Graphical norms via conceptual graphs

Madalina Croitoru, Oren Nir, Miles Simon, Luck Michael

Knowledge-Based Systems 29 (2012) 31–43

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Graphical norms via conceptual graphs

Madalina Croitoru [a], Nir Oren [b,*], Simon Miles [c], Michael Luck [c]

[a] LIRMM, University Montpellier II, France
[b] Department of Computing Science, University of Aberdeen, UK
[c] Department of Informatics, King's College London, UK

ARTICLE INFO

Article history:
Available online 14 July 2011

Keywords:
Norms
Conceptual graphs
Reasoning
Graph-based reasoning
Normative violations

ABSTRACT

The specification of acceptable behaviour can be achieved via the use of obligations, permissions and prohibitions, collectively known as *norms*, which identify the states of affairs that should, may, or should not hold. Norms provide the ability to constrain behaviour while preserving individual agent autonomy. While much work has focused on the semantics of norms, the *design* of normative systems, and in particular understanding the impact of norms on a system, has received little attention. Since norms often interact with each other (for example, a permission may temporarily derogate an obligation, or a prohibition and obligation may conflict), understanding the effects of norms and their interactions becomes increasingly difficult as the number of norms increases. Yet this understanding can be critical in facilitating the design and development of effective or efficient systems. In response, this paper addresses the problem of norm explanation for Naïve users by providing of a graphical norm representation that can explicate why a norm is applicable, violated or complied with, and identify the interactions between permissions and other types of norms. We adopt a *conceptual graph* based semantics to provide this graphical representation while maintaining a formal semantics.

1. Introduction

Solutions to problems arising in the domain of multi-agent systems have often been inspired by approaches from human societies. Nowhere is this more evident than in addressing the problem of controlling the behaviour of agents within open systems. Here, interactions between agents can cause unexpected system behaviour, and traditional procedural approaches fail due to the unpredictability and complexity of these interactions, as well as the inherent autonomy of the agents involved. In human societies, behavioural control is achieved in a declarative manner, by specifying expectations regarding the behaviour of others, such as with laws or rules. These specifications, or *norms*, identify obligations, permissions and prohibitions that individuals are expected to comply with in particular situations. Drawing on this, there has been much work concerning the application of norms to artificial systems, in which agents are able to make use of concepts such as obligations, permissions, and prohibitions, to represent and reason about socially imposed goals and their execution. Such *norm aware* agents are able to decide whether to act in a manner consistent with norms, or whether to ignore them. In this context, norms are generally imposed on a set of agents in order to increase the overall utility of a system (often at the cost of individual utility) [18], or to reduce computational or communication overhead [4].

While the design and architecture of norm aware agents is critically important, this is not the only problem that must be addressed when utilising norms. Perhaps more interesting (and more challenging) is the problem of design time identification of which norms are needed in order to achieve some desired behaviour. Norms can interact with each other in unpredictable ways, and determining the effects of a norm on a system can thus be difficult. To identify these problematic norm interactions requires us to be able to *explain* the effects of a norm, and why, in some specific situation, it is applicable, violated, complied with, or in some other state, yet this has not been investigated to any real depth. Moreover, the ability to provide such explanations can enable designers to better understand the interactions between different norms, thereby allowing them to avoid introducing redundant norms [3], and to specify norms more precisely. Norm explanations can thus provide vital support for the design a normative system. In addition, from the perspective of users, norm explanation can facilitate a more intuitive appreciation of a system by providing a stronger understanding of the *reasons* why particular norms may have been brought to certain states in response to system events. Such a facility can increase and enhance the trust of a user in relation to operation of the system, providing confidence that it is in fact operating correctly.

Since much of the research into the formal properties of norms has taken place within the area of philosophy and deontic logic

* Corresponding author.
E-mail addresses: croitoru@lirmm.fr (M. Croitoru), n.oren@abdn.ac.uk (N. Oren), simon.miles@kcl.ac.uk (S. Miles), michael.luck@kcl.ac.uk (M. Luck).

0950-7051/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2011.06.025

[13,23], norms are typically specified within a knowledge-based system (KBS) using a logic which, for non-technical users, is often difficult to understand. However, in order for a KBS to be usable by such users, it is essential that they can understand and control not only the knowledge base construction process, but also how results are obtained from the running system. It should be easy for users not only to enter different pieces of knowledge and to understand their *meaning* but also to understand the *results* of the system, and *how* the system computed these results. This latter aspect, namely the ability to understand *why* the system gives a certain answer, is especially important since the expertise of different users may vary, and explaining each step of the logical inference process poses a difficult problem.

However, due to the core properties of norms, providing such explanations is not trivial. First, norms can be applicable only in specific circumstances, rather than over a system's entire lifetime. Thus, examining norms in isolation from a running system may not provide any useful explanation regarding an individual agent's behaviour. Second, multiple norms can interact with each other, collectively placing complex expectations on the various agents involved. Thus, while it may appear that an agent is violating some obligation, it may actually be the case that the agent is either currently exempt from this obligation due to it not being applicable in the current situation, or due to there being some *permission* that applies in the current circumstances, overriding the obligation. Given this, it should be clear that it is extremely difficult for non-technical users (indeed, also for technical experts) to interpret a large set of textually (logically) specified norms and identify their effects, and that an alternative solution to norm understanding is required.

In response, our aim in this paper is to provide a sound graphical representation of norms, by adopting a graph-based semantics and applying the semantics to normative systems. To do so, we adopt the normative framework of Oren et al. [17], a generic framework that enables updating and monitoring of the changing *status* of norms, and supports the normative reasoning process. Now, in order to provide such a graphical representation, we must be able to provide a sound and complete translation between the operations of the normative framework and the operations on the graph-based representation. Not only can this help in *understanding* the results of an update to the status of a norm, but it also allows for structural *optimisations* of norms that might not be obvious from the textual (logical) representation of the norm. Each of these is a significant challenge; in this paper, we focus on the former aspect of the graphical representation, leaving the latter for future work.

