

# Combining Existential Rules with Network Diffusion Processes for Automated Hypothesis Generation

Jose Nicolas Paredes<sup>a</sup>, Marcelo Alejandro Falappa<sup>a</sup>, Paulo Shakarian<sup>b</sup>, Maria Vanina Martinez<sup>c,\*</sup> and Gerardo Ignacio Simari<sup>a</sup>

<sup>a</sup> *Departamento de Ciencias e Ingenieria de la Computacion, Universidad Nacional del Sur (UNS) & Instituto de Ciencias e Ingenieria de la Computacion (UNS-CONICET), Argentina*

*E-mails: jose.paredes@cs.uns.edu.ar, mfalappa@cs.uns.edu.ar, gis@cs.uns.edu.ar*

<sup>b</sup> *Syracuse University, United States*

*E-mail: pashakar@syr.edu*

<sup>c</sup> *Artificial Intelligence Research Institute (IIIA CSIC), Spain*

*E-mail: vmartinez@iia.csic.es*

**Abstract.** There are many applications in which decision support systems can benefit from the ability to automatically generate hypotheses—two salient examples are detection of malicious behavior (such as in social media platforms or cyber threat analysis in enterprise systems) and preventive healthcare that continuously monitors patients’ readings towards early detection of conditions requiring medical attention. In this paper, we continue work on the recently-proposed NETDER architecture (Network Diffusion and ontological reasoning based on Existential Rules), addressing the technical issues towards its effective implementation. The working hypothesis behind our model is that three key elements can be leveraged towards the automatic generation of hypotheses: (i) combining multiple, ever-evolving data sources, (ii) maintaining a knowledge base using logic-based formalisms capable of value invention to support reasoning about unknown objects based on available data, and (iii) maintaining a related knowledge base for actors and relationships among them, and how information flows across such a network. After developing the formal machinery behind the model, we focus on the main task of query answering (QA), studying three different policies that can be used towards this end. One of our main contributions in this regard is the analysis about computational cost of the proposed QA procedures, including cases in which termination of associated procedures is not guaranteed. Finally, we present a use case showing how the formalism can be applied in a real-world setting involving applications to cybersecurity.

**Keywords:** Ontological Languages, Socio-Technical Systems, Hybrid Artificial Intelligence

## 1. Introduction and Motivation

There are many real world domains in which solving even seemingly basic problems requires carrying out reasoning tasks based on well-founded data management. Even though many basic problems have been addressed in the literature of different subareas of AI and Databases, the more general problem of query answering (QA) towards the *automatic generation of hypotheses* has not received much attention.

---

\*Corresponding author. E-mail: vmartinez@iia.csic.es.

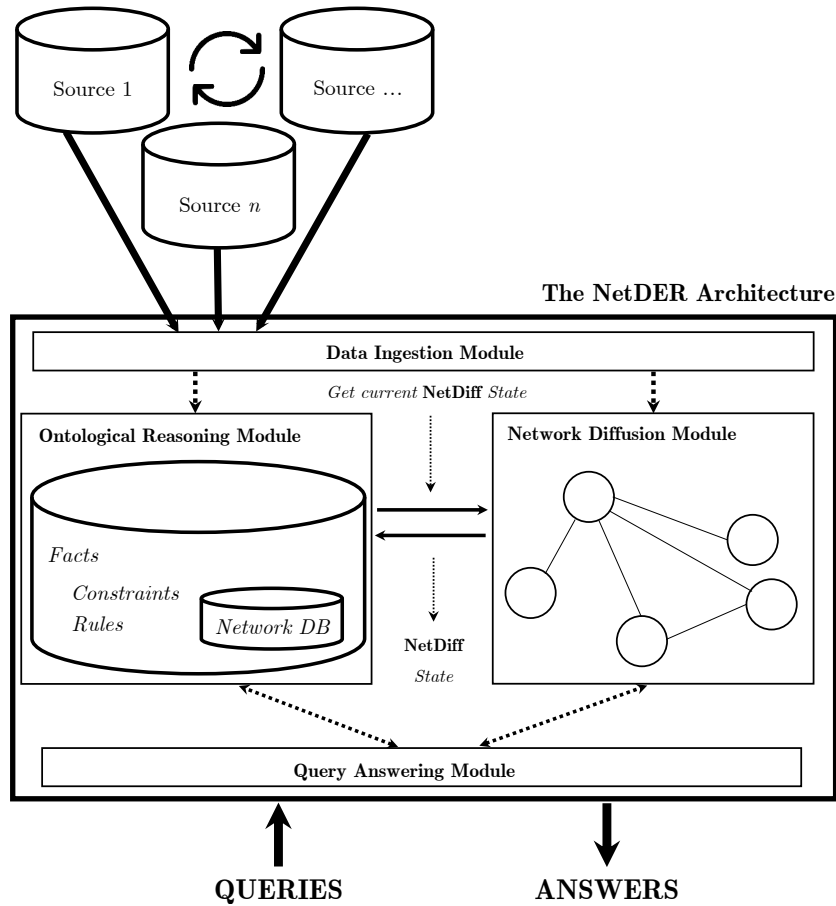


Fig. 1. Schematic view of the NETDER architecture; solid lines indicate data access, and dashed ones data modification. The architecture has four modules; the main task is handled by the Query Answering Module, which may involve reasoning about unknown objects.

Consider for instance how medical doctors take into account a host of information sources before giving one or more possible diagnoses, how government auditors gather information looking for possible fraud in the use of public funds, or how cybersecurity analysts essentially carry out the same process but with different underlying data when looking for the culprit of a cyber attack. In all these cases—and many others of this sort—automating this kind of process (at least partially) requires several high-level capabilities such as logical reasoning, handling unknown entities and objects, modeling networks of interconnected objects and diffusion of properties throughout such networks, and QA algorithms that can yield the best possible answers under resource and data availability constraints.

This is the motivation behind the recently proposed NETDER architecture [41], which initially was described mainly from a system design point of view. Figure 1 shows an overview of the model; a variety of data sources (assumed to be updating at independent rates) feeds a Data Ingestion Module (DIM) that handles issues such as data cleaning, schema matching, inconsistency, incompleteness, etc., as well as other higher-level issues such as trust and uncertainty management. The main objective of the DIM is to prepare the data and build an ontological knowledge base, including a network database—these components are key to the functions of the Ontological Reasoning Module and the Network Diffusion Module, which can be described as follows:

- *Ontological Reasoning Module (ORM)*: Maintains the knowledge base (comprised of facts, rules, and constraints) both for background knowledge as well as for the network (see Network Diffusion Module). The rea-

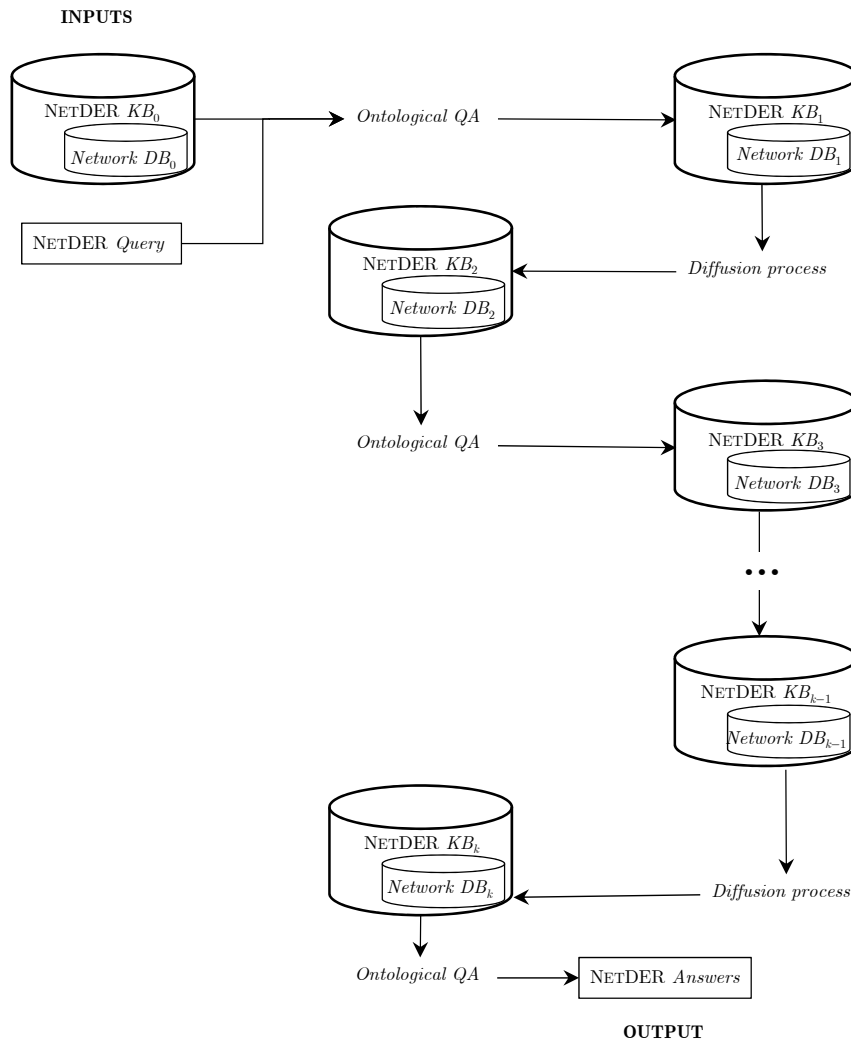


Fig. 2. Overview of the general process carried out by the Query Answering Module.

soning processes implemented here are the main drivers behind ontological QA; the network module plays a supporting role, offering access to the current network state as a service to this module.

- *Network Diffusion Module (NDM)*: Based on the network DB stored in the ORM, this module handles the *dynamic* aspects of networks in the form of diffusion processes. The main services implemented here are therefore the simulation of such processes and answering subsidiary queries over the network for the ORM.

Finally, the *Query Answering Module (QAM)* is responsible for implementing the process that carries out the main task, which we refer to as NETDER query answering. As we will see, this involves coordinating the execution of both the ORM and NDM modules. In Figure 2, we provide a high-level view of the process performed by this module; starting from the basic inputs (knowledge base and a query), a sequence of ontological QA and diffusion process is carried out repeatedly until an answer is delivered. As we will show later, this process is governed by a *query answering policy* that, depending on its nature, may or may not be guaranteed to terminate.

In this paper, we continue developing the initial proposal of the model [41] by developing the full details needed to implement the architecture. Our previous work [40] presents a preliminary validation of this architecture, where the focus was on developing and evaluating a basic instantiation of NETDER for a specific set of tasks—this also required developing a testbed for synthetically generating sets of full traces that afford access to ground truth for

empirical evaluations. Furthermore, we are also developing a line of work based on explainability using this model. In [39] we focused on different ways in which explanations can be presented to users of various types (such as basic users, power users, and analysts), and showed how it can be used to develop a recommender system for detecting hate speech in social platforms. Then, in [58] we studied how the different modules can be applied towards delivering a wide range of explanations, presenting a “*build your own explanation*” algorithm that users can leverage towards deploying explanations that are tailored to their needs.

For the NDM, we will assume that a small set of basic services are available that can be implemented in a wide variety of ways; however, in order to be able to provide a theoretical analysis of one possible implementation, we will make use of the NetDiff language, which is directly based on the MANCaLog formalism [52, 53]. Much simpler alternatives for implementing the NDM include, for instance, the well-known agent-based model (ABM); for example, an ABM model could be used to represent how opinions arise in a group of people [32]. The main issues involved towards the full NETDER implementation are:

- Designing the representation of the ontological knowledge base maintained in the ORM. In particular, this KB also contains the representation of a network of entities with labels that apply to nodes and edges—each of these labels have assigned an uncertainty interval that is subject to being updated by the diffusion processes administered by the NDM.
- Adapting the MANCaLog [52, 53] formalism proposed for general network diffusion processes to function as the specification language for diffusion processes in the NDM. This involves developing a procedure for querying network states, and studying conditions under which it is guaranteed to terminate and enjoy computational tractability.
- Developing a modified chase procedure—analogue to the well-known procedure for QA in existential rules—to be used in answering queries over the ORM knowledge base. This involves adapting the procedure to handle the ontological rules extended with the possibility of checking conditions over the network, both locally with respect to individual components and globally, as summaries of the state of the structure.
- Developing a general NETDER query answering procedure in charge of orchestrating the interaction between the two main query module-level QA procedures—the NETDER chase and network state querying—via the application of what we call *query answering policies*.

With all these elements in place, we study the properties of the general QA procedure and learn that even under strict conditions ensuring that both subsidiary procedures terminate, the general procedure is not guaranteed to terminate. We also prove a result stating that, under certain conditions, the computational cost of the general QA procedure is determined by the complexity of classical query answering over a set of classical existential rules derived via a translation from the NETDER KB—the computational cost will thus depend on the specific fragment to which the translated KB belongs.

As an illustrative application domain, cybersecurity provides particularly rich, noisy, and continuously evolving data sources for risk assessment. A central class of tools in this field are those that issue warnings about assets potentially under threat, often by integrating structured reasoning components with complementary analytic techniques [28]. In particular, consider a cybersecurity firm in charge of scouring the deep and darkweb in search of evidence that its clients’ systems are at risk of being the target of a cyber attack. Available information takes the form of forum and market posts written by actors searching for and selling malware, discussing attack vectors and vulnerabilities, etc. Once a vulnerability is discovered, discussions regarding exploits can be seen as different diffusion processes among different kinds of actors and at different levels of expertise. Here, the main QA tasks could be asking questions such as:

*Q<sub>1</sub>: Are there systems running on company C’s computers at risk of being attacked at this time?*

An analyst may be interested in what systems are at risk, or they may have suspicions regarding a particular system, and this could lead to posing a query related to the hypotheses generated about systems running on company C’s computers at risk of being attacked at this time. As expected, the answers could change when new data is available—systems at risk at time  $\tau$  may no longer be so at time  $\tau + 1$ , systems that are not at risk at time  $\tau$  could become so at time  $\tau + 1$ , and it may also be necessary to consider new systems at time  $\tau + 1$ .

1 *Q<sub>2</sub>: What products from vendor V have unpatched vulnerabilities at this time?* 1

2 Here, the interest is focused on unpatched vulnerabilities of the products, which are a necessary condition for an 2  
 3 attack to occur effectively; that is, these answers can be useful to answer *Q<sub>1</sub>*. Hence, an analyst will need to pose a 3  
 4 query to generate hypotheses with the aim of unveiling unpatched vulnerabilities of a product at this time. Again, 4  
 5 answers could change over time—unpatched vulnerability of a product at time  $\tau$  may not remain at time  $\tau + 1$ , and 5  
 6 it may be necessary to consider new products and vulnerabilities at time  $\tau + 1$ . 6  
 7

8 *Q<sub>3</sub>: Is there any evidence that a viable exploit for vulnerability X is being developed at this time?* 8

9 Now, the question is centered on what exploits exist in order to take advantage of one or more vulnerabilities at 9  
 10 this time, and this is also an issue with a direct connection to *Q<sub>1</sub>* since its answers are other necessary conditions 10  
 11 for an attack to take place (that is, these answers can be useful to answer *Q<sub>1</sub>*). Then, a domain expert can specify a 11  
 12 query to generate hypotheses about exploits that use a set of vulnerabilities at this time. As before, answers could be 12  
 13 different for each time point, since unpatched vulnerability change over time, and it may be necessary to consider 13  
 14 new products and vulnerabilities as times progresses. 14  
 15

16 *Q<sub>4</sub>: What actors may be interested in attacking system S running on company C's computers?* 16

17 Finally, in this last case, an analyst could be concerned with determining who is interested in a cyber attack targeting 17  
 18 the systems running on our computers—answers to this question are also dependent on the answers to *Q<sub>1</sub>* because 18  
 19 in order for there to exist a party responsible for an attack, there must exist a system at risk of being attacked. In this 19  
 20 context, an analyst will define a query to generate hypotheses with the aim of determining the (probably incomplete) 20  
 21 identity of malicious actors who might be interested in carrying out a cyber attack at this time. Once more, the 21  
 22 answers will probably change over time, since malicious actors who are interested in perpetrating a cyber attack at 22  
 23 time  $\tau$  may no longer be interested at time  $\tau + 1$ , malicious actors who are not interested in perpetrating a cyber 23  
 24 attack at time  $\tau$  could become so at time  $\tau + 1$ , and it may be necessary to consider new cyber attacks and malicious 24  
 25 actors at time  $\tau + 1$ . 25  
 26

27 Note that the above questions are all interconnected, since answers to one can be useful to find answers to others. 27  
 28 A novel aspect of NETDER is that it can be used to address two or more related problems at once. Likewise, 28  
 29 NETDER has the ability to reason about unknown objects, which can be essential in tackling issues such as: i) to 29  
 30 answer *Q<sub>3</sub>*, we may know that there *exists* an exploit and its code for vulnerability X, but we may not have the code 30  
 31 available; or ii) to answer *Q<sub>4</sub>*, we may know that there *exists* a malicious actor who is interested in perpetrating a 31  
 32 cyber attack, has the username “john420” on darkweb market Y, and that they have an associated IP address that 32  
 33 they control for launching attacks, but for now we do not know its value. 33  
 34

35 These and other questions are examples of queries in which ontological reasoning (leveraging value invention and 35  
 36 complex inference) and reasoning about network structures is paramount to effectively combining available knowl- 36  
 37 edge (that is in constant evolution) affected by uncertainty, and thus human-in-the-loop approaches are especially 37  
 38 useful. Other examples that are closely related to our cybersecurity motivational example include reasoning about 38  
 39 malicious behavior in other scenarios in which data may be available to different degrees, from detecting bad actors 39  
 40 in the Ethereum blockchain—where all transactions are publicly available—to tagging fake news posts on social 40  
 41 platforms, where depending on the degree to which data can be accessed the effort may range from shallow to quite 41  
 42 involved. We will come back to this scenario through simple examples used in the paper to illustrate the intuition 42  
 43 behind the main concepts, as well as in a full example developed as a preliminary practical evaluation of the model's 43  
 44 capabilities. 44  
 45

46 The rest of this paper is then organized as follows. After covering preliminary material on existential rules and 46  
 47 diffusion processes in Section 2, Section 3 presents the details of the knowledge representation and reasoning ma- 47  
 48 chinery, as well as subsidiary QA procedures of the ORM and NDM that define the operational semantics of the 48  
 49 corresponding reasoning tasks in each module. In Section 4, we develop and study the properties of NETDER's op- 49  
 50 erational semantics by specifying the NETDER QA procedure, which depends on the operational semantics of the 50  
 51 ORM and NDM defined previously. Sections 5 and 6 discuss related work and conclusions, respectively. Addition- 51

ally, Appendix A shows how the NETDER ontological language can be mapped in Datalog+/-, Appendix B includes proofs of theorems, observations and lemmas, whereas Appendix C details an extended use case in a cybersecurity scenario.

## 2. Preliminaries

### 2.1. Existential Rules (also known as Datalog+/-)

We now present some brief preliminaries about the family of ontological languages called existential rules *Datalog+/-* [10]. *Datalog+/-* is a family of rule based ontological languages developed for the Semantic Web, designed to balance expressive power with computational tractability; in this respect, it is closely related to lightweight Description Logics [36], which pursue the same trade-off.

We assume: (i) an infinite universe of (*data*) constants  $\Delta$  (which constitute the “normal” domain of a database), (ii) an infinite set of (*labeled*) nulls  $\mathcal{N}$  (used as “fresh” Skolem terms, which are placeholders for unknown values, and can thus be seen as variables), and (iii) an infinite set of variables  $\mathcal{V}$  (used in queries and dependencies). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. We assume a lexicographic order on  $\Delta \cup \mathcal{N}$ , with every symbol in  $\mathcal{N}$  following all symbols in  $\Delta$ .

We assume a *relational schema*  $\mathcal{R}$ , which is a finite set of *predicate symbols* (or simply *predicates*). A *term*  $t$  is a constant, null, or variable. An *atomic formula* (or *atom*)  $a$  has the form  $P(t_1, \dots, t_n)$ , where  $P$  is an  $n$ -ary predicate, and  $t_1, \dots, t_n$  are terms. A conjunction of atoms is often identified with the set of all its atoms. An instance  $\mathcal{I}$  for a relational schema  $\mathcal{R}$  is a (possibly infinite) set of atoms with predicates from  $\mathcal{R}$  and arguments from  $\Delta \cup \mathcal{N}$ . Likewise, a *database*  $D$  for a relational schema  $\mathcal{R}$  is a finite set of atoms with predicates from  $\mathcal{R}$  and arguments from  $\Delta$ , this is,  $D$  is a finite instance which contains only constants.

A *homomorphism* is a mapping  $\mu : \Delta \cup \mathcal{V} \cup \mathcal{N} \rightarrow \Delta \cup \mathcal{V} \cup \mathcal{N}$  such that (i) if  $c \in \Delta$  then  $\mu(c) = c$ , (ii) if  $c \in \mathcal{V} \cup \mathcal{N}$  then  $\mu(c) \in \Delta \cup \mathcal{N}$ , and (iii)  $\mu$  is naturally extended to atoms, sets of atoms, and conjunctions of atoms, that is,  $\mu(p(t_1, \dots, t_n)) = p(\mu(t_1), \dots, \mu(t_n))$ . A *grounding* is a homomorphism  $\rho$  such that  $\rho(c) \in \Delta$  for every  $c \in \Delta \cup \mathcal{V} \cup \mathcal{N}$ . Next, we show the different kinds of rules available in this formalism.

**TGDs.** A *tuple generating dependency* (TGD) is a first-order formula:

$$\sigma : \forall \mathbf{X} \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$$

where  $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$  are sequences of variables,  $\Phi(\mathbf{X}, \mathbf{Y})$  and  $\Psi(\mathbf{X}, \mathbf{Z})$  are conjunctions of atoms over  $\mathcal{R}$  (called the *body* and *head* of  $\sigma$ , denoted  $body(\sigma)$  and  $head(\sigma)$ , respectively). For ease of presentation, we sometimes omit the quantifiers, and assume that all variables without quantifier are universally quantified.

**EGDs.** An *equality generating dependency* (EGD) is a first-order formula:

$$\epsilon : \forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow X_i = X_j$$

where  $\Phi(\mathbf{X})$  is a conjunction of atoms,  $\mathbf{X}$  is a sequence of variables, and  $X_i, X_j \in \mathbf{X}$ . As with TGDs, we sometimes omit the universal quantifiers. The body of an EGD  $\epsilon$ , denoted  $body(\epsilon)$ , corresponds to  $\Phi(\mathbf{X})$ , and the head, denoted  $head(\epsilon)$ , corresponds to the equality atom on the right hand side of the rule.

**NCs.** A *negative constraint* (NC) is a first-order formula of the form:

$$\mu : \forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow \perp$$

where  $\Phi(\mathbf{X})$  is a conjunction of atoms, and  $\mathbf{X}$  is a sequence of variables. As with the other rules, we sometimes omit the universal quantifiers. The body of a NC  $\mu$ , denoted  $body(\mu)$ , corresponds to  $\Phi(\mathbf{X})$ , and the head, denoted  $head(\mu)$ , corresponds to the logical contradiction on the right hand side of the rule.

Satisfaction of these rules in an instance is determined as follows:

**TGD Satisfaction.** An instance  $\mathcal{I}$  for  $\mathcal{R}$  satisfies a TGD  $\sigma$ , denoted  $\mathcal{I} \models \sigma$ , if whenever there exists a homomorphism  $h$  such that  $h(\text{body}(\sigma)) \subseteq \mathcal{I}$ , then there exists an extension  $h'$  of  $h$  such that  $h'(\text{head}(\sigma)) \subseteq \mathcal{I}$ .

**EGD Satisfaction.** An instance  $\mathcal{I}$  satisfies an EGD  $\epsilon$ , denoted  $\mathcal{I} \models \epsilon$ , if whenever there exists a homomorphism  $h$  such that  $h(\text{body}(\epsilon)) \subseteq \mathcal{I}$ , then it must hold that  $h(X_i) = h(X_j)$ . Otherwise, we say that  $\epsilon$  is *violated* in  $\mathcal{I}$ .

**NC Satisfaction.** An instance  $\mathcal{I}$  satisfies a NC  $\mu$ , denoted  $\mathcal{I} \models \mu$ , if there does not exist a homomorphism  $h$  such that  $h(\text{body}(\mu)) \subseteq \mathcal{I}$ . Otherwise, we say that  $\mu$  is *violated* in  $\mathcal{I}$ .

Finally, we describe *Datalog*+/- queries and answers. A *conjunctive query* (CQ) over  $\mathcal{R}$  has the form  $Q(\mathbf{X}) = \exists \mathbf{Y} \Upsilon(\mathbf{X}, \mathbf{Y})$ , where  $\Upsilon(\mathbf{X}, \mathbf{Y})$  is a conjunction of atoms (possibly equalities, but not inequalities) with the variables  $\mathbf{X}$  and  $\mathbf{Y}$ , and possibly constants, but without nulls. A *Boolean CQ* (BCQ) over  $\mathcal{R}$  is a CQ of the form  $Q()$ , often written as the set of all its atoms, without quantifiers. Answers are defined via *homomorphisms*, and the set of all *answers* to a CQ  $Q(\mathbf{X}) = \exists \mathbf{Y} \Upsilon(\mathbf{X}, \mathbf{Y})$  over an instance  $\mathcal{I}$ , denoted  $Q(\mathcal{I})$ , is the set of all tuples  $t$  over  $\Delta$  for which there exists a homomorphism  $\mu: \mathbf{X} \cup \mathbf{Y} \rightarrow \Delta \cup \mathcal{N}$  such that  $\mu(\Upsilon(\mathbf{X}, \mathbf{Y})) \subseteq \mathcal{I}$  and  $\mu(\mathbf{X}) = t$ . The *answer* to a BCQ  $Q()$  over an instance  $\mathcal{I}$  is *Yes*, denoted  $\mathcal{I} \models Q$ , if and only if  $Q(\mathcal{I}) \neq \emptyset$ .

## 2.2. The NetDiff Formalism

In this section, we will present the main elements involved in the NetDiff formalism, which will be used in some theoretical analyses to model diffusion processes in complex networks. Our presentation is a (slight) generalization of the one first introduced in prior work [52, 53]—we modify the syntax by adding a new kind of rule (*global rules*) to derive global knowledge that allows us to summarize the local knowledge about nodes in the network, generalizing the notion of influence function, and incorporating variables into rules.

We assume that individuals (such as persons or agents) are arranged in a directed graph (or network) denoted *Graph*  $= (V, E)$ , where the set of nodes  $V$  corresponds to the individuals, and the edges  $E$  model the relationships between them. Let  $\mathcal{P}$  be a set of predicate symbols; we will use *node*  $\in \mathcal{P}$  and *edge*  $\in \mathcal{P}$  to identify the components in the network in the following way: *node* $(t_1), \dots, \text{node}(t_n) \in V$  and *edge* $(t'_1, t'_2), \dots, \text{edge}(t'_m, t'_{m+1}) \in E$ . That is, *node* $(t_1)$  states that “ $t_1$  is a node” and *edge* $(t'_1, t'_2)$  states that “ $(t'_1, t'_2)$  is an edge”. Additionally, we will use  $\mathcal{G} = V \cup E$  to denote the set of all *components* (nodes and edges) in the network—thus,  $c \in \mathcal{G}$  could be either a node or an edge—and the special constant  $\Omega$  will be used as an abstraction of the whole network. We also assume a possibly infinite data domain  $\Delta$ . Finally, we assume that  $\mathcal{P}$  contains three different sets of *label predicate symbols*, namely  $\mathcal{P}_{loc}$ ,  $\mathcal{P}_{eloc}$ , and  $\mathcal{P}_{glo}$ , and based on this we have a finite set of labels  $\mathcal{L}$ , which is partitioned into three sets, respectively:

- *node local labels*  $\mathcal{L}_{nloc}$ , which are labels applied to nodes of the network and that are built from predicate symbols in  $\mathcal{P}_{nloc}$ .
- *edge local labels*  $\mathcal{L}_{eloc}$ , those that are applied to edges of the network and that are built from predicate symbols in  $\mathcal{P}_{eloc}$ .
- *global labels*  $\mathcal{L}_{glo}$ , a generalization of local labels, refer to global knowledge about nodes in the network. They cannot be applied to any node or edge, but rather to the whole network, and are built from predicate symbols in  $\mathcal{P}_{glo}$ .

Labels will be associated to entities (nodes), their connections (edges), and the whole network representing different attributes. Note that labels do not reference explicitly nodes, edges, or the whole network because they are always applied to all elements of each type of set, respectively—that is, every label in  $\mathcal{L}_{nloc}$  is applied to every node in  $V$ , and they will be distinguished according to their truth value assigned to them. Later, we will see that a default value is assigned, which will be changed using facts and rules. Each label  $\mathcal{L}$  has the form  $p(t_1, \dots, t_n)$ , where  $t_i \in \Delta$  and  $p \in \mathcal{P}$ .

**Example 2.1.** Consider the structure of an online social network:

$$\text{Graph}_{soc} = (\{\text{node}(n_1), \text{node}(n_2)\}, \{\text{edge}(n_1, n_2)\}).$$

We have  $\mathcal{L}_{nloc} = \mathcal{L}_1 \cup \mathcal{L}_2$  where:

$$\mathcal{L}_1 = \{\text{expert}(\text{ddos}), \text{expert}(\text{botnet}), \text{expert}(\text{trojan}), \text{expert}(\text{malware})\}.$$

These labels represent the field of expertise of the node to which they are applied (we will see below how this association is made syntactically). That is, a node could be an expert on DDoS (distributed denial of service attacks), botnets, trojans, and on malware, respectively.

Additionally, we use  $\mathcal{L}_2 = \{\text{bel\_dang}(\text{gooligan}), \text{bel\_dang}(\text{luabot})\}$  to denote labels representing that the node to which they are applied believes that gooligan and luabot are dangerous terms discovered on the Web, respectively. We have a single edge local label  $\mathcal{L}_{eloc} = \{\text{close}(\text{trojan}, \text{win-10})\}$ , which allows us to represent that the nodes connected by an edge to which the label applies are close regarding their interests with respect to “trojans in Windows 10”. Finally, we also have  $\mathcal{L}_{glo} = \{\text{dangerous}(\text{gooligan}), \text{dangerous}(\text{luabot})\}$  to denote the corresponding global labels, in this case representing the global state of the network regarding the belief that terms gooligan and luabot are dangerous. ■

Since most interesting domains involve uncertainty, it is important to have the ability to represent this situation in the labels; *Network atoms* are the formal units that allow us to do so, such that every *network atom*  $\langle L, bnd \rangle$  associates a label  $L$  with an interval  $bnd \subseteq [0, 1]$ <sup>1</sup>. Additionally, these elements can be used to define a *world*  $W$ , which is a set of network atoms where for each  $L \in \mathcal{L}$  there exists exactly one atom  $\langle L, bnd \rangle \in W$ . Worlds allow us to describe the state of the network—the state can be seen as the association of a world to each component of the network (node or edge), or the whole network. Therefore, if we know the state of the network then we can know what knowledge is held (or not); a world  $W$  satisfies a network atom  $\langle L, bnd \rangle$  iff there exists  $\langle L, bnd' \rangle \in W$  such that  $bnd' \subseteq bnd$ . Likewise, network atoms can be used to build *network formulas* using variables, substitutions, and the standard connectives  $\wedge, \vee, \neg$ , and their satisfaction is defined via substitutions; satisfaction generalizes inductively over connectives as usual.

Knowledge about the domain can be defined using atomic structures and rules. Given  $\mathcal{P}, \Delta, \mathcal{L}$ , and  $Graph = (V, E)$ , a *component state* over  $\mathcal{L}$  and  $Graph$  is a structure of one of the following forms:

$$(v, \langle L_{nloc}, bnd \rangle), (e, \langle L_{eloc}, bnd \rangle), \text{ or } (\Omega, \langle L_{glo}, bnd \rangle),$$

where  $v \in V, e \in E, L_{nloc} \in \mathcal{L}_{nloc}, L_{eloc} \in \mathcal{L}_{eloc}, L_{glo} \in \mathcal{L}_{glo}$ , and  $bnd \subseteq [0, 1]$ . We denote with  $\mathcal{CS}$  the set of all possible component states over  $\mathcal{L}$  and  $Graph$ .

Now, we need to introduce the concept of NetDiff fact. This first requires the notion of time, which we will model as a finite set of discrete values between 0 and a constant maximum value  $t_{max}$ —this number specifies the total amount of time we are considering for diffusion processes, that is, we work with a fixed horizon. A NetDiff *fact* is therefore a component state  $cs$  annotated with a time interval  $[\tau_1, \tau_2] \subseteq [0, t_{max}]$ , as follows:

$$cs : [\tau_1, \tau_2]$$

where  $[\tau_1, \tau_2] \subseteq [0, t_{max}]$ . Such facts state that the network atom  $\langle L_{nloc}, bnd \rangle$  (and  $\langle L_{eloc}, bnd \rangle, \langle L_{glo}, bnd \rangle$ , respectively) is true for the node  $v$  (edge  $e$ , the whole network  $\Omega$ , respectively) during each time point  $\tau \in [\tau_1, \tau_2]$ .

A simple example of a fact based on our running example is:

$$(node(n_1), \langle \text{expert}(\text{malware}), [1, 1] \rangle) : [0, t_{max}],$$

specifying that individual  $n_1$ , which is a node in the network is considered, with certainty, to be an expert on malware throughout the entire set of time points considered<sup>2</sup>.

<sup>1</sup>Intervals in network atoms represent how confident we are in the piece of knowledge. They could be written in alternative ways; for example, the interval  $[0.3, 0.7]$  could also be specified as  $0.5 \pm 0.2$ .

<sup>2</sup>This NetDiff fact is equivalent to the set of facts  $\{(node(n_1), \langle \text{expert}(\text{malware}), [1, 1] \rangle) : [0, 2], (node(n_1), \langle \text{expert}(\text{malware}), [1, 1] \rangle) : [1, t_{max}]\}$ .

We now turn to NetDiff rules, of which we consider two kinds: (a) *local* rules, which have a label atom in the head with variables and/or constants that is used to build local labels; and (b) *global* rules, which have a label atom in the head with variables and/or constants, that is used to build global labels. The idea behind *local* rules is simple: a node that meets some criteria is influenced by the set of its neighbors who possess certain properties. The amount of influence exerted on a node by its neighbors is specified by an *influence function*. As a result, a local rule consists of four major parts: (i) an influence function, (ii) neighbor criteria, (iii) target criteria, and (iv) a target. Intuitively, (i) specifies how the neighbors influence the node in question, (ii) specifies which of the neighbors can influence the node, (iii) specifies the criteria that cause the node to be influenced, and (iv) is the property of the node that changes as a result of the influence.

In a similar manner, *global* rules define how the state of one or more *global* labels is updated by a set of components of the network that possess certain properties. The updated value of any *global* label is determined by an aggregation function (defined as usual, such as mean, min, max, etc.).

For the first kind of rule, the *local rules*, the structure is the following:

$$\underbrace{\alpha(\mathbf{S})}_{\text{target}} \xleftarrow{\Delta t} \left( \underbrace{nc_{edge}(\mathbf{X}), nc_{node}(\mathbf{Y})}_{\text{neighbor criteria}}, \underbrace{if}_{\text{influence function}} \right) \underbrace{tc(\mathbf{T})}_{\text{target criteria}}$$

where  $\alpha \in \mathcal{P}_{nloc}$  is a label predicate symbol ( $\alpha(\mathbf{S})$  is the target),  $\Delta t$  is a natural number, *if* (influence function) is a function that determines the amount of influence exerted, and *tc*( $\mathbf{T}$ ) (target criteria) is a conjunction of atomic formulas about the “target” node, that is a node for which the bound of the label is updated by the rule;  $nc_{edge}(\mathbf{X})$ ,  $nc_{node}(\mathbf{Y})$  (neighbor criteria) are conjunctions of atomic network facts about the edges connected to the target node, and about the nodes connected with those edges to the target node. Moreover,  $\mathbf{S}, \mathbf{T}, \mathbf{X}, \mathbf{Y}$  are sequences of variables; in the particular case that  $\mathbf{S}, \mathbf{T}, \mathbf{X}, \mathbf{Y}$  are all empty, we say that the rule is *ground*.

Influence functions are of the form:

$$if : 2^V \times V \times 2^{CS} \times \mathcal{L} \rightarrow \mathcal{K},$$

where  $V$  is the set of nodes in  $Graph = (V, E)$ ,  $\mathcal{L}$  is the set of labels,  $CS$  is the set of all possible component states over  $\mathcal{L}$  and  $Graph$ , and  $\mathcal{K}$  is the set of all possible intervals between 0 and 1. Intuitively, the first term is a set of neighbors of a node  $v$  that is specified in the second term. Likewise,  $v$  is the target node of a local rule that is applied (and which has invoked *if*), the third term is the set of component states that represent the network state at time  $\tau$  when the local rule is applied, and the fourth is the label that will change its bound in the target node when the rule is applied. Then, the result of the function is an interval between 0 and 1 that represents the influence exerted.

Note that the target (also referred to as the head) of the rule involves a formula with a single label predicate symbol; essentially, the body of the rule characterizes a set of nodes, and this atomic formula allows us to identify the labels that could be modified for each node in this set. More specifically, the rule states that when certain conditions for a node and its neighbors are met, the bound (*bnd*) changes for those network atoms formed with some label based on the predicate symbol  $\alpha$  on that node. If the rule is applicable at time  $\tau$ , then the change will be made at time  $\tau + \Delta t$ .

An example of a local rule is the following:

$$lr_1 : \underbrace{bel\_dang(X_1)}_{\text{target}} \xleftarrow{1} \left( \underbrace{\langle expert(X_2), [0, 0] \rangle}_{\text{target criteria}}, \underbrace{\langle \langle close(X_3, X_4), [1, 1] \rangle, \langle bel\_dang(X_1), [1, 1] \rangle \rangle}_{\text{neighbor criteria}} \wedge \underbrace{\langle expert(X_2), [1, 1] \rangle}_{\text{neighbor criteria}}, \underbrace{if_{one}}_{\text{influence function}} \right)$$

Intuitively, this rule says that “nodes that are not experts in domain  $X_2$  with certainty, and have close neighbors (regarding their interests with respect to domain  $X_3$  in platform  $X_4$ ) that are experts on domain  $X_2$  and believe that

$X_1$  is dangerous with certainty, will update the belief that  $X_1$  is dangerous according to function  $if_{one}$ , in the next time point”. In this case, influence function  $if_{one}$  is a simple function that simply returns the interval  $[1, 1]$ . Moreover, if we assume that  $\mathcal{L}_{nloc} = \{bel\_dang(luabot), expert(trojan), expert(ddos)\}$  and  $\mathcal{L}_{eloc} = \{close(trojan, win-10)\}$ , then the current state of the network must satisfy the following groundings of  $lr_1$ :

$$lr_{1.1} : bel\_dang(luabot)_{\langle expert(trojan), [0,0] \rangle} \leftarrow \frac{1}{2} \left( \langle close(trojan, win-10), [1, 1] \rangle, \right. \\ \left. \langle bel\_dang(luabot), [1, 1] \rangle \wedge \langle expert(trojan), [1, 1] \rangle, if_{one} \right)$$

$$lr_{1.2} : bel\_dang(luabot)_{\langle expert(ddos), [0,0] \rangle} \leftarrow \frac{1}{2} \left( \langle close(trojan, win-10), [1, 1] \rangle, \right. \\ \left. \langle bel\_dang(luabot), [1, 1] \rangle \wedge \langle expert(ddos), [1, 1] \rangle, if_{one} \right)$$

For the second kind of rule, the *global rules*, the structure is the following:

$$\beta(\mathbf{X}) \leftarrow (\alpha(\mathbf{Y})_{lt(\mathbf{Z})}, af)$$

where  $\beta \in \mathcal{P}_{glo}$  is a label predicate symbol,  $\alpha \in \mathcal{P}_{nloc}$ ,  $lt(\mathbf{Z})$  is a conjunction of atomic network formulas, and  $af$  is an aggregation function. As before, when sequences of variables  $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$  are all empty, the rule is *ground*.

This kind of rule allows us to obtain a kind of summary of the state of the network—the value of the bound of the global labels formed with the predicate symbol  $\beta$  is the result of applying the function  $af$  to a set of bounds. Every one of these bounds is associated with some local label formed with the predicate symbol  $\alpha$  belonging to some of the components of the graph (the ones meeting criteria  $lt(\mathbf{Z})$ ).

An example of a global rule is the following:

$$gr_1 : dangerous(X) \leftarrow (bel\_dang(X)_{\langle expert(Y), [1,1] \rangle}, af_1)$$

This rule can be read: “the global state of the network regarding whether  $X$  is dangerous depends on the value of the function  $af_1$  applied to the levels of belief that  $X$  is dangerous in the nodes that are experts with certainty”. In this case, function  $af_1$  could obtain the output bound by averaging the upper and lower bounds of the input. For example, this function could be defined:  $af_1(\{[0.5, 1.0], [0.5, 0.7]\}) = [(0.5 + 0.5)/2, (1.0 + 0.7)/2] = [0.5, 0.85]$

We have now all the elements to define a NetDiff knowledge base containing the following elements:

- (1) Ground atoms of the form  $node/1$  and  $edge/2$  to represent the structure of the graph. For example:  $node(n_1)$ ,  $node(n_2)$ , and  $edge(n_1, n_2)$ .
- (2) Labels defined in  $\mathcal{L}$ . For example,  $dangerous(luabot)$ .
- (3) A protected ground atom (which will remain fixed) of the form  $tmax/1$  to represent the value of  $t_{max}$ . For example,  $tmax(2)$ .
- (4) NetDiff facts. For instance,  $f_1$  as defined in Example 2.2 below.
- (5) NetDiff rules that can be local rules, or global rules. For instance,  $lr_1$  in Example 2.2 below.

Formally, we say that a NetDiff KB is a pair  $DiffKB = (G, P)$  where  $G$  (Network DB in Figure 1) contains elements from (1) to (4) and  $P$  only contains elements from (5).

**Example 2.2.** Continuing from the running example, we define the following NetDiff facts and rules:

$$f_1 : (node(n_1), \langle expert(ddos), [1.0, 1.0] \rangle) : [0, t_{max}]$$

$$f_2 : (node(n_1), \langle expert(trojan), [0.9, 1.0] \rangle) : [0, t_{max}]$$

$$f_3 : (node(n_2), \langle expert(ddos), [0.6, 1.0] \rangle) : [0, t_{max}]$$

$$f_4 : (node(n_1), \langle bel\_dang(luabot), [0.7, 1.0] \rangle) : [0, t_{max}]$$

$$f_5 : (node(n_1), \langle bel\_dang(gooligan), [0.8, 1.0] \rangle) : [0, t_{max}]$$

$$f_6 : (edge(n_1, n_2), \langle close(trojan, win-10), [0.8, 1.0] \rangle) : [0, t_{max}]$$

$$lr_1 : bel\_dang(luabot) \stackrel{2}{\leftarrow} T, \langle expert(ddos), [0.9, 1.0] \rangle \wedge \\ \langle expert(trojan), [0.9, 1.0] \rangle \wedge \\ \langle bel\_dang(luabot), [0.5, 1.0] \rangle, sftTp$$

$$lr_2 : bel\_dang(gooligan) \stackrel{1}{\leftarrow} \langle close(trojan, win-10), [0.7, 1.0] \rangle, \\ \langle expert(trojan), [0.85, 1.0] \rangle \wedge \\ \langle bel\_dang(gooligan), [0.5, 1.0] \rangle, sftTp$$

$$gr_1 : dangerous(luabot) \leftarrow$$

$$bel\_dang(luabot)_{\langle expert(ddos), [0.9, 1.0] \rangle \wedge \langle expert(trojan), [0.9, 1.0] \rangle}, af_1$$

$$gr_2 : dangerous(gooligan) \leftarrow bel\_dang(gooligan)_{\langle expert(trojan), [0.85, 1.0] \rangle}, af_2$$

$$sftTp(V', v, NS, L) = \begin{cases} [0.7, 1.0] & \text{if } |V'| / |neigh(v)| > 0.5 \\ [0.0, 1.0] & \text{otherwise} \end{cases}$$

$$af_1(B) = [l, u] \text{ where: } l = \sum_{[l', u'] \in B} \frac{l'}{|B|} \quad \text{and} \quad u = \sum_{[l', u'] \in B} \frac{u'}{|B|}$$

$$af_2(B) = [l, u] \text{ where: } [l, u] \in B \text{ and } (l + u)/2 = \max\{(l' + u')/2 \text{ s.t. } [l', u'] \in B\}$$

Facts  $f_1$  and  $f_2$  state that  $node(n_1)$  is an expert on DDoS attacks with the highest confidence and an expert on trojan attacks with a confidence of at least 0.9. Facts  $f_4$  and  $f_5$  state that  $node(n_1)$  believes that luabot and gooligan are dangerous with a confidence of at least 0.7 and 0.8, respectively. Likewise,  $f_3$  describes  $node(n_2)$  as an expert on DDoS attacks with a confidence of at least 0.6, and fact  $f_6$  states that  $node(n_1)$  and  $node(n_2)$  have close interests regarding trojans in Windows 10 with a confidence of at least 0.8. Here, every NetDiff fact holds at every possible time step (from 0 to  $t_{max}$ ).

Additionally, rule  $lr_1$  says that a node will believe luabot to be dangerous with a weight of at least 0.7 (this is specified in the  $sftTp$  influence function) in the next two time steps, if at least half of their neighbors (also specified in  $sftTp$ ) are experts on DDoS and trojan attacks with a confidence of at least 0.9, and they believe luabot to be dangerous with a weight of at least 0.5. Function  $sftTp$  represents a ‘‘soft’’ tipping function [25, 49] where the affected nodes do not necessarily have to adopt exactly the same behavior as those that influence them, but can be partially influenced through the use of confidence intervals. Rule  $lr_2$  can be read analogously.

Finally,  $gr_1$  and  $gr_2$  are global rules and they can be read similarly; hence, we only intuitively describe  $gr_1$  as a rule that says that the confidence of the network regarding the danger of luabot (global label  $dangerous(luabot)$ ) is updated by applying function  $af_1$  to a set of local beliefs about the danger of luabot (local label  $bel\_dang(luabot)$ ), which come from the nodes that are experts on DDoS and trojan attacks with a confidence of at least 0.9. Function  $af_1$  yields as output a bound where the lower limit is the average of the lower limits of the input bounds and the upper limit is obtained in an analogous way. On the other hand, function  $af_2$  yields as output the interval whose average between the lower and upper bound is the highest. For example:  $af_2(\{[0.5, 1.0], [0.7, 1.0]\}) = [0.7, 1.0]$  because  $(0.7 + 1.0)/2 = \max\{(0.5 + 1.0)/2 = 0.75, (0.7 + 1.0)/2 = 0.85\}$ . ■

### 3. The Network Diffusion and Ontological Reasoning Modules

In this section, we will focus on describing the Network Diffusion Module (NDM) and Ontological Reasoning Module (ORM) of the NETDER architecture. In Section 3.1, we propose an implementation of the NDM on NetDiff

as introduced above, specify its semantics in an operational way by providing the details of a procedure to answer NetDiff queries, and define two new termination conditions for the diffusion process (which can be considered in conjunction with the one already studied in MANCaLog [52, 53]). Then, in Section 3.2, we propose the implementation of the ORM by extending the *Datalog+/-* formalism so that rules can take into account network knowledge; the operational semantics of this extension is defined via a modified chase procedure.

### 3.1. Network Diffusion

One of our main interests in implementing the NDM is to be able to represent network knowledge; though we can do this directly using NetDiff knowledge bases, we also want to reason about the (complex) dynamics that occur in these networked systems. Therefore, we need to have the capability to answer queries about the network; Boolean NetDiff queries are of the form:

$$\left( \underbrace{\bigwedge_{i \geq 0} (v_i, \text{condition}_i)}_{(a)} \wedge \underbrace{\bigwedge_{k \geq 0} (e_k, \text{condition}_k)}_{(b)} \wedge \underbrace{\bigwedge_{m \geq 0} (\Omega, \text{condition}_m)}_{(c)} \right) : \underbrace{[\tau_1, \tau_2]}_{(d)}$$

where  $\text{condition}_i = \bigwedge_{j > 0} \langle L_j, \text{bnd}_j \rangle$ ,  $\text{condition}_k = \bigwedge_{l > 0} \langle L_l, \text{bnd}_l \rangle$ , and  $\text{condition}_m = \bigwedge_{n > 0} \langle L_n, \text{bnd}_n \rangle$ ,  $v_i \in V$ ,  $e_j \in E$ ,  $L_j \in \mathcal{L}_{nloc}$ ,  $L_l \in \mathcal{L}_{eloc}$ ,  $L_n \in \mathcal{L}_{glo}$ , and  $[\tau_1, \tau_2] \subseteq [0, t_{max}]$ . Part (a) is a conjunction that requires the satisfaction of a conjunction of network atoms (i.e., a conjunction of labels with their bounds) in different nodes ( $v_i$ ), part (b) is an analogous requirement for edges ( $e_k$ ), and part (c) refers to network atoms for the whole network ( $\Omega$ ). Finally, part (d) determines the times at which (a), (b), and (c) must be satisfied.

The main data structure that we use in the following is called the NetDiff *state*, and it maps time steps into sets of component states (see Figure 4 for an example illustrating this structure).

**Definition 3.1** (NetDiff state). *Given a NetDiff knowledge base  $\text{DiffKB} = (G, P)$ , a NetDiff state for  $\text{DiffKB}$  is a mapping  $\text{net\_state} : [0, t_{max}] \rightarrow 2^{\mathcal{CS}}$ , where  $\mathcal{CS}$  is set of all possible component states over  $\mathcal{L}$  and  $\text{Graph} = (V, E)$  such that  $\mathcal{L} \cup V \cup E \subset G$ , and for each  $t \in [0, t_{max}]$ ,  $v \in V$ ,  $e \in E$ ,  $L_{nloc} \in \mathcal{L}_{nloc}$ ,  $L_{eloc} \in \mathcal{L}_{eloc}$ ,  $L_{glo} \in \mathcal{L}_{glo}$ , there exist intervals  $\text{bnd}_1, \text{bnd}_2, \text{bnd}_3 \subseteq [0, 1]$  such that:*

$$\langle L_{nloc}, \text{bnd}_1 \rangle, \langle e, \langle L_{eloc}, \text{bnd}_2 \rangle \rangle, \langle \Omega, \langle L_{glo}, \text{bnd}_3 \rangle \rangle \in \text{net\_state}(t)$$

Algorithm 1 (Figure 3) implements the NetDiff diffusion process for a KB  $\text{DiffKB} = (G, P)$ —note that it terminates when the application of rules no longer changes the NetDiff state. The procedure considers the NetDiff initial state as the state where the uncertainty is maximum (every label is associated with the bound  $[0, 1]$ ), and its output is the NetDiff state that is obtained when a fixed point is reached<sup>3</sup>. Note that when a rule cannot be applied on a NetDiff state, then the output of the “*apply*” function is the same input NetDiff state.

Now, given NetDiff query  $Q$  and knowledge base  $\text{DiffKB} = (G, P)$ , if  $\text{net\_state}$  is the NetDiff state obtained from Algorithm 1 over  $\text{DiffKB}$ , then the query  $Q$  can be answered based on  $\text{net\_state}$  and the following function:

$$\text{checkCondInState}(\text{cond}, \text{net\_state}) \tag{1}$$

defined as follows;  $\text{cond}$  is a NetDiff query of the form:

$$\left( \bigwedge_{i \geq 0} \left( v_i, \bigwedge_{j > 0} \langle L_j, \text{bnd}_j \rangle \right) \wedge \bigwedge_{k \geq 0} \left( e_k, \bigwedge_{l > 0} \langle L_l, \text{bnd}_l \rangle \right) \wedge \bigwedge_{m \geq 0} \left( \Omega, \bigwedge_{n > 0} \langle L_n, \text{bnd}_n \rangle \right) \right) : [\tau_1, \tau_2]$$

<sup>3</sup>Though we do not provide details of its implementation, we refer the reader to works on logic-based diffusion models [52, 53] for one possibility.

---

**Algorithm 1** NetDiff Diffusion
 

---

**Require:** NetDiff knowledge base  $DiffKB = (G, P)$

```

1:  $ans := \emptyset$ 
2: // Every label is associated with a bound  $[0, 1]$ 
3:  $net\_state_{prev} := \text{NetDiff Initial State}$ 
4:  $net\_state_{now} := \text{NetDiff Initial State}$ 
5: for  $fact$  in  $G$  do
6:   // NetDiff state is updated to enforce each NetDiff fact
7:    $net\_state_{now} := \text{apply}(net\_state_{now}, fact)$ 
8: end for
9: for  $r$  in  $P$  do
10:  // NetDiff state is updated by applying each NetDiff rule  $r$  once
11:   $net\_state_{now} := \text{apply}(net\_state_{now}, r)$ 
12: end for
13: while  $net\_state_{prev} \neq net\_state_{now}$  do
14:  // While loop implementing a fixed point operator
15:   $net\_state_{prev} := net\_state_{now}$ 
16:  for  $r$  in  $P$  do
17:    // NetDiff state is updated by applying each NetDiff rule  $r$  once
18:     $net\_state_{now} := \text{apply}(net\_state_{now}, r)$ 
19:  end for
20: end while
21: return  $net\_state_{now}$ 

```

---

Fig. 3. The NetDiff Diffusion Algorithm

and  $checkCondInState$  returns “Yes” iff for each  $\tau \in [\tau_1, \tau_2]$  with

$$(v_i, \langle L_j, bnd'_j \rangle), (e_k, \langle L_l, bnd'_l \rangle), (\Omega, \langle L_n, bnd'_n \rangle) \in net\_state(t)$$

it holds that  $bnd'_j \subseteq bnd_j$ ,  $bnd'_l \subseteq bnd_l$ , and  $bnd'_n \subseteq bnd_n$ .

Intuitively, this function simply checks over a NetDiff state ( $net\_state$ ) the conditions (network atoms in  $cond$ ) for each node, edge, and  $\Omega$  in every specified time step, as illustrated in Example 3.1. To answer the query,  $Q$  is evaluated over  $net\_state$  by invoking  $checkCondInState(Q, net\_state)$ , which checks the conditions specified in  $Q$  over the NetDiff final state that is output by Algorithm 1. Note that the termination of Algorithm 1 depends on the evolution of the NetDiff states; we will analyze this issue later on in the context of Theorem 3.1.

**Example 3.1.** Consider the labels  $\mathcal{L} = \mathcal{L}_{nloc} \cup \mathcal{L}_{eloc} \cup \mathcal{L}_{glo}$ , where:

$$\begin{aligned} \mathcal{L}_{nloc} &= \{l_1 : \text{expert}(\text{trojan}), l_2 : \text{bel\_dang}(\text{gooligan})\}, \\ \mathcal{L}_{eloc} &= \{l_3 : \text{close}(\text{trojan}, \text{win-10})\}, \\ \mathcal{L}_{glo} &= \{l_4 : \text{dangerous}(\text{gooligan})\}. \end{aligned}$$

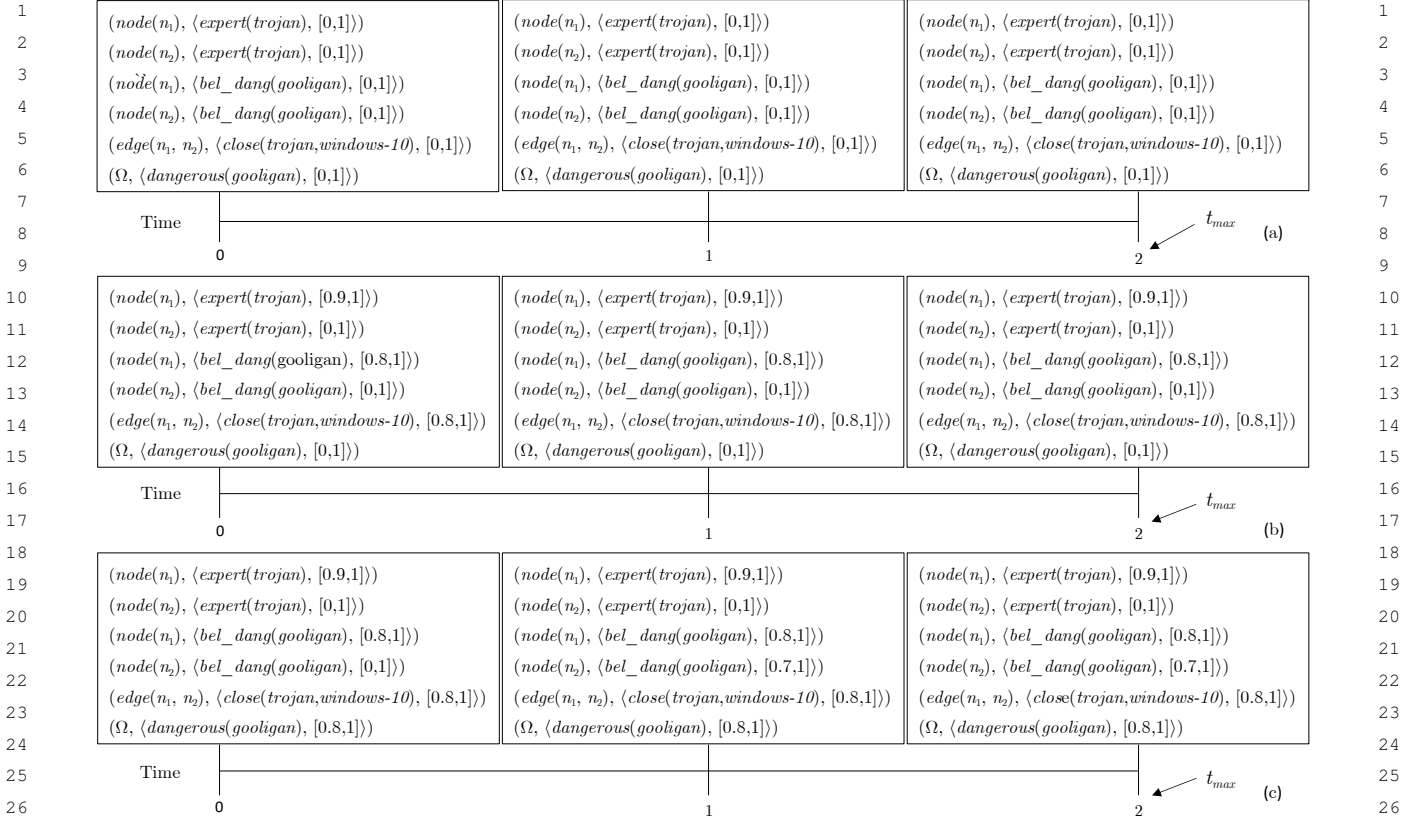


Fig. 4. Evolution of the network for the NetDiff diffusion process (cf. Algorithm 1) from Example 3.1. (a) Initial state of the network; (b) State after applying facts  $f_2, f_5$ , and  $f_6$ ; (c) State of the network after applying rules  $lr_2$ , and  $gr_2$ .

Let NetDiff knowledge base  $DiffKB = (G, P)$  where

$$G = \{g_1 : node(n_1), g_2 : node(n_2), g_3 : edge(n_1, n_2)\} \cup \mathcal{L} \cup \{tmax(2)\} \cup \{f_2, f_5, f_6\}$$

and  $P = \{lr_2, gr_2\}$  obtained from a simplification of the running example and Example 2.2. Consider the following Boolean NetDiff queries:

$$Q_1 = (node(n_1), \langle expert(trojan), [0.5, 1.0] \rangle \wedge \langle bel\_dang(gooligan), [0.8, 1.0] \rangle) : [0, 2]$$

$$Q_2 = (node(n_1), \langle expert(trojan), [0.6, 1.0] \rangle) \wedge (edge(n_1, n_2), \langle close(trojan, win-10), [0.8, 1.0] \rangle) : [0, 2]$$

$$Q_3 = (node(n_2), \langle bel\_dang(gooligan), [0.6, 1.0] \rangle) : [0, 0]$$

$$Q_4 = (node(n_2), \langle bel\_dang(gooligan), [0.6, 1.0] \rangle) : [2, 2]$$

Query  $Q_1$  is asking whether, within the interval time  $[0, 2]$ ,  $n_1$  is an expert in topic “trojan” with certainty above 0.5 and she believes that “gooligan” is dangerous with certainty above 0.8.

In order to answer queries, Algorithm 1 starts from the NetDiff state with maximum uncertainty, where every label has bound  $[0, 1]$  (top of Figure 4). Next, the NetDiff state is updated to enforce the facts  $f_2, f_5, f_6$  (middle part of Figure 4) and a new NetDiff state is obtained. Then, the NetDiff state is updated by applying each rule once (bottom row of Figure 4) and a new NetDiff state is obtained, which in this case is the fixed point since no further changes can occur. Finally, whenever  $DiffKB = (G, P)$  reaches a fixed point, queries can be answered using the final state and

function  $checkCondInState(cond, net\_state)$  (Eq. 1): the answer for  $Q_1$  is “yes” because (a) the label  $expert(trojan)$  has bounds  $[0.9, 1.0]$ , which is a subset of  $[0.5, 1.0]$  (specified in the query), for node( $n_1$ ) in each time  $\tau \in [0, 2]$ , and (b) the label  $bel\_dang(gooligan)$  has bounds  $[0.8, 1.0]$ , exactly as specified in the query, for node( $n_1$ ) in each time  $\tau \in [0, 2]$ .

The other queries can be solved analogously—the answer for query  $Q_2$  is “yes”, the answer for query  $Q_3$  is “no”, and the answer for query  $Q_4$  is “yes” (note that the only difference between  $Q_3$  and  $Q_4$  is that they refer to different times). ■

It is important to note that the monotonicity property holds with respect to the same label at the same time point, as can be seen by reading Figure 4 vertically (for instance, focusing on time point 0), but does *not* necessarily hold with respect to the same label in different time points. In other words, intervals are always tightened by Algorithm 1 when updating the NetDiff state, but for a given NetDiff state it is possible for an interval that is associated with a label at time point 0, to start with  $[0, 1]$ , change to  $[0.5, 1]$  at time point 1, and then grow again to  $[0.25, 1]$  at time 2.

Now, we will analyze the conditions under which Algorithm 1 terminates, building on the analysis done by Shakarian et al. [52, 53] where fixed points are guaranteed to be reached in finite time because influence functions are restricted (condition (a) of Theorem 3.1 below). Even though NetDiff knowledge bases as defined here have a new kind of rule (global rules), their application depends on the local labels and not the global ones; hence, if a fixed point is reached for local labels then a fixed point must be reached for global labels, and we can therefore ignore global rules for this analysis. Likewise, though our NetDiff knowledge bases include variables, they can be easily transformed into ground knowledge bases since we know beforehand which labels are available (and the set of labels is finite); therefore, these ground knowledge bases (ignoring global rules) are equivalent to MANCaLog programs with respect to termination. As a result, the only difference with the original framework is that our NetDiff knowledge bases can define more general influence functions; however, if condition (a) of Theorem 3.1 holds, then the termination of Algorithm 1 is guaranteed. In the following theorem we define two additional conditions to get termination guarantees (proof in Appendix B).

**Theorem 3.1.** *Given a ground NetDiff knowledge base  $DiffKB = (G, P)$ , Algorithm 1 is guaranteed to terminate if one of the following conditions holds:*

- a) *Every local rule in  $P$  only contains influence functions that depend on the relationship between the total number of (eligible) neighbors of the target node and a subset of these that satisfy a certain criterion (qualifying neighbors).*
- b) *The number of bits used to represent the confidence intervals assigned to every label is bounded by a constant  $b$ .*
- c) *Every local rule in  $P$  uses a value of  $\Delta t \neq 0$ .*

Additionally, the running time of Algorithm 1 is as follows, depending on the specific conditions:

- Case (a):  $O(|DiffKB| \cdot t_{max} \cdot |V| \cdot (IF_{time} \cdot d_{\Omega}^{in} + |\mathcal{L}_{nl}| \cdot |\mathcal{L}_{el}| \cdot d_{\Omega}^{in^2}))$
- Case (b):  $O(|DiffKB| \cdot t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{el}| \cdot |\mathcal{L}_{nl}| \cdot d_{\Omega}^{in}) \cdot 2^{2b})$
- Case (c):  $O(|V| \cdot (IF_{time} + |\mathcal{L}_{el}| \cdot |\mathcal{L}_{nl}| \cdot d_{\Omega}^{in}) \cdot |DiffKB|^{t_{max}})$

In each case,  $d_{\Omega}^{in}$  is the maximum in-degree in the network and  $IF_{time}$  is the worst case cost associated with any influence function used in  $P$ 's local rules.

Note that for non-ground knowledge bases,  $|DiffKB|$  depends on the size of the grounding of  $DiffKB$ . The following corollary is a direct consequence of the proof of Theorem 3.1.

**Corollary 3.1.** *Given NetDiff knowledge base  $DiffKB = (G, P)$ , if the time required to apply every influence function of local rules in  $P$  is bounded by a polynomial in the size of the network, and one of conditions (a)–(c) from Theorem 3.1 holds, then Algorithm 1 for  $DiffKB$  terminates in polynomial time in the data complexity (i.e., only network  $G$  is considered as part of the input).*

Theorem 3.1 and Corollary 3.1 state that we can efficiently compute answers to NetDiff queries, given a NetDiff knowledge base  $DiffKB = (G, P)$ , using Function  $checkCondInState$  (Eq. 1) and Algorithm 1 under certain conditions. The following observation simply states that convergence is not always guaranteed.

Step	Time	G	L
0	0	node(n <sub>1</sub> )	[0.7, 1.0]
0	0	node(n <sub>2</sub> )	[0.7, 1.0]
0	0	node(n <sub>3</sub> )	[1.0, 1.0]
1	0	node(n <sub>1</sub> )	[0.85, 1.0]
1	0	node(n <sub>2</sub> )	[0.85, 1.0]
1	0	node(n <sub>3</sub> )	[1.00, 1.0]
2	0	node(n <sub>1</sub> )	[0.925, 1.0]
2	0	node(n <sub>2</sub> )	[0.925, 1.0]
2	0	node(n <sub>3</sub> )	[1.000, 1.0]
...	...	...	...

Fig. 5. Networks evolution as the single local rule is applied. Edges are not included since there are no edge local labels defined in the example.

**Observation 3.1.** *There exists a NetDiff knowledge base  $\text{DiffKB} = (G, P)$  such that Algorithm 1 over  $\text{DiffKB}$  never terminates.*

The following example suffices to show that the observation is true.

**Example 3.2.** *We assume that we have the following components:*

- Three nodes:  $\text{node}(n_1)$ ,  $\text{node}(n_2)$ , and  $\text{node}(n_3)$ .
- Six edges:  $\text{edge}(n_1, n_2)$ ,  $\text{edge}(n_2, n_1)$ ,  $\text{edge}(n_1, n_3)$ ,  $\text{edge}(n_3, n_1)$ ,  $\text{edge}(n_2, n_3)$ , and  $\text{edge}(n_3, n_2)$ .
- A single node local label  $L$ , this is,  $\mathcal{L} = \mathcal{L}_{nloc} \cup \mathcal{L}_{eloc} \cup \mathcal{L}_{glo}$ ,  $\mathcal{L}_{nloc} = \{L\}$ ,  $\mathcal{L}_{eloc} = \{\}$ , and  $\mathcal{L}_{glo} = \{\}$ .
- A single influence function:  $\text{if}_1(V', v, \text{net\_state}, L, t) = [l_1, u_1]$  where:

$$l_1 = \sum_{\text{condition}} l' / |V'|, \quad u_1 = \sum_{\text{condition}} u' / |V'| \text{ and}$$

where “condition” is defined as

$$\text{checkCondInState}((v', \langle L, [l', u'] \rangle)) : [\tau, \tau], \text{net\_state}) \text{ is “yes” } \wedge v' \in V'.$$

Note that “condition” states that the output of  $\text{if}_1$  is obtained from bounds of label  $L$  (target in the rule applied) for neighbors of target node at time  $\tau$  when the rule was applied.

- Three NetDiff facts and one rule, respectively:

$$f_1 : (\text{node}(n_1), \langle L, [0.7, 1.0] \rangle) : [0, t_{max}]$$

$$f_2 : (\text{node}(n_2), \langle L, [0.7, 1.0] \rangle) : [0, t_{max}]$$

$$f_3 : (\text{node}(n_3), \langle L, [1.0, 1.0] \rangle) : [0, t_{max}]$$

$$lr_1 : L_T \stackrel{0}{\leftarrow} (\mathcal{T}, \langle L, [0.5, 1.0] \rangle, \text{if}_1)$$

In this context, we define the NetDiff knowledge base  $\text{DiffKB} = (G, P)$  where:

$$G = \{\text{node}(n_1), \text{node}(n_2), \text{node}(n_3), \text{edge}(n_1, n_2), \text{edge}(n_2, n_1), \text{edge}(n_1, n_3), \\ \text{edge}(n_3, n_1), \text{edge}(n_2, n_3), \text{edge}(n_3, n_2)\} \cup \mathcal{L} \cup \{t_{max}(0)\} \cup \{f_1, f_2, f_3\}$$

$$P = \{lr_1\}$$

Clearly, none of conditions (a)–(c) from Theorem 3.1 hold (since a priori we cannot assume that the number of bits to represent intervals is bounded); hence, we cannot ensure termination of Algorithm 1 on  $\text{DiffKB} = (G, P)$ . Likewise, Figure 5 shows the evolution of the NetDiff states when Algorithm 1 is executed on  $\text{DiffKB} = (G, P)$ . Note that the network will never reach a stable state at time 0 because the intervals for network atoms formed with  $L$  in  $\text{node}(n_1)$  and  $\text{node}(n_2)$  can always be tighter when the influence function is applied. The reason is that the influence function takes an average of the lower and upper values of the intervals of network atoms formed with  $L$

1 for each neighbor of the target node in the applied rule, and therefore the process (the execution of Algorithm 1 on  
2  $DiffKB = (G, P)$ ) never terminates. ■

3 Note that even though it is easy to encounter examples—like the one above—in which diffusion processes do not  
4 theoretically terminate, in practice the condition regarding bounded number of bits in the representation of intervals  
5 will be satisfied.

### 7 3.2. Ontological Modeling: An Implementation of the ORM

9 In this section, we implement the Ontological Reasoning Module of the NETDER architecture. We first introduce  
10 the NETDER ontology language, which is an extension of the well-known family of ontology languages known  
11 as existential rules or *Datalog+/-*; as we will show later, the extension is only a syntactic one for representational  
12 convenience to be able to easily refer to network knowledge. After presenting its syntax and semantics, we develop  
13 a modified chase procedure for query answering.

#### 14 3.2.1. Syntax

15 We now set up the basic units of language syntax and how they allow us to combine classical ontological and  
16 network knowledge. Let  $DiffKB = (G, P)$  be a NetDiff knowledge base with a finite set of labels  $\mathcal{L} = \mathcal{L}_{glo} \cup \mathcal{L}_{nloc} \cup$   
17  $\mathcal{L}_{eloc}$ , where  $\mathcal{L}_{glo}$  is a set of global labels,  $\mathcal{L}_{nloc}$  is a set of node local labels, and  $\mathcal{L}_{eloc}$  is a set of edge local labels.  
18 We have following language elements:

19 **Definition 3.2** (Network local annotation). A network local annotation relative to  $P$ , is a (finite) conjunction of pairs  
20  $\lambda(\mathbf{U}) = \langle A_1, bnd_1 \rangle \wedge \dots \wedge \langle A_k, bnd_k \rangle$  where every  $A_{1 \leq i \leq k}$  is an atom over  $\mathcal{R}, \mathbf{U}$ , and  $\Delta, \mathbf{U}$  is a sequence of variables  
21 such that  $\mathbf{U} \subset \mathcal{V}$ , every  $bnd_{1 \leq i \leq k} \subseteq [0, 1]$ , and there exists a substitution  $\sigma$  such that  $\sigma A_{1 \leq i \leq k} \in \mathcal{L}_{eloc} \cup \mathcal{L}_{nloc}$ .

22 **Definition 3.3** (Network global annotation). A network global annotation relative to  $P$  is a (finite) conjunction of  
23 pairs  $\gamma(\mathbf{U}) = \langle B_1, bnd_1 \rangle \wedge \dots \wedge \langle B_k, bnd_k \rangle$  where  $B_{1 \leq i \leq k}$  is an atom over  $\mathcal{R}, \mathbf{U}$ , and  $\Delta, \mathbf{U}$  is a sequence of variables  
24 such that  $\mathbf{U} \subset \mathcal{V}$ ,  $bnd_{1 \leq i \leq k} \subseteq [0, 1]$ , and there exists a substitution  $\sigma$  such that  $\sigma B_{1 \leq i \leq k} \in \mathcal{L}_{glo}$ .

25 **Definition 3.4** (Network component target). A network component target is a pair  $(c, \lambda(\mathbf{U}))$  where  $c$  is an atom  
26 over  $\mathcal{R}, \mathcal{V}$ , and  $\Delta, \lambda(\mathbf{U})$  is a network local annotation, and there exists a substitution  $\sigma$  such that  $\sigma c \in \mathcal{G}$ .

27 These elements simply assign bounds to labels, and such structures are assigned to components—we will illustrate  
28 them in an example soon. We now have the necessary elements to define the first kind of NETDER rule.

29 **Definition 3.5** (NETDER tuple-generating dependency). A NETDER tuple-  
30 generating dependency (TGD) is a formula of the form:

$$31 \quad \forall \mathbf{X} \mathbf{Y} \mathbf{Q} \mathbf{S} \mathbf{U} \Upsilon(\mathbf{X}) \wedge \Phi(\mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{Q}, \mathbf{R}) \wedge \Xi(\mathbf{S}, \mathbf{T}) : \gamma(\mathbf{U})$$

32 where  $\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{Q}, \mathbf{R}, \mathbf{S}, \mathbf{T}, \mathbf{U} \subset \mathcal{V}$  are sequences of variables,  $\mathbf{Q} \cup \mathbf{S} \subseteq \mathbf{X} \cup \mathbf{Y} \cup \mathbf{U}$ ,  $\mathbf{R} \cup \mathbf{T} = \mathbf{Z}$ ,  $\Upsilon(\mathbf{X})$  and  $\Psi(\mathbf{Q}, \mathbf{R})$   
33 are conjunctions of atoms,  $\Phi(\mathbf{Y})$  and  $\Xi(\mathbf{S}, \mathbf{T})$  are conjunctions of network component targets, and  $\gamma(\mathbf{U})$  is a network  
34 global annotation.

35 These rules can be read as follows: whenever the conditions specified in formula  $\Upsilon$  hold in the ontology, the  
36 conditions in formula  $\Phi$  hold in the network at a local level, and the conditions in  $\gamma$  hold at a global level, then the  
37 conditions specified in the head also hold (again divided into those for the ontology via  $\Psi$  and the network at a local  
38 level via  $\Xi$ ). Figure 6 illustrates this visually with an example. NETDER EGDs and NCs are analogous extensions:  
39

40 **Definition 3.6** (NETDER equality-generating dependency). A NETDER equality-generating dependency (EGD) is  
41 a formula of the form:

$$42 \quad \forall \mathbf{S} \mathbf{T} \mathbf{U} \Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow (X_i = X_j) : \gamma(\mathbf{U})$$

43 where  $\mathbf{S}, \mathbf{T}, \mathbf{U} \subset \mathcal{V}$  are sequences of variables,  $X_i, X_j \in \mathbf{S} \cup \mathbf{T} \cup \mathbf{U}$ ,  $\Upsilon(\mathbf{S})$  is a conjunction of atoms,  $\Phi(\mathbf{T})$  is a  
44 conjunction of network component targets, and  $\gamma(\mathbf{U})$  is a network global annotation.

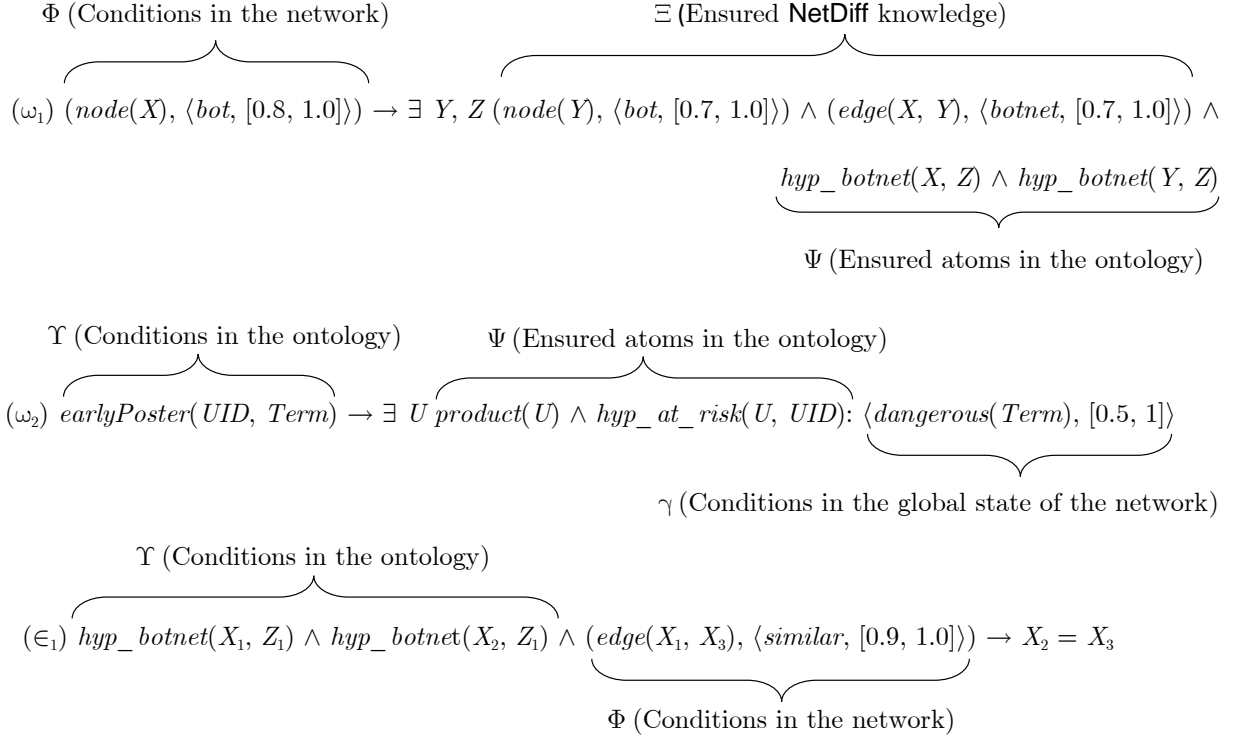


Fig. 6. Illustration of instantiations of two NETDER TGDs and one EGD (subformula names are as they appear in Definitions 3.5 and 3.6).

**Definition 3.7** (NETDER negative constraint). A NETDER negative constraint (NC) is a formula of the form:

$$\forall \mathbf{STU} \Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow \perp : \gamma(\mathbf{U})$$

where  $\mathbf{S}, \mathbf{T}, \mathbf{U} \in \mathcal{V}$  are sequences of variables,  $\Upsilon(\mathbf{S})$  is a conjunction of atoms,  $\Phi(\mathbf{T})$  is a conjunction of network component targets, and  $\gamma(\mathbf{U})$  is a network global annotation.

In these formulas, we sometimes omit the universal quantifiers. NETDER knowledge bases comprise then both the ontological knowledge and the network knowledge.

**Definition 3.8** (NETDER knowledge base). A NETDER knowledge base is a tuple  $KB = (D, G, \Sigma, P)$ , where  $D$  is a finite set of atoms with predicates from a relational schema  $\mathcal{R}$  and arguments from a set of constants  $\Delta$ ,  $\Sigma$  is a set of NETDER TGDs, EGDs, and NCs over  $\mathcal{R}$ , and  $DiffKB = (G, P)$  is a NetDiff knowledge base (cf. Page 10).

**Example 3.3.** Continuing from Example 3.1, we show an instance of a NETDER knowledge base  $KB = (D, G', \Sigma, P)$  where  $G'$  extends  $G$  from Example 3.1 with additional elements, and  $P$  contains the same NetDiff rules from Example 3.1. We define other additional local labels: (i) *bot*, which is a node local label to represent a bot actor; (ii) *similar*, which is an edge local label to represent two similar actors, and (iii) *botnet*, which is an edge local label to represent two actors in the same botnet.  $D$  contains the following atoms:

$$D = \{d_1 : user(n_1, ponzio), d_2 : earlyPoster(n_1, gooligan), d_3 : product(win-10)\}$$

$G'$  is an extension of  $G$  from Example 3.1 that contains the following elements:

$$\begin{aligned}
G' = G \cup \{ & g_4 : \text{node}(n_3), g_5 : \text{edge}(n_2, n_3), \\
& g_6 : \text{bot}, g_7 : \text{botnet}, g_8 : \text{similar}, \\
& g_9 : \left( \text{node}(n_1), \langle \text{bot}, [0.9, 1.0] \rangle \right) : [0, t_{max}], \\
& g_{10} : \left( \text{node}(n_2), \langle \text{bot}, [0.6, 1.0] \rangle \right) : [0, t_{max}], \\
& g_{11} : \left( \text{edge}(n_2, n_3), \langle \text{botnet}, [0.75, 1.0] \rangle \right) : [0, t_{max}], \\
& g_{12} : \left( \text{edge}(n_1, n_2), \langle \text{similar}, [0.9, 1.0] \rangle \right) : [0, t_{max}] \}
\end{aligned}$$

$\Sigma$  contains the following set of NETDER TGDs and EGDs:

$$\begin{aligned}
(\omega_1) & \left( \text{node}(X), \langle \text{bot}, [0.8, 1.0] \rangle \right) \rightarrow \exists Y, Z \left( \text{node}(Y), \langle \text{bot}, [0.7, 1.0] \rangle \right) \wedge \\
& \left( \text{edge}(X, Y), \langle \text{botnet}, [0.7, 1.0] \rangle \right) \wedge \\
& \text{hyp\_botnet}(X, Z) \wedge \text{hyp\_botnet}(Y, Z) \\
(\omega_2) & \text{earlyPoster}(UID, Term) \rightarrow \exists U \text{product}(U) \wedge \text{hyp\_at\_risk}(U, UID) : \langle \text{dangerous}(Term), [0.5, 1.0] \rangle \\
(\omega_3) & \left( \text{node}(X), \langle \text{expert}(S), [0.9, 1.0] \rangle \wedge \langle \text{bel\_dang}(Term), [0.8, 1.0] \rangle \right) \\
& \rightarrow \exists U, UID, N \text{product}(U) \wedge \text{user}(UID, N) \wedge \text{hyp\_at\_risk}(U, UID) \\
(\omega_4) & \text{earlyPoster}(UID, Term) \rightarrow \exists N \text{user}(UID, N) \\
(\epsilon_1) & \text{hyp\_botnet}(X_1, Z_1) \wedge \text{hyp\_botnet}(X_2, Z_1) \wedge \left( \text{edge}(X_1, X_3), \langle \text{similar}, [0.9, 1.0] \rangle \right) \rightarrow X_2 = X_3 \\
(\epsilon_2) & \text{hyp\_botnet}(X_1, Z_1) \wedge \text{hyp\_botnet}(X_2, Z_2) \wedge \left( \text{edge}(X_1, X_2), \langle \text{botnet}, [0.7, 1.0] \rangle \right) \rightarrow Z_1 = Z_2
\end{aligned}$$

Rule  $\omega_1$  can be read as follows: “if the network contains a node  $X$  that is believed to be a bot with confidence at least 0.8, then there exists another node  $Y$  believed to be a bot with confidence at least 0.7, an edge between  $X$  and  $Y$  labeled as part of a botnet also with confidence at least 0.7, and two hypotheses stating that  $X$  and  $Y$  belong to the same botnet  $Z$ ”.

Similarly, rule  $\epsilon_1$  states “If we have the hypothesis that  $X_1$  and  $X_2$  belong to the same botnet  $Z_1$ , and  $X_1$  is similar to  $X_3$  with confidence at least 0.9 in the network, then  $X_2$  and  $X_3$  should be equal”. The remaining rules can be read similarly.

Also, NETDER TGDs  $\omega_1$  and  $\omega_2$ , and NETDER EGD  $\epsilon_1$  are used in Figure 6 to show how rules can be instantiated, according to Defs. 3.5 and 3.6. ■

With the syntax in place, in the next section we develop the semantics of the NETDER language.

### 3.2.2. Semantics: A Modified Chase

In this section, we address the semantics for reasoning tasks *in the ORM only* (the full semantics for NETDER reasoning tasks will be discussed in the next section); this semantics will be defined in an operational way. Therefore, we focus on extending the well-known *chase* procedure to work with NETDER rules, which will be essential towards computing query answers in our model. In the following, we modify the chase procedure in order to consider the network component targets in the TGDs and EGDs, and the network global annotations in the formulas when we obtain the atomic consequences derived from the ontology. Query answering under general classical TGDs is undecidable [5], but under different sets of conditions a universal model can be obtained and used to answer queries. Given a program  $\Sigma$  with only classical TGDs, this can be achieved by exploiting the chase procedure, originally introduced for checking implication of dependencies and query containment [5, 24, 31]. In the rest of this work, by “chase” we refer to the version of the procedure commonly known as the *oblivious chase*.

NETDER TGD Chase Rule. Consider a NETDER KB =  $(D, G, \Sigma, P)$ , a time frame  $[\tau_1, \tau_2] \subseteq [0, t_{max}]$ , and a NETDER TGD  $\sigma \in \Sigma$  of the form  $\Upsilon(\mathbf{X}) \wedge \Phi(\mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{Q}, \mathbf{R}) \wedge \Xi(\mathbf{S}, \mathbf{T}) : \gamma_\sigma(\mathbf{U})$ . Then,  $\sigma$  is applicable to  $(D, G)$  at time frame  $[\tau_1, \tau_2]$  if and only if:

- there exists homomorphism  $h$  mapping atoms of  $\Upsilon(\mathbf{X})$  s.t.  $h(\Upsilon(\mathbf{X})) \subseteq D$ ,
- there exists homomorphism  $h_1$  that extends  $h$  s.t. the output of function  $checkCondInState(Q_1, net\_state)$  (Equation 1, Page 12) is “yes”, where  $Q_1 = (h_1(\Phi(\mathbf{Y}))) : [\tau_1, \tau_2]$  and  $net\_state$  is the output of Algorithm 1 over the NetDiff knowledge base  $DiffKB = (G, \emptyset)^4$  (cf. “get current NetDiff state” in Figure 1), and
- the output of function  $checkCondInState(Q_2, net\_state)$  is “yes”, where  $Q_2 = (\Omega, h_1(\gamma_\sigma(\mathbf{U}))) : [\tau_1, \tau_2]$  and  $net\_state$  is the same as above.

Let  $\sigma$  be applicable to  $(D, G)$  at time frame  $[\tau_1, \tau_2]$ , and  $h_2$  be a homomorphism that extends  $h_1$  as follows: for each  $Q_i \in \mathbf{Q} \cup \mathbf{S}$ ,  $h_2(Q_i) = h_1(Q_i)$ ; for each  $Z_j \in \mathbf{Z}$ ,  $h_2(Z_j) = z_j$ , where  $z_j \in \mathcal{N}$  is a fresh null,  $z_j$  does not occur in  $D \cup G$ , and  $z_j$  lexicographically follows all other nulls already introduced. When  $\sigma$  is applied on  $(D, G)$  then:

- Atoms  $\alpha_1, \dots, \alpha_m$  are added to  $D$  where  $h_2(\Psi(\mathbf{Q}, \mathbf{R})) = \alpha_1 \wedge \dots \wedge \alpha_m$ , if not already in  $D$ ,
- If  $\Xi(\mathbf{S}, \mathbf{T}) = (c_1(X_1), na_1(\mathbf{Y}_1)) \wedge \dots \wedge (c_{m'}(X_{m'}), na_{m'}(\mathbf{Y}_{m'}))$  and the answer of function  $checkCondInState(Q_{1 \leq i \leq n}, net\_state)$  is “no”, where
  - \*  $Q_{1 \leq i \leq n} = (c'_1, na'_1) \wedge \dots \wedge (c'_{m'}, na'_{m'})$ ,
  - \*  $c'_1 = h_2(c_1(X_1)), \dots, c'_{m'} = h_2(c_{m'}(X_{m'}))$ ,
  - \*  $na'_1 = h_1(na_1(\mathbf{Y}_1)), \dots, na'_{m'} = h_1(na_{m'}(\mathbf{Y}_{m'}))^5$ , and
  - \*  $net\_state$  is the output of Algorithm 1 over NetDiff base  $DiffKB = (G, \emptyset)$ .

Then,

- \* Atoms  $c'_1, \dots, c'_{m'}$  are added to  $G$ , if not already in  $G$ ,
- \* NetDiff facts  $(c'_1, na'_1) : [\tau_1, \tau_2], \dots, (c'_{m'}, na'_{m'}) : [\tau_1, \tau_2]$  are added to  $G^6$ .

The chase of level up to 0 of  $(D, G)$  at time frame  $[\tau_1, \tau_2] \subseteq [0, t_{max}]$  relative to the set of NETDER TGDs  $\Sigma$ , denoted  $chase_{[\tau_1, \tau_2]}^0(D, G)$ , is defined assigning to every element in  $D$  and  $G$  the (derivation) level 0. For every  $k \geq 1$ , the chase of level up to  $k$  of  $(D, G)$  at time frame  $[\tau_1, \tau_2] \subseteq [0, t_{max}]$  relative to the set of NETDER TGDs  $\Sigma$ , denoted  $chase_{[\tau_1, \tau_2]}^k(D, G) = (D^k, G^k)$ , is constructed as follows: let  $(I_1, I'_1, I''_1), \dots, (I_n, I'_n, I''_n)$  be all possible images of (both) bodies and global annotations of NETDER TGDs in  $\Sigma$  where  $I_i$  are images of the conjunction of atoms in the body,  $I'_i$  are images of the conjunction of network component targets in the body, and  $I''_i$  are images of global annotations, such that:

- $I_1, \dots, I_n \subseteq D^{k-1}$  for some homomorphism  $h$ ,
- the output of function  $checkCondInState(Q_{1 \leq i \leq n}, net\_state)$  is “yes”, where we have  $Q_{1 \leq i \leq n} = (I'_i) : [\tau_1, \tau_2]$  and  $net\_state$  is the output of Algorithm 1 over the NetDiff knowledge base  $DiffKB = (G^{k-1}, \emptyset)$  for some other homomorphism  $h_1$  that extends  $h$ , and
- the output of function  $checkCondInState(Q_{1 \leq i \leq n}, net\_state)$  is “yes”, where we have  $Q_{1 \leq i \leq n} = (\Omega, I''_i) : [\tau_1, \tau_2]$  ( $net\_state$  is the same as the one in the previous item).

Then, every corresponding NETDER TGD application can be performed, choosing the applied NETDER TGDs and two appropriate homomorphisms in a (fixed) linear and lexicographic order, respectively, and assigning to every new atom the (derivation) level  $k$ .

The chase of  $(D, G)$ , denoted  $chase_{[\tau_1, \tau_2]}(D, G)$ , is then defined as the limit for  $k \rightarrow \infty$  of  $chase_{[\tau_1, \tau_2]}^k(D, G)$ . The chase graph for a NETDER KB is the directed graph consisting of  $chase_{[\tau_1, \tau_2]}(D, G)$  as the set of nodes and having an edge from  $a$  to  $b$  if and only if  $b$  is obtained from a one-step application of a NETDER TGD.

<sup>4</sup>In this case, the NetDiff knowledge base has no rules because we only need the state from NetDiff facts  $G$ .

<sup>5</sup>Here, we use  $h_1$  as labels cannot have null values.

<sup>6</sup>We write  $(c, na_1 \wedge \dots \wedge na_k) : [\tau_1, \tau_2]$  as a summarized form for the NetDiff facts  $(c, na_1) : [\tau_1, \tau_2], \dots, (c, na_k) : [\tau_1, \tau_2]$ .

NETDER EGD Chase Rule. Let NETDER be a knowledge base  $KB = (D, G, \Sigma, P)$ , a time frame  $[\tau_1, \tau_2] \subseteq [0, t_{max}]$ , and a NETDER EGD  $\epsilon \in \Sigma$  of the form  $\Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow (X_i = X_j) : \gamma_\epsilon(\mathbf{U})$ . Then,  $\epsilon$  is applicable to  $(D, G)$  at time frame  $[\tau_1, \tau_2]$  if and only if:

- there exists a homomorphism  $h$  mapping atoms of  $\Upsilon(\mathbf{S})$  s.t.  $h(\Upsilon(\mathbf{S})) \subseteq D$ ,
- there exists a homomorphism  $h_1$  that extends  $h$  s.t. the output of function  $checkCondInState(Q_1, net\_state)$  is “yes”, where  $Q_1 = (h_1(\Phi(\mathbf{T}))) : [\tau_1, \tau_2]$  and  $net\_state$  is the output of Algorithm 1 over the NetDiff knowledge base  $DiffKB = (G, \emptyset)$ , and
- the output of function  $checkCondInState(Q_2, net\_state)$  is “yes”, where  $Q_2 = (\Omega, h_1(\gamma_\epsilon(\mathbf{U}))) : [\tau_1, \tau_2]$  ( $net\_state$  and  $h_1$  are the same as the ones in the previous item).

Let  $\epsilon$  be applicable to  $(D, G)$  at time frame  $[\tau_1, \tau_2]$ ; then, when  $\epsilon$  is applied to  $(D, G)$  one of the following can occur: (i)  $h(X_i)$  and  $h(X_j)$  are constants and the chase fails, or (ii)  $h(X_i)$  is replaced with  $h(X_j)$  if  $X_i$  is a null and  $h(X_j)$  is a constant or  $h(X_i)$  is preceded by  $h(X_j)$  in the lexicographic order.

It is well known that undesired interactions between EGDs and TGDs must be handled with care because they can lead to undecidability state. Towards this end, the *separability* condition is one way to avoid such interactions [10].

NETDER NC Chase Rule. Let NETDER be knowledge base  $KB = (D, G, \Sigma, P)$ , a time frame  $[\tau_1, \tau_2] \subseteq [0, t_{max}]$ , and a NETDER NC  $\mu \in \Sigma$  of the form  $\Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow \perp : \gamma_\mu(\mathbf{U})$ . As usual, the chase must only check:

- there does not exist a homomorphism  $h$  s.t.  $h(\Upsilon(\mathbf{S})) \subseteq D$ ; or
- there does not exist a homomorphism  $h_1$  that extends  $h$  s.t. the output of function  $checkCondInState(Q_1, net\_state)$  is “yes”, where  $Q_1 = (h_1(\Phi(\mathbf{T}))) : [\tau_1, \tau_2]$  and  $net\_state$  is the output of Algorithm 1 over the NetDiff knowledge base  $DiffKB = (G, \emptyset)$ , and
- the output of function  $checkCondInState(Q_2, net\_state)$  is “yes”, where  $Q_2 = (\Omega, h_1(\gamma_\mu(\mathbf{U}))) : [\tau_1, \tau_2]$  ( $net\_state$  and  $h_1$  are the same as the ones in the previous item).

If one of these checks fails, then the chase fails; otherwise, the constraints can be ignored during the remaining execution of the chase.

**Example 3.4.** Consider the NETDER knowledge base  $KB = (D, G, \Sigma, P)$  where  $D, \Sigma, P$  are the same from Example 3.3 but  $G$  is from Example 3.1. Then, NETDER TGD  $\omega_3 \in \Sigma$  is applicable to  $(D, G)$  at time  $[0, t_{max}]$  because: there exists a homomorphism  $h_1$  that extends  $h$  (in this case,  $h$  is an empty mapping) such that the output of function  $checkCondInState(Q_1, net\_state)$  is “yes”, where:

- $Q_1 = (h_1((node(X), \langle expert(S), [0.9, 1.0] \rangle) \wedge \langle bel\_dang(T), [0.8, 1.0] \rangle))) : [0, t_{max}]$
- $h_1((node(X), \langle expert(S), [0.9, 1.0] \rangle) \wedge \langle bel\_dang(T), [0.8, 1.0] \rangle)) = (node(n_1), \langle expert(trojan), [0.9, 1.0] \rangle) \wedge \langle bel\_dang(gooligan), [0.8, 1.0] \rangle)$ .  
So, the homomorphism maps  $X$  to  $n_1$ ,  $S$  to  $trojan$ , and  $T$  to  $gooligan$ .
- $net\_state$  is the output of Algorithm 1 over the NetDiff knowledge base  $DiffKB = (G, \emptyset)$ , which can be seen in Figure 4b (only NetDiff facts are applied).

When  $\omega_3$  is applied to  $(D, G)$  at time frame  $[0, t_{max}]$ , then the following atoms with null values  $z_1, z_2$ , and  $z_3$  are added to  $D$  (here, nothing is added to  $G$ ):

$$\begin{aligned} h_2(product(U)) &= product(z_1), \\ h_2(user(UID, N)) &= user(z_2, z_3), \\ h_2(hyp\_at\_risk(U, UID)) &= hyp\_at\_risk(z_1, z_2). \end{aligned}$$

Although our ontological language is defined as an extension of existential rules, the network DB ( $G$ ) and the ontological part of the NETDER KB ( $D$  and  $\Sigma$ ), with the new kinds of rules, can be embedded in a classical Datalog+/- representation—see Appendix A for details of this mapping. ■

#### 4. NETDER Query Answering

In this section, we provide details for implementing the Query Answering Module (QAM, cf. Figure 1) and describe how the query answering process can be carried out as a collaboration between the Ontological Reasoning and Network Diffusion modules (ORM and NDM, respectively). These two modules operate as *synchronous* processes; the query answering procedure therefore needs to address how the *interactions* between them occur—as we will see below, this leads to a non-trivial analysis of the conditions under which such a process is guaranteed to terminate. First, we define the kind of queries that we will consider.

**Definition 4.1** (NETDER conjunctive query). *Let  $[\tau_1, \tau_2] \subseteq [0, t_{max}]$  be a time frame; a NETDER conjunctive query (CQ) is of the following form:*

$$Q(\mathbf{X}) = \exists \mathbf{Y} \rho(\mathbf{X}_1, \mathbf{Y}_1) \wedge \phi(\mathbf{X}_2, \mathbf{Y}_2) \wedge \gamma(\mathbf{X}_3, \mathbf{Y}_3) : [\tau_1, \tau_2]$$

where  $\mathbf{X}, \mathbf{Y}, \mathbf{X}_1, \mathbf{Y}_1, \mathbf{X}_2, \mathbf{Y}_2, \mathbf{X}_3, \mathbf{Y}_3 \subset \mathcal{V}$  are sequences of variables,  $\mathbf{X} = \mathbf{X}_1 \cup \mathbf{X}_2 \cup \mathbf{X}_3$ ,  $\mathbf{Y} = \mathbf{Y}_1 \cup \mathbf{Y}_2 \cup \mathbf{Y}_3$ ,  $\rho(\mathbf{X}_1, \mathbf{Y}_1)$  is a conjunction of atoms over a relational schema  $\mathcal{R}$  with arguments from a set of variables  $\mathcal{V}$  and constants  $\Delta$ ,  $\phi(\mathbf{X}_2, \mathbf{Y}_2)$  and  $\gamma(\mathbf{X}_3, \mathbf{Y}_3)$  are a conjunction of network component targets and a network global annotation, respectively, over a set of label predicate symbols  $\mathcal{P}$  with arguments from a set of variables  $\mathcal{V}$  and constants  $\Delta$ .

As usual, a NETDER query is said to be Boolean when  $\mathbf{X} = \{\}$ ; such a query can only have “Yes” or “No” as an answer. Clearly, there could be many different ways to compute answers to such queries; in this section, we see how three particular classes of answers of interest can be obtained, and focus on the problem of the feasibility of their computation. Such classes arise from how the interaction between the modules is handled.

In order to structure this interaction, we propose two main policies and a third that is a particular case of the others (cf. Figure 7). As we will see, these policies will help to define conditions under which the query answering process terminates. In each case, the chase and network diffusion processes are carried out in turn; the difference lies in how this interaction stops:

- *Unbounded-Int*: The process terminates when the query can be answered or a given condition is satisfied over the network and/or the ontology (for instance, a fixed point is reached). The number of interactions between the ORM and NDM are therefore not a priori bounded.
- *Bounded-Int*: The process terminates when the query can be answered or the number of iterations reaches a given number.
- *One-Shot*: The process terminates when the query can be answered or after performing a single interaction between the ORM and NDM. That is, if the chase cannot answer the query, then the diffusion process is executed and when it is completed the chase tries to answer the query one more time before completing the process. Note that this policy is a specific case of the *Bounded-Int* policy.

Algorithm 2 (Figure 8) describes this query answering process in high-level pseudocode. The first step is to execute the chase, then the query is evaluated and if it can already be answered or the termination conditions are satisfied, *done* is set to *true*. Otherwise, the diffusion process is carried out (cf. Algorithm 1) and a new NetDiff state *net\_state* is obtained, then  $G$  is updated from *net\_state*. The query is evaluated again and if flag *done* is still *false*, then the procedure iterates until *EvaluateQuery* sets *done* to *true* in accordance with the corresponding policy  $QAP$ . Note that *ExecuteChase* never fails because, as mentioned in Section 1, the DIM handles inconsistencies. Next, we define what constitutes an answer to a NETDER conjunctive query.

**Definition 4.2** (Answers to NETDER conjunctive query). *Given a NETDER knowledge base  $KB$ , NETDER conjunctive query  $Q$ , and  $QAP$  Policy, the set of answers to  $Q$  over  $KB$  using Policy, denoted  $ans(Q, Policy, KB)$ , is the set of all tuples that are output by Algorithm 2 with parameters  $Q$ , Policy, and  $KB$ . For the Boolean case, the answer to  $Q()$  is yes iff the set of answers is non-empty, and no otherwise.*

The following example illustrates the NETDER query answering process.

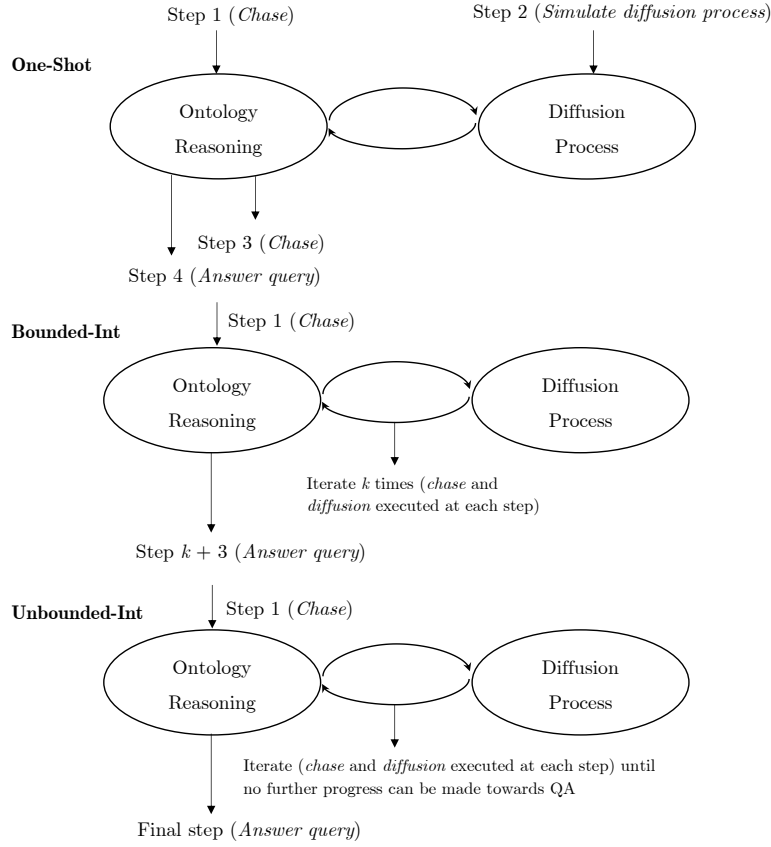


Fig. 7. Graphical representation of the three possible interactions considered between the ORM and NDM during the query answering process.

**Example 4.1.** Consider running Algorithm 2 over Boolean NETDER query:

$$Q() = \exists U, UID \left( \text{product}(U) \wedge \text{hyp\_at\_risk}(U, UID) \right) : [0, t_{max}],$$

*QAP Policy = One-Shot, and the knowledge base  $KB = (D, G, \Sigma, P)$  where  $D, G, P$  are the same from Example 3.4 and  $\Sigma = \{\omega_2\}$  from Example 3.3. The first execution of the chase (line 4 in Algorithm 2) does not add new knowledge to  $(D, G)$  because  $\omega_2$  cannot be applied. When the diffusion process is executed (line 8 in Algorithm 2), a final NetDiff state is obtained, which can be seen in part (c) of Figure 4. Then,  $G$  is updated with that final NetDiff state such that now  $G$  contains<sup>7</sup>:*

$$G = \{ \text{node}(n_1), \text{node}(n_2), \text{edge}(n_1, n_2) \} \cup \mathcal{L} \cup \{ t_{max}(2) \} \cup \\ \{ (\text{node}(n_1), \langle \text{expert}(\text{trojan}), [0.9, 1] \rangle) : [0, t_{max}], \\ (\text{node}(n_1), \langle \text{bel\_dang}(\text{gooligan}), [0.8, 1] \rangle) : [0, t_{max}], \\ (\text{node}(n_2), \langle \text{bel\_dang}(\text{gooligan}), [0.7, 1] \rangle) : [1, t_{max}], \\ (\text{edge}(n_1, n_2), \langle \text{close}(\text{trojan}, \text{windows}), [0.8, 1] \rangle) : [0, t_{max}], \\ (\Omega, \langle \text{dangerous}(\text{gooligan}), [0.8, 1] \rangle) : [0, t_{max}] \}$$

*Now, the chase is executed again with the new knowledge base (line 10 in Algorithm 2). In this case,  $\omega_2$  is applicable to  $(D, G)$  and the atoms  $h_2(\text{product}(U)) = \text{product}(z_1)$ ,  $h_2(\text{hyp\_at\_risk}(U, UID)) = \text{hyp\_at\_risk}(z_1, n_1)$  are added*

<sup>7</sup>Recall that NetDiff facts about labels with bounds  $[0, 1]$  can be omitted.

**Algorithm 2** NETDER Query Answering**Require:** Query  $Q$ , QAP Policy,  $KB = (D, G, \Sigma, P)$ 1:  $done := false$ 2:  $ans := \emptyset$ 3: **while**  $!done$  **do**4:  $(D, G) := ExecuteChase(Q, KB)$  [ORM]

// Computes the necessary finite part of the chase to answer the query

5:  $ans := ans \cup EvaluateQuery(Q, Policy, (D, G), done)$ 6: **if** not  $done$  **then**7:  $DiffKB := (G, P)$ 8:  $net\_state := ExecuteDiffProc(DiffKB)$  // Algorithm 1 [NDM]9:  $update(G, net\_state)$  [ORM]10:  $(D, G) := ExecuteChase(Q, KB)$  [ORM]11:  $ans := ans \cup EvaluateQuery(Q, Policy, (D, G), done)$ 12: **end if**13: **end while**14: **if**  $Q$  is Boolean **then**15: **return**  $ans \neq \emptyset$ 16: **else**17: **return**  $ans$ 18: **end if**

Fig. 8. The NETDER Query Answering Algorithm

to  $D$  and nothing is added to  $G$ . Finally, the interactions between ORM and NDM terminate because QAP is One-Shot, and the answer for the query  $Q$  is “yes”. ■

We now present our main result regarding computational cost, which states conditions under which the computational cost of Algorithm 2 is  $O(poly(|G|))$  or it preserves the cost of carrying out QA over the underlying ontology language (the proof can be found in Appendix B). For this purpose, in the appendix we show that given a NETDER knowledge base  $KB = (D, G, \Sigma, P)$  and a NETDER conjunctive query  $Q$ , it is possible to obtain a classical Datalog+/- knowledge base, denoted  $trans(KB) = (D_t, \Sigma_t)$ , from  $KB$  according to Definition A.1, and a classical conjunctive query, denoted  $trans(Q)$ , from  $Q$  according to Definition A.2, such that each answer obtained via the NETDER chase can also be obtained via the (oblivious) classical chase<sup>8</sup>. Note that this translation does not include the diffusion process; hence, general NETDER query answering cannot be solved completely by using  $trans(Q)$  and  $trans(KB)$ , but it could be used to carry out ontological NETDER query answering (ORM). Then, we could address NETDER query answering in the following way:

Step 1) Try to answer query  $trans(Q)$  using  $D_t^{(1)} = chase(trans(KB))$ ;

Step 2) Execute the diffusion process over NetDiff knowledge base  $(G_1, P)$  (where  $G_1$  is obtained from  $D_t^{(1)}$ ) to yield the new NetDiff knowledge base  $(G_2, P)$ ;

Step 3) Repeat Step 1 over  $trans((D_2, G_2, \Sigma, P))$  (where  $D_2$  is obtained from  $D_t^{(1)}$ ), and then Step 2 until those conditions are met according to the chosen QAP policy; and

Step 4) Answer query  $trans(Q)$ .

However, the purpose of this translation is not to provide a way to implement NETDER QA, but to use it as a resource to prove computational cost results.

<sup>8</sup>Proposition A.1 shows that this transformation process enjoys this completeness property; note that the converse, or soundness, is not required for the purposes of this result, which only seeks to establish upper bounds on computational cost.

Therefore, we can study the computational cost of Algorithm 2 based on the analysis of (classical) query answering in *Datalog+/-*. For this we need to consider three different aspects of a NETDER query answering instance: (i) the kind of QAP (bounded or unbounded); (ii) whether or not network modification is allowed (which simply refers to allowing NETDER rules or the environment to make changes to the structure or state of the network, such as adding a node or changing a bound); and (iii) whether or not there are guarantees that the diffusion process (Algorithm 1) will terminate, such as the ones provided in Theorem 3.1.

**Theorem 4.1.** *Let  $KB = (D, G, \Sigma, P)$  be a NETDER knowledge base,  $Q(\mathbf{X})$  be a conjunctive query,  $KB_t = \text{trans}(KB) = (D_t, \Sigma_t)$  be the transformation of  $KB$  into a classical *Datalog+/-* knowledge base, and  $Q_t(\mathbf{X}) = \text{trans}(Q(\mathbf{X}))$  be the transformed query (classical *Datalog+/-* query). If classical conjunctive query answering of  $Q_t$  over  $KB_t$  is decidable, and its associated computational data time complexity class has a worst case cost of  $O(f(|D_t|))$ , then the computational cost of Algorithm 2 is  $O(\text{poly}(|G|)) + O(f(|D_t|))$  under the following conditions:*

- *Bounded-Int policy with guaranteed termination of the diffusion process and ability to modify the network, and*
- *Unbounded-Int policy with guaranteed termination of the diffusion process and without ability to modify the network.*

Intuitively, this theorem identifies conditions under which the computational cost of Algorithm 2 can be bounded by the computational cost of classical *Datalog+/-* QA over the translated NETDER knowledge base and query—note that we focus on the case in which only the database is assumed to be non-fixed.

Thus, first we need to analyze if the underlying classical rules belong to a fragment of *Datalog+/-* for which conjunctive queries are decidable; this can be guaranteed via *syntactic conditions* such as guardedness [10] or frontier-guardedness [4], which are PTIME-complete in the data complexity (that is, worst case cost of  $O(\text{poly}(|D_t|))$ ), stickiness [11] and acyclicity, which are  $AC_0$  in the data complexity (i.e., worst case cost of  $O(\log(|D_t|))$ ), or their weak counterparts [9, 11, 18], which are also PTIME-complete in the data complexity. Other possibilities are *semantic conditions* such as finite expansion sets (fes), bounded treewidth sets (bts), and finite unification sets (fus) [4]—cf. the work by Simari et al. [56, Section 1.3.3] for a brief overview of many such conditions. A deeper analysis of constraints over NETDER rules that ensure that the translated rules fall into tractable fragments of classical rules is outside the scope of this work, but an important next step.

In the following, we show an example focusing on the translation of the NETDER knowledge base from Example 3.3 and a query based on this example (cf. Definition A.1 and Definition A.2, respectively). In addition, we analyze if QA over the translated knowledge base and query is decidable and, if so, the worst case computational cost of the associated complexity class.

**Example 4.2.** *Consider the NETDER knowledge base  $KB = (D, G', \Sigma, P)$  from Example 3.3; the transformed knowledge base  $KB_t = (D_t, \Sigma_t)$  is a *Datalog+/-* ontology instance with the following elements:*

$$\begin{aligned}
 D_t = \{ & \\
 & \text{trans}(d_1) : \text{user}(n_1, \text{ponzio}, 0, t_{\max}), \text{trans}(d_2) : \text{earlyPoster}(n_1, \text{gooligan}, 0, t_{\max}), \\
 & \text{trans}(d_3) : \text{product}(\text{windows-10}, 0, t_{\max}), \text{trans}(g_1) : \text{node}(n_1, 0, t_{\max}), \\
 & \text{trans}(g_2) : \text{node}(n_2, 0, t_{\max}), \text{trans}(g_4) : \text{node}(n_3, 0, t_{\max}), \\
 & \text{trans}(g_3) : \text{edge}(n_1, n_2, 0, t_{\max}), \text{trans}(g_5) : \text{edge}(n_2, n_3, 0, t_{\max}), \\
 & \text{trans}(l_1) : \text{expert}(\text{trojan}, 0, t_{\max}), \text{trans}(l_2) : \text{bel\_dang}(\text{gooligan}, 0, t_{\max}), \\
 & \text{trans}(l_3) : \text{close}(\text{trojan}, \text{windows-10}, 0, t_{\max}), \\
 & \text{trans}(l_4) : \text{dangerous}(\text{gooligan}, 0, t_{\max}), \\
 & \text{trans}(f_2) : \text{aux\_expert}(\text{trojan}, n_1, 0.9, 1, 0, t_{\max}), \\
 & \text{trans}(f_5) : \text{aux\_bel\_dang}(\text{gooligan}, n_1, 0.8, 1, 0, t_{\max}), \\
 & \text{trans}(f_6) : \text{aux\_close}(\text{trojan}, \text{windows-10}, n_1, n_2, 0.8, 1, 0, t_{\max}), \\
 & \text{trans}(g_6) : \text{bot}(0, t_{\max}), \text{trans}(g_7) : \text{botnet}(0, t_{\max}), \text{trans}(g_8) : \text{similar}(0, t_{\max}), \\
 & \text{trans}(g_9) : \text{aux\_bot}(n_1, 0.9, 1, 0, t_{\max}), \text{trans}(g_{10}) : \text{aux\_bot}(n_2, 0.6, 1, 0, t_{\max}), \\
 & \text{trans}(g_{11}) : \text{aux\_botnet}(n_2, n_3, 0.75, 1, 0, t_{\max}), \\
 & \text{trans}(g_{12}) : \text{aux\_similar}(n_1, n_2, 0.9, 1, 0, t_{\max}), \text{trans}(t_{\max}) : t_{\max}(2) \}
 \end{aligned}$$

$\Sigma_t$  contains the following transformed TGDs and EGDs:

$$\begin{aligned}
& (trans(\omega_1)) \underbrace{node(X, 0, 2) \wedge aux\_bot(X, L_1, U_1, T_1, T_2)}_{trans(\Phi)} \wedge \underbrace{gte(L_1, 0.8) \wedge gte(1, U_1)}_{trans(\Phi)} \rightarrow \\
& \quad \exists Y, Z \underbrace{node(Y, 0, 2) \wedge aux\_bot(Y, 0.7, 1, T_1, T_2)}_{trans(\Xi)} \wedge \\
& \quad \underbrace{edge(X, Y, 0, 2) \wedge aux\_botnet(X, Y, 0.7, 1, T_1, T_2)}_{trans(\Xi)} \wedge \\
& \quad \underbrace{hyp\_botnet(X, Z, 0, 2) \wedge hyp\_botnet(Y, Z, 0, 2)}_{trans(\Psi)} \\
& (trans(\omega_2)) \underbrace{aux\_dangerous(\Omega, Term, L_1, U_1, T_1, T_2) \wedge gte(L_1, 0.5) \wedge gte(1, U_1)}_{trans(\Upsilon)} \wedge \\
& \quad \underbrace{earlyPoster(UID, Term, 0, 2)}_{trans(\Upsilon)} \rightarrow \exists U \underbrace{product(U, 0, 2)}_{trans(\Psi)} \wedge \underbrace{hyp\_at\_risk(U, UID, 0, 2)}_{trans(\Psi)} \\
& (trans(\omega_3)) \underbrace{node(X, 0, 2) \wedge aux\_expert(X, S, L_1, U_1, T_1, T_2)}_{trans(\Phi)} \wedge \\
& \quad \underbrace{gte(L_1, 0.9) \wedge gte(1, U_1) \wedge aux\_bel\_dang(X, Term, L_2, U_2, T_1, T_2)}_{trans(\Phi)} \\
& \quad \wedge \underbrace{gte(L_2, 0.8) \wedge gte(1, U_2)}_{trans(\Phi)} \rightarrow \exists U, UID, N \underbrace{product(U, 0, 2)}_{trans(\Psi)} \wedge \underbrace{user(UID, N, 0, 2)}_{trans(\Psi)} \wedge \underbrace{hyp\_at\_risk(U, UID, 0, 2)}_{trans(\Psi)} \\
& (trans(\omega_4)) \underbrace{earlyPoster(UID, Term, 0, 2)}_{trans(\Upsilon)} \rightarrow \exists N \underbrace{user(UID, N, 0, 2)}_{trans(\Psi)} \\
& (trans(\epsilon_1)) \underbrace{hyp\_botnet(X_1, Z_1, 0, 2) \wedge hyp\_botnet(X_2, Z_1, 0, 2) \wedge edge(X_1, X_3, 0, 2) \wedge aux\_similar(X_1, X_3, L_1, U_1, T_1, T_2)}_{trans(\Phi)} \wedge \\
& \quad \underbrace{gte(L_1, 0.9) \wedge gte(1, U_1)}_{trans(\Upsilon)} \rightarrow X_2 = X_3 \\
& (trans(\epsilon_2)) \underbrace{hyp\_botnet(X_1, Z_1, 0, 2) \wedge hyp\_botnet(X_2, Z_2, 0, 2) \wedge edge(X_1, X_2, 0, 2) \wedge aux\_botnet(X_1, X_2, L_1, U_1, T_1, T_2)}_{trans(\Phi)} \wedge \\
& \quad \underbrace{gte(L_1, 0.9) \wedge gte(1, U_1)}_{trans(\Upsilon)} \rightarrow Z_1 = Z_2
\end{aligned}$$

Also,  $\Sigma_t$  contains further TGDs to ensure the default interval for each label:

$$\begin{aligned}
& tgd_1 : node(X, 0, t_{max}) \wedge expert(trojan, 0, t_{max}) \rightarrow aux\_expert(X, trojan, 0, 1, 0, t_{max}) \\
& tgd_2 : node(X, 0, t_{max}) \wedge bel\_dang(gooligan, 0, t_{max}) \rightarrow aux\_bel\_dang(X, gooligan, 0, 1, 0, t_{max}) \\
& tgd_3 : node(X, 0, t_{max}) \wedge bot(0, t_{max}) \rightarrow aux\_bot(X, 0, 1, 0, t_{max}) \\
& tgd_4 : edge(X, Y, 0, t_{max}) \wedge close(trojan, windows-10, 0, t_{max}) \rightarrow \\
& \quad aux\_close(X, Y, trojan, windows-10, 0, 1, 0, t_{max}) \\
& tgd_5 : edge(X, Y, 0, t_{max}) \wedge botnet(0, t_{max}) \rightarrow aux\_botnet(X, Y, 0, 1, 0, t_{max}) \\
& tgd_6 : edge(X, Y, 0, t_{max}) \wedge similar(0, t_{max}) \rightarrow aux\_similar(X, Y, 0, 1, 0, t_{max}) \\
& tgd_7 : dangerous(gooligan, 0, t_{max}) \rightarrow aux\_dangerous(\Omega, gooligan, 0, 1, 0, t_{max})
\end{aligned}$$

For NETDER knowledge base KB, we could pose the NETDER query:

$$\begin{aligned}
Q_1(UID) &= \exists B hyp\_botnet(UID, B) : [t_{max}, t_{max}] \\
&= \exists B hyp\_botnet(UID, B) : [2, 2]
\end{aligned}$$

which can be translated as:

$$\text{trans}(Q_1(\text{UID})) = \exists B \text{ hyp\_botnet}(\text{UID}, B, T_1, T_2) \wedge \text{gte}(2, T_1) \wedge \text{gte}(T_2, 2)$$

Note that we use  $\text{trans}(\text{elem})$  to denote the translated NETDER element  $\text{elem}$ ,  $\text{trans}(\Upsilon)$  and  $\text{trans}(\Psi)$  to denote the conjunction of translated ontological atoms,  $\text{trans}(\Phi)$  and  $\text{trans}(\Xi)$  to denote the conjunction of translated network component targets, and  $\text{trans}(\gamma)$  to denote the translated network global annotation, which are involved in each rule. Furthermore, NETDER elements with references to  $t_{\max}$  value were translated with the reference replaced by the appropriate value 2.

In accordance with Theorem 4.1, in order to assess the computational cost of Algorithm 2 for the task of answering NETDER query  $Q_1(\text{UID})$  over  $KB$ , first we must analyze the cost of answering the translated query  $\text{trans}(Q_1(\text{UID}))$  over  $\text{trans}(KB)$ . With this in mind, we examine each translated TGD in  $\Sigma_t$  to determine if they belong to some tractable Datalog+/- fragment, and we reach the following conclusions:

- $\text{trans}(\omega_1)$  is guarded;
- $\text{trans}(\omega_2)$  is frontier-one (only  $UID$  is in the frontier);
- $\text{trans}(\omega_3)$  is disconnected (it has an empty frontier);
- $\text{trans}(\omega_4)$  is linear;
- $\text{tgd}_1$ – $\text{tgd}_7$  are guarded.

So, the set of TGDs taken as a whole is frontier-guarded (which is a direct consequence of containment between classes). Additionally, the EGDs ( $\text{trans}(\epsilon_1)$ ) and ( $\text{trans}(\epsilon_2)$ ) do not increase the data complexity of answering  $\text{trans}(Q_1(\text{UID}))$  over  $\text{trans}(KB)$  when they are added to translated TGDs in  $\Sigma_t$  because the only atom in ( $\text{trans}(\epsilon_1)$ ) and ( $\text{trans}(\epsilon_2)$ ) with variables in positions that may contain fresh nulls generated by the chase is  $\text{hyp\_botnet}/4$ , which is not used as part of the body in any TGD. Hence, we conclude that answering the query  $\text{trans}(Q_1(\text{UID}))$  over  $\text{trans}(KB)$  is decidable in PTIME in the data complexity [4], and therefore the worst case computational cost is  $O(\text{poly}(|D_t|))$ . However, this is only the computational cost of the ontological query answering, and the computational cost of Algorithm 2 requires analyzing the chosen policy and the diffusion process conditions. ■

The following observation refers to the termination of Algorithm 2.

**Observation 4.1.** Algorithm 2 may not terminate for the following reasons:

- (a) QA over the transformed query and knowledge base is undecidable,
- (b) there are no guarantees that a fixed point will be reached when computing the result of the diffusion process (see Theorem 3.1).

It is clear that reason (a) arises from the fact that any procedure to solve an undecidable problem may never terminate. Likewise, reason (b) comes from the fact that it is possible to define influence functions that return intervals that are a proportion of the input values; so, for instance, if the function returns the bounds multiplied by 0.5, a fixed point will never be reached (cf. Observation 3.1).

Two more observations can be made. First, when Algorithm 2 under *One-Shot* or *Bounded-Int* policy and with diffusion termination guarantees terminates, it has computational cost  $O(\text{poly}(|G|)) + O(f(|D_t|))$  because we assume that the diffusion process takes polynomial time in the size of the NetDiff database (since the whole network needs to be scanned at least once). In general, function “ $f(|D_t|)$ ” depends on the characteristics of the underlying ontology language, as mentioned above.

Second, Algorithm 2 under the *Unbounded-Int* policy with diffusion termination guarantees and without ability to modify the network, is a particular case of the *One-Shot* policy under the same conditions, and hence the results are the same. This is due to the fact that the network cannot be modified, and therefore executing the diffusion process  $k$  times is the same as executing it only once, as is done under the *One-Shot* policy.

A final result that we show seeks to analyze the expressive power of NETDER query answering, even under what appear to be safe assumptions:

**Theorem 4.2.** NETDER query answering (Algorithm 2) over knowledge bases  $KB = (D, G, \Sigma, P)$ , such that ontological query answering over  $\langle D, \Sigma \rangle$  is decidable and diffusion process  $P$  is guaranteed to terminate, is Turing-complete.

Intuitively, a deterministic universal TM can be simulated using NetDiff as the diffusion modeling language, ontological rules that have the ability to modify the network, and the *Unbounded-int* query answering policy—the full proof of this result is included in Appendix B. In future work, we plan to address the definition of a model semantics, which will provide the basis for an analysis of the computational complexity of associated problems. This will also directly yield an undecidability result by establishing a reduction from the halting problem, using the same approach as the one taken in the proof of Theorem 4.2.

With the full details for the main NETDER modules in place, we refer the interested reader to Appendix C, where we develop a use case to illustrate the operation of the framework in a real-world cybersecurity application domain.

## 5. Related Work

We now discuss work related to our framework which, apart from classical approaches to logic-based abduction and automated generation of hypotheses, is most closely related to extensions of ontology languages and modeling of diffusion processes. Though the application we focus on in this paper is related to cybersecurity, the automated generation and evaluation of hypotheses finds applications in many other domains. Two examples of other applications that could clearly benefit from a knowledge-based human-in-the-loop approach are automated health care and detection of corruption in government agencies. We refer the reader to Davoudi et al. [14], Velasco et al. [46], and Siciliani et al. [55] for recent works in these areas that, although take different approaches, clearly show the potential to apply a framework like ours. There is not much research that recognizes and develops solutions around the concept of automated hypothesis generation; one of the earliest approaches is by Morgan [34]. Other interesting works that are similar in spirit but adopt very different approaches are those by Elsaesser et al. [17] and Sybrandt et al. [57].

The rest of this section is therefore organized into three parts: logic-based abduction and automated generation of hypotheses, extensions to ontology languages, and network diffusion processes and cascades.

*Logic-based Abduction and Automated Generation of Hypotheses* Abductive reasoning, the process of finding explanations for observations, was first introduced in the 19th century by philosopher Charles S. Peirce [8] as a form of reasoning distinct from deduction and induction. Abductive inferences are best interpreted as hypotheses that can later be confirmed or discarded. In artificial intelligence, abductive reasoning naturally finds a place as a tool that complements deductive and inductive approaches in a vast array of applications, such as diagnosis [13, 42], non-monotonic reasoning [15], planning [45, 54], and logic programming [16, 29]. Also related to our approach incorporating uncertainty is work developing combinations of probabilistic and logic-based reasoning—some well-known examples of such work are David Poole’s Theorist system [44] and the Independent Choice Logic [43], probabilistic logic [37], and probabilistic logic programming [35]. Though these and other classical works have laid the foundation for our model, there are significant differences at the core: value invention is not considered (which is a fundamental aspect of reasoning about unknown objects), handling of uncertainty is typically constrained by specific models or assumptions such as pairwise independence, and diffusion in networks is not a built-in feature.

*Extensions to Ontology Languages* Our work builds on a large body of work on extensions to ontological languages, such as those based on description logics (DLs) [2, 3] and existential rules (also known as Datalog+/-) [10] which are two well-known families of knowledge representation formalisms. The work of Gottlob et al. [23] is one of the closest to our research line, in which the authors extend existential rules using probabilistic annotations in order to progress towards a principled way to handle query answering tasks under uncertainty and inconsistency. Similarly to our work, they modify the chase procedure to consider the propagation of probabilistic events through the atoms and rules of the ontology. Additionally, this extension also models ontological and uncertain knowledge by means of two separate worlds and their interactions. However, although probabilistic events and network global and local annotations are similar in that they allow us to handle uncertainty, the semantics and overall goal is dif-

ferent in that our proposal is especially designed for domains where information is propagated between entities that interact with each other. In this same direction, we can also mention the work of Lukasiewicz [30], where the author presents an extension of  $\mathcal{SHLF}(\mathbf{D})$  and  $\mathcal{SHOLN}(\mathbf{D})$ , two expressive DLs that are related to sublanguages of the Web Ontology Language (OWL), known as OWL Lite and OWL DL, respectively.  $\mathcal{P}\text{-}\mathcal{SHLF}(\mathbf{D})$  and  $\mathcal{P}\text{-}\mathcal{SHOLN}(\mathbf{D})$  are the languages resulting from these extensions, and they allow us to represent certain and uncertain knowledge, either in the form of assertional and/or terminological statements about instances of concepts and roles. Both approaches use probability distributions over interpretations obtained from the available knowledge to carry out probabilistic query answering tasks. Another closely related work, by Gutiérrez-Basulto et al. [26], also defines probabilistic DLs by means of extensions of the sublanguages  $\mathcal{ALC}$  and  $\mathcal{EL}$ . Their proposal differs from the other works mentioned previously in that their formalism not only allows us to represent probabilities regarding TBox statements and ABox assertions, but also about concept and role names; an additional difference is that its semantics is monotonic in nature.

Also related to this research line is an early proposal in which we made preliminary efforts to combine logic-based formalisms and probabilistic knowledge to address problems such as adversarial deduplication [38]. This is a slightly different version of the problem known as adversarial/entity resolution in classical databases, but in this case we assume that there are actors controlling multiple accounts who seek to avoid being identified as the same entity. We proposed to generate hypotheses about duplicate actors based on the detection of suspicious behavior and, as in this work, the value invention capability in order to complete them when new information becomes available. Note that this problem can be seen as a particular kind of malicious behavior; however, that formalism does not have the capability to model diffusion processes.

*Network Diffusion Processes and Cascades* The NetDiff language is based on the MANCaLog formalism [52, 53], originally presented as a standalone language for modeling cascading diffusion processes. The proposal was based on a set of design criteria that are closely related to the kind of applications that we envision for NETDER: (i) *Multiply labeled and weighted nodes and edges*: allows us to represent different properties of nodes and edges; (ii) *Explicit representation of time*: allows us to represent more expressive models with temporal relations between conditions in the network structure, the current state of the diffusion process, and propagation of influence; (iii) *Non-Markovian temporal relationships*: “memorylessness” is not enough to represent temporal relations over multiple time units; (iv) *Representation of uncertainty*: many real-world settings cannot be represented without this capability; (v) *Competing processes*: many practical problems have competing processes by nature; (vi) *Non-monotonic Processes*: allows us to represent situations where different properties of nodes that are satisfied at a given time may not be satisfied later; and (vii) *Tractability*: practical feasibility is a very important feature when non-trivial amounts of information need to be handled. These criteria were chosen based on a thorough analysis of related literature; though there exist formalisms with different subsets of these features, none enjoy all of them at once. The resulting language is therefore capable of representing multiple non-trivial diffusion processes instead of single simple ones, which is the most typical case handled in the literature. This language is also the basis for the PyReason software<sup>9</sup> [1], which is currently being developed as a toolkit to support efforts in the area of neuro-symbolic reasoning and learning [50, 51].

Other proposals to tackle logic-based reasoning problems over diffusion processes on networks include those by Christoff et al. [12] and Hansen [27], which are founded on extensions of modal logic and fuzzy logic, respectively. However, they do not have the same representation capability as our proposal for diffusion process. In the former, the authors extend a kind of modal logic that provides the capability to reason about agents in the network and define different network structural properties. As a result, the dynamics of diffusion processes in the network is modeled using modal formulas. Likewise, the latter proposes a blend of modal and fuzzy logic for the purpose of focusing on opinion dynamics and answering questions about *truth-tracking*, *opinion leaders*, *convergence and consensus*, and *speed of convergence*. However, even though the problems they address are related to our approach, significant differences include the fact that their model cannot represent rules that modify properties based on neighbors’ criteria, and therefore cascades cannot be represented; the approach cannot represent uncertainty (as we do with

<sup>9</sup><https://pyreason.syracuse.edu/>

weighted labels); and the network evolution is carried out in such a way that the available knowledge at each time step can only directly affect that of the next time step.

Finally, there are other works that are related to ours in different ways but do not quite fit in any of the above categories. First, the BLOG model [33] focuses on defining probabilistic models over worlds with unknown objects. Though this capability is in line with that of value invention in a general sense, its instantiation in BLOG is geared towards defining first-order structures that contain varying and unbounded *number* of objects, such as an urn containing an unknown number of balls. Next, NETDER can be seen as a kind of multi-context system [6, 7, 22], since these systems allow us to combine different first-order logical languages, structuring them using a hierarchy. Lower levels provide symbols for higher levels, and the knowledge is exchanged between them via bridge rules, which consist of a body to represent knowledge that holds or not in other contexts and they enable us to add knowledge in the context described in the head when the body is satisfied. An interesting avenue for future work, as already mentioned above, is to develop a model semantics for NETDER, and multi-context systems may provide a vehicle to achieve this. Lastly, another related area that we would like to investigate is that of knowledge dynamics, that is considering the formalization of how the proposed framework may change over time as knowledge in the domain naturally changes. In order to do this, existing work in the literature on knowledge dynamics, such as [19, 20], becomes highly relevant. In those works, a formalism is presented to deal with knowledge dynamics equipped with argumentative inference, in which information is organized in layers by means of stratified belief bases. It is important to note that the complexity of the NETDER architecture and the evolving nature of the applications that we envision for it, would require to address several challenges not considered in more classical settings. For instance, in [21] we have recently initiated a line of work related to this, with a preliminary study of a multi-agent system designed to address cybersecurity issues in social platforms that makes use of the HEIST (*Hybrid Explainable and Interpretable Socio-Technical systems*) application framework [58], which is based on the NETDER architecture studied here.

## 6. Conclusions

In this work, we have continued a line of research started with our recent proposal of the NETDER architecture for the automated generation and evaluation of hypotheses for decision support in human-in-the-loop systems. The initial work was focused on high level system design that included the definition of the four main modules in NETDER: Data Ingestion, Ontological Reasoning, Network Diffusion, and Query Answering; in this paper, we focused on developing the full details of an extended ontology language, used in ORM, with the ability to query conditions over networks. Then, we used the NetDiff language (a slight extension of MANCaLog formalism) for developing the NDM in order to make available all the necessary machinery for the operation of our model. We investigated a procedure for the NETDER query answering task, and arrived at an interesting set of results regarding its computational cost, ranging from polynomial time tractability to the possibility of non-guaranteed termination, depending on the features that are made available. Finally, we developed a use case to illustrate how the approach can be applied in a cybersecurity domain to reason about vulnerabilities based on darkweb forum posts.

Future work in this direction involves full implementation and more extensive evaluations with real and synthetic data. Another important next step is to define a model semantics and study the complexity of fragments of NETDER rules and queries that can deliver tractability guarantees for the the query answering process.

**Acknowledgments.** This work was funded in Argentina in part by the following institutions: Universidad Nacional del Sur (UNS) under grant PGI 24/N057, CONICET under grant PIP 11220210100577CO, and ANPCyT under grant PICT-2018-0475 (PRH-2014-0007). M. Vanina Martinez also acknowledges support by the Spanish MCIN/AEI (CHIST-ERA iTrust) project PCI2022-135010-2, project PID2022-139835NB-C21, and PIE 2023-5AT010 CSIC. Paulo Shakarian is supported by ARO grant W911NF-24-1-0007.

## References

- [1] Aditya, D., Mukherji, K., Balasubramanian, S., Chaudhary, A., and Shakarian, P. (2023). PyReason: Software for open world temporal logic. In *AAAI Spring Symposium (Mar. 2023)*.
- [2] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.
- [3] Baader, F., Horrocks, I., Lutz, C., and Sattler, U. (2017). *An Introduction to Description Logic*. Cambridge University Press.
- [4] Baget, J., Mugnier, M., Rudolph, S., and Thomazo, M. (2011). Walking the complexity lines for generalized guarded existential rules. In *Proc. IJCAI*, pages 712–717.
- [5] Beeri, C. and Vardi, M. Y. (1981). The implication problem for data dependencies. In *Automata, Languages and Programming, 8th Colloquium, Acre (Akko), Israel, July 13-17, 1981, Proceedings*, pages 73–85.
- [6] Brewka, G. and Eiter, T. (2007). Equilibria in heterogeneous nonmonotonic multi-context systems. In *AAAI*, volume 7, pages 385–390.
- [7] Brewka, G., Roelofsens, F., and Serafini, L. (2007). Contextual default reasoning. In *Proc. IJCAI*, pages 268–273.
- [8] Buchler, J., editor (1940). *The Philosophy of Peirce: Selected Writings*. Harcourt, New York, NY.
- [9] Cali, A., Gottlob, G., and Kifer, M. (2013). Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174.
- [10] Cali, A., Gottlob, G., and Lukasiewicz, T. (2012a). A general Datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics*, 14:57–83.
- [11] Cali, A., Gottlob, G., and Pieris, A. (2012b). Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128.
- [12] Christoff, Z. and Hansen, J. U. (2015). A logic for diffusion in social networks. *J. Applied Logic*, 13(1):48–77.
- [13] Console, L. and Torasso, P. (1991). A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7(3):133–141.
- [14] Davoudi, A., Malhotra, K. R., Shickel, B., Siegel, S., Williams, S., Ruppert, M., Bihorac, E., Ozrazgat-Baslanti, T., Tighe, P. J., Bihorac, A., et al. (2019). Intelligent ICU for autonomous patient monitoring using pervasive sensing and deep learning. *Scientific reports*, 9(1):1–13.
- [15] Eiter, T. and Gottlob, G. (1995). The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42.
- [16] Eiter, T., Gottlob, G., and Leone, N. (1997). Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science*, 189(1-2):129–177.
- [17] Elsaesser, C. and Tanner, M. C. (2001). Automated diagnosis for computer forensics. *The Mitre Corporation*, pages 1–16.
- [18] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. (2005). Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124.
- [19] Falappa, M. A., García, A. J., and Simari, G. R. (2016). A set of operations for stratified belief bases. In Beierle, C., Brewka, G., and Thimm, M., editors, *Computational Models of Rationality, Essays dedicated to Gabriele Kern-Isberner on the occasion of her 60th birthday*, pages 223–242. College Publications.
- [20] Falappa, M. A., García, A. J., and Simari, G. R. (2023). Merging operators on stratified belief bases equipped with argumentative inference. *Journal of Applied Non Classical Logics*, 33(3-4):387–420.
- [21] Garcia, A. C., Martinez, M. V., Deagustini, C. A. D., and Simari, G. I. (2023). A multi-agent system for addressing cybersecurity issues in social networks. In *Proc. ENIGMA@KR*, volume 3495, pages 43–54. CEUR-WS.org.
- [22] Giunchiglia, F. and Serafini, L. (1994). Multilanguage hierarchical logics or: How we can do without modal logics. *Artif. Intell.*, 65(1):29–70.
- [23] Gottlob, G., Lukasiewicz, T., Martinez, M. V., and Simari, G. I. (2013). Query answering under probabilistic uncertainty in Datalog+/- ontologies. *Ann. Math. Artif. Intell.*, 69(1):37–72.
- [24] Gottlob, G., Orsi, G., Pieris, A., and Simkus, M. (2012). Datalog and its extensions for semantic web databases. In *Proc. Reasoning Web – Semantic Technologies for Advanced Query Answering*, pages 54–77.
- [25] Granovetter, M. (1978). Threshold models of collective behavior. *American journal of sociology*, 83(6):1420–1443.
- [26] Gutiérrez-Basulto, V., Jung, J. C., Lutz, C., and Schröder, L. (2017). Probabilistic description logics for subjective uncertainty. *J. Artif. Intell. Res.*, 58:1–66.
- [27] Hansen, J. U. (2019). Reasoning about opinion dynamics in social networks. *J. Log. Comput.*, 29(7):1121–1137.
- [28] Iqbal, M., Kormiltsyn, A., Dwivedi, V., and Matulevičius, R. (2024). Blockchain-based ontology driven reference framework for security risk management. *Data & Knowledge Engineering*, 149:102257.
- [29] Kakas, A., Michael, A., and Mourlas, C. (2000). ACLP: Abductive constraint logic programming. *The Journal of Logic Programming*, 44:129–177(49).
- [30] Lukasiewicz, T. (2008). Expressive probabilistic description logics. *Artif. Intell.*, 172(6-7):852–883.
- [31] Maier, D., Mendelzon, A. O., and Sagiv, Y. (1979). Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469.
- [32] Mastroeni, L., Vellucci, P., and Naldi, M. (2019). Agent-based models for opinion formation: A bibliographic survey. *IEEE Access*, 7:58836–58848.
- [33] Milch, B., Marthi, B., Russell, S. J., Sontag, D. A., Ong, D. L., and Kolobov, A. (2005). BLOG: Probabilistic models with unknown objects. In *Proc. IJCAI*, pages 1352–1359.
- [34] Morgan, C. O. (1975). Automated hypothesis generation using extended inductive resolution. In *Proc. of IJCAI*, pages 351–356.
- [35] Ng, R. T. and Subrahmanian, V. S. (1992). Probabilistic logic programming. *Information and Computation*, 101(2):150–201.

- [36] Nguyen, L. A., Nguyen, T.-B.-L., and Szalas, A. (2015). Towards richer rule languages with polynomial data complexity for the semantic web. *Data & Knowledge Engineering*, 96-97:57–77.
- [37] Nilsson, N. (1986). Probabilistic logic. *Artificial Intelligence*, 28:71–87.
- [38] Paredes, J., Martinez, M. V., Simari, G. I., and Falappa, M. A. (2018). Leveraging probabilistic existential rules for adversarial deduplication. In *Proc. PRUV@IJCAR*.
- [39] Paredes, J., Teze, J. C., Martinez, M. V., and Simari, G. I. (2022). The HEIC application framework for implementing XAI-based socio-technical systems. *Online Social Networks and Media*, 32:100239.
- [40] Paredes, J. N., Simari, G. I., Martinez, M. V., and Falappa, M. A. (2021a). Detecting malicious behavior in social platforms via hybrid knowledge- and data-driven systems. *Future Gener. Comput. Syst.*, 125:232–246.
- [41] Paredes, J. N., Simari, G. I., Martinez, M. V., and Falappa, M. A. (2021b). NetDER: An architecture for reasoning about malicious behavior. *Inf. Syst. Frontiers*, 23(1):185–201.
- [42] Poole, D. (1989). Explanation and prediction: An architecture for default and abductive reasoning. *Computational Intelligence*, 5(2):97–110.
- [43] Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56.
- [44] Poole, D., Goebel, R., and Aleliunas, R. (1987). Theorist: A logical reasoning system for defaults and diagnosis. In Cercone, N. and McCalla, G., editors, *The Knowledge Frontier*, pages 331–352. Springer, New York.
- [45] Riabov, A. V., Sohrobi, S., Sow, D., Turaga, D., Udrea, O., and Vu, L. (2015). Planning-based reasoning for automated large-scale data analysis. In *Proc. ICAPS*.
- [46] Rotberg, R. I., editor (2019). *Identifying Corruption Risk in Brazil: New Measures for Effective Oversight*, pages 57–91. Springer International Publishing, Cham.
- [47] Sapienza, A., Bessi, A., Damodaran, S., Shakarian, P., Lerman, K., and Ferrara, E. (2018). Early warnings of cyber threats in online discussions. *CoRR*, abs/1801.09781.
- [48] Sarkar, S., Almukaynizi, M., Shakarian, J., and Shakarian, P. (2018). Predicting enterprise cyber incidents using social network analysis on the darkweb hacker forums. *CoRR*, abs/1811.06537.
- [49] Schelling, T. C. (1978). Micromotives and macrobehavior. *WW Norton and Company, New York*.
- [50] Shakarian, P., Baral, C., Simari, G. I., Xi, B., and Pokala, L. (2023). *Neuro Symbolic Reasoning and Learning*. Springer Briefs in Computer Science. Springer.
- [51] Shakarian, P., Simari, G. I., and Bastian, N. D. (2025). Probabilistic foundations for metacognition via hybrid AI. In Petrick, R. P. A. and Geib, C. W., editors, *Proceedings of the 2025 AAAI Spring Symposium Series, San Francisco, CA, USA, March 31-April 2, 2025*, pages 389–393. AAAI Press.
- [52] Shakarian, P., Simari, G. I., and Callahan, D. (2013a). Reasoning about complex networks: A logic programming approach. *Theory Pract. Log. Program.*, 13(4-5-Online-Supplement).
- [53] Shakarian, P., Simari, G. I., and Schroeder, R. (2013b). MANCaLog: A logic for multi-attribute network cascades. In *Proc. AAMAS*, pages 1175–1176.
- [54] Shanahan, M. (2000). An abductive event calculus planner. *Journal of Logic Programming*, 44:207–239.
- [55] Siciliani, L., Taccardi, V., Basile, P., Di Ciano, M., and Lops, P. (2023). AI-based decision support system for public procurement. *Information Systems*, 119:102284.
- [56] Simari, G. I., Molinaro, C., Martinez, M. V., Lukasiewicz, T., and Predoiu, L. (2017). *Ontology-Based Data Access Leveraging Subjective Reports*. Springer Briefs in Computer Science. Springer.
- [57] Sybrandt, J., Carrabba, A., Herzog, A., and Safro, I. (2018). Are abstracts enough for hypothesis generation? In *Proc. IEEE (Big Data)*, pages 1504–1513. IEEE.
- [58] Teze, J. C., Paredes, J., Martinez, M. V., and Simari, G. I. (2024). Engineering user-centered explanations to query answers in ontology-driven socio-technical systems. *Semantic Web*, 15(4):991–1020.

## Appendix A. Embedding the NETDER Ontological Language into Classical Existential Rules

We first show how the NETDER ontological language can be mapped into classical existential rules (this family of languages is also known as Datalog+/-, so we will sometimes refer to it by that name). The following definition presents the details of this embedding.

**Definition A.1** (Transformed knowledge base). *Let  $KB = (D, G, \Sigma, P)$  be a NETDER knowledge base, then the transformed knowledge base  $\text{trans}(KB) = (D_t, \Sigma_t)$ , with  $D_t = \text{trans}(D)$  and  $\Sigma_t = \text{trans}(\Sigma)$ , is a Datalog+/- ontology (also denoted with  $KB_t$ ) obtained by applying the following rules:*

- a)  $D_t$  contains the atom  $\text{tmax}(c)$  ( $c$  is a constant) from  $G$ .
- b)  $D_t$  contains every atom concerning the labels from  $G$ .
- c)  $D_t$  contains every transformed ontological atom from  $D$  and every transformed atom concerning the nodes, edges from  $G$  according to the mapping presented in Figure 9—item 1.
- d)  $D_t$  contains every corresponding transformed NetDiff fact from  $G$ . NetDiff facts (for node local labels, edge local labels, and global labels) and network component targets are transformed according to the mapping in Figure 9—items 2.x) and 3.x), respectively. In these and the following analogous cases, the new predicates “aux\_X” introduced in the transformations simply encode the same relation as “X” and incorporate the uncertainty bounds and the time steps as additional positions.
- e) For each NETDER TGD  $\sigma \in \Sigma$  of the form:

$$\Upsilon(\mathbf{X}) \wedge \Phi(\mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{Q}, \mathbf{R}) \wedge \Xi(\mathbf{S}, \mathbf{T}) : \gamma(\mathbf{U})$$

$\Sigma_t$  contains a TGD  $\sigma_t$  of the form:

$$\gamma_t(\mathbf{U}, T_1, T_2) \wedge \Upsilon_t(\mathbf{X}) \wedge \Phi_t(\mathbf{Y}, T_1, T_2) \rightarrow \exists \mathbf{Z} \Psi_t(\mathbf{Q}, \mathbf{R}) \wedge \Xi_t(\mathbf{S}, \mathbf{T}, T_1, T_2)$$

where  $\Upsilon_t(\mathbf{X}) = \text{trans}(\Upsilon(\mathbf{X}))$  and  $\Psi_t(\mathbf{Q}, \mathbf{R}) = \text{trans}(\Psi(\mathbf{Q}, \mathbf{R}))$  are the conjunctions of transformed ontological atoms of  $\Upsilon(\mathbf{X})$  and  $\Psi(\mathbf{Q}, \mathbf{R})$ , respectively—based on item 1 in Figure 9;  $\gamma_t(\mathbf{U}, T_1, T_2) = \text{trans}(\gamma(\mathbf{U}))$  is the transformed network global annotation of  $\gamma(\mathbf{U})$ —based on item 4 in Figure 9; and  $\Phi_t(\mathbf{Y}, T_1, T_2) = \text{trans}(\Phi(\mathbf{Y}))$  and  $\Xi_t(\mathbf{S}, \mathbf{T}, T_1, T_2) = \text{trans}(\Xi(\mathbf{S}, \mathbf{T}))$  are the conjunctions of transformed network component targets of  $\Phi(\mathbf{Y})$  and  $\Xi(\mathbf{S}, \mathbf{T})$ , respectively—based on items 3.x in Figure 9.

- f) For each NETDER EGD  $\epsilon \in \Sigma$  of the form:

$$\Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow (X_i = X_j) : \gamma(\mathbf{U})$$

$\Sigma_t$  contains an EGD  $\epsilon_t$  of the form:

$$\gamma_t(\mathbf{U}, T_1, T_2) \wedge \Upsilon_t(\mathbf{S}) \wedge \Phi_t(\mathbf{T}, T_1, T_2) \rightarrow (X_i = X_j)$$

where  $\Upsilon_t(\mathbf{S}) = \text{trans}(\Upsilon(\mathbf{S}))$  is the conjunction of transformed ontological atoms of  $\Upsilon(\mathbf{S})$ —based on item 1 in Figure 9;  $\gamma_t(\mathbf{U}, T_1, T_2) = \text{trans}(\gamma(\mathbf{U}))$  is the transformed network global annotation of  $\gamma(\mathbf{U})$ —based on item 4 in Figure 9; and  $\Phi_t(\mathbf{T}, T_1, T_2) = \text{trans}(\Phi(\mathbf{T}))$  is the conjunction of transformed network component targets of  $\Phi(\mathbf{T})$ —based on items 3.x in Figure 9.

- g) For each NETDER NC  $\mu \in \Sigma$  of the form:

$$\Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow \perp : \gamma(\mathbf{U})$$

$\Sigma_t$  contains a NC  $\mu_t$  of the form:

$$\gamma_t(\mathbf{U}, T_1, T_2) \wedge \Upsilon_t(\mathbf{S}) \wedge \Phi_t(\mathbf{T}, T_1, T_2) \rightarrow \perp$$

where  $\Upsilon_t(\mathcal{S}) = \text{trans}(\Upsilon(\mathcal{S}))$  is conjunction of transformed ontological atoms of  $\Upsilon(\mathcal{S})$ —based on item 1 in Figure 9;  $\gamma_t(\mathbf{U}, T_1, T_2) = \text{trans}(\gamma(\mathbf{U}))$  is the transformed network global annotation of  $\gamma(\mathbf{U})$ —based on item 4 in Figure 9; and  $\Phi_t(\mathbf{T}, T_1, T_2) = \text{trans}(\Phi(\mathbf{T}))$  is the conjunction of transformed network component targets of  $\Phi(\mathbf{T})$ —based on items 3.x in Figure 9.

h) For each  $\text{nloc\_pred}(u_1, \dots, u_k) \in \mathcal{L}_{\text{nloc}}$ ,  $\text{eloc\_pred}(v_1, \dots, v_l) \in \mathcal{L}_{\text{eloc}}$ , and  $\text{glo\_pred}(w_1, \dots, w_m) \in \mathcal{L}_{\text{glo}}$ ,  $\Sigma_t$  contains TGDs of the form:

$$\text{tgd}_{\text{nloc}} : \text{node}(X, 0, t_{\text{max}}) \wedge \text{nloc\_pred}(u_1, \dots, u_k, 0, t_{\text{max}}) \rightarrow \text{aux\_nloc\_pred}(X, u_1, \dots, u_k, 0, 1, 0, t_{\text{max}})$$

$$\text{tgd}_{\text{eloc}} : \text{edge}(X, Y, 0, t_{\text{max}}) \wedge \text{eloc\_pred}(v_1, \dots, v_l, 0, t_{\text{max}}) \rightarrow \text{aux\_eloc\_pred}(X, Y, v_1, \dots, v_l, 0, 1, 0, t_{\text{max}})$$

$$\text{tgd}_{\text{glo}} : \text{glo\_pred}(w_1, \dots, w_m, 0, t_{\text{max}}) \rightarrow \text{aux\_glo\_pred}(\Omega, w_1, \dots, w_m, 0, 1, 0, t_{\text{max}})$$

This TGDs allow us to represent the defaults interval  $[0, 1]$  (maximum uncertainty) for each label.

Note that  $T_1, T_2$  are two shared variables in every network component target and network global annotation of each TGD, EGD, or NC, and they allow us to represent in an abstract way the given time frame in a query. Likewise, we sometimes use  $\Upsilon_t$  (instead of  $\text{trans}(\Upsilon)$ ) to denote the transformed version of a NETDER element or the conjunction of them  $\Upsilon$ . Finally, we also need to translate queries.

**Definition A.2** (Transformed NETDER conjunctive query). *Let the sentence*

$$Q(\mathbf{X}) = \exists \mathbf{Y} \rho(\mathbf{X}_1, \mathbf{Y}_1) \wedge \phi(\mathbf{X}_2, \mathbf{Y}_2) \wedge \gamma(\mathbf{X}_3, \mathbf{Y}_3) : [\tau_1, \tau_2]$$

be a NETDER CQ, the transformed NETDER CQ  $\text{trans}(Q(\mathbf{X}))$  of  $Q(\mathbf{X})$  is of the form:  $\exists \mathbf{Y} \rho_t(\mathbf{X}_1, \mathbf{Y}_1) \wedge \phi_t(\mathbf{X}_2, \mathbf{Y}_2, T_1, T_2) \wedge \gamma_t(\mathbf{X}_3, \mathbf{Y}_3, T_1, T_2) \wedge \text{gte}(\tau_1, T_1) \wedge \text{gte}(T_2, \tau_2)$  where  $\rho_t(\mathbf{X}_1, \mathbf{Y}_1)$ ,  $\phi_t(\mathbf{X}_2, \mathbf{Y}_2, T_1, T_2)$ , and  $\gamma_t(\mathbf{X}_3, \mathbf{Y}_3, T_1, T_2)$  are the transformed versions of  $\rho(\mathbf{X}_1, \mathbf{Y}_1)$ ,  $\phi(\mathbf{X}_2, \mathbf{Y}_2)$ , and  $\gamma(\mathbf{X}_3, \mathbf{Y}_3)$ , respectively. Sometimes, we denote  $\text{trans}(Q(\mathbf{X}))$  as  $Q_t(\mathbf{X})$ .

The following result states that each answer to a query (in the ORM) computed using our modified chase can also be computed using the classical chase procedure.

**Proposition A.1.** *Given NETDER conjunctive query:*

$$Q(\mathbf{X}) = \exists \mathbf{Y} \rho(\mathbf{X}_1, \mathbf{Y}_1) \wedge \phi(\mathbf{X}_2, \mathbf{Y}_2) \wedge \gamma(\mathbf{X}_3, \mathbf{Y}_3) : [\tau_1, \tau_2]$$

if  $t$  is an answer to  $Q$  over a knowledge base  $KB$  computed using the NETDER chase, then  $t$  is an answer to query  $\text{trans}(Q)$  over  $\text{trans}(KB)$  using the (oblivious) classical Datalog+/- chase.

*Proof sketch:* To prove this result, we need to first show that: (i) whenever TGDs are applicable in the NETDER chase their transformation will be applicable in the classical one; and (ii) the result of applying TGDs in the NETDER chase can also be obtained by applying their transformation in the classical one. The proof for EGDs is analogous.

Given a TGD  $\Upsilon(\mathbf{X}) \wedge \Phi(\mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{Q}, \mathbf{R}) \wedge \Xi(\mathbf{S}, \mathbf{T}) : \gamma(\mathbf{U})$ , this rule is applicable for the NETDER chase when  $\Upsilon(\mathbf{X})$  (the ontological conditions) is applicable in the classical chase, and,  $\Phi(\mathbf{Y})$  and  $\gamma(\mathbf{U})$  (the network conditions) are applicable to  $(D, G)$  for our NETDER chase at time frame  $[\tau_1, \tau_2]$ . That is, the output of function  $\text{checkCondInState}(Q_1, \text{net\_state})$  is “yes”, where  $Q_1 = \left( h_1(\Phi(\mathbf{Y})) \wedge (\Omega, h_1(\gamma(\mathbf{U}))) \right) : [\tau_1, \tau_2]$ ,  $h_1$  is an adequate homomorphism, and  $\text{net\_state}$  is the output of Algorithm 1 over the NetDiff knowledge base  $\text{DiffKB} = (G, \emptyset)$ . The transformed rule according to item (e) in Definition A.1 incorporates two new conjunctions:  $\Phi_t$  and  $\gamma_t$ . Since the NETDER TGD is applicable, we know that there exist corresponding NetDiff facts that ensure the applicability of  $\Phi(\mathbf{Y})$  and  $\gamma(\mathbf{U})$ , which means that their transformed versions will also hold with appropriate homomorphisms. To see this, we need to consider how applicability is decided for the network conditions, which essentially requires

combining the mappings of the network conditions and the time frame defined in the NETDER query to obtain NetDiff queries. Consider w.l.o.g. a NetDiff query  $Q = ((c, \langle L, [l_1, u_1] \rangle) : [\tau_1, \tau_2])$  where  $c \in V \cup E \cup \{\Omega\}$ ,  $L \in \mathcal{L}$ ,  $[l_1, u_1] \subseteq [0, 1]$ , and  $[\tau_1, \tau_2] \subseteq [0, t_{max}]$  (recall that, by definition of the NETDER TGD chase rule, the NETDER chase queries the network state using Algorithm 1).

If the output of  $checkCondInState(Q, net\_state)$  is “yes”, then the output of  $checkCondInState(Q', net\_state)$  is “yes” for every NetDiff query  $Q' = ((c, \langle L, [l'_1, u'_1] \rangle) : [\tau'_1, \tau'_2])$  where  $[l_1, u_1] \subseteq [l'_1, u'_1]$  and  $[\tau'_1, \tau'_2] \subseteq [\tau_1, \tau_2]$ , which is represented via atoms “ $gte(L'_1, l_1) \wedge gte(u_1, U'_1) \wedge \dots \wedge gte(L'_k, l_k) \wedge gte(u_k, U'_k)$ ” of transformed versions in rule bodies—cf. item f) in Definition A.1, and items 3.x) and 4 in Figure 9—and atoms “ $gte(\tau_1, T_1) \wedge gte(T_2, \tau_2)$ ” of transformed NETDER CQ (cf. Definition A.2). Therefore, this completes the proof that a NETDER TGD is applicable over a KB for the NETDER chase if and only if the transformed TGD is applicable over the transformed  $KB_t$  for the classical *Datalog*+/- chase.

For the second part (ii), when the TGD is applied, both chase procedures add the mapped atoms from  $\Psi(\mathbf{Q}, \mathbf{R})$  to the evolving structure; and both chase procedures add the mapped atoms from  $\Xi(\mathbf{S}, \mathbf{T})$  with interval  $[\tau_1, \tau_2]$ . Therefore, the answers to query  $Q(\mathbf{X})$  over  $KB$  computed using the NETDER chase are the same as those obtained using the classical procedure with the transformed query  $Q_t(\mathbf{X})$  over the transformed knowledge base  $KB_t$ .  $\square$

## Appendix B. Proofs

**Theorem 3.1.** Given a ground NetDiff knowledge base  $DiffKB = (G, P)$ , Algorithm 1 is guaranteed to terminate if one of the following conditions is satisfied:

- Every local rule in  $P$  only contains influence functions that depend on the relationship between the total number of (eligible) neighbors and a fraction of these that satisfy a certain criterion (qualifying neighbors).
- The number of bits used to represent the confidence intervals assigned to every label is bounded by a constant  $b$ .
- Every local rule in  $P$  uses a value of  $\Delta\tau \neq 0$ .

Additionally, the running time of Algorithm 1 is as follows, depending on the specific conditions:

- Case (a):  $O(|DiffKB| \cdot t_{max} \cdot |V| \cdot (IF_{time} \cdot d_{\Omega}^{in} + |\mathcal{L}_{nl}| \cdot |\mathcal{L}_{el}| \cdot d_{\Omega}^{in^2}))$
- Case (b):  $O(|DiffKB| \cdot t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{el}| \cdot |\mathcal{L}_{nl}| \cdot d_{\Omega}^{in}) \cdot 2^{2b})$
- Case (c):  $O(|V| \cdot (IF_{time} + |\mathcal{L}_{el}| \cdot |\mathcal{L}_{nl}| \cdot d_{\Omega}^{in}) \cdot |DiffKB|^{t_{max}})$

In each case,  $d_{\Omega}^{in}$  is the maximum in-degree in the network and  $IF_{time}$  is the worst case cost associated with any influence function used in  $P$ 's local rules.

*Proof:* The proof of condition (a) in the theorem is a slight adaptation of the proof presented in [52]; for greater clarity, the details are reproduced here. For a given  $node(n_i) \in V$ , we will use the notation  $d_{n_i}^{in}$  to denote the number of incoming neighbors (of any edge type) of  $node(n_i)$ , and  $d_{\Omega}^{in} = \max(\{d_{n_k}^{in} \text{ s.t. } node(n_k) \in V\})$ . Moreover, we assume that the worst case of any influence function is  $IF_{time}$  and we use the “satisfaction check” term to refer to the task of checking if a network atom holds in a NetDiff state. For greater readability, we divide the proof into several parts:

*Claim 1.* Let  $lr = L_{tc} \stackrel{\Delta t}{\leftarrow} (nc_{edge}, nc_{node}, if)$  be a ground NetDiff local rule. For each time point,  $lr$  causes the bounds on a network atom (associated with a certain  $node(n_i)$ ) to tighten, there are at most  $|\mathcal{L}_{nloc}| + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{n_i}^{in}$  satisfaction checks, and one influence function calculation performed. There are  $|\mathcal{L}_{nloc}|$  satisfaction checks in the worst case for  $L_{tc}$ , and  $|\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{n_i}^{in}$  satisfaction checks in the worst case for  $nc_{edge}$  and  $nc_{node}$ . Note that this claim holds under all three conditions.

*Claim 2.* If condition (a) is satisfied, then a given ground NetDiff local rule  $r$  can tighten the bound on a network atom of a node at a defined time no more than  $d_{n_i}^{in}$  times. This is because if all the influence functions only depend on the relationship between the total (eligible) neighbors, and a fraction of these that satisfy a certain criterion (qualifying

neighbors), then the total number of different bounds that can be assigned is  $d_{n_i}^{in}$ . Note that this is not only the total number of possibilities for a single application of the rules, but of all applications of the rules before convergence.

*Claim 3.* If condition (a) is satisfied, then each NetDiff local rule contributes a cost in  $O(t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}))$ . Clearly, there are only  $t_{max}$  time steps that can be affected and, for each node and each time step, there is only one label that can be affected by a NetDiff local rule. Hence, by Claims 1-2, we can say that each NetDiff local rule takes at most

$$t_{max} \cdot \sum_{i=1}^{|V|} \left( (IF_{time} + |\mathcal{L}_{nloc}| + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{n_i}^{in}) \cdot d_{n_i}^{in} \right),$$

satisfaction checks, which has complexity  $O(t_{max} \cdot |V| \cdot (IF_{time} \cdot d_{\Omega}^{in} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in^2}))$ .

Then, if condition (a) is satisfied, as Algorithm 1 is dominated by the while loop implementing the fixed point operator, when the time complexity for the convergence of that while loop is considered, we have:

$$O(|DiffKB| \cdot t_{max} \cdot |V| \cdot (IF_{time} \cdot d_{\Omega}^{in} + |\mathcal{L}_{nloc}| \cdot |\mathcal{L}_{eloc}| \cdot d_{\Omega}^{in^2})),$$

which follows from Claim 3 and the size of the NetDiff knowledge base.

*Claim 4.* If condition (b) is satisfied, then a given network atom of a given node considering each time step causes at most  $(t_{max} + 1) \cdot 2^{2b}$  modifications. This is true because the bound on the number of bits, which allows us to represent at most  $2^b$  different numbers and  $2^b + (2^b - 1) + \dots + 1 \leq 2^b \cdot 2^b = 2^{2b}$  different intervals, whereby the amount of modifications of every network atom of a node considering all time steps is at most  $(t_{max} + 1) \cdot 2^{2b}$ .

*Claim 5.* If condition (b) is satisfied, then each NetDiff local rule contributes a cost in  $O(t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot 2^{2b})$ . Similarly to Claim 3, we have from Claims 1 and 4 that each NetDiff local rule takes at most:

$$t_{max} \cdot \sum_{i=1}^{|V|} \left( (IF_{time} + |\mathcal{L}_{nloc}| + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{n_i}^{in}) \cdot 2^{2b} \right),$$

satisfaction checks, which is  $O(t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot 2^{2b})$ .

Then, if condition (b) is satisfied, as Algorithm 1 is dominated by the while loop implementing the fixed point operator, when the time complexity for the convergence of that while loop is considered, we have:

$$O(|DiffKB| \cdot t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot 2^{2b}),$$

which follows from Claim 5 and the size of the NetDiff knowledge base.

*Claim 6.* If condition (c) is satisfied, then the number of applications for a given ground NetDiff local rule is in  $O(|DiffKB|^{t_{max}})$  because the number of applications for a ground NetDiff local rule that can affect a network atom at time  $\tau \in [0, t_{max}]$  depends on the number of modifications at times  $\tau' < \tau$ . Thus, we have that any network atom for any node at time  $\tau = 0$  can only be changed by some NetDiff fact, and then the number of modifications is at most  $|DiffKB|$ . Similarly, any network atom for any node at time  $\tau = 1$  only could be changed by some NetDiff fact (at most  $|DiffKB|$  modifications) and at most  $|DiffKB|$  applications (because it depends on modifications in times  $\tau < 1$ ) of  $|DiffKB|$  rules—thus, at most  $|DiffKB| + |DiffKB|^2$  modifications.

Continuing with this reasoning, for  $\tau = 2$  the number of modifications is  $|DiffKB|$  (from facts) plus  $|DiffKB| + |DiffKB| + |DiffKB|^2$  applications of  $|DiffKB|$  rules: that is, at most  $2 \cdot |DiffKB|^2 + |DiffKB|^3 + |DiffKB|$  modifications. Consequently, we can deduce that the number of modifications at time  $\tau$  grows as  $O(|DiffKB|^{\tau})$ , and when the sum of all the modifications at every time is considered, the complexity is  $O(|DiffKB|^{t_{max}})$ . As the application of a local

rule at time  $\tau$  depends on the modifications that were made at time  $\tau' < \tau$ , then also the number of applications of a given NetDiff local rule grows as  $O(|DiffKB|^{t_{max}})$ .

*Claim 7.* If condition (c) holds, even when  $\Delta\tau = 1$  in all NetDiff local rules of *DiffKB* the number of applications for a given NetDiff local rule has order  $O(|DiffKB|^{t_{max}})$ . This results from the fact that, distinct from the case in Claim 6, the number of modifications at time  $\tau = 2$  is at most  $|DiffKB| + |DiffKB|^2 + |DiffKB|^3$ , at time  $\tau = 3$  is at most  $|DiffKB| + |DiffKB|^2 + |DiffKB|^3 + |DiffKB|^4$ , and so on. Clearly, the number of modifications at time  $\tau \in [0, t_{max}]$  has order  $O(|DiffKB|^\tau)$ , and when the sum of all modifications at every time is considered the number grows as  $O(|DiffKB|^{t_{max}})$ . However, though the complexity is the same using any  $\Delta\tau > 0$ , using a smaller  $\Delta\tau$  is better in practical applications because the probability of interaction between local rules is smaller<sup>10</sup>.

*Claim 8.* If condition (c) is satisfied, then each NetDiff local rule contributes  $O(|V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot |DiffKB|^{t_{max}})$ . Similarly to Claims 3 and 5, we have from Claims 1 and 6 that each NetDiff local rule takes at most:

$$\sum_{i=1}^{|V|} \left( (IF_{time} + |\mathcal{L}_{nloc}| + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{n_i}^{in}) \cdot |DiffKB|^{t_{max}} \right),$$

satisfaction checks, which is  $O(|V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot |DiffKB|^{t_{max}})$ .

Then, if condition (c) is satisfied, as Algorithm 1 is dominated by the while loop implementing the fixed point operator, when the time complexity for the convergence of that while loop is considered, we have:

$$O(|V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot |DiffKB|^{t_{max}}),$$

which follows from Claim 8 even considering the size of the NetDiff knowledge base.  $\square$

**Theorem 4.1.** Let  $KB = (D, G, \Sigma, P)$  be a NETDER knowledge base,  $Q(\mathbf{X})$  be a conjunctive query,  $KB_t = trans(KB) = (D_t, \Sigma_t)$  be the transformation of  $KB$  into a classical *Datalog*+/- knowledge base, and  $Q_t(\mathbf{X}) = trans(Q(\mathbf{X}))$  be the transformed query (classical *Datalog*+/- query). If classical conjunctive query answering  $Q_t$  over  $KB_t$  is decidable, and its associated computational complexity class has a worst case cost of  $O(f(|D_t|))$ , then the computational cost of Algorithm 2 is  $O(poly(|G|)) + O(f(|D_t|))$  under the following conditions:

- *Bounded-Int* policy with diffusion guarantees and ability to modify the network, and
- *Unbounded-Int* policy with diffusion guarantees and without ability to modify the network.

*Proof:* Given a NETDER  $KB = (D, G, \Sigma, P)$ , the computational cost of answering NETDER queries results from: (i) the computational of executing the NETDER chase based on  $KB$ , (ii) the computational cost of executing the diffusion process based on  $(G, P)$ , and (iii) repeating these two steps a number of times dictated by the query answering policy.

The cost of executing the NETDER chase is at most the same as that of executing the classical chase over the transformed knowledge base, which can be inferred from Proposition A.1. The cost of executing the diffusion process with **guarantees** is bounded by a polynomial in the size of the network database  $G$ . Therefore, if the number of times that these steps are repeated is bounded by some constant, we can conclude that the computational cost of answering NETDER queries is  $O(poly(|G|) + O(f(|D_t|)))$ , where  $f(|D_t|)$  is the computational cost function inherent to the complexity class associated with the QA problem over the transformed query and knowledge base (when it is decidable). Clearly, such a constant exists when the *Bounded-Int* policy is used. In the case of the *Unbounded-Int* policy, the theorem only considers the case in which the network cannot be modified; hence, executing the diffusion process only once is enough to reach a fixed point.  $\square$

<sup>10</sup>One way to tackle this complexity source might therefore be to analyze the case in which the number of interactions between local rules is bounded using a notion similar to treewidth; this is the topic of future work.

**Theorem 4.2.** NETDER query answering (Algorithm 2) over knowledge bases  $KB = (D, G, \Sigma, P)$ , such that ontological query answering over  $\langle D, \Sigma \rangle$  is decidable and diffusion process  $P$  is guaranteed to terminate, is Turing-complete.

*Proof:* In order to show that this result holds, we provide the intuitions behind how a deterministic universal TM with an empty input tape can be simulated using a NETDER knowledge base, which is based on the proof of Theorem 3.5 in [9]. We make the following assumptions and initial definitions:

- a) *Unbounded-int* query answering policy is chosen and the ability to modify the network structure from ontological QA is allowed.
- b) The NetDiff knowledge base is defined with  $t_{max} = 1$ , and influence functions are restricted so that termination guarantees hold for the diffusion process (cf. Theorem 3.1).
- c) Influence functions  $if_{one}$  and  $if_{zero}$  are defined such that they always return  $[1, 1]$  and  $[0, 0]$ , respectively. These functions are very restrictive and they allow us to obtain termination guarantees in the diffusion process, because every network atom can change its bound to  $[1, 1]$  or  $[0, 0]$  only once in a given time step.
- d)  $trans(S_1, A_1, S_2, A_2, Dir)$  states that if the current state is  $S_1$  and a symbol  $A_1$  is read, then switch to state  $S_2$ , write  $A_2$ , and move the head in direction  $Dir$  (“right”, “left”, or “stay”).
- e)  $cursor(Y_1, X_1)$  states that the cursor is in position  $X_1$  at time  $Y_1$ .
- f)  $state(Y_1, S_1)$  states the MT is in state  $S_1$  at time  $Y_1$ .
- g)  $content(X_1, Y_1, A_1)$  states that the content of position  $X_1$  at time  $Y_1$  is  $A_1$ .

In this case, a key aspect to simulate a TM is to make an infinite grid appear; hence, we need to create a NETDER knowledge base such that includes:

- 1) The initial ontological database  $D$  with an atom to indicate the initial state ( $d_1$ ), the single final state ( $d_2$ ), the initial content of the tape (blank symbol, which we denote by  $\square$ ) ( $d_3$ ), the initial cursor position ( $d_4$ ), the transitions ( $d_5-d_j$ ), the initial network database  $G$  with an only node to represent the initial point of the grid  $g_1$ , and which meets a specific network condition ( $g_2$  and  $g_3$ ). This is,  $D$  has the following form:

$$D = \{d_1 : state(0, s_1), d_2 : halt(s_0), d_3 : content(0, 0, \square), d_4 : cursor(0, 0), d_5 : trans(s_2, a_1, s_3, a_2, dir_1), \dots, d_j : trans(s_k, a_k, s_{k+1}, a_{k+1}, dir_k)\}$$

and  $G$  has the following form:

$$G = \{g_1 : node(0)\} \cup \mathcal{L} \cup \{g_3 : (node(0), \langle index, [1, 1] \rangle) : [0, t_{max}]\}$$

where  $\mathcal{L} = \mathcal{L}_{glo} \cup \mathcal{L}_{nloc} \cup \mathcal{L}_{eloc}$ ,  $\mathcal{L}_{glo} = \emptyset$ ,  $\mathcal{L}_{nloc} = \{g_2 : index\}$ , and  $\mathcal{L}_{eloc} = \emptyset$ .

- 2) Two rules to create an edge from a node, and a node from an edge (they have the ability to modify the network structure), respectively, which will be used as the basis for an infinite grid to appear. Only one of these has an existential variable in the head, and it can only be applied when a specific network condition is met.

$$tgd_1 : (node(X), \langle index, [1, 1] \rangle) \rightarrow \exists Y edge(X, Y)$$

$$tgd_2 : edge(X, Y) \rightarrow node(Y)$$

- 3) A full TGD to create the default content of each new position.

$$tgd_3 : node(X) \rightarrow content(X, X, \square)$$

- 4) A full TGD to create a grid atom from a transition and two edges.

$$tgd_4 : transition(\mathbf{T}) \wedge edge(X_1, X_2) \wedge edge(Y_1, Y_2) \rightarrow grid(\mathbf{T}, X_1, Y_1, X_2, Y_2)$$

- 5) Three full TGDs to carry out the transitions in the directions “right”, “left”, and “stay”, respectively, based on each grid atom.

$$tgd_5 : grid(S_1, A_1, S_2, A_2, right, X_1, Y_1, X_2, Y_2) \wedge cursor(Y_1, X_1) \wedge$$

$$state(Y_1, S_1) \wedge content(X_1, Y_1, A_1) \rightarrow cursor(Y_2, X_2) \wedge content(X_1, Y_2, A_2) \wedge$$

$$state(Y_2, S_2) \wedge mark(Y_1, X_1)$$

$$tgd_6 : grid(S_1, A_1, S_2, A_2, left, X_1, Y_1, X_2, Y_2) \wedge cursor(Y_1, X_2) \wedge$$

$$state(Y_1, S_1) \wedge content(X_2, Y_1, A_1) \rightarrow cursor(Y_2, X_1) \wedge content(X_2, Y_2, A_2) \wedge state(Y_2, S_2) \wedge mark(Y_1, X_2)$$

$$tg d_7 : grid(S_1, A_1, S_2, A_2, stay, X_1, Y_1, X_2, Y_2) \wedge cursor(Y_1, X_1) \wedge state(Y_1, S_1) \wedge content(X_1, Y_1, A_1) \rightarrow cursor(Y_2, X_1) \wedge content(X_1, Y_2, A_2) \wedge state(Y_2, S_2) \wedge mark(Y_1, X_1)$$

6) A set of full TGDs to represent “inertia”, this is, grid positions that were not modified keep their content.

$$tg d_8 : mark(Y_1, X_1) \wedge grid(\mathbf{T}, X_1, Y_1, X_2, Y_2) \rightarrow keep_f(Y_1, X_2)$$

$$tg d_9 : keep_f(Y_1, X_1) \wedge grid(\mathbf{T}, X_1, Y_1, X_2, Y_2) \rightarrow keep_f(Y_1, X_2)$$

$$tg d_{10} : mark(Y_1, X_2) \wedge grid(\mathbf{T}, X_1, Y_1, X_2, Y_2) \rightarrow keep_p(Y_1, X_1)$$

$$tg d_{11} : keep_p(Y_1, X_2) \wedge grid(\mathbf{T}, X_1, Y_1, X_2, Y_2) \rightarrow keep_p(Y_1, X_1)$$

For each symbol in the tape alphabet  $\{a_1, \dots, a_l, \square\}$ , we have two full TGDs:

$$tg d_{12} : keep_f(Y_1, X_1) \wedge grid(\mathbf{T}, X_1, Y_1, X_2, Y_2) \wedge content(X_1, Y_1, a_l) \rightarrow content(X_1, Y_2, a_l)$$

$$tg d_{13} : keep_p(Y_1, X_1) \wedge grid(\mathbf{T}, X_1, Y_1, X_2, Y_2) \wedge content(X_1, Y_1, a_l) \rightarrow content(X_1, Y_2, a_l)$$

...

$$tg d_{2^*l+10} : keep_f(Y_1, X_1) \wedge grid(\mathbf{T}, X_1, Y_1, X_2, Y_2) \wedge content(X_1, Y_1, a_l) \rightarrow content(X_1, Y_2, a_l)$$

$$tg d_{2^*l+11} : keep_p(Y_1, X_1) \wedge grid(\mathbf{T}, X_1, Y_1, X_2, Y_2) \wedge content(X_1, Y_1, a_l) \rightarrow content(X_1, Y_2, a_l)$$

$$tg d_{2^*l+12} : keep_f(Y_1, X_1) \wedge grid(\mathbf{T}, X_1, Y_1, X_2, Y_2) \wedge content(X_1, Y_1, \square) \rightarrow content(X_1, Y_2, \square)$$

$$tg d_{2^*l+13} : keep_p(Y_1, X_1) \wedge grid(\mathbf{T}, X_1, Y_1, X_2, Y_2) \wedge content(X_1, Y_1, \square) \rightarrow content(X_1, Y_2, \square)$$

7) A full TGD to represent that the MT has to stop when a final state is reached.

$$tg d_{14} : state(Y, S) \wedge halt(S) \rightarrow stop$$

8) An only one NetDiff rule which represents that each node that meets a specific network condition, then it will influence its neighbors in order to meet it the next time as well.

$$diff\_rule : index_T \stackrel{1}{\leftarrow} T, \langle index, [1, 1] \rangle, inf_{one}$$

Note that this NETDER knowledge base has a single TGD with an existential variable that can only be applied when a specific network condition is met. This allows us to create an edge, and then a full TGD creates another node that does not meet the specific network condition, which limits the amount of nulls created during the ontological QA process. For this reason, in this case, ontological QA creates a finite grid, which causes this process being decidable. However, as we described in Figure 7, the next step after ontological QA (*chase* in ORM) is the diffusion process (in NDM), which enables the new node created by ORM to now meet the specific network condition. In this way, the ontological QA has a new grid point available, and since this sequence is infinitely repeated (as made possible when the *Unbounded-int* policy is chosen), then the infinite grid appears. Therefore, we can perfectly mimic a deterministic universal TM even when query answering in the ORM is decidable and the diffusion process in the NDM is guaranteed to terminate.  $\square$

### Appendix C. A Use Case in the Cybersecurity Domain

In this section, we describe a use case for the application of NETDER to address a real-world problem in a cybersecurity domain. This extended example is initially inspired in works that tackle different problems in this domain, such as those by Sapienza et al. [47] and Sarkar et al.[48]. In the former, the authors propose a system to generate warnings about cyber-threats using information on malicious actors in the darkweb and information from cybersecurity expert users on Twitter. On the other hand, in the latter the authors seek to predict cyber incidents using the reply network structure of user interactions from darkweb forums. However, two aspects that we wish to highlight in this section are: (i) frameworks used for such predictions are typically limited to external data (i.e., darkweb, social media, etc.) and are not extensible to consider other cybersecurity data sources such as DNS sink-holes, IDS/IPS data, and honeypots; (ii) it is often difficult to obtain explanations for how a result is reached, and

although the problem of providing explanations is outside the scope of this paper, our proposal has the advantage of being supported by well-founded rule-based formalisms, which can be argued to be a valuable feature towards achieving these capabilities.

Next, we describe how NETDER can be instantiated to address the problem of detecting terms that can be considered to be “dangerous” in that they may lead to the conclusion that IT products are at risk at a specific time, and which products are at risk. Throughout this section, we make the following assumptions:

- We have available information regarding posts written by different actors, and their interactions through forums in online hacker communities such as the deepweb and darkweb. This is a realistic assumption given the body of work that has already been carried out with such sources<sup>11</sup>.
- It is possible to determine when an actor is a *domain expert*; there is work on this problem, such as that of Sarkar et al. [48], in which the authors consider that actors are experts based on their mentions of Common Vulnerabilities and Exposures (CVE)<sup>12</sup> related to important (relative to frequency in the data) Common Platform Enumeration (CPE)<sup>13</sup> groups and their replies to other actors. Other works, such as that of Sapienza et al. [47], rely on the existence of “well-known experts” on social platforms like Twitter.
- A given actor can be an expert only in some domains. For this we use MITRE Corporation’s classification for attack patterns based on the following domains: *software*, *hardware*, *communications*, *supply chain*, *social engineering*, and *physical security*. In this example, we only consider the first three for simplicity reasons.
- Some of the reasoning tasks rely on the *Data Ingestion Module (DIM)* that, as we discussed in the introduction, is in charge of pre-processing and maintaining data. In this case, it builds and maintains the database of users, products, their vendors, and their platforms, detects and maintains who is an early poster about a specific term, processes and maintains the frequency of mentions for terms and products, vendors, or platforms in forum posts, and builds and maintains the network DB (cf. Figure 1). Note that some of the basic tasks carried out by the DIM can be solved in a simple way—for example, data engineers design the data model based on the content of sources, and then databases are built by mapping the data to the model. However, the number and complexity of the data sources used by the system can easily increase the complexity of the tasks if, for example, higher-level data processing such as trust, uncertainty, inconsistency, and incompleteness management are required, or very low update rates are necessary; but even in this case, these issues can be overcome by using data feeds that have already been pre-processed by third-parties; for instance, many cybersecurity companies provide services for carrying out data cleaning tasks, choosing sources based on trustworthiness, and providing a unified schema for its customers to issue queries.

The rest of this section is then organized as follows: we begin with a simplified setup, describing first the network side in Section C.1 and then the ontology side in Section C.2. Then, in Section C.3 we go into greater depth, developing the example into more detail.

### C.1. Network Knowledge

We have a directed graph in which each node represents an actor in some forum on the darkweb, and two nodes are connected if one of them has replied to a post by the other<sup>14</sup>. The predicate symbols that can be used to instantiate the local labels are:

- *bel\_dang*: allows us to instantiate node local labels to represent the degree to which each node believes that a given term is dangerous.

<sup>11</sup>This data kind of data is also available via cybersecurity companies.

<sup>12</sup>CVE numbers are assigned to each vulnerability included in the National Vulnerability Database, which provides definitions for publicly disclosed cybersecurity vulnerabilities and exposures. Its goal is to make it easier to share data among different tools. Entries contain an identification number, a description, and at least one public reference.

<sup>13</sup>CPE is a standardized method of describing and identifying classes of applications, operating systems, and hardware devices.

<sup>14</sup>This connection could be augmented with a label about closeness based on rate of reply between them—we do not consider edge local labels for simplicity reasons.

- 1 – *expert*: allows us to instantiate node local labels to represent the degree to which each node is an expert in a particular domain of attack.
- 2
- 3 – *related*: allows us to instantiate node local labels to represent the degree to which a node believes that some domain of attack and some term are related.
- 4

5 The Data Ingestion Module could obtain *bel\_dang* by using Natural Language Processing (NLP) tools to detect unknown terms in posts and assess the degree of danger based on CVEs and/or known cyber-security words mentioned. On the other hand, *expert* could be obtained by mapping every CVE mentioned to some domain of attack defined by MITRE; expertise could then be a ratio of CVEs in each domain. Finally, *related* could be the result of applying NLP tools to the post where a specific term is mentioned; the relation can be measured between a term in a post and a domain of attack.

6 Analogously, the global labels only can be instantiated with the predicate symbol *dangerous*, representing the degree to which each term is considered dangerous by the network as a whole. Next, we describe the NetDiff facts and rules in  $G$  and  $P$ , respectively. On the one hand,  $G$  contains the following:

$$\begin{aligned}
 G = \{ & \text{node}(n_1), \text{node}(n_2), \text{node}(n_3), \text{node}(n_4), \text{edge}(n_1, n_4), \text{edge}(n_2, n_1), \\
 & \text{edge}(n_3, n_1), \text{edge}(n_2, n_4), \text{edge}(n_3, n_2), \text{expert}(\text{hardware}), \\
 & \text{expert}(\text{software}), \text{expert}(\text{communications}), \text{dangerous}(\text{double-kill}), \\
 & \text{bel\_dang}(\text{double-kill}), \text{related}(\text{software}, \text{double-kill}), t_{\max}(5), \\
 & (\text{node}(n_1), \langle \text{expert}(\text{software}), [0, 0] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_1), \langle \text{expert}(\text{hardware}), [0, 0] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_1), \langle \text{expert}(\text{communications}), [0, 0] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_2), \langle \text{expert}(\text{software}), [0, 0] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_2), \langle \text{expert}(\text{hardware}), [0, 0] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_2), \langle \text{expert}(\text{communications}), [0, 0] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_3), \langle \text{expert}(\text{software}), [0.7, 1] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_3), \langle \text{expert}(\text{hardware}), [0.4, 1] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_3), \langle \text{expert}(\text{communications}), [0.2, 1] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_3), \langle \text{related}(\text{software}, \text{double-kill}), [0.2, 1] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_3), \langle \text{bel\_dang}(\text{double-kill}), [0.2, 1] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_4), \langle \text{expert}(\text{software}), [0.8, 1] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_4), \langle \text{expert}(\text{hardware}), [0.6, 1] \rangle) : [0, t_{\max}], \\
 & (\text{node}(n_4), \langle \text{expert}(\text{communications}), [0.5, 1] \rangle) : [0, t_{\max}] \}
 \end{aligned}$$

7 On the other hand,  $P$  contains the following NetDiff rules:

- 8 –  $lr_1 : \text{bel\_dang}(Y)_T \stackrel{1}{\leftarrow} (\top, \langle \text{bel\_dang}(Y), [0.7, 1.0] \rangle \wedge \langle \text{expert}(X), [0.7, 1.0] \rangle \wedge \langle \text{related}(X, Y), [0.8, 1.0] \rangle, if_1)$

9 This rule states that every node with at least half of its neighbors (as defined in  $if_1$ , see below) who believe  $Y$  is dangerous with confidence between 0.7 and 1.0, are experts on domain  $X$  with confidence between 0.7 and 1.0, and  $X$  is related to  $Y$  with confidence between 0.8 and 1.0, will be affected (involves updating the bound) regarding their degree of belief that the term  $Y$  is dangerous. In this case, the update will depend on the definition of the influence function  $if_1$  and will take effect in the next time step.

- 10 –  $lr_2 : \text{bel\_dang}(Y)_T \stackrel{1}{\leftarrow} (\top, \langle \text{bel\_dang}(Y), [0.5, 1.0] \rangle \wedge \langle \text{expert}(X), [0.0, 0.0] \rangle, if_2)$

11 Similarly, this rule says that every node with at least a ratio of 0.7 of its neighbors who believe  $Y$  to be dangerous with confidence of at least 0.5 (and are non-experts), will be affected regarding their degree of belief that the term  $Y$  is dangerous. In this case, the update will depend on the definition of the influence function  $if_2$ , which requires a greater number of neighbors than  $if_1$  and has a smaller effect on the label target (label obtained from

$bel\_dang(Y)$ ) because we assume that nodes that are not experts have less influence. As in the previous rule, the update will take effect at the next time step.

$$- gr_1 : dangerous(Y) \leftarrow (bel\_dang(Y)_{(expert(X), [0.7, 1.0])}, af_1)$$

Finally, rule  $gr_1$  states that the degree to which  $Y$  is globally believed to be dangerous is based on the application of the function  $af_1$  to the belief of each node that is an expert with confidence at least 0.7.

The influence functions referred to above are defined as follows:

$$if_1(V', node(n), NS, L) = \begin{cases} [0.7, 1.0] & \text{if } |V' \cap A|/|A| \geq 0.5 \\ [0.0, 1.0] & \text{otherwise} \end{cases}$$

where  $A = \{node(n') \in V \text{ s.t. } edge(n', n) \in E\}$ .

$$if_2(V', node(n), NS, L) = \begin{cases} [0.5, 1.0] & \text{if } |V' \cap A|/|A| \geq 0.7 \\ [0.0, 1.0] & \text{otherwise} \end{cases}$$

where  $A = \{node(n') \in V \text{ s.t. } edge(n', n) \in E\}$ . The aggregation function  $af_1$  is defined as  $af_1(B) = [l, u]$  where:

$$l = \sum_{[l_i, u_i] \in B} \frac{l_i}{|B|} \quad \text{and} \quad u = \sum_{[l_i, u_i] \in B} \frac{u_i}{|B|}$$

### C.2. Ontological Knowledge

In the Ontology Reasoning Module we have available information on users from different forums, who are early posters about specific terms, the referred products, their vendors, platforms, and the frequency of co-occurrence between a specific term and a product, a vendor, or a platform. A very reduced database could be the following:

$$D = \{ d_1 : user(n_1, maximus), d_2 : user(n_2, magik), \\ d_3 : earlyPoster(n_1, double-kill), \\ d_4 : co\_occur(double-kill, windows-10, 0.75), \\ d_5 : co\_occur(double-kill, internet-explorer, 0.7), \\ d_6 : platform(application), d_7 : platform(operating-system), \\ d_8 : vendor(microsoft), d_9 : product(windows-10), \\ d_{10} : parent(microsoft, windows-10), \\ d_{11} : parent(operating-system, microsoft) \}.$$

Also, we have defined one NETDER TGD and one EGD in  $\Sigma$ :

$$\omega_1 : user(UID, N) \wedge earlyPoster(UID, T) \rightarrow \exists U product(U) \wedge \\ hyp\_at\_risk(U, UID) \wedge hyp\_vulnerability(T, U) : \langle dangerous(T), [0.5, 1.0] \rangle$$

Intuitively, the rule says that if a user is an early poster about a term  $T$  that is dangerous according to the network with confidence at least 0.5, then we can formulate the hypothesis that some product is at risk, the user in question is responsible for this, and there is also a hypothesis that  $T$  uses exploits to vulnerabilities of  $U$ .

$$\epsilon_1 : product(U_1) \wedge hyp\_vulnerability(T, U_1) \wedge product(U_2) \wedge co\_occur(T, U_2, RF) \wedge RF > \theta \rightarrow U_1 = U_2$$

This says that if there is a hypothesis that  $T$  uses exploits to vulnerabilities of some product  $U_1$  and another product  $U_2$  that co-occurs with  $T$  on posts from forums with a given relative frequency, then the products are the same.

We assume the NETDER query  $Q = \exists X, Y hyp\_vulnerability(X, Y) : [t_{max}, t_{max}]$ , the One-Shot policy, and the NETDER  $KB = (D, G, \Sigma, P)$ . Next, we show the operation of the chase in the context of the NETDER query answering process for  $Q$  over  $KB$  under the One-Shot policy.

Applying ontological rules. Figure 10 shows the result of applying the *chase* procedure to NETDER query  $Q$  and knowledge base  $KB = (D, G, \Sigma, P)$  at time  $[t_{max}, t_{max}]$ . In this case, the execution of the chase corresponds to Step 3 of the One-Shot policy in Figure 7; therefore, the diffusion process has already been executed once and  $G$  has been updated. When  $\omega_1$  is applied, a new null value  $z_1$  is created and three new atoms are produced:

$$product(z_1), \quad hyp\_at\_risk(z_1, n_1), \quad hyp\_vulnerability(double-kill, z_1).$$

Then, when  $\epsilon_1$  is applied  $z_1$  is mapped to a known value and we have:  $z_1 = windows-10$ . At this point,  $\omega_1$  is applied again (this could be done any number of many times), a new null value  $z_2$  is created, and three new atoms are produced:

$$product(z_2), \quad hyp\_at\_risk(z_2, n_1), \quad hyp\_vulnerability(double-kill, z_2).$$

As before,  $\epsilon_1$  can be applied and  $z_2$  is mapped to the known value created in the previous step:  $z_2 = internet-explorer$ .

Note that given the data available up to this point, two hypotheses are generated regarding two different products being vulnerable to *double-kill*. However, when more information is obtained about the workings of *double-kill*, it turns out that *internet-explorer* is only used as a tool to download malware, exploiting a particular weakness in that browser. This subtle difference highlights the importance of adopting a *human in the loop* approach where experts work together with automated tools to obtain better results.

*Network knowledge and diffusion.* Figure 10 only includes the necessary details about the network knowledge that is used to apply TGD  $\omega_1$ . On the other hand, Figure 11 shows a detailed representation of the final state of the network with bounds for each node local label (recall that there are no edge local labels) and the sole global label. This final state is obtained after executing Step 2 of the One-Shot policy in Figure 7. Now, the question is *how* the network reaches this state—to illustrate this as clearly as possible, we describe the evolution process in Figure 12.

The initial state at time 0 is modified by rule  $lr_2$  at time 1, and therefore the label *bel\_dang(double-kill)* is updated in *node*( $n_1$ ) and *node*( $n_2$ ) in Steps 1 and 2, respectively. Then, this same label is updated in *node*( $n_4$ ) by rule  $lr_1$  at time 2. The process finishes when rule  $gr_3$  is applied at time 2, causing the global label *dangerous(double-kill)* to be updated with bound  $[0.75, 1.0]$ , which is obtained from the bounds on the label *bel\_dang(double-kill)* in *node*( $n_3$ ) and *node*( $n_4$ ).

### C.3. A More Complete Set of Rules

In the previous section, we focused on how the *chase* procedure works with ontological and network knowledge, and the diffusion process to produce the latter, and for this initial example we only used one TGD and one EGD. However, in a more realistic setting we require a larger set of rules. We now present a more complete set of rules to better illustrate the capabilities of the approach.

$$(\omega_2): product(U) \rightarrow \exists V vendor(V) \wedge parent(V, U)$$

$$(\omega_3): vendor(V) \rightarrow \exists S platform(S) \wedge parent(S, V)$$

Intuitively, rules  $\omega_2$  and  $\omega_3$  say that every product has an associated vendor, and every vendor has an associated platform, respectively. This encodes a simple subset of the CPE hierarchy described above.

$$(\epsilon_2): product(U_1) \wedge hyp\_vulnerability(T, U_1) \wedge product(U_2) \wedge co\_occur(T, U_2, RF) \wedge (RF > \theta) \wedge vendor(V) \wedge parent(V, U_1) \wedge parent(V, U_2) \rightarrow U_1 = U_2.$$

This EGD states that if there is a hypothesis that  $T$  uses exploits for vulnerabilities of some product  $U_1$ , there is another product  $U_2$  that is mentioned together with  $T$  in forum posts with a given relative frequency (this threshold should be smaller than the one used in  $\epsilon_1$ ), and these products have the same vendor, then there is reason to believe that these products might be the same.

$$(\epsilon_3): product(U) \wedge hyp\_vulnerability(T, U) \wedge vendor(V_1) \wedge parent(V_1, U) \wedge co\_occur(T, V_2, RF) \wedge (RF > \theta) \rightarrow V_1 = V_2.$$

The following rules are similar to the previous one but focus on vendors and platforms. Rule ( $\epsilon_3$ ) says that if there is a hypothesis that  $T$  uses exploits for vulnerabilities of some product  $U$ , and there is another vendor  $V_2$  that co-occurs with  $T$  in posts with a given relative frequency, then  $V_1$  and  $V_2$  may refer to the same vendor.

$$(\epsilon_4): \text{product}(U) \wedge \text{hyp\_vulnerability}(T, P) \wedge \text{vendor}(V_1) \wedge \text{parent}(V_1, P) \wedge \\ \text{platform}(S_1) \wedge \text{parent}(S_1, V_1) \wedge \text{co\_occur}(T, S_2, RF) \wedge (RF > \theta) \rightarrow S_1 = S_2.$$

Finally, if there is a hypothesis that  $T$  uses exploits for vulnerabilities of some product  $U$ , and there is a platform  $S_2$  that co-occurs with  $T$  in forums with a given relative frequency (this threshold should be even smaller than that of  $\epsilon_2$ ), then  $S_1$  and  $S_2$  may refer to the same platform.

As can be seen from inspecting these rules, the knowledge engineering process required to produce them is a complex one, possibly involving the combination of domain experts and semi-automated approaches. One aspect worth highlighting again at this point is that TGDs and EGDs are used as drivers of hypothesis generation and evaluation—so, for instance, even though ( $\epsilon_4$ ) concludes that  $S_1 = S_2$ , all this is doing is assigning a value to a null generated previously by ( $\omega_3$ ); since the chase generates as many of these values as needed, multiple such hypotheses can be entertained at once or eventually dropped.

1) Ontological atom (could be ground)	$pred(S_1, \dots, S_k)$
Transformed version	$pred(S_1, \dots, S_k, 0, t_{max})$
2.1) NetDiff fact (for node local labels)	$(node(n), \langle lab\_pred_1(s_1, \dots, s_k), [l_1, u_1] \rangle) : [\tau_1, \tau_2]$
Transformed version	$\{node(n, 0, t_{max}), aux\_lab\_pred_1(n, s_1, \dots, s_k, l_1, u_1, \tau_1, \tau_2)\}$
2.2) NetDiff fact (for edge local labels)	$(edge(n_1, n_2), \langle lab\_pred_1(s_1, \dots, s_k), [l_1, u_1] \rangle) : [\tau_1, \tau_2]$
Transformed version	$\{edge(n_1, n_2, 0, t_{max}), aux\_lab\_pred_1(n_1, n_2, s_1, \dots, s_k, l_1, u_1, \tau_1, \tau_2)\}$
2.3) NetDiff fact (for global labels)	$(\Omega, \langle lab\_pred_1(s_1, \dots, s_k), [l_1, u_1] \rangle) : [\tau_1, \tau_2]$
Transformed version	$\{aux\_lab\_pred_1(\Omega, s_1, \dots, s_k, l_1, u_1, \tau_1, \tau_2)\}$
3.1) Net. Component Target (for rules referred to nodes)	$(node(X), \langle lab\_pred_1(S_1), [l_1, u_1] \rangle \wedge \dots \wedge \langle lab\_pred_k(S_k), [l_k, u_k] \rangle)$
Transformed version (for rule bodies)	$node(X, 0, t_{max}) \wedge aux\_lab\_pred_1(X, S_1, L'_1, U'_1, T_1, T_2) \wedge gte(L'_1, l_1) \wedge gte(u_1, U'_1) \wedge \dots \wedge aux\_lab\_pred_k(X, S_k, L'_k, U'_k, T_1, T_2) \wedge gte(L'_k, l_k) \wedge gte(u_k, U'_k)$
Transformed version (for rule heads)	$node(X, 0, t_{max}) \wedge aux\_lab\_pred_1(X, S_1, l_1, u_1, T_1, T_2) \wedge \dots \wedge aux\_lab\_pred_k(X, S_k, l_k, u_k, T_1, T_2)$
3.2) Net. Component Target (for rules referred to edges)	$(edge(X, Y), \langle lab\_pred_1(S_1), [l_1, u_1] \rangle \wedge \dots \wedge \langle lab\_pred_k(S_k), [l_k, u_k] \rangle)$
Transformed version (for rule bodies)	$edge(X, Y, 0, t_{max}) \wedge aux\_lab\_pred_1(X, Y, S_1, L'_1, U'_1, T_1, T_2) \wedge gte(L'_1, l_1) \wedge gte(u_1, U'_1) \wedge \dots \wedge aux\_lab\_pred_k(X, Y, S_k, L'_k, U'_k, T_1, T_2) \wedge gte(L'_k, l_k) \wedge gte(u_k, U'_k)$
Transformed version (for rule heads)	$edge(X, Y, 0, t_{max}) \wedge aux\_lab\_pred_1(X, Y, S_1, l_1, u_1, T_1, T_2) \wedge \dots \wedge aux\_lab\_pred_k(X, Y, S_k, l_k, u_k, T_1, T_2)$
4) Net. Global Annotation	$(\langle lab\_pred_1(S_1), [l_1, u_1] \rangle \wedge \dots \wedge \langle lab\_pred_k(S_k), [l_k, u_k] \rangle)$
Transformed version	$aux\_lab\_pred_1(\Omega, S_1, L'_1, U'_1, T_1, T_2) \wedge gte(L'_1, l_1) \wedge gte(u_1, U'_1) \wedge \dots \wedge aux\_lab\_pred_k(\Omega, S_k, L'_k, U'_k, T_1, T_2) \wedge gte(L'_k, l_k) \wedge gte(u_k, U'_k)$

Fig. 9. Mapping from NETDER ontological language elements to elements of classical Datalog+/-.

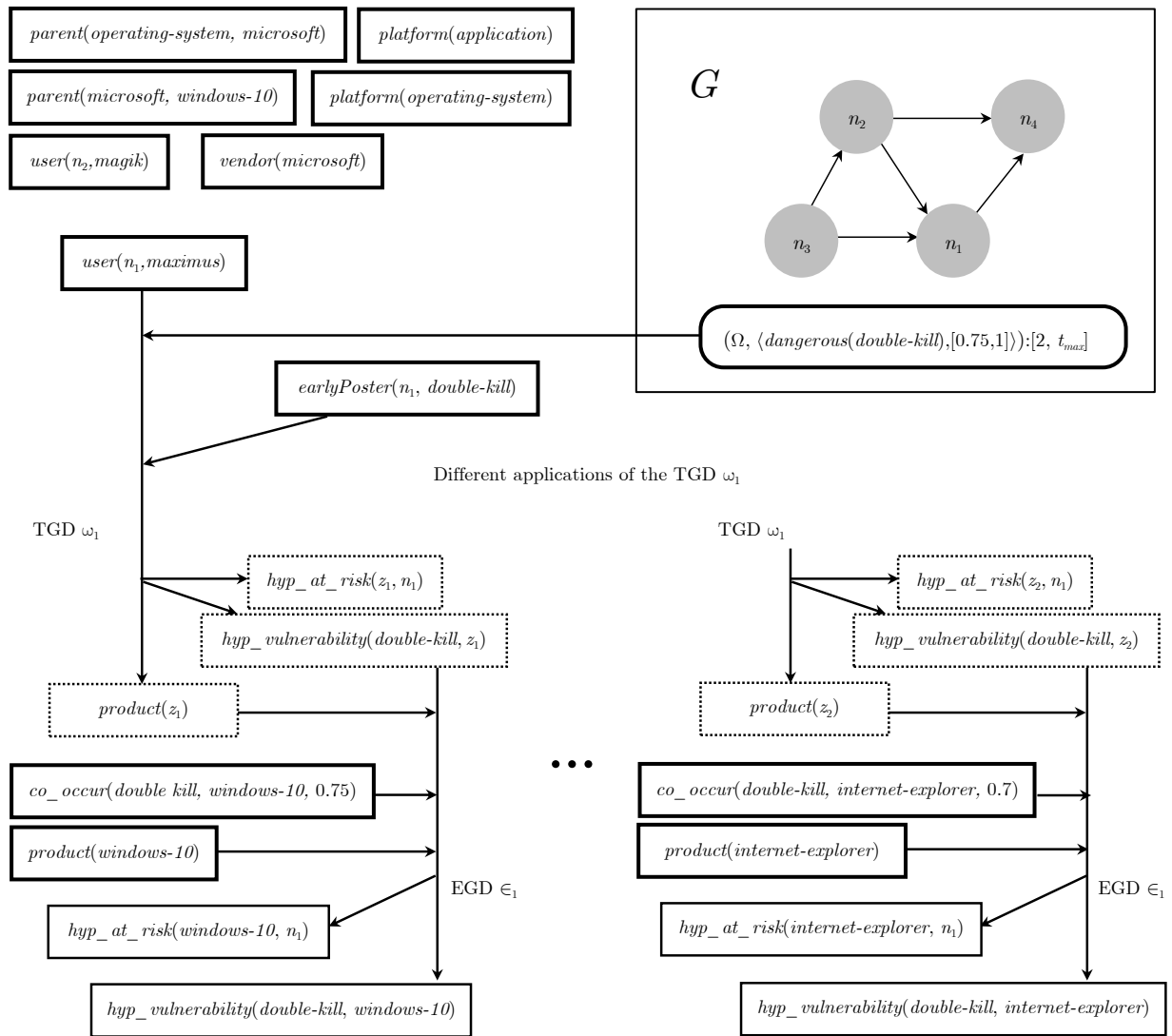


Fig. 10. Example of the chase based on ontological and network knowledge. Boxes with thick outlines represent ontological facts, those with dotted lines represent facts with null values (a kind of partial knowledge), and arrows represent the application of NETDER TGDs and EGDs. The box labeled “ $G$ ” is used to distinguish network knowledge and contains a graphical representation of relationships between the nodes and an example of NetDiff fact (cf. Figure 11 for details regarding the network’s state).

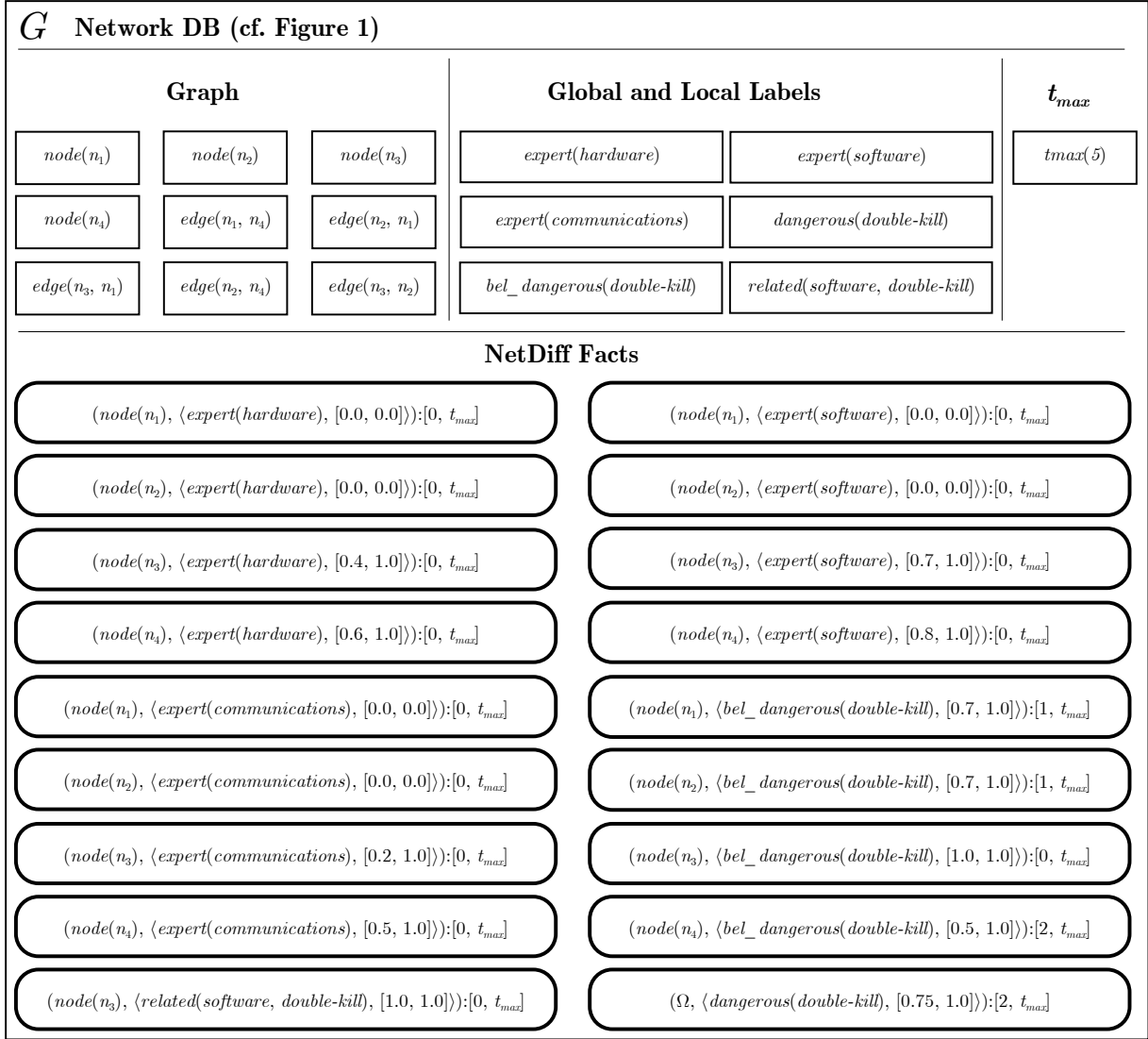


Fig. 11. Detailed view of the state of the network at the moment when the *chase* is carried out as seen in Figure 10. Box with thick outline define a graph with nodes and their connections, and those with rounded corners represent NetDiff facts. Labels with bound  $[0.0, 1.0]$  are not explicitly represented.

<i>Time</i>	<i>Step</i>	<i>G</i>	$L_{loc1}$	$L_{loc2}$	$L_{loc3}$	$L_{loc4}$	$L_{loc5}$	$L_{loc6}$	$L_{loc7}$
0	0	$node(n_1)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	?
0	0	$node(n_2)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	?
0	0	$node(n_3)$	[0.7, 1.0]	[0.4, 1.0]	[0.2, 1.0]	[1.0, 1.0]	?	?	[1.0, 1.0]
0	0	$node(n_4)$	[0.8, 1.0]	[0.6, 1.0]	[0.5, 1.0]	?	?	?	?
1	1	$node(n_1)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	[0.7, 1.0]
1	1	$node(n_2)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	?
1	1	$node(n_3)$	[0.7, 1.0]	[0.4, 1.0]	[0.2, 1.0]	[1.0, 1.0]	?	?	[1.0, 1.0]
1	1	$node(n_4)$	[0.8, 1.0]	[0.6, 1.0]	[0.5, 1.0]	?	?	?	?
1	2	$node(n_1)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	[0.7, 1.0]
1	2	$node(n_2)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	[0.7, 1.0]
1	2	$node(n_3)$	[0.7, 1.0]	[0.4, 1.0]	[0.2, 1.0]	[1.0, 1.0]	?	?	[1.0, 1.0]
1	2	$node(n_4)$	[0.8, 1.0]	[0.6, 1.0]	[0.5, 1.0]	?	?	?	?
2	3	$node(n_1)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	[0.7, 1.0]
2	3	$node(n_2)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	[0.7, 1.0]
2	3	$node(n_3)$	[0.7, 1.0]	[0.4, 1.0]	[0.2, 1.0]	[1.0, 1.0]	?	?	[1.0, 1.0]
2	3	$node(n_4)$	[0.8, 1.0]	[0.6, 1.0]	[0.5, 1.0]	?	?	?	[0.5, 1.0]

Fig. 12. State of the network at times 0, 1, and 2. We use the following names:  $L_{loc1} = expert(software)$ ,  $L_{loc2} = expert(hardware)$ ,  $L_{loc3} = expert(communications)$ ,  $L_{loc4} = related(software, double-kill)$ ,  $L_{loc5} = related(hardware, double-kill)$ ,  $L_{loc6} = related(communications, double-kill)$ ,  $L_{loc7} = bel\_dang(double-kill)$ .