IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

SEMANTIC AND SYNTAX OF WEB UTILIZINGHEX PROGRAMME

[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, often known as Web 3.0, is a set of standards developed by the World Wide Web Consortium (W3C) that extends the World Wide Web... Tim Berners-Lee created the word to describe a network of data (or data web) that computers can process—that is, one in which most of the meaning is machine-readable. Through outside atoms, hex programs are able to model several essential extensions to ASP, and therefore are a useful application for expressing different applications. In this paper we examine the Semantic Web Through Hex Programme. We define semantics and syntax of hex programs and show the way they could be used in the context of the Semantic Web. Ultimately, we clearly show the usability as well as usefulness of hex programs as well as the prototype implementation of ours on the foundation of concrete, real world scenarios. Real-world utilizations of hex-programs and the particular solver, showing their value and adaptability, are additionally exhibited in the theory.

Keywords *- Semantic Web, Hex Programme,ASP etc.*

1. INTRODUCTION

The Semantic Web initiative of the World Wide Web Consortium (W3C) has been active for the final couple of years and has attracted scepticism and interest in equal measure. The effect of the Semantic Web is actually apt to be especially strong in distance learning, libraries as well as info management, and collaborative research; we shall check out each. 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

The Semantic Web has potential that is great, and with direct application to the HE and FE sector. Nevertheless, it's been a rather long time of growth and does call for an expenditure of time, resources and expertise. Nevertheless, the time does appear appropriate to begin to consider how better to use the simpler applications of the technology.

Prior to the Semantic Web can easily end up worldwide functional, there does need to be more often, and it must be a lot more readily accessible. There's a unique overhead to utilizing the Semantic Web in phrases of establishing shared vocabularies as well as ontologies, and also in giving the correct annotations to energy that make them

noticeable to the Semantic Web. This's a non-trivial job and sometimes customers will sometimes not have the time period to include things like this, or maybe the experience to do it effectively. A missing part of the Semantic Web is actually a simple means supporting this, like the editors as well as resources for the traditional Web. Unquestionably the ease of the HTML language used in the present Web was obviously a significant impact on the success of its and as a way for the Semantic Web to break out from narrow towns to common use it must deal with the

problems of making it so easy to use and accessible to other.

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.

1.1 HEX Programme

We define the syntax as well as answer set semantics of hex plans, extending ASP with higher order functions and strong interfacing of outside computation solutions. While answer set semantics for higher order logic plans has been recommended earlier by Ross [1994], additional extension of that proposal to accommodate outside atoms is commercially difficult since the technique of Ross is founded on the idea of unfounded set, which can't be very easily generalized to this particular environment. The strategy of ours, instead, is actually based on a recent idea of system reduct as a result of Faber et al. [2004], which admits an all-natural definition of answer set semantics.

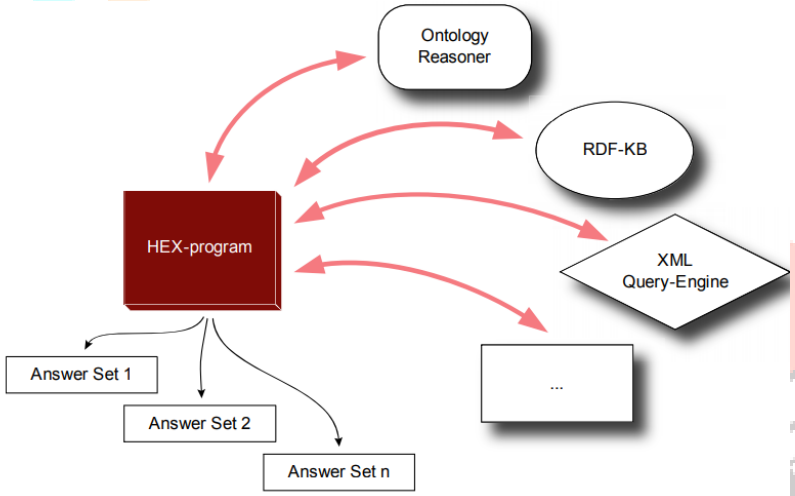

Figure 1: hex-program Concept

- We are going to discuss outside atoms like a helpful abstraction of several extensions to ASP including, among others, aggregates, explanation logic atoms, or maybe agent programs. Outside atoms thus facilitate investigating typical qualities of such extensions, and may function as a consistent framework for defining semantics of more identical extensions of ASP. Moreover, hex programs are actually a foundation for the handy look of generic evaluation algorithms for this kind of extensions in this particular framework.

- By means of hex programs, effective meta reasoning becomes readily available in a decidable context, e.g., for Semantic Web applications, for meta interpretation found ASP itself, or even for defining policy languages. For instance, advanced closed the definition or community reasoning of constructs for a prolonged ontology language (e.g., of RDF Schema) is actually well supported. Because of the higher order functions, the representation is actually succinct.

- Eventually, we are going to give a comprehensive account of strategies the best way to calculate the answer sets of a hex program under the state of utilizing a current solver for conventional ASP. To this conclusion, we'll determine structural qualities of a hex program which will allow us to assess it by splitting it within elements. Furthermore, the intricacy of fixing a hex

program is surveyed.

Remember that various other logic-based formalisms, including TRIPLE or even FLogic, feature also higher order predicates for meta reasoning wearing Semantic Web applications. Nevertheless, TRIPLE is low level focused & lacks exact semantics, while F Logic within the implementations of its (Flora, Florid, Ontoweb) restricts the expressiveness of its to well-founded semantics for negation, in order to get effectiveness. The formalism of ours, instead, is completely declarative and presents the chance of nondeterministic predicate characterization with higher complexity. This proved currently helpful & moderately effective for a range of applications with inherent nondeterminism, like diagnosis, preparation, or maybe configuration, and hence offers a rich ground for integrating these places with meta reasoning. In the course of Section 5.3, the place we introduce by now implemented outside atoms, a selection of illustrative examples will present the comfort of hex programs.

2. REVIEW OF LITERATURE

Falkner et al., (2018) Automated problem solving in conjunction with declarative specifications of search problems have proven to considerably enhance the implementation as well as maintenance bills along with the man machine interaction of deployed industrial uses. The knowledge representation as well as reasoning (KRR) framework of solution set programming (ASP) comes with a rich representation language as well as high end solvers. Thus, ASP has grown to be extremely appealing for the representation and solving of search problems both for business and academia. This particular post concentrates on probably the latest manufacturing uses of ASP.

Eiter et al., (2017) Access to outside info is actually a crucial necessity for Answer Set Programming (ASP), that is a booming declarative problem-solving strategy these days. Outside access not just contains data in formats that are several, but much more basic also the outcomes of computations, and perhaps in a two-way info exchange. Supplying these kinds of access is actually a significant challenge, and particularly in case it must be supported at a generic level, both regarding the semantics as well as effective computation. With this post, we think about problem solving with ASP below outside info access using the dlvhex system. The latter facilitates the access through specific outside atoms, that are two-way API like interfaces involving the rules of the system as well as an outside source. The dlvhex system features a flexible plugin architecture which enables one to utilize several predefined and user defined outside atoms which may be implemented, e.g., in C or Python.

Jain, Sarika & Mishra Tiwari, Sanju. (2014) As a backbone of the Semantic Web, Ontologies present a shared comprehension of a domain name of written text. Ontologies, with the appearance of theirs, consumption, and classification address for concrete ontology language which is essential for the Semantic Web. They may be utilized to help an excellent variety of jobs in various domains including knowledge representation, natural language processing, info retrieval, info exchange, collaborative methods, databases, knowledge management, database integration, digital libraries, info retrieval, or maybe multi-agent methods.

Alviano, Faber and Mario, Wolfgang. (2013) Recently, Answer Set Programming (ASP), logic programming underneath the stable style or maybe solution set semantics, has seen a few extensions by generalizing the idea of an atom in these programs: whether it is aggregate atoms, HEX atoms, generalized quantifiers, or maybe abstract constraints, the concept is actually having more complex satisfaction patterns of the lattice of Herbrand interpretations compared to conventional, easy atoms. Within this paper we refer to any of those constructs as generalized atoms. Many semantics with differing attributes have been suggested for these extensions, rendering the real picture relatively blurry. With this paper, we examine the category of applications which have convex generalized atoms (originally suggested by Truszczynski and Liu) in principle bodies and show this- Positive Many Meanings- because of this category a lot of the proposed semantics coincide.

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. HEX PROGRAM SYNTAX

Let c, X, and G be mutually disjoint sets whose elements are actually known as regular names, adjustable names, and outside predicate names, respectively. Unless explicitly specified, components from X (resp., C) are actually denoted with original letter in top case (resp., lower case), while components from G are actually prefixed with &. We remember that consistent labels work both as specific and predicate labels.

Elements from C ∪ X are called terms. A higher-order atom (or atom) is a tuple (Y0, Y1, . . . , Yn), where Y0, . . . , Yn are terms; n ≥ 0 is the arity of the atom. Intuitively, Y0 is the predicate name, and we thus also use the more familiar notation Y0(Y1, . . . , Yn). The atom is ordinary, if Y0 is a constant.

For example, (x, rdf : type, c), node(X), and D(a, b), are atoms; the rst two are ordinary atoms.

An external atom is of the form

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m), \quad (1)$$

where Y1, . . . , Yn and X1, . . . , Xm are two lists of terms (called input and output lists, respectively), and & g ∈ G is an external predicate name. We assume that &g has xed lengths in(&g) = n and out(&g) = m for input and output lists, respectively. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates.

Example 1 The external atom &reach [edge, a](X) may be devised for computing the nodes which are reachable in the graph edge from the node a. Here, we have that in(&reach) = 2 and out(&reach) = 1.A rule r is of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, not\ \beta_{n+1}, \ldots, not\ \beta_m. \quad (2)$$

where m, k ≥ 0, $\alpha_1, \ldots, \alpha_K$ are atoms, and β1, . . ., βm are either atoms or external atoms. We deFIne H(r) = $\{\alpha_1, \ldots, \alpha_K\}$ and B(r) = B+(r) ∪B−(r), where B+(r) = {β1, . . . , βn} and B−(r) = {βn+1, . . . , βm}. If H(r) = ∅ and B(r) ≠ ∅, then r is a constraint, and if B(r) = ∅ and H(r) ≠∅, then r is a fact; r is ordinary, if it contains only ordinary atoms. Note that in contrast to dl-programs, hex-programs allow for disjunctive heads and constraints. A hex-program is a finite set P of rules. It is ordinary, if all rules are ordinary.

4. SEMANTICS OF HEX-PROGRAMS

We define the semantics of hex programs by generalizing the answer set semantics by Lifschitz as well as Gelfond. To this conclusion, we use the latest idea of a reduct as outlined by Faber et al. (referred to as FLP reduct henceforth) rather than to the standard reduct by Lifschitz and Gelfond. The FLP reduct admits a natural and elegant characterization of answer sets for programs with aggregate atoms, since it guarantees answer set minimality, even though the definition depending on the standard reduct lacks this critical element. In the sequel, allow P be a hex program. The Herbrand platform of P, denoted HBP, would be the set of all the potential ground variations of atoms in addition to outside atoms occurring in P received by replacing variables with constants from C. The grounding of a rule r, grnd(r), is actually defined appropriately, and also the grounding of system P is actually provided by $grnd(P) = \cup r \in P\ grnd(r)$. Unless specified C, X , and G otherwise are implicitly provided by P.

Theorem 1- The answer set semantics of hex programs extends the answer set semantics of average applications as outlined by Gelfond and Lifschitz [1991], and the answer set semantics of HiLog plans as defined by Ross [1994].

Proof. Let P be a hex program with no outside atoms. The semantics of P directly match to the classical answer set semantics.

The succeeding property, that is readily proved, expresses that answer sets adhere to the concept of minimality.

Theorem 2 Every answer set of a hex program P is actually a little style of P.

Proof For starters, we show that a solution set A of P is as well a version of P. This follows out of the reality that each answer set A is actually probably a least type of the FLP reduct of P. Hence, A must gratify each rule r of fP^A. If a principal r was eliminated by the reduct, it's trivially satisfied by A. As a result, A is a unit of P.

Theorem 3 Let P be a hex program without not and constraints. In the event that all outside atoms in grnd(P) are actually monotonic family member to P, then P has several answer set. Additionally, if P is actually disjunction free, it's a single answer set.

Proof. A good system with just montonic outside atoms must have an unit and therefore also a little style. Then again, it additionally has a solution set, since each little design is a response set.

If perhaps P is disjunction free, it's Horn and hence should have a distinctive least model. Notice that this particular property fails whether outside atoms might be non-monotonic. In fact, we are able to quickly model default negation not p(a) by an external atom &not[p](a); the HEX-program p(a) ← &not[p](a) amounts then to the ordinary program p(a) ← not p(a), that has no answer set.

5. COMPUTATION OF HEX-PROGRAMS

The strategy of ours to implementing as well as developing a reasoner for hex programs was using existing solvers as efficiently as you possibly can by integrating them right into a reasoning framework, rather than producing a model generator from scratch. We recognized that existing implementations of ASP reasoners employ extremely advanced as well as techniques that are powerful, which could be reused for this particular novel semantics to an excellent degree. In this chapter we current concepts as well as algorithms for fixing hex programs.

The challenge of employing a reasoner for hex programs lies in the interaction between outside atoms as well as the typical part of an application. Because of the bidirectional flow of info represented by the input list of its, an outside atom can't be evaluated just before the majority of the system. Nevertheless, the presence of efficient and established reasoners for answer set traffic programs led us to the thought of splitting and rewriting the application so that a current answer set solver could be used in turn with the outside atoms' evaluation operates. The reasoner for dl programsby now implemented a naive model of this particular technique, attempting to sort the application in a stratified and an unstratified component and hence speed up the computation. At this point, we wish to get a far more advanced idea of processing the system.In the subsequent subsection, we are going to define ideal notions of dependency which allow us to see a hex program as being a graph to be able to recognize subgraphs with particular properties which could be evaluated individually. This particular dependency info is going to be not unlike the one which was created for dl programs, but additionally much more general, since we've to account for disjunctive heads and also higher order syntax. As a result, we are going to repeat the notion of dependency at a logic plan and improve it exactly where necessary. Additionally, we are going to outline syntactic criteria for protection constraints of hex programs, guaranteeing a limited reasoning domain. In contrast to the therapy of outside evaluations, the next element of hex programs, the higher order syntax, doesn't include this kind of advanced mechanisms. The notion of ours of higher order could essentially be regarded as syntactic sugar and converted to a first order logic software by shifting the predicate within the tuple of debates. As a result, it's adequate to carry out the following replacement before any kind of system evaluation: Each typical atom of the type p(X), the place that the predicate sign p could additionally be a variable, is actually replaced by a first order atom an (p, X), in which n is actually the arity of X.

5.1 Dependency Information

To take the dependency involving bodies as well as heads into consideration is a very common tool for devising an operational semantics for regular logic plans, e.g., by means of the notions of nearby stratification or stratification, or maybe via splitting sets or modular stratification. In contrast to the standard meaning of dependency, like in, we've to consider that in hex programs, dependency involving heads as well as bodies isn't the one likely source of interaction among predicates. Additionally, permitting greater order atoms to possess non ground predicates, we work with a modified notion of dependency between atoms, taking the whole atom and not just the predicate sign of its into consideration. Particularly we are able to have:

Dependency between increased order atoms: For example, p(A C(a and)) are absolutely linked. Intuitively, since C is able to unify with the regular sign p, rules which define C(a) might implicitly determine the predicate p. This is not necessarily the case: for example, rules defining the atom p(X) don't meet up with rules determining a(X), and also H(a, Y) doesn't meet up with H(b, Y)

Dependency through outside atoms: Outside atoms are able to take predicate extensions as feedback: as a result, outside atoms might count on the input predicates of theirs. This is the sole environment where predicate names play an unique function.

Disjunctive dependency: Atoms showing up in the same disjunctive head have a small interaction, since they intuitively are a means for defining a typical nondeterministic search room.

In the following we remember the standard notion of stratification, supplementing the definition probably provided. A plan P is known as stratified, if there's a partition

$$P = P_1 \dot{\cup} \ldots \dot{\cup} P_n \qquad (3)$$

such that the following conditions hold for i = 1, . . . , n:

1. if a relation symbol r occurs positively (i.e., is contained in a positive literal) in a rule in P_i , then its Definition (i.e., the subset of P consisting of all rules where r occurs in the head) within $\cup_{j \le i} Pj$.

2. if a relation symbol occurs negatively (i.e., is contained in a negative literal) in a rule in P_i , then its definition is contained within $\cup_{j \le i} Pj$

According to this definition, P is stratified by $P1 \cup \dot{} \ldots \cup \dot{} P_n$ and each P_i is called a stratum of P.

Naturally, this definition is insufficient for hex-programs, considering that not only external atoms depend from other atoms without occurring in any head, but also external atoms can have non-monotonic behavior and thus must be treated like weakly negated literals regarding stratification. We already noted that, while theoretically an external atom depends on the entire model(s) of the program, in practice we can restrict the input interpretation to specific relations.

Definition 1. Let &g be an external predicate, $f_{\&g}$ its evaluation function, I an interpretation, and X1, . . . , Xn its input list. Then &g is associated with a type signature (t1, . . . , tn), where each t_i is the type associated with Xi and can either be c or p. If t_i is c, then we assume that Xi is a constant, otherwise we assume that Xi is a predicate symbol. f&g depends only on those atoms in I that have a predicate symbol p equal to some $Xi \in X1, \ldots, Xn$ with t_i = p.

In order to be able to identify a reasonable dependency structure, In practice we do not allow to specify variables for input terms of type p. Otherwise the calculation of the part of the program that such an external atom depends on would quickly become very complex.

Definition 2. Let P be a program and a, b atoms occurring in some rule of P. Then, a depends positively on b ($a \rightarrow_p b$), if one of the following conditions holds:

1. There is some rule r ∈ P such that a ∈ H(r) and b ∈ B^+(r)

Example: $r1 : p(X) \leftarrow q(X), r(X).$

Clearly, we have $p(X) \rightarrow {}_q q(X)$ and $p(X) \rightarrow {}_p r(X)$.

2. There are some rules r1, r2 ∈ P such that a ∈ B(r1) and b ∈ H(r2) and there exists a partial substitution θ of variables in a such that either aθ = b or a = bθ. E.g., H(a, Y) unifies with p(a, X).]

Example: $r1 : p(X) \leftarrow q(X), r(X).$

$r2 : q(Y) \leftarrow s(Y).$

Since q(X) unifies with q(Y), we have $q(X) \rightarrow {}_p q(Y)$.

3. There is some rule r ∈ P such that a, b ∈ H(r). Note that this relation is symmetric.

Example: $r1 : p(X) \vee q(X) \leftarrow r(X).$

From this we get $p(X) \rightarrow {}_p q(X)$ and $q(X) \rightarrow {}_p p(X)$.

Furthermore, a depends externally on b ($a \rightarrow_e b$), if one of the following conditions holds:

1. a is an external predicate of form $\&g[\bar{X}](\bar{Y})$ with a type signature (t1, . . . , tn), where $\bar{X}$ = X1, . . . , Xn, b is of form $p(\bar{Z})$, and, for some i, X_i = p and t_i = p.

Example: $r1: num(N) \leftarrow \&count[item](N).$

$r2: item(X) \leftarrow part(X).$

Here we have &count[item](N)→$_e$ item(X), if the input term item is of type p instead of merely denoting a constant string.

2. There is some rule r ∈ P with a, b ∈ B(r) such that a is an external predicate of form &g[$\bar{X}$]($\bar{Y}$) where $\bar{X}$ = X_1, . . . , Xn, and b is of form p($\bar{Z}$), and $X^- \cap Z^- 6= \emptyset$. Example: r1 : reached(X) ← &reach[N, edge](X), startnode(N). This causes &reach[N, edge](X)→$_e$ startnode(N).

Moreover, a depends negatively on b (a→$_n$b), if there is some rule r ∈ P such that either a ∈ H(r) and b ∈ B^-(r) or b is a non-monotonic external atom.

We say that a depends on b, if a→ b, where → = →p ∪ →e ∪ →n . The relation →+ denotes the transitive closure of →. We say that a strictly depends on b, or a 7→ b, if a→$^+$ b, but not b $b \nrightarrow^+$ a.

These dependency relations let us construct a graph GP, which we call the dependency graph of the corresponding program P.

Definition 3 Let P be a hex-program. A dependency graph GP of P consists of the set VP that contains all atoms in P (i.e., the vertices of GP) and the set EP of dependency relations contained in P according to Definition 4.6.2 (i.e., the edges of GP).

Note that this definition is based on a non-ground hex-program P.

Example 1 Consider the following program, modeling the search for personal contacts that stem from a FOAF-ontology,3 which is accessible by a URL.

(1) $url(\text{“}http://www.kr.tuwien.ac.at/staff/roman/foaf.rdf\text{”}) \leftarrow;$

(2) $url(\text{“}http://www.mat.unical.it/\tilde{}ianni/foaf.rdf\text{”}) \leftarrow;$

(3) $\neg input(X) \vee \neg input(Y) \leftarrow url(X), url(Y), X\ 6 = Y;$

(4) $input(X) \leftarrow not\ \neg input(X), url(X);$

(5) $triple(X,Y,Z) \leftarrow \&rdf\ [A](X,Y,Z), input(A);$

(6) $name(X,Y) \leftarrow triple(X, \text{“}http://xmlns.com/foaf\ /0.1\ /name\text{”}, Y);$

(7) $knows(X,Y) \leftarrow name(A,X), name(B,Y),$

$triple(A, \text{“}http://xmlns.com/foaf\ /0.1\ /knows\text{”}, B).$

Probably the first 2 facts specify the URLs of the FOAF ontologies we would like to query. Rules three as well as four make certain that each answer set is going to be based on an one-time URL only. Rule five extracts all triples from an RDF file specified by the extension of input. Rule six converts triples that assign names to people to the predicate name. Lastly, the final rule traverses the RDF graph to put together the relation understands. Figure 2 displays the dependency graph of P.

We are able to now define several structural qualities of hex-programs.

Definition 4 Let P be a hex-program and → the relation defined above. We say that P is

(i) nonrecursive, if → is acyclic;

(ii) stratied, if there is no cycle in → containing some atom a and b such that a→$_n$b;

(iii) e-stratified, if there is no cycle in → containing some atom a and b such that a→$_e$b; and

(iv) totally stratified, if it is both stratified and e-stratified.

For example, the application in Example 1 is completely stratified as the sole cycle is actually brought on by the disjunction in Rule (three) and doesn't include negation. Just before we show the way to process the graph to be able to calculate the answer sets of a hex program, we first have to reply to the question how you can deal with the likely infinite domain of a hex program.

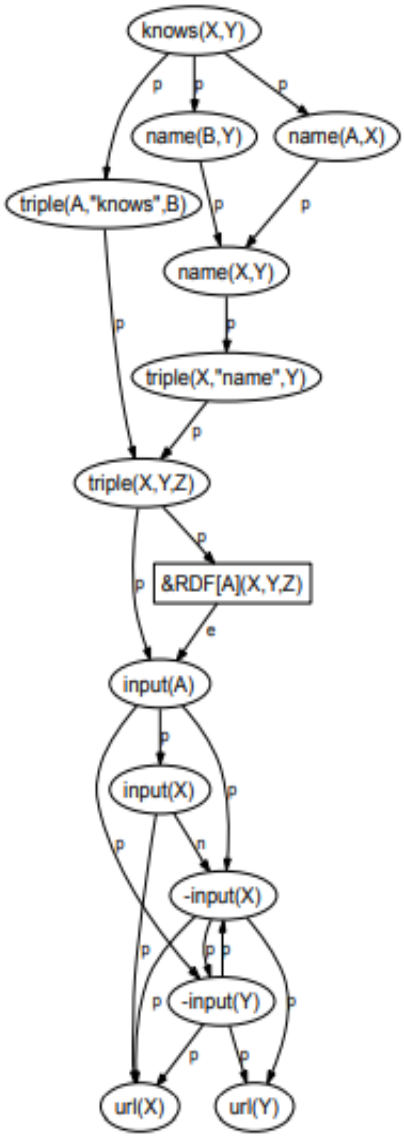

Figure 2: FOAF program graph

5.2 Infinite Domains

Provided a hex program P, its grounding grnd(P) is actually infinite in common, as well as can't be cut down straightforwardly to a limited portion since, provided an outside predicate &g, the co-domain of F&g is actually unfamiliar and perhaps infinite. It's therefore vital that you impose 2 restrictions:

1. We think that for a single particular ground input tuple, an outside atom just returns a limited set of paper tuples. Or else, finiteness might certainly not be assured, independently of the program's structure.

2. We limit the use of outside predicates inside a hex program in phrases of stratification to be able to bound the selection of symbols to be taken into consideration to a limited number, whilst outside knowledge in terms of new symbols can continue to be brought into an application.

In the following, we are going to describe the next state of detail, starting with the definition of rule safety

Definition 5 Given a rule r, the set of safe variables in r is the smallest set X of variables such that

(i) X appears in a positive ordinary atom in the body of r, or

(ii) X appears in the output list of an external atom &g[Y1, . . . , Yn](X1, . . . , Xm) in the body of r and Y1, . . . , Yn are safe.

A rule r is safe, if each variable appearing in a negated atom and in any input list is safe, and variables appearing in H(r) are safe.

For instance, the rule$r: C(X) \leftarrow url(U), \&rdf\ [U](X, "rdfs: subClassOf", C)$ is healthy. Intuitively, this particular notion captures those rules for which input to outside atoms will be driven by means of additional atoms in the identical rule. Because of the extension of the predicate url, the amount of pertinent ground instances of r intuitively is limited and could be driven by repeated calls to F&rdf.In certain instances, safety isn't sufficient for determining finiteness of the set of pertinent symbols to be had in account. This inspires the following stronger notion:

Definition 6 Let r be a rule in P with external atoms $\&f_1[\bar{Y}_1](\bar{X}_1), \ldots \&fn[\bar{Y}_n](\bar{X}_n)$ in B(r) and E be the set of all variables in $\cup_{i \leq n} \bar{Y}_i$. Moreover, let S be the set of atoms b ∈ B^+(r) such that each atom a ∈ H(r) strictly depends on b. Let V be the set of all variables that occur in the ordinary atoms in S and all variables in the output list of the external atoms in S. Let G be the set of all predicate symbols in $\cup_{i \leq n} \bar{Y}_i$. Then, r is strongly safe, iff (i) E ⊆ V and (ii) each atom a ∈ H(r) strictly depends on all p ∈ G.

Informally, a principle is clearly safe, in case the outside atoms of its receive the feedback of theirs from a lower stratum of the system. By doing this, even in case they arise in a cycle, the output can't of theirs grow infinitely, since the dimensions of the input of theirs is actually fixed before entering" the cycle. The rule r above isn't really protected. In fact, in case some outside URL invoked by means of &rdf has several triple of form $(X, "rdfs: subClassOf\ ", url)$, the extension of the url predicate is potentially innite. The rule

$r': instanceOf\ (C, X)$
$\leftarrow url(U), \&rdf\ [U](X, "rdfs: subClassOf\ ", C)$

is strongly safe, if url(U) does not depend transitively on instance Of (C, X).

The strong safety condition is, anyway, only needed for rules which are involved in cycles of →. In other settings, the ordinary safety restriction is enough. This leads to the following notion of a domain-expansion safe program. Let $grnd_U$(P) be the ground program generated from P using only the set U of constants.

Definition 7 A hex-program P is domain-expansion safe iff each rule r ∈ P is safe and each rule r ∈ P containing some external atom b ∈ B(r) is strongly safe.The following theorem states that we can effectively reduce the grounding of domain expansion safe programs to a finite portion.

Theorem 4 For any domain- expansion safe hex-program P, there exists a finite set D ⊆ C such that $grnd_D$(P) is equivalent to $grnd_C$ (P) (i.e., has the same answer sets).

Proof (sketch).

The evidence proceeds by given that, though the Herbrand universe of P is actually in concept infinite, just a limited set D of constants may be taken into consideration. From D, a limited ground plan, $grnd_D$(P), could be utilized for computing answer sets. Provided P is actually domain expansion safe, it could be proven that $grnd_D$(P) has the identical answer sets as $grnd_C$(P).

A system which incrementally creates Grndd(P and D) could be sketched as follows: We upgrade a set of energetic regular atoms A along with a set R of ground rules (both of them at first empty) by means of a characteristic ins(r, A), and that is frequently invoked over all rules $r \in P$ until A and R achieve a fixed point. The feature ins(r, A) is actually so that, given a secure rule r along with a set A of atoms, it returns the set of all the soil variations of r so that each of its body atom an is actually either (i) such that a ∈ A or (ii) if a is external, f_a is true. D is the final value of A, and R = $grnd_A$(P). It can be shown that the above algorithm converges and $grnd_D$(P) ⊆ grnd C (P). The program $grnd_C$(P) can be split into two modules: N_1 = $grnd_D$(P) and N2 = $grnd_C$(P) \ $grnd_D$(P). It holds that each answer set S of $grnd_C$(P) is such that S = $S_1 \cup S_2$, where S_1∈ AS(N'$_1$) and S2 ∈ AS(N'$_2$). N'$_1$is a version of N_1 enriched with all the ground facts in AS(N_2). Also, we can show that the only answer set of N_2 is the empty set. From this the proof follows.

5.3 Splitting Algorithm

Like the techniques discussed in Subsection 3.9.1, the concept of analysis of a hexprogram relies on the concept of splitting sets. Intuitively, provided a system P, a splitting set S is actually a set of terrain atoms which cause a sub program $grnd(P') \subset grnd(P)$ whose models$M = \{M_1, \ldots, M_n\}$ may be evaluated individually. Next, a sufficient splitting theorem shows how you can plug the designs Mi

from M into an altered version of P/P' so that the complete variants might be computed. At this point, we work with a modified idea of splitting set, accommodating non ground applications and suited to the definition of ours of dependency graph.

Definition 8 A worldwide splitting set for a hex program P is actually a set of atoms A appearing in P, so that every time $a \in A$ and a→b for several atom b appearing in P, and then also b∈ A.

Furthermore, we determine another kind of splitting set:

Definition 9 A local splitting set for a hex-program P is a set of atoms A ⊆ VA, such that for each atom a ∈ A there is no atom b ∉ A such that a→ b and b $\rightarrow^{+}$ a.

Thus, contrary to a global splitting set, a local splitting set does not necessarily include the lowest layer of the program, but it never breaks a cycle.

Definition 10 The bottom of P w.r.t. a set of atoms A is the set of rules $b_A(P) = \{r \in P \mid H(r) \cap A\, 6 = \emptyset\}$.

In other words, the bottom of P w.r.t. a set of atoms A includes all those rules that define A, i.e., whose head atoms occur in A.

Theorem 5 Let P be a hex program and permit A be a worldwide splitting set for P. Then M is actually a solution set of P I M is actually an answer set of P zero, in which P zero will be the system received by removing b_A (P) from P and adding the literals in N as facts to N and P is actually an answer set of b_A(P).

Proof. The evidence is actually mutatis mutandis as the one of Theorem 3.9.1, changing the solid reduct by the FLP reduct.

Apart from these definitions, we'll next explain several preparations of the initial hex program to be able to have the ability to apply the thought of splitting sets by processing the program's dependency graph.

To prepare the Program

From the viewpoint of program evaluation, it turns out to be impractical to determine the semantics of an outside predicate by means of a Boolean feature. Instead, we want a purposeful characterization, which gives a set of paper tuples for a certain input tuple. To this conclusion, we define $F_{\&g} : 2^{HBp} \times D1 \times \cdots \times Dn \rightarrow 2^{R_c^m}$ with $F_{\&g}$(I, y1, . . . , yn) =(x1, . . . , xm) iff $f_{\&g}$(I, y_1, . . . , yn, x1, . . . , xm) = 1, where R_c^m is the set of all tuples of arity m that can be built with symbols from C and Di = C for 1 ≤ i ≤ n. With this notion, we can compute the entire output of an external atom with a ground input list.

If the input list y1, . . . , yn isn't ground in the initial application, the restriction of domain expansion security for hex programs guarantees that the values of its could be driven out of the remaining rule physique. Nevertheless, since the fundamental concept of the algorithm of ours is usually to split up the application along outside atoms and assess the ensuing areas in turn, we may lose precisely this context of an outside atom. As an example, this, let us 1st think about the next principle from the introduction:

$$reached(X) \leftarrow \&reach[edge, a](X) \qquad (4)$$

the place that the very first input phrase is actually of sort p, i.e., edge is actually translated as a predicate name along with an is actually of sort c, i.e., a constant. At this point, the feedback is totally determined w.r.t. to an interpretation and the outside evaluation feature could F& reach could be evaluated with the feedback I, edge, a (assuming I has facts about advantage along with which have been computed before). These days' things aren't as straightforward if variables are actually utilized at the input list, as in Rule (five) of Example 1:

$$triple(X, Y, Z) \leftarrow \&rdf\ [A](X, Y, Z), input(A) \qquad (5)$$

Obviously, before we are able to assess this atom the input of its must be grounded. But since these ground values count on a predicate which happens in the same rule frame, feedback, we have to include an auxiliary rule to the program:

$$triple(X, Y, Z) \leftarrow \&rdf\ [A](X, Y, Z), input(A)$$

$$\mathrm{rd}_{\mathrm{finp}}(A) \leftarrow input(A). \quad (6)$$

Obviously, the mind predicates of such rules should be exclusively connected with the respective outside atom. Putting in these kinds of rules will in addition affect GP appropriately.

We are able to generalize these auxiliary rules by the following definition:

Definition 11 Let P be a hex-program and &g[$\bar{Y}$]($\bar{X}$) be some external atom with input list $\bar{Y}$occurring in a rule r ∈ P. Then, for each such atom, a rule $r^{\&g_{inp}}$ is composed as follows:

- The head H($r_{inp}^{\&g}$) contains an atom ginp($\bar{Y}$) with a fresh predicate symbol g_{inp}.
- The body B($r_{inp}^{\&g}$) of the auxiliary rule contains all body literals of r other than &g[$\bar{Y}$]($\bar{X}$)that have at least one variable in its arguments (resp. in its output list if b is another external atom) that occurs also in $\bar{Y}$.

For each external atom in P we can create such a rule. We denote the set of all such rules with $P_{inp.}$

The evaluation algorithm will ensure that the extension of g_{inp} is known before an external atom &g i.e., the function $F_{\&g}(I, y_1, \ldots, y_n)$ needs to be evaluated. For atoms with an input list which was currently ground in the initial application, this extension coincides with the one this kind of input tuple. For enter prospect lists with variables, the extension might include much more ground tuples or zero, each of that will be feedback to the outside evaluation feature. Remember that there's no need to change the initial rule. The goal of the auxiliary rules is simply to introduce an extra advantage in the dependency graph, giving us the chance to split the graph there and calculate the input of the outside atom before proceeding with the first rule itself. We all know right now how you can put together the info flow from the system to the outside atom. The sole part of our interfacing machinery we're currently lacking before we are able to process the dependency graph is actually a method of importing the outside evaluation consequence into the system. To this conclusion we want another definition:

5.4 Evaluation Algorithm

The evaluation algorithm within utilizes the following subroutines:

$eval(comp, I)$ Computes the styles of an outside component comp (which is actually of one of the kinds discussed above) for the interpretation I; I is actually added as a set of facts to each outcome model

$solve(P, I)$ Returns the answer sets of P ∪ A, in which P doesn't include some outside atom and A is actually the set of facts which corresponds to I.

Intuitively, the algorithm traverses the dependency graph from bottom to top, slowly pruning it while computing the respective versions. Just before entering the loop, the auxiliary rules from Definition 10 are actually added to the application in Step (a), followed by the identification of the dependency graph as well as the outside components. M belongs to the set of the present designs after each iteration, beginning with the individual set of facts F of the system. Step (b) singles through just about all outside parts which don't rely on any additional atom or maybe component, i.e., which are actually on the bottom part of the dependency graph. Next, the algorithm loops with all present versions M ∈ M. For this kind of model those elements are actually evaluated in Step (c) and also may be taken out of the list of outside parts which are left to be resolved. In this particular innermost loop, the outcomes of all elements are actually built up in M' and after that added to M'' Moreover, Step (d) guarantees that almost all rules of the elements are actually taken out of the system. M has the outcome of all parts with all present versions. By the staying a part of the graph, Step (e) extracts probably the largest possible subprogram which doesn't rely on any remaining outside component i.e., that's once again on the bottom part of the graph. Following computing the styles of this particular subprogram with regard to the present consequence, it's taken out of the system resp. the dependency graph.

Algorithm 1: Dependency graph evaluation.

Input: a HEX-program P
Result: a set of models $\mathcal{M}$
(a) $P \Leftarrow P \cup P_{inp}$
Determine the dependency graph G_P for P
Find all external components C_i of P and build $Comp = \{C_1, \ldots, C_n\}$
$T \Leftarrow Comp$
$\mathcal{M} \Leftarrow \{F\}$, where F is the set of all facts originally contained in P
while $P \neq \emptyset$ **do**
(b) $\overline{T} \Leftarrow \{C \in T \mid \forall a \in C$: if $a \rightarrow b \in V_G$ then $b \in C\}$
$\mathcal{M}'' \Leftarrow \emptyset$
forall $M \in \mathcal{M}$ **do**
$\mathcal{M}' \Leftarrow \{M\}$
$\mathcal{C} \Leftarrow \emptyset$
forall $C \in \overline{T}$ **do**
(c) $\mathcal{C} \Leftarrow \bigcup_{M' \in \mathcal{M}'} \mathsf{eval}(C, M')$
$\mathcal{M}' \Leftarrow \mathcal{C}$
end
$\mathcal{M}'' \Leftarrow \mathcal{M}'' \cup \mathcal{M}'$
end
$Comp \Leftarrow Comp \setminus \overline{T}$
(d) $P \Leftarrow P \setminus b_c(P)$ with $c \in \overline{T}$
$\mathcal{M} \Leftarrow \mathcal{M}''$
$\overline{C} \Leftarrow \{u \in V_G \mid u \in C$ or $u \rightarrow^+ c, c \in C$ for any $C \in Comp\}$
(e) $P' \Leftarrow P_{hex} \setminus b_{\overline{C}}$
$\mathcal{M} \Leftarrow \bigcup_{M \in \mathcal{M}} \mathsf{solve}(P', M)$
$P \Leftarrow P \setminus P'$
remove all atoms from the graph that are not in $\overline{C}$
end
return $\mathcal{M}$

Essentially, the iteration traverses the system graph by using 2 distinct evaluation capabilities each turn. While eval solves little subprograms with outside atoms, comp solves maximum non external subprograms.

6. CONCLUSION

We defined the semantics of common hex programs by generalizing the conventional answer set semantics, defining the model of an outside atom and making use of the FLP reduct rather than the classical GL reduct. We've sketched common examples of the usability of outside atoms in the domain of Semantic Web applications, like importing and manipulation outside theories or perhaps defining certain ontology semantics by rules of a hex program. We then have outlined the normal idea of computation of a hex program, relying on the dependency info which underlies the system. We targeted at integrating present reasoners rather than producing a completely brand new one and thus created a framework which builds upon readily available answer set solver and also the engines behind outside atoms. From this particular viewpoint, we made an algorithm which creates the styles of a hex program by decomposing it and consequently calling the outside reasoners. Additionally, we specified syntactic protection constraints to guarantee decidability. A prototype implementation, known as dlvhex, of the techniques was developed, together with a selection of so-called plugins which offer a number of outside atoms, interfacing for example RDF repositories, DL knowledge bases, the WordNet database or perhaps easy, but helpful string operations. We created a software program architecture which adheres to prevalent software standards and also allows for fast improvement of custom plugins.

REFERENCES

[1] Alviano, Mario & Faber, Wolfgang. (2013). Properties of Answer Set Programming with Convex Generalized Atoms.

[2] Banbara, Mutsunori & Soh, Takehide & Tamura, Naoyuki & INOUE, KATSUMI & Schaub, Torsten. (2013). Answer set programming as a modeling language for course timetabling. Theory and Practice of Logic Programming. 13. 10.1017/S1471068413000495.

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

[4] Eiter, Thomas & Ianni, Giovambattista & Krennwallner, Thomas. (2009). Answer Set Programming: A Primer. Lecture Notes in Computer Science. 5689. 40-110. 10.1007/978-3-642-03754-2_2.

[5] 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.

[6] Falkner, Andreas & Friedrich, Gerhard & Schekotihin, Konstantin & Taupe, Richard & Teppan, Erich. (2018). Industrial Applications of Answer Set Programming. KI - Künstliche Intelligenz. 32. 1-12. 10.1007/s13218-018-0548-6.

[7] Jain, Sarika & Mishra Tiwari, Sanju. (2014). Knowledge Representation with Ontology

Tools & Methodology. International Journal of Computer Applications. 6. 975-8887.

[8] Mallat, Souheyl & Hkiri, Emna & Maraoui, Mohsen & Zrigui, Mounir. (2015). Semantic Network Formalism for Knowledge Representation:. International Journal on Semantic Web and Information Systems. 11. 64-85. 10.4018/IJSWIS.2015100103.

[9] Niepert, Mathias & Buckner, Cameron & Allen, Colin. (2008). Answer Set Programming on Expert Feedback to Populate and Extend Dynamic Ontologies.. 500-505.

[10] Rettinger, Achim & Lösch, Uta & Tresp, Volker & d'Amato, Claudia & Fanizzi, Nicola. (2012). Mining the Semantic Web. Data Mining and Knowledge Discovery. 24. 10.1007/s10618-012-0253-2.