Oren et al.'s framework represents norms by means of sets of first order logic tuples, which are manipulated using a set of rules that can be reduced to first order logic subsumption on the individual tuple elements. The contribution of this paper is to map norms onto *conceptual graphs* [19,20], the only graph based formalism to have a sound and complete semantics corresponding to deduction (via subsumption) in first order logic. This formal semantics enables us to easily link Oren et al.'s norms, with their textual representation, to the conceptual graph's graphical representation, thereby providing a graphical explanation regarding the system's normative state to non-technical users. This aspect of our work was first discussed in [6], in which it was shown how individual obligations can be represented graphically. Representing permissions, and their interaction with obligations, introduces further complications, but we can extend the basic model to address this, as originally outlined in [16].

The remainder of this paper is structured as follows. In the next section, we provide the necessary formal background to the paper by briefly reviewing the normative framework and introducing the conceptual graph formalism. In Section 3, we show how the status of norms can be computed graphically. Section 4 then considers the graphical representation of interactions between permissions and other norm types. In Section 5, the paper provides a discussion in two parts: first it offers an evaluation of the effectiveness of our approach, together with an assessment of what is needed for more substantial user studies; second, it reviews some important related work. Finally, Section 6 concludes the paper by considering possible extensions to our work.

2. Background

In order to provide the requisite context for the contributions of the paper, and the basis on which we are able to develop norm explanations, we begin in this section by reviewing the formal model of norms. The model focuses on the problem of monitoring in that it facilitates identification of the *status* of norms as the environment changes over time. We then introduce the graphical formalism used in the remainder of this paper, conceptual graphs (CGs), which we map to the normative model in Section 3. This mapping allows us to address the problem of *explanation*, identifying why a norm has a particular status at some point in time.

2.1. The normative model

We introduce the normative model in a somewhat informal manner, motivating it in the context of a small example and examining how the model can be applied. Consider a situation in which an agent takes their car to a repair shop in order to be repaired. This repair shop provides a guarantee to its customers that their cars will be repaired within seven days, and thus has an *obligation* upon it, whenever a car arrives, to repair it within seven days. Clearly, once this obligation is fulfilled, it is lifted, and the repair shop no longer needs to repair the car. However, the obligation remains on the repair shop *as long as* the car is not repaired (even after seven days have passed). Finally, circumstances beyond the repair shop's control (for example, a power failure), will give the repair shop permission to repair the car seven days later than otherwise required.

The requirement on the repair shop to mend a car within seven days only obliges the repair shop to take action once a car actually arrives. Until then, the norm is an *abstract norm*. When a customer brings in a car, the norm is instantiated, thereby obtaining normative force over the repair shop and obliging it to repair the car within seven days. A single abstract norm can result in multiple *instantiated norms*; if two cars arrive at the repair shop, two instantiations of the abstract norm will occur.

Given this example, we observe that a norm may be defined in terms of five components. First, a norm has a *type*, such as an obligation, or a permission. Second, a norm has an *activation condition*, identifying the situations in which the norm affects some agents. Third, a norm imposes some *normative condition* on the affected agents; if this normative condition does not hold, then the norm is not being complied with (or made use of in the case of a permission). Fourth, norms have an *expiration condition*, identifying the situations after which the norm no longer affects the agent. Finally, the norm must identify the agents to which it is directed (i.e. those it affects), referred to as the *norm targets*.

More formally, we assume that the permissions and obligations represented by the norm refer to states and events in some environment, represented by some logical predicate language $\mathcal{L}$, such as first order logic. A norm is then a tuple of the form:

$\langle NormType,$
$NormActivation,$
$NormCondition,$
$NormExpiration,$
$NormTarget\rangle,$

where

1. $NormType \in \{obligation, permission\}$; and
2. $NormActivation, NormCondition, NormExpiration, NormTarget$ are all well formed formulae (*wff*) in $\mathcal{L}$.

Thus, for example, the following abstract norm represents the idea that a repair shop must repair a car within seven days of its arrival at the shop[1]:

$$\begin{aligned}&\langle obligation,\\ &arrivesAtRepairShop(X, Car, T_1),\\ &repaired(Car) \vee (currentTime(CurrentTime) \wedge\\ &before(CurrentTime, T_1 + 7days)),\\ &repaired(Car),\\ &repairShop(X)\rangle.\end{aligned}$$

For ease of presentation, we have taken a relaxed approach to the notation, and mixed events and states within this norm; a more complex underlying language, such as the Event Calculus [11], would allow disambiguation of these concepts (as well as providing a richer typology of temporal notions).

If, at any point, an abstract norm's *NormActivation* condition holds, an instantiated version of the norm is created (subject to the additional constraint that the norm is not already instantiated for the same reason, as discussed in detail in [17]). The instantiation of a norm involves creating a copy of the abstract norm in which the norm's variables are bound to the values that caused the *NormActivation* condition to evaluate to true. When instantiated, the individuals included in *NormTarget* are identified. These individuals are then either obliged or permitted to bring about the normative goal specifed by *NormCondition*, until such a time as the conditions specified by *NormExpiration* hold. In this way, instantiated norms *persist* until they expire. (Note that a more complete logical semantics for the instantiation and processing of norms in this way is provided in [17].)

Now, if a car, car_1, arrives at Bob's repair shop at time 12, we can instantiate the abstract norm above and obtain the following instantiated norm:

$$\begin{aligned}&\langle obligation,\\ &arrivesAtRepairShop(bob, car_1, 12),\\ &repaired(car_1) \vee (currentTime(CurrentTime) \wedge\\ &before(CurrentTime, 19)),\\ &repaired(car_1),\\ &repairShop(bob)\rangle.\end{aligned}$$

It should be noted that there can be variables within an instantiated norm (as is the case for *CurrentTime* in the above norm), and that the norm target refers to both the agent (*bob*), and the role (*repairShop*) undertaken by the agent in the context of the norm. Given a *NormTarget*, one simple way of computing the set of agents affected by the norm, is as follows: $\{X | NormTarget \vdash role(X)\}$ for any predicate *role*.

A key aspect of the normative framework is that it enables the identification of the changing *status* of norms over time. This status can include the fact that it is instantiated or abstract, whether it is being complied with or violated, and whether it has expired. This is critical in understanding the impact of norms on behaviour and determining what actions to take as a result; the work in [17] introduces several distinct predicates that capture these different possibilities for status. For example, *violation* of a norm may require some remedial action, and is thus a relevant status value, with an associated predicate. Importantly, the status can also be referred to by other norms. For example, a norm stating that "*if a car has not been repaired after seven days, the repair must be free*", can be represented as follows (assuming that the norm above is labelled $n1$):

$$\begin{aligned}&\langle obligation,\\ &violated(n1),\\ &repairCost(Car, 0),\\ &false,\\ &repairShop(X)\rangle.\end{aligned}$$

Here, the $violated(n1)$ predicate refers to the norm's status, and evaluates to true if and only if $n1$ is an instantiated obligation whose normative condition evaluates to false, and for which there is no permission that allows the *negation* of the normative condition. This, and other such predicates are formally defined in [17].

As seen in this example, norms can *explicitly* refer to other norms and the variables found within them (such as *Car* in the example above). In addition, as we will see later there may also be *implicit* references to other norms (most notably in the case of permissions). Determining the status of any particular norm thus requires an examination of the interactions between multiple norms; when a system contains many norms connected to each other by such implicit and explicit references, it can be extremely difficult to identify precisely why some norm has a particular associated status. In order to address this difficulty of understanding and identification, we seek an alternative means of examining the status of norms in such systems. In particular, since humans find it much easier to assimilate large amounts of graphical information, as opposed to information in other forms, it is appropriate to make use of a graphical model to represent and visualise norms. In doing so, we are able to make explicit the links between the norms described above in a way that is amenable to human inspection and understanding. Of course, since such a representation must also to be able to be processed by machine, the best choice of representation to use for this purpose is one that is well understood and has a formal semantics. In consequence, therefore, we adopt *conceptual graphs* as the foundation for our graphical representation mechanism. In the next subsection, we introduce and describe this conceptual graph formalism.

2.2. Conceptual graphs

Due to their visual qualities, *semantic networks*, which were originally developed as cognitive models, have been used for knowledge representation since the early days of artificial intelligence, especially in natural language processing. Different kinds of semantic networks all share the basic idea of representing domain knowledge using a graph, but there are differences concerning notation, as well as rules or inferences supported by the language. In semantic networks, *diagrammatical reasoning* is mainly based on path construction in the network.

In this context, we can distinguish two major families of languages resulting from work on *semantic networks*: KL-ONE and *conceptual graphs*. KL-ONE [22] is considered to be the ancestor of *description logics* (DLs) ([1]), which form the most prominent family of knowledge representation languages dedicated to reasoning about ontologies. However, description logics have now lost their graphical origins. In contrast, *conceptual graphs* were introduced by Sowa (cf. [19,20]) as a diagrammatic system of logic intended "to express meaning in a form that is logically precise, humanly readable, and computationally tractable" (cf. [20]). Throughout the remainder of this paper we use the term

[1] Unless otherwise stated, we make use of Prolog notation within our logical formulae. More specifically, variables are written with an initial capital letter, while constants begin with a lowercase letter.

"conceptual graphs" to denote the family of formalisms rooted in Sowa's work and then enriched and further developed with a graph-based approach (cf. [5]).

In the conceptual graph (CG) approach, all kinds of knowledge can be encoded as graphs and can thus be naturally visualised. More specifically, a CG partitions knowledge into two types, the first of which identifies the CG's *vocabulary*, and can be seen as a basic ontology, and the second of which, (referred to as the *basic graph*) stores facts encoded using the CG's vocabulary. The vocabulary, referred to as the CG's *support*, is composed of two distinct parts, namely a partial order of concepts, and a partial order of relations (of any arity). Since both parts of the support are partial orders, they can be visualised by their Hasse diagram, where the partial order represents a specialisation relation, $t' \leqslant t$, indicates that t' is a specialisation of t. More specifically, if t and t' are concepts, $t' \leqslant t$ indicates that every instance of the concept t' is also an instance of the concept t. Fig. 1 provides an example of a concept hierarchy constructed in this way, and which is used in the illustrative example throughout this paper. Similarly, if t and t' are relations, and these relations have the same arity, say k, then $t' \leqslant t$ means that if t' holds between k entities, then t also holds between these k entities. Fig. 2 shows the relation hierarchy that is used in the example throughout this paper. These relations are organised by arity — unary, binary and ternary and so on — with a separate graph for each.

Now, a CG's basic graph encodes knowledge based on the representation of entities and their relationships. This encoding takes the form of a bipartate graph, consisting of *concept nodes* that represent entities, and *relation nodes* that represent relationships between these entities or their properties. A concept node is labelled by a pair, $t{:}m$, where t is a concept drawn from the concept hierarchy, and m is called the *marker* of the node. Markers consist of either a specific individual name, or a marker, denoted $*$, which acts as a generic marker, and is used if the concept node refers to an unspecified entity. A relation node is labelled by a relation r taken from the relation hierarchy and, if r has an arity of k, the relation node must be incidental to k totally ordered edges. Classically, concept nodes are drawn as rectangles and relation nodes as ovals. The order on edges incidental to a *k-ary* relation node is then represented by labelling the edges with numbers from 1 to k. Fig. 3 provides an example of a basic graph that expresses the fact that a *vehicle arrived* at the *RepairShop* at a certain *Time*. Finally, since the notion of *instantiated norm* is central to our framework, we occasionally refer to an *instantiated basic graph*, which is simply a basic graph with no generic markers, as shown in Fig. 4.

Given these basic notions of conceptual graphs, a mapping between a CG and first order logic can be used to provide the CG with a semantics. This mapping, denoted by Φ in the conceptual graphs literature, utilises a first order language corresponding to the elements of the conceptual graph's vocabulary (i.e. its relation and concept hierarchies). Elements from the concept hierarchy are translated into unary predicates, and elements from the relation hierarchy with an arity of k are mapped into k-ary predicates. Individual names are then constants in the logic. Formulae are added to the logic based on the partial orders of concepts and relations: if t and t' are concepts, and $t' < t$, then the formula $\forall x((t'(x) \rightarrow t(x))$ is obtained. Similarly, if r and r' are k-ary relations, with $r' < r$, then the formula $\forall x_1 \ldots x_k(r'(x_1,\ldots,x_k) \rightarrow r(x_1,\ldots,x_k))$ is obtained. A fact G obtained from the basic graph can then be translated into a positive, conjunctive and existentially closed formula (via the mapping $\Phi(G)$), with each concept node being translated into a variable or a constant. If the concept node is a generic node (as in the concept nodes on the left hand side of Fig. 3), then $\Phi(G)$ results in a variable, otherwise $\Phi(G)$ returns the concept node's individual marker.

While we have shown how CGs can be mapped to a first order logic formula, mapping between CGs and first order logic in order to perform reasoning is cumbersome, and a large part of the power of the CG approach is obtained from the ability to perform reasoning over the graphs themselves. The fundamental operation used to perform such reasoning is *projection*. In order to describe this concept, we must first define the notion of a *homomorphism*.

Let G and H be two basic graphs (BGs). A *homomorphism* π, from G to H, is a mapping, from the concept node set of G to the concept node set of H, and from the relation node set of G to the relation node set of H, that preserves edges and may decrease concept and relation labels. That is:

- for any edge labelled i between the concept node c and relation node r in G, there is an edge labelled i between the nodes $\pi(c)$ and $\pi(r)$ in H; and
- for any (concept or relation) node x in G, the label of its *image* $\pi(x)$ in H is a specialisation of the label of x; that is, $\pi(x) \leqslant x$.

Homomorphisms are used to form *projections* between two BGs, as illustrated in Fig. 5. Here, the BG on the left hand side of the figure models the situations when some *Car* arrives at bob's repair shop at *Time* 12, while the right hand side of the figure models the case when some *Vehicle* arrives at some *RepairShop* at some *Time*. The homomorphism (indicated using dashed lines) indicates that node *RepairShop* on the right can be mapped onto the

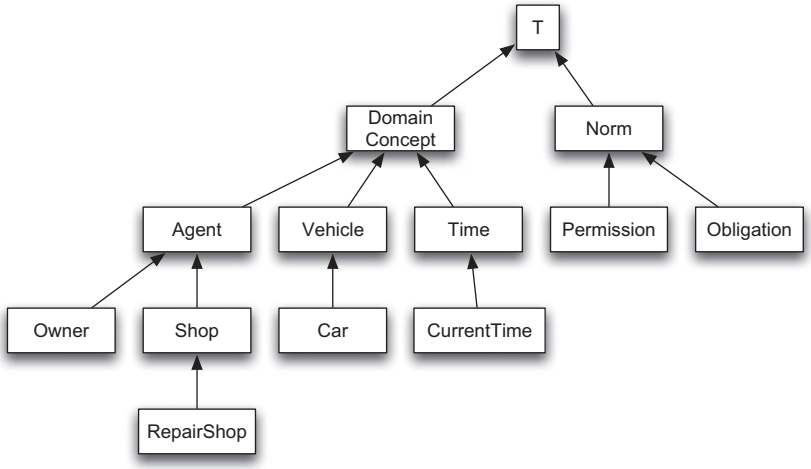

Fig. 1. Conceptual graph support: the *concept hierarchy*.

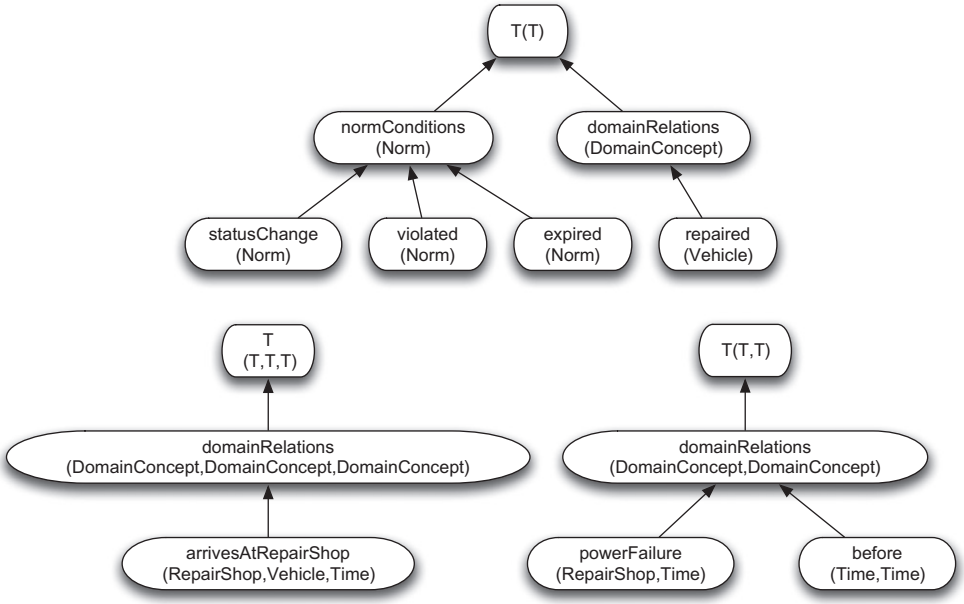

Fig. 2. Conceptual graph support: the *relation hierarchy*.

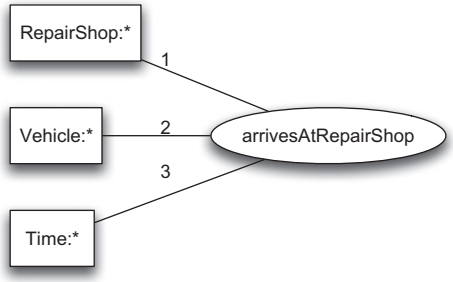

Fig. 3. A generic *basic conceptual graph fact*.

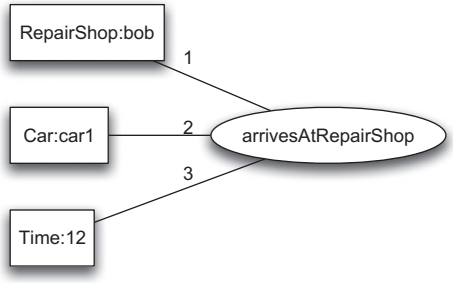

Fig. 4. A ground, or instantiated, *basic conceptual graph fact*.

RepairShop node on the left with marker *bob*, that the *Vehicle* node on the right can be mapped to *Car* on the left, and that the generic *Time* node on the right can be mapped to a specific *Time* node. Finally, the *arrivesAtRepairShop* relation maps between the two BGs.

Now, an important theorem in the CG literature (referred to as the *fundamental theorem*) also allows us to map back from first order logic to conceptual graphs. This theorem states that, given two BGs, G and H, there is a homomorphism from G to H if and only if $\Phi(G)$ is a semantic consequence of $\Phi(H)$ and the logical translation of the vocabulary: $\Phi(\mathcal{V}), \Phi(\mathcal{H}) \models \Phi(\mathcal{G})$. This is a soundness and completeness theorem of BG homomorphism with respect to first order logic entailment, the consequence of which is that a homomorphism between two graphs is, in effect, an explanation as to why logical subsumption takes place. Since such homomorphisms can be represented graphically, this allows for visual representations of logical subsumption, the explanation of which is a unique feature of CGs. Any alternative logic-based graphical representation language would have to include an additional separate explanation layer as well as the representation layer itself.

Given the fundamental building blocks we have now introduced, of the normative model and conceptual graphs, we can proceed to detail how a norm can be represented within a CG-based framework.

3. Graphically computing the status of norms

3.1. Modelling norms with CGs

By encoding structured knowledge graphically, CGs can provide a way to represent, illustrate and interpret the states through which norms proceed; that is, whether they have been activated, violated, fulfilled, or expired. Then, by connecting such representations (or depictions) of permissions and obligations, it is possible to interpret whether an obligation has truly been violated, or whether a permission derogates this obligation under particular circumstances.

One commonly encountered problem is that norms can sometimes be fulfilled by multiple different actions, events or states. Intuitively, if these conditions are separated by disjunctions, they can be evaluated in a tree-like structure by the norm reasoner. We make this explicit by representing norms in such a structure, with every level of the tree corresponding to one type of condition in the norm. Moreover, at every level, we break the condition into a disjunction of positive first order logic conjunctions. This representation ensures that normative reasoning is sound and complete with respect to a particular kind of path-finding in the norm tree (finding at least one satisfied level node). Now, when instantiated, a norm's activation condition becomes fixed, and its normative and expiration conditions are used to determine its status. This suggests that the tree structure is indeed suitable for use in representing a norm. In what follows, we proceed to define this tree structure, which we refer to as a *norm tree*.

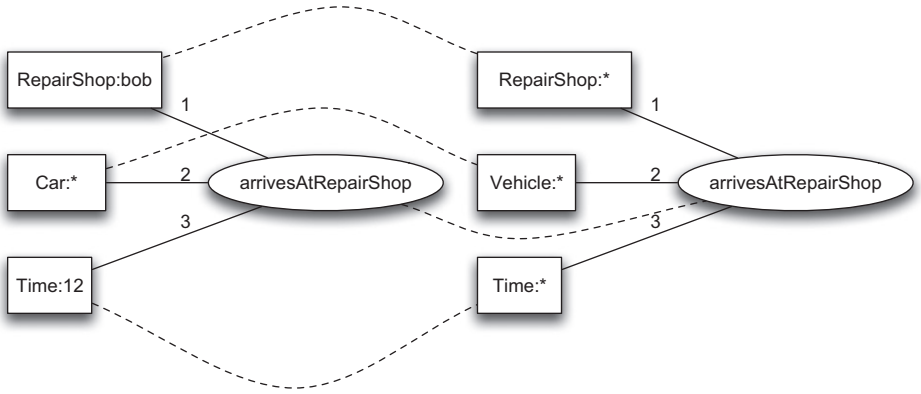

Fig. 5. A projection between two basic conceptual graph fact.

A norm tree represents both abstract and instantiated norms. Its root is associated with the entire norm (more specifically, its *type* and *target*), while the remaining levels represent different parts of the norm. (For this purpose, we also assume that a norm's *target* is a conjunctive formula, and can thus be represented as a conceptual graph). Nodes in the second level are associated with the activation condition, nodes in the third level are associated with the normative condition, and nodes in the fourth level with the expiration condition. Each of the nodes within the tree has an associated CG representation of its content, as illustrated in Fig. 6.

Given this basic structure, different branches of the norm tree can be used to represent disjunctive conditions within a specific norm attribute. Thus, for example, a norm with a normative condition of the form $a \vee b$ would have two branches at the norm tree's third level. As indicated above, we assume that the norm target parameter consists of a conjunctive combination of predicates (in other words, a norm is associated with a specific group of individuals rather than applying to some subgroup *or* another), and that all other parameters (except for norm type), may contain disjunctions. In this way, in order to represent the norm as a norm tree, we transform all of its attributes into disjunctive normal form, to get a norm represented as follows:

$$\left\langle Type, \bigvee_{i=1,a} AC_i, \bigvee_{j=1,c} NC_j, \bigvee_{k=1,e} EC_k, NT \right\rangle, \tag{1}$$

where AC_i, NC_j, EC_k and NT are all conjunctive first order formulae so that, for example, $AC = \bigvee_{i=1,a} AC_i$. Furthermore, by assuming negation as failure, we can ensure that all of these formulae are positive (by introducing an explicit predicate for negation), and can therefore represent each as a conceptual graph, defined on some given support (i.e. the domain ontology).

Given a norm N in disjunctive normal form as in Eq. (1) above, we define its norm tree as a tree for which each node contains a norm and is labelled by a CG as follows.

1. The root node of the tree contains norm N and is labelled by a CG identifying the norm's type and targets (i.e. *Type* and *NT*).
2. The root node has a child nodes (i.e. nodes at level one) where, for $i = 1 \ldots, a$, child node i is labelled with the CG representing AC_i and contains a norm N^i of the form:
$$\left\langle Type, AC_i, \bigvee_{j=1,c} NC_j, \bigvee_{k=1,e} EC_k, NT \right\rangle.$$
3. Each node at level two, which is a child of N^i, and is labelled with a CG representing NC_j, contains a norm N^{ij} for $j = 1, \ldots, c$ of the form:
$$\left\langle Type, AC_i, NC_j, \bigvee_{k=1,e} EC_k, NT \right\rangle.$$
4. Each node at level three, which is a child of N^{ij}, and is labelled with a CG representing EC_k, contains a norm N^{ijk} for $k = 1, \ldots, e$ of the form:
$$\langle Type, AC_i, NC_j, EC_k, NT \rangle.$$

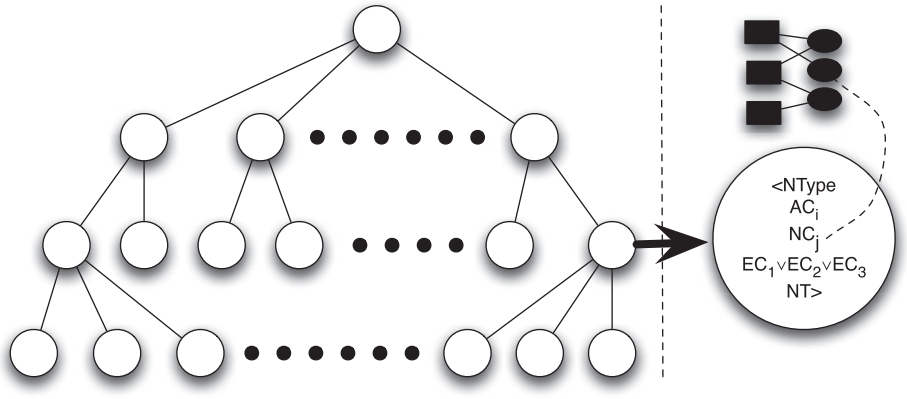

Fig. 6. A conceptual representation of a norm tree.

3.2. Modelling norms in the repair domain

Consider the norm of our car repair example, which obliges a repair shop to repair a car within seven days of its arrival. The left hand side of Fig. 7 illustrates the norm tree associated with this norm. For simplicity, we have ignored the norm target parameter, assuming that it is present in the root node. The dotted line between the nodes and CGs identifies which nodes are labelled with which CGs. It should be noted that the *function* relation, found in the right hand normative condition node, is used to compute whether the current time is greater than seven days from the time the car arrived for repair. This is used to simplify the CG shown in the figure; within a complete system, this CG would make use of an arithmetic function to add seven days to the car's arrival time, and then make use of an additional function or predicate to compare the current time to the deadline to determine whether the car has been repaired in time. Now again consider the nodes at the third level of the norm tree. These correspond to the norm condition and, when translated to first order logic, yield a formula of the form $repaired(Car) \vee function(CurrentTime, Time + 7\ days)$.

Note that there is a separation between the semantics of the normative model and its norms, and the semantics of the knowledge-based system. For a parameter (such as the normative condition) in the norm to evaluate to true, any of the disjunctions from which it is composed must evaluate to true (e.g. *repaired(Car)* in the above example). This aspect of a norm is captured by the normative model's semantics, and is thus represented by the norm tree structure. However, reasoning within the knowledge-based system is kept separate from the norm model semantics by means of conceptual graph annotations of the nodes in the normative tree. Thus, the knowledge-based system identifies *which* of the normative condition's disjunctions actually evaluated to true in the case where the normative condition is true. A user of the system could then be presented with the explanation of why the norm condition is valid: in the context of the repair shop norm, at least one node is satisfied (or both). While the figures in this paper are monochrome, colour can be added to a running system in order to identify the validity of a node (for example, red could mean invalid, while green could mean valid).

Finally, the right hand side of Fig. 7 illustrates the norm tree for the permission (to repair a car later than 7 days if there is a power failure) found in our example. Since no disjunctions exist within the activation, expiration and normative conditions, the norm tree has no branches.

This conceptual graph representation provides us with two advantages over a textual representation of the norm. First, the conceptual graph representation makes the types of concepts linked by predicates visually explicit (for example, *RepairShop:** as opposed to *X*). While this problem is easily addressed by manually changing the variable names of the textual logic representation (using *meaningful* literals), the heuristic employed could be confusing. Second, and more importantly, for *elaborated* pieces of knowledge (namely conjunctions with common variables) the translation between natural language and logical formulae becomes very difficult. For example, suppose that we are trying to represent the fact that a car arrives at a *Volvo* repair shop, that the repair shop accepts only cars of the same make, that the time at which the car arrives at the repair shop must be later than 9, and that this is the opening time of the repair shop. While the conceptual graph depiction is intuitive given its visual nature, the logic-based (textual) approach can be difficult to follow.

3.3. Instantiating norms

Now, consider the abstract norm illustrated on the left hand side of Fig. 7, and suppose that a new fact—that some car, c_1 arrived at the repair shop belonging to *bob* at time 12—is added to the knowledge base. In predicate form, we write $arrivesAtRepairShop(bob, c_1, 12)$. This piece of knowledge is *projected* to all the norm conditions in the system in the following way. Using projection, the fact is mapped onto the abstract norm of Fig. 7, and a new norm tree, with the appropriate CG nodes now labelled by constants, is created. This CG is shown in Fig. 8 in which, for clarity, nodes in the norm tree belonging to an abstract norm are depicted in white (c.f. Fig. 7).

It should be noted that there can be multiple instantiated versions of the same abstract norm simultaneously. However, each of these will have a different set of variable bindings, and thus a different CG associated with the norm tree.

3.4. Computing the status of norms

So far, we have shown how abstract and instantiated norms may be represented as norm trees, but we have not yet considered how to determine the status of a norm using our norm tree structure. Consider the left hand side of Fig. 8, which represents the instantiated norm from the repair shop example. The right hand side of the figure shows the CG representation of the environment,

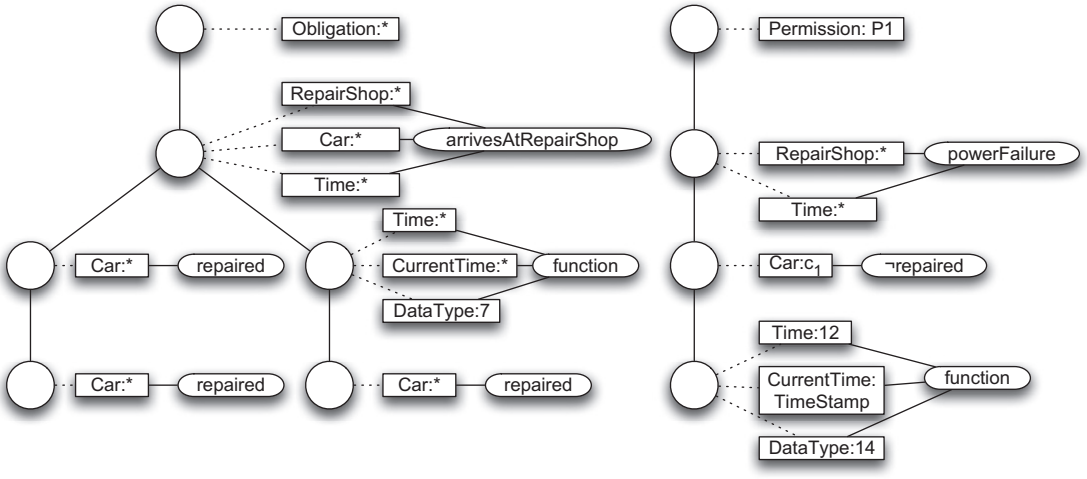

Fig. 7. The norm tree for the abstract obligation norm (left) and abstract permission norm (right) found in the repair shop example.

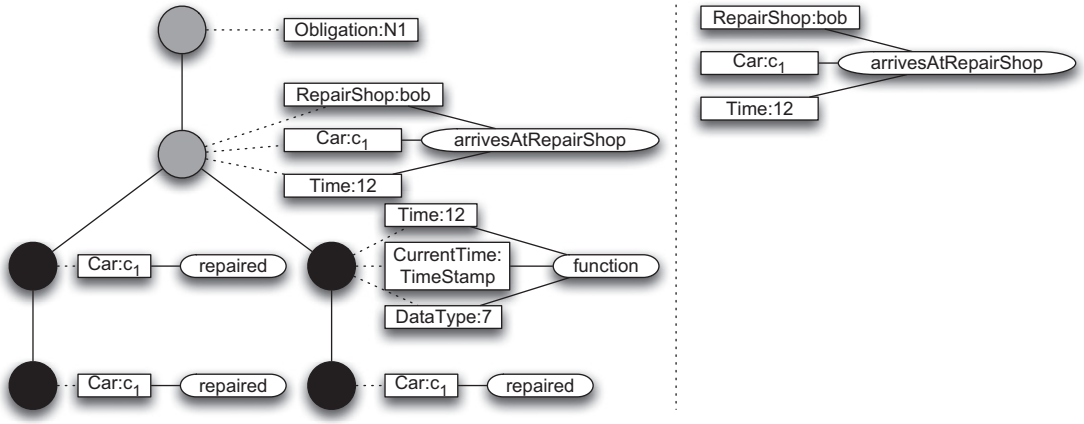

Fig. 8. An instantiated norm for the repair shop example.

as stored within the knowledge base. This CG represents the fact that a car c_1 arrived at the repair shop at time 12. To distinguish between abstract and instantiated norms, we colour the nodes of an instantiated norm using different colours (which are always non-white), as opposed to white abstract norms.

Now, as new facts appear and disappear within the knowledge base, the status of norms also changes. Determining this status may be achieved by checking for the existence of projections between the facts in the environment and the conceptual graph annotations of the norm tree. Fig. 9 illustrates the situation when an additional fact—namely that the current time is before time 19—is added to the environment (represented by the two CGs on the right of the figure). The norm tree on the left of Fig. 9 now contains a mixture of black and grey nodes. A grey node corresponds to the fact that the node is satisfied; that is, there is a projection between the environment and the corresponding CG annotation. The remaining nodes are black: they are not satisfied. Thus, in Fig. 9, illustrating the car repair example, there is no projection between the CG node representing the expiration condition, which states that the car is repaired, and the CG on the right of Fig. 9. Similarly, there is a projection (and thus the node is grey) between the CG on the right, and the CG linked to the node at the normative condition level stating that the current time is before 19 (the condition in this latter node is represented by the function taking in the datatype, time and current time). If, at some later point, the car is repaired, the black nodes within the norm tree will turn grey.

During its lifecycle, an abstract norm becomes instantiated. While instantiated, its normative condition may evaluate to true or false at different times. Eventually, the norm's expiration condition evaluates to true, after which the instantiated norm is deleted. We have already seen how one may determine whether a norm may be instantiated using a norm tree. A norm's normative condition is satisfied (that is, it evaluates to true), if any of the nodes at the norm condition level are grey. Similarly, a norm expires if any of the nodes at the expiration condition level are grey.

A norm's status includes whether it is activated or expiring, and whether it is being satisfied, and it is trivial to determine this from the norm tree. It is also possible to determine more sophisticated aspects providing a richer notion of the status of a norm from a norm tree. As an example, in the next section, we discuss how to determine whether an obligation has been *violated*. All aspects of the status of a norm can be computed by posing queries to the knowledge base, and thus, it is possible to *visually* determine the status of a norm.

4. Computing violation with permissions

One critical aspect of normative state that cannot be computed directly form a norm tree is whether the norm is violated. This is because of the way in which we treat permissions. In [2], Boella and van der Torre point out that permissions can be viewed as exceptions to obligations and prohibitions, and this is how

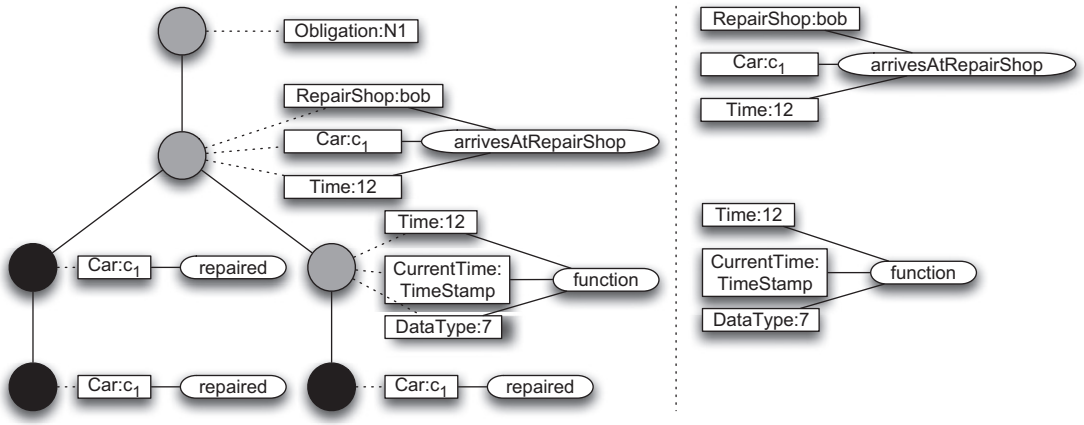

Fig. 9. A norm tree evaluated according to the knowledge base shown on the right.

permissions are handled by our model. Thus, for example, given an obligation on the repair shop to repair a car within 7 days, a permission to instead repair the car within 14 days *derogates* the obligation. While the obligation may not be complied with (because the car may not be repaired within 7 days), the repair shop will not be in violation of the obligation unless 14 days have expired.

Permissions thus do not exist in isolation, but instead act as exceptions to other types of norms. This means that in evaluating whether an obligation or prohibition is violated, one must consider not only the possibly violated norm itself, but also the permissions present. However, given a large normative system, identifying the appropriate permission that may prevent a violation from occurring can be challenging. Our visual approach can help overcome the cognitive load imposed by this problem by highlighting any relevant permissions that prevent a norm from being violated.

To illustrate, we return to our car repair example. If a power failure occurred at time 14, then the (instantiated) permission allowing Bob to repair car c_1 within 14 days (i.e. by day 28) is as follows:

$$\langle permission,\\ powerFailure(bob, 14),\\ \neg repaired(c_1),\\ currentTime(CurrentTime) \wedge before(CurrentTime, 28days),\\ repairShop(bob)\rangle.$$

Conceptually, in order to determine whether an instantiated and un-expired permission derogates an obligation or prohibition, we must check whether the permission's norm condition is *consistent* with the obligation. If it is not consistent, in the sense that the permission allows the negation of the obligation, then derogation takes place, otherwise the permission does not affect the obligation. In our example, $\neg repaired(c_1)$ is inconsistent when evaluated against $repaired(c_1)$, and the permission thus derogates the obligation. This check for consistency thus lies at the heart of our work.

Clearly, consistency checking requires the ability to represent and reason about the negation of a relation. However, the standard CG formalism is unable to represent such negated relations, and we make use of an extension to CGs first proposed by Mugnier and Leclère [15] to show how the consistency check can be performed from within the CG formalism. Mugnier and Leclère introduce the idea of a *negative relation node* which, when present as a node in a CG, identifies the fact that the named relation does not exist between the concepts incident on the node. Now, the approach we adopt here makes use of the closed world assumption, and extends a CG to include its negative relation nodes. More specifically, we add all possible negative relation nodes that do not make the graph contradictory to the CG's basic graph. Thus, