Answering Why and How questions with respect to a frame-based knowledge base: a preliminary report

Chitta Baral, Nguyen Ha Vo, and Shanshan Liang

School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, Arizona, USA
chitta@asu.edu, nguyen.h.vo@asu.edu, shanshan.liang@asu.edu

Abstract

Being able to answer questions with respect to a given text is the cornerstone of language understanding and at the primary school level students are taught how to answer various kinds of questions including why and how questions. In the building of automated question answering systems the focus so far has been more on factoid questions and comparatively little attention has been devoted to answering why and how questions. In this paper we explore answering why and how questions with respect to a frame-based knowledge base and give algorithms and ASP (answer set programming) implementation to answer two classes of questions in the Biology domain. They are of the form: "How are X and Y related in the process Z?" and "Why is X important to Y?"

1998 ACM Subject Classification D.1.6 Logic Programming, H.3.4 Question-answering (fact retrieval) systems, I.2.4 Frames and scripts

Keywords and phrases answer set programming, frame based knowledge representation, question answering.

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

In recent years question answering (QA) has become more prominent via efforts such as the Google Knowledge Graph [11] and systems such as Watson [7]. However, most question answering efforts remain focused on factoid questions; a notable exception being navigational "How" questions answered by Siri.

"How" and "Why" questions are important types of questions that are introduced to students at primary school level in their reading and comprehension classes. At the school level answering why questions involves finding the reason or cause of a thing that happened and answering how questions involves finding the way something is done. Answering such questions become more elaborate in Biology where some researchers suggest [15] three kinds of answers to "Why" questions: teleological answer about effects, proximate answers about immediate causes and evolutionary answers based on natural selection; while others [16] propose an even more elaborate categorization of questions and answers such as: How is X used (asked for the biological role/function), How does X work (asked for physiological explanation), and Why does X has a certain item/behavior (asked for the functional significance of certain biological roles). In the literature [1] "How" questions have been referred to as procedural questions.

At present automatic answering of "Why" and "How" questions with respect to large text corpuses [12] are based on factoid extraction where answers are located by looking for

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–11

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

associate words and phrases such as "because of" and "causes". In this paper we take a different approach. Instead of answering "Why" and "How" questions with respect to natural language text [3] , we explore answering them with respect to a frame based knowledge base. Our motivation behind that is to first formalize the notion of answers to such questions; I.e., define what are answers to "Why" and "How" questions with respect to a knowledge base.

We use the frame based biology knowledge base AURA [5] and while identifying several question forms we focus on two specific question forms as a start: "How are X and Y related in the process Z?" and "Why is X important to Y?" Looking at examples in the frame based knowledge representation in AURA we define the notion of an *event description graph* and formalize the answers to our two question types with respect to such graphs. We then give an answer set programming formalization of the reasoning process to find the answers (and thus give an implementation) and conclude with future research directions. Our answer set programming formalization builds up on our earlier work [4] to reason with frame based knowledge using answer set programming.

2 Background

2.1 Frame-Based Knowledge Base

The basic aspects of a frame-based knowledge base (KB) is to represent classes and objects (instances). For classes, the most important information is the class hierarchy. For example[1], the highest class in the AURA[2] [5] hierarchy is "Thing", with two children classes "Entity" and "Event". "Entity" can have descendent classes such as "Cell", "Sunlight", "Sugar" that are biological entities, while "Event" can have descendent classes such as "Photosynthesis", "Mitosis" that are biological processes. We also need to represent objects that may belong to the same classes (share the same basic features), but have their own specific properties. To represent the shared features amongst objects (in order to prevent repetitive encoding of the same set of knowledge entries), "prototypes" of classes are encoded and during reasoning they are cloned by all the objects from that class. The KB normally supports the encoding of multiple inheritance, meaning that a class need to inherit from all of its ancestor classes in the class hierarchy. In this case, in order to obtain the full information for an object, the object needs to clone from all the prototypes of the class it belongs to, as well as the prototypes of all its ancestor classes. When merging the information together, the process of "unification" [6] is introduced to make sure that any conflicts are dealt with properly.

