \subseteq clos(C, \Sigma)$

$D = D_1 \sqcap D_2, \{D_1, D_2\} \subseteq clos(C, \Sigma)$

$D = \exists R.D_1, \{R, D_1\} \subseteq clos(C, \Sigma)$

$D = \forall R.D_1, \{D_1, \exists R.\neg D_1\} \subseteq clos(C, \Sigma)$

$D = (\leq n\, Q.D_1)$, then $\{(\geq n+1\, Q.D_1)\} \subseteq clos(C, \Sigma)$

$D = (\geq n\, Q.D_1)$, then $\{Q, D_1\} \subseteq clos(C, \Sigma)$

For all $R* \in clos(C, \Sigma), R \in clos(C, \Sigma)$,

For all$D \in clos(C, \Sigma), \neg D \in clos(C, \Sigma)$.

Theorem 3. Conceptualization of the SHIQ* It's possible to satisfy C using a SHIQ* knowledge base. Σiff C(X) may be satisfied in terms of $\Phi(C, \Sigma)$.

There exists a model $I = (\Delta^I, .^I)$with as $C^I \neq$ 0satisfiability for. As a result of this interpretation, we may derive the answer set A = (M, H_M) as follows:

$$M = \{C(a) \mid a \in C^{\mathcal{I}}, C \in clos(C, \Sigma)\} \cup \{\neg C(a) \mid a \notin C^{\mathcal{I}}, C \in clos(C, \Sigma)\}$$

$$\cup \{Q(a, b), Q^{-}(b, a) \mid (a, b) \in Q^{\mathcal{I}}, Q \in clos(C, \Sigma)\}$$

$$\cup \{\neg Q(a, b), \neg Q^{-}(b, a) \mid (a, b) \notin Q^{\mathcal{I}}, Q \in clos(C, \Sigma)\} \quad (22)$$

As a result, it is simple to demonstrate that (M, H_M) is a solution set of $\Phi(C, \Sigma)$.However, not all CLPs can be rewritten to represent the same information in a SHIQ* knowledge base, as stated in Theorem. Take the following rule, for instance:

$$g(X, Y) \leftarrow a(X), f(X, Y), b(Y) \quad (23)$$

Assuming the projection of f on both its first and second coordinates, g is stated to be precisely the same projection f. The three axioms $\top \sqsubseteq \forall g^{-}.a, \top \sqsubseteq \forall g\ .b$ *and* $g \sqsubseteq f$can be used to imitate one direction (the minimality). More expressive DLs, on the other hand, need concepts and roles to be expressed in more ways.

5. CONCLUSION

The concept of this particular effort was contributing to the continuing advancements in the Semantic Web as well as logic programming communities about an integration of a rule layer in the present Semantic Web structure. Specifically, we tried to exploit the functions of non-monotonic reason in the type of answer set programming towards reasoning duties in the Semantic Web. Right after a short survey with the status quo as well as an introduction to the fundamental ideas of answer set programming as well as knowledge representation in the Semantic Web via Description Logics as well as ontologies. Importantly, we chose an approach of rigid semantic separation, in order to stay away from common pitfalls when combining some diverge formalisms as logic programming as well as description logics, that are fragment of first order logic. Specifically, we stay away from undecidability problems because of their completely different model theories.

REFERENCES

[1] Corapi, Domenico & Russo, Alessandra &Lupu, Emil. (2011). Inductive Logic Programming in Answer Set Programming. 91-97. 10.1007/978-3-642-31951-8_12.

[2] Eiter, Thomas &Ianni, Giovambattista& Lukasiewicz, Thomas &Schindlauer, Roman &Tompits, Hans. (2008). Combining Answer Set programming with Description Logics for the Semantic Web. Artificial Intelligence. 172. 1495-1539. 10.1016/j.artint.2008.04.002.

[3] Eiter, Thomas & Kaminski, Tobias &Redl, Christoph &Schüller, Peter &Weinzierl, Antonius. (2017). Answer Set Programming with External Source Access. 10.1007/978-3-319-61033-7_7.

[4] Eiter, Thomas. (2007). Answer Set Programming for the Semantic Web. 23-26. 10.1007/978-3-540-74610-2_3.

[5] Erdem, Esra&Gelfond, Michael & Leone, Nicola. (2016). Applications of Answer Set Programming. AI Magazine. 37. 53. 10.1609/aimag.v37i3.2678.

[6] Faber, Wolfgang. (2020). An Introduction to Answer Set Programming and Some of Its Extensions. 10.1007/978-3-030-60067-9_6.

[7] Gelfond, Michael & Kahl, Y.. (2013). Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach. Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. 1-348. 10.1017/CBO9781139342124.

[8] M. Denecker, J. Vennekens, S. Bond, M. Gebser, and M. Truszczynski, "The second answer set programming competition," in Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009), ser. LNCS, vol. 5753. Springer, 2009, pp. 637–654.

[9] Nickles, Matthias &Mileo, Alessandra. (2014). Web Stream Reasoning Using Probabilistic Answer Set Programming. Lecture Notes in Computer Science. 8741. 197. 10.1007/978-3-319-11113-1_16.

[10] S. Heymans, D. V. Nieuwenborgh, and D. Vermeir. Semantic web reasoning with conceptual logic programs. In Rules and Rule Markup Languages for the Semantic Web: Third International Workshop (RuleML 2004), pages 113–127, Hiroshima, Japan, Nov. 2004.