In general, although there is a large body of knowledge bases that use the frame based approach [8], there hasn't been much research on how to use the knowledge encoded in frames declaratively, especially in the realm of question answering applications. In our earlier work [4] we investigated how to utilize the KB for answering "what" questions in a declarative way, as opposed to the procedural approach adopted by the original AURA system. There we gave an abstract definition of a KB, and a declarative implementation of "clone and unify". From here on, whenever we refer to an object we use the complete information for that object (after the cloning and unification process), which is obtained by the declarative implementation mentioned earlier.

[1] Note that the various examples mentioned in this paper are from the AURA knowledge base, some with slight modifications.

[2] The AURA knowledge base is a frame-based KB developed manually by knowledge experts. AURA contains large amount of frames describing biology concepts and biology processes, and has been used to answer a wide variety of "what" questions [5].

To the best of our knowledge, there has been little research on answering "How" and "Why" questions with respect to frame-based knowledge bases. The main goal of this paper is to provide insight on how frame based KB can be used to answer some "Why" and "How" questions. To do that we use an "abstract view" of the KB that allows a better illustration of the semantics behind the KB and how they can be used for QA purposes.

2.2 Answer Set Programming

We use Answer Set Programming (ASP) [10] as our knowledge representation language for its strong theoretical foundation [2], expressiveness, the availability of various solvers [9, 14, 13] and its earlier use in the declarative implementation of "clone and unify".

An ASP program is a collection of rules of the form:

$$a \leftarrow a_1, \ldots, a_m, \mathit{not}\ b_1, \ldots, \mathit{not}\ b_n$$

where a, $a_1, ..., a_m$ and $b_1, ..., b_n$ are atoms. The rule reads as "a is true if $a_1 ... a_m$ are all known to be true and $b_1 ... b_n$ can be assumed to be false". The semantics of answer set programs are defined using answer sets (earlier called stable models).

3 Answering two Why/How Questions

As mentioned earlier, in this paper we consider two particular types of Why and How questions: "How are X and Y related in process Z?" and "Why is X important to Y?".

Let us illustrate them with respect to a knowledge base about the process of photosynthesis. The following component of an event description graph (to be formally defined later) expresses the knowledge about photosynthesis.

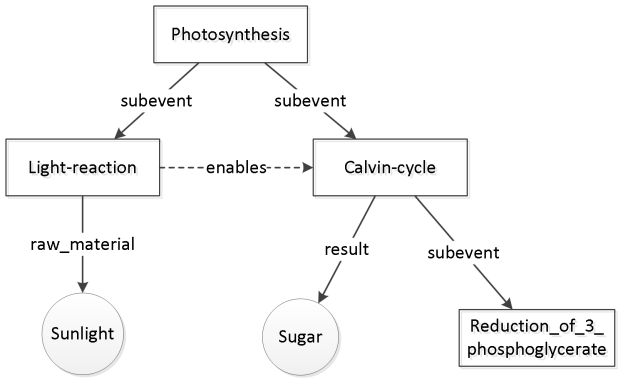

Figure 1 *The event description graph of photosynthesis. Events and entities are depicted by rectangles and circles respectively. Compositional edges are represented by solid lines and behavioral edges by dashed lines.*

Now consider the "How" question: How are sunlight and sugar related in photosynthesis?

An intuitive answer to this question is: *Photosynthesis has two subevents: light reaction and calvin cycle. The light reaction needs sunlight as its raw material, and later enables the calvin cycle which produces sugar as the result.* This answer can be obtained from the graph in Fig. 1 constructed from the frame based knowledge base AURA by using the information that "raw material", "enables", and "result" are the key slots used by AURA.

Now let us consider the "Why" question: Why is sunlight important to photosynthesis?

An intuitive answer to this question is: *Sunlight is the raw material of light reaction thus sunlight is important to light reaction; light reaction is an important sub-event of photosynthesis; therefore sunlight is also important for photosynthesis.* This answer can be obtained from the graph in Fig. 1 when augmented with information about "importance". Following is such an augmented graph.

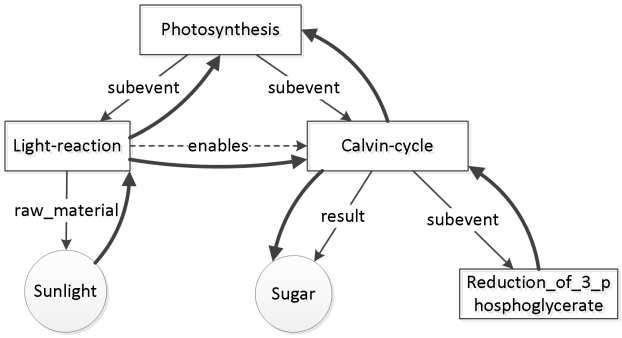

Figure 2 *The event description graph of photosynthesis with the "important edges" marked by bold arrow.*

Using the augmented graph we need to follow the "important" edges that link "sunlight" to photosynthesis.

The above examples suggest close relationships between the answers of why and how questions and the graph representation of processes. In the following we give a formal representation of processes as graphs, define some generic operations on the graphs and use them in formulating answers to our two kinds of why and how questions.

3.1 Knowledge Bases of Biological Processes

In the frame representation that we use in [4] the Knowledge Base has the generic encoding format: $has(X, S, V)$, where X can be either a class or an object, S refers to a "slot", which describes the property of X, and V is the value for that slot. While the KB may contain a large amount of information, we do not need all of that for our specific types of question answering. Thus we consider and define a simplified view of the KB through the notion of Event Description Graphs.

There are two important aspects of a Knowledge Base of Biological Processes: Events and Entities. Each biological process is a event, which can often be broken down to several sub-events (and sub-events of sub-events). Entities can be involved in the processes as raw materials, results, bases, objects, etc.[3] Using that we now define Event Description Graphs.

▶ **Definition 1.** An Event Description Graph is a directed graph with two types of nodes: event nodes and entity nodes; two types of directed edges: compositional edges and behavioral edges; and a special node referred to as the main event node or the root node, which has no incoming edge. An Event Description Graph satisfies the following conditions:

1. All other nodes beside the root are reachable from the root via compositional edges.
2. There are no directed cycle of only compositional edges.

[3] For a complete list such relations (slot names), please refer to the Slot Dictionary in the Component Library (http://www.cs.utexas.edu/ mfkb/RKF/tree/).

3. There are no directed cycle of only behavioral edges.
4. There are no outgoing edges from the entity nodes.

We use $EDG(Z)$ to denote the Event Description Graph with root Z.

"Event nodes" and "entity nodes" represent biological processes and biological entities respectively. The "compositional edges" and "behavioral edges" are categorized based on specific event-event and event-entity relations. Table 1 shows some example relations that can be viewed as compositional and behavioral edges. For event-to-event relation, only the "sub-event" relation is viewed as a compositional edge, while others are viewed as behavioral edges. All the event-to-entity relations are considered to be compositional edges.

Each Event Description Graph describes its root event which is a biological process defined in the KB. As all the sub-events are also biological processes, the subgraph with a sub-event as root and that contains all the accessible nodes/edges from that sub-event is considered the Event Description Graph for that sub-event.

Category	Type	Slot names
Event-to-Event	compositional	sub-event
Event-to-Event	behavioral	next_event, enables, causes, prevents...
Event-to-Entity	compositional	raw_material, result, site, location, base, agent...
Event-to-Entity	behavioral	(null)

Table 1 The slot names indicating "compositional"/"behavioral" edges.

A *cpath* from a node X to a node Y in $EDG(Z)$, denoted as $cpath(X, Y)$, is a path consisting of only compositional edges. Similarly, a $bpath(X, Y)$ is a path consisting of only behavioral edges, and an $ipath(X, Y)$, is a path consisting of only "important edges". While $cpath(X, Y)$ and $bpath(X, Y)$ reflect how X and Y are connected compositionally or behaviorally, sometimes we need to add richer semantic information such as an edge being important which is then used to define $ipath(X, Y)$. Intuitively we say that there is an "important edge" from X to Y iff Y can not function properly without X. The following Table 2 shows several functionally important relations.

Category	Slot names
Entity-to-Event	raw_material, site, base
Event-to-Event(explicit)	enables, causes, regulates, prevents, subevent
Event-to-Event (implicit)	(sample rule) the result of E1 is the raw_material of E2
Event-to-Entity	result

Table 2 The slot names indicating "functional importance".

3.2 Answers to two types of Why/How Questions

In this subsection we will formally define the answers to two types of Why and How questions. We will illustrate the definitions and algorithms using the following event description graph.

Given the event description graph of process 1 in Figure 3 consider answering the question "How are process 8 and entity 10 related in process 1?". Intuitively, it seems the answer should only contain important information to understand the relation between 8 and 10 such as: compositional path from process 3 to process 8 through process 5 and compositional

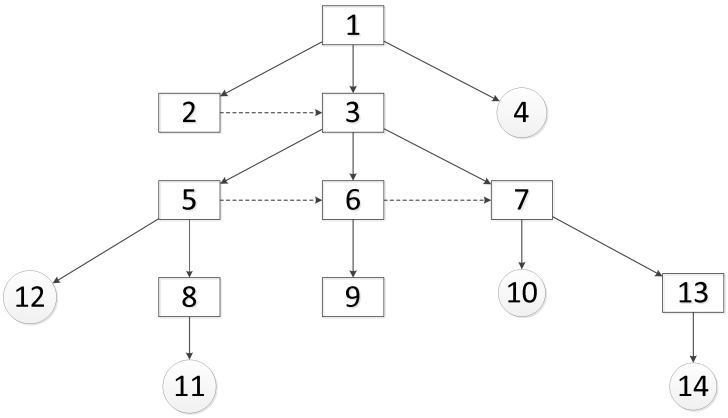

Figure 3 Event Description Graph of process 1. Events and entities are depicted by rectangles and circles respectively. Compositional edges are represented by solid lines and behavioral edges by dashed lines.

path from 3 to 10 through 7 to explain compositional relations between 8 and 10; behavioral path from 5 to 7 through 6 to explain behavioral relations; and compositional edge from 1 to 3 and then to 6 to clarify process 6. Information about process/entity 2, 4, 9, 11, 12, 13 and 14 can be omitted since they are not important for the connection between 8 and 10. Following the above intuition, we formally define the answer for question "How are X and Y related in process Z" as the graph denoted by $MIN\_EDG^{Z}_{X,Y}$ defined as follows. First, $LCA(X, Y)\, in\, EDG(Z)$, denotes the lowest common ancestor of X and Y in $EDG(Z)$.

▶ **Definition 2.** Given an event description graph EDG(Z) and two nodes X and Y in that graph, $MIN\_EDG^{Z}_{X,Y}$ is the subgraph of EDG(Z) consisting of the following:

- The set of nodes $V_X \cup V_Y \cup V_{behavioral} \cup V_{ZT}$, where $T = LCA(X, Y)$ in $EDG(Z)$; V_x and V_y are the set of nodes in $cpath(T, X)$ and $cpath(T, Y)$ respectively; $V_{behavioral}$ is the set of nodes on any $bpath(X', Y')$, where $X' \in V_x$, and $Y' \in V_y$; and V_{ZT} is the set of nodes in $cpath(Z, T)$.
- The set of edges consisting of the union of the edges obtained from $EDG(T)$ by removing all edges that connect to the nodes in $EDG(T) \setminus V$ and the edges in $cpath(Z, T)$ from Z to T in $EDG(Z)$.

The path $cpath(Z, T)$ from Z to T helps clarify what T is with respect to the process Z. With respect to relating 8 and 10 in Figure 3, this $cpath(Z, T)$ is the path from process 1 to process 3.

▶ **Example 3.** Let us consider the photosynthesis example. $LCA(sunlight, sugar)$ in $EDG(photosynthesis)$ is photosynthesis itself. Using definition 2 we have: V_x = *{photosynthesis, light_reaction, sunlight}*, $V_y = \{photosynthesis, calvin\_cycle, sugar\}$, and $V_{behavioral} = \{\}$.

$MIN\_EDG(photosynthesis)^{photosynthesis}_{sunlight,sugar}$ thus has nodes V = *{photosynthesis, light_reaction, calvin_cycle, sunlight, sugar}* and edges: E = *{(photosynthesis, light_reaction), (photosynthesis, calvin_cycle), (light_reaction, sunlight), (calvin_cycle, sugar), (light_reaction, calvin_cycle)}*.

This subgraph expresses the answer to the question "How are sunlight and sugar related in photosynthesis?". We can also answer the question "How are sunlight and sugar related?",

by finding the MIN_EDG for sunlight and sugar in the entire KB, rather than in the Event Description Graph of photosynthesis.

Now let us consider the question: "Why is X important to Y?" In order to answer it we need both $MIN\_EDG(event)^T_{X,Y}$ and the notion of path, where *event* is the ancestor of all events in the KB. Using them we have the following definition.

▶ **Definition 4.** The answer for "Why is X important to Y?" is the combination of: (i) $MIN\_EDG(event)^T_{X,Y}$ where $T = LCA(X, Y)\, in\, EDG(event)$ and (ii) ipath(X,Y).

4 ASP Encodings for General Reasoning Rules

In this section we discuss the encoding for all the defined components discussed in the previous section.

4.1 Encoding the Semantics of Slots

We encode the slot names that indicates a compositional/behavioral edges (Table 1) as follows:

```
cedge(subevent; raw_material; result; site; location; base; agent).
bedge(next_event; enables; causes; prevents).
iedge(raw_material; site; base; subevent).
iedge(enables; causes; regulates; supports; prevents; result).
```

4.2 Compositional-Connected, Behavioral-Connected, & Importantly-Connected

The following rules define "directly-compositionally-connected" (dcconnects), "directly-behaviorally-connected" (dbconnects) and "directly-importantly-connected" (diconnects).

```
dcconnects(X, Y) :-  has(X, S, Y), event(X), cedge(S).
dbconnects(X, Y) :-  has(X, S, Y), event(X), event(Y), bedge(S).
diconnects(X, Y) :-  has(X, S, Y), iedge(S).
```

The predicates *cconnect*, *bconnect* and *iconnect* denoting "compositionally-connected", "behaviorally-connected" and "importantly-connected" are transitive closures of "dcconnects", "dbconnects", and "dcconnects" respectively and are defined in the standard way.

4.3 Cpath, Bpath & Ipath

We can utilize the above defined relations to enumerate all the nodes on the compositional/behavioral path from a node to another. We define $cpath(A, Z, I, C)$ which means C is the Ith node in the path from A to Z.

```
cpath(A, Z, 0, A)   :- cconnects(A, Z).
cpath(A, Z, T+1, C) :- cpath(A, Z, T, B), dcconnects(B,C), step(T),
                       cconnects(C, Z).
```

We similarly define *bpath* and *ipath* and use them.

4.4 Finding Common Ancestor

Now we encode the rules for finding common ancestor for X and Y. The first two rules encode the special cases where either X is the ancestor of Y or Y is the ancestor of X. The 3rd rule means that Z is a common ancestor of X and Y if Z cconnects to both X and Y.

```
common_ancestor(X,X,Y) :- cconnects(X, Y), X != Y.
common_ancestor(Y,X,Y) :- cconnects(Y, X), X != Y.
common_ancestor(Z,X,Y) :- cconnects(Z, X), cconnects(Z, Y), X != Y.
```

Following the algorithm, the next step is to find the lowest-common-ancestor. We say that Z1 is not a lowest common ancestor if there exist another common ancestor Z2 which is a descendant of Z1 (Z1 cconnects to Z2). And then we can define the lowest-common-ancestor(lcs) using default negation.

```
not_lcs(Z1, X, Y) :- common_ancestor(Z1,X,Y), common_ancestor(Z2,X,Y),
                     Z1 != Z2, cconnects(Z1, Z2).
lcs(Z, X, Y)      :- common_ancestor(Z, X, Y), not not_lcs(Z, X, Y).
```

4.5 Correctness of the General Reasoning Rules

Proposition 1. Z is the lowest common ancestor of X and Y w.r.t. the KB of process P iff: $lcs(Z, X, Y)$ is entailed by the program described above.

5 ASP Encoding of How/Why Question and Answering

In this section we present the encoding for the general reasoning rules used in answering the how and why questions. We provide the template for encoding both the questions and the answers in a generic and easy-to-expand fashion. Our encoding is sufficient for a large list of questions. However, there are questions that themselves encompass a complicated semantic meaning, which needs additional representations that are beyond the scope of this work.

5.1 Question Encoding

To encode the semantics in the questions properly, we use the following template. Each question has a QID, $Type$, $Category$, two $Parameters$, and optionally the $Scope$.

```
question(QID).                 has(QID, type, Type).
has(QID, category, Category).  has(QID, param1, XClass).
has(QID, param2, YClass).      has(QID, scope, ScopeClass).
```

In the following we illustrate the encodings for the questions “How are sunlight and sugar related in photosynthesis?” and “Why is sunlight important to photosynthesis?”, respectively.

```
question(q1).
has(q1, type, how).
has(q1, category, relation).
has(q1, param1, sunlight).
has(q1, param2, sugar).
has(q1, scope, photosynthesis).
```

```
question(q2).
has(q2, type, why).
has(q2, category, important_to).
has(q2, param1, sunlight).
has(q2, param2, photosynthesis).
```

5.2 Answer Graph

According to the definitions of the answers for how and why questions, we use $\_answer\_graph(Q, Z, X, Y)$ to denote the answer $MIN\_EDG(Scope)^{Z}_{X,Y}$ of question Q. The rule head $\_answer\_graph(Q, Z, X, Y)$ denotes the answer as a graph with root Z, and two descendant X and Y, in which X, Y and Z are instances of XClass, YClass, and ScopeClass. Z is the lowest common ancestor for X and Y.

```
_answer_graph(Q, Z, X, Y) :-
question(Q),
has(Q, type, how),
has(Q, category, relation),
has(Q, param1, XClass),
has(Q, param2, YClass),
has(Q, scope, ScopeClass),
has(X, instance_of, XClass),
has(Y, instance_of, YClass),
has(Z, instance_of, ScopeClass),
lcs(Z, X, Y).
```

```
_answer_graph(Q, Z, X, Y) :-
question(Q),
has(Q, type, why),
has(Q, category, important_to),
has(Q, param1, XClass),
has(Q, param2, YClass),
XClass != YClass,
has(X, instance_of, XClass),
has(Y, instance_of, YClass),
iconnects(X, Y),
lcs(Z, X, Y).
```

Similar to the "How" question, for the "Why" question we also find the lowest common ancestor (without the scope information) Z of X and Y to form the answer graph, while enforcing that there must exist an ipath from X to Y.

5.3 Obtaining Complete Answer: Output All Nodes/Edges in the Answer Graph and Answer Path

We use *_answer_node(Q, AnswerGraph, node, E)* to denote all the nodes E in the *AnswerGraph*. The first two rules encode that if the question has a answer graph (Q, Z, X, Y), then all the nodes E on the compositional path from both Z to X and Z to Y will be answer nodes. The 3rd rule encodes that all the nodes on the behavioral paths linking every pair of nodes on the compositional paths are also answer nodes. The last rule encodes that all the nodes on compositional paths from the scope of the question to Z (to clarify the role of Z with respect to the given scope) are also answer nodes. Note that Scope, a prototype of ScopeClass, is the instance of the ScopeClass class. In our DB, prototype is always defined for each class.

```
_answer_node(Q, _answer_graph(Q, Z, X, Y), node, E) :-
    _answer_graph(Q, Z, X, Y), cpath(Z, X, T, E), step(T).

_answer_node(Q, _answer_graph(Q, Z, X, Y), node, E) :-
    _answer_graph(Q, Z, X, Y), cpath(Z, Y, T, E), step(T).

_answer_node(Q, AnswerGraph, node, E) :-
    _answer_node(Q, AnswerGraph, node, X),
    _answer_node(Q, AnswerGraph, node, Y),
    bpath(X, Y, T, E), step(T).

_answer_node(Q, _answer_graph(Q, Z, X, Y), node, E) :-
    _answer_graph(Q, Z, X, Y), has(Q, scope, ScopeClass),
    has(Scope, prototype_of, ScopeClass), cpath(Scope, Z, T, E), step(T).
```

Next the final answer is the collection of nodes and edges in the answer graph and appropriate rules are written for that. For lack of space we skip the propositions that relate the earlier definition of an answer with the answer obtained using the ASP rules.

6 Conclusion, Discussion and Future work

With good progress in information retrieval, natural language processing, speech recognition and associated fields, question answering systems are becoming a reality. However, most question answering systems are about factoid questions. But various applications, such as building intelligent tutoring systems need more general form of question answering, especially involving why and how questions. To develop systems that can answer why and how questions with respect to text, we first need to be clear about correct answers to why and how questions in a more formal setting. In other words, we need to develop a formal theory of answers to why and how questions. Towards that end, we made a start in this paper with focus on why and how question answering with respect to a structured knowledge base. We developed an abstract notion of an event description graph and used that to formalize answers with respect to two kinds of why and how questions. We then gave an ASP implementation of our formalization. The motivation behind using ASP is that as a prerequisite to answering questions with respect to a frame based knowledge base we need to implement issues such as inheritance and cloning in making inferences about facts of the form $has(X, S, Y)$. In an earlier paper we showed how ASP can be used to implement inheritance and cloning. Hence our use of ASP in this paper. Moreover we are not aware of any other declarative implementation or formalization of cloning in any other language.

Although, so far in this paper we only considered two kinds of why and how questions, our approach generalizes beyond those two to additional types. Below, we give a couple of examples on that. In the future we will consider additional types of why and how questions.

1. To answer questions of the form, "How does X occur?", we just need to define an answer graph as:

```
_answer_graph(Q, X, First_subevent_of_X, Last_subevent_of_X), ...
```

in which the first and last subevents of X can be easily obtained if all the subevents are properly ordered using the "next_event" relation.

2. Similarly, to answer questions of the form, "How does X produce Y?", the answer graph is defined as:

```
_answer_graph(Q, X, null, Y), ...
```

so that only the cpath from X to Y is in the answer graph, and this chain of reaction is "how X produces Y" if the last event in the chain has Y as result.

3. Similarly, to answer questions of the form, "Why does X have Property Y?", the answer graph is defined as:

```
_answer_graph(Q, X, Y, Subevent_of_X_that_involves_Y), ...
```

where for each subevents of X that involves Y, we generate an answer graph and output "why is Y important for that subevent".

References

1. F. Aouladomar. Towards answering procedural questions. page 21. Proc. of KRAQ'05, an IJCAI05 workshop., 2005.
2. C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.
3. C. Baral, S. Liang, and V. Nguyen. Towards deep reasoning with respect to natural language text in scientific domains. *DeepKR Workshop*, 2011.
4. Chitta Baral and Shanshan Liang. From knowledge represented in frame-based languages to declarative representation and reasoning via asp. *13th International Conference on 13th International Conference on Principles of Knowledge Representation and Reasoning*, 2012.
5. Vinay K. Chaudhri, Peter E. Clark, Sunil Mishra, John Pacheco, Aaron Spaulding, and Jing Tien. Aura: Capturing knowledge and answering questions on science textbooks. Technical report, SRI International, 2009.
6. P. Clark, B. Porter, and B.P. Works. Km: The knowledge machine 2.0: Users manual, 2004.
7. D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A.A. Kalyanpur, A. Lally, J.W. Murdock, E. Nyberg, J. Prager, et al. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79, 2010.
8. R. Fikes and T. Kehler. The role of frame-based representation in reasoning. *Communications of the ACM*, 28(9):904–920, 1985.
9. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. A user's guide to gringo, clasp, clingo, and iclingo. *November*, 77:78–80, 2008.
10. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080. MIT Press, 1988.
11. Google Knowledge Graph. http://www.google.com/insidesearch/features/search/knowledge.html.
12. R. Higashinaka and H. Isozaki. Corpus-based question answering for why-questions. *Proc. of IJCNLP*, 1:418–425, 2008.
13. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7(3):499–562, 2006.
14. I. Niemelä and P. Simons. Smodels—an implementation of the stable model and well-founded semantics for normal logic programs. *Logic Programming and Nonmonotonic Reasoning*, pages 420–429, 1997.
15. Tom Shellberg. Teaching how to answer 'why'questions about biology. *The American Biology Teacher*, 63(1):16–19, 2012/06/17 2001.
16. A. Wouters. The functional perspective of organismal biology. *Current Themes in Theoretical Biology*, pages 33–69, 2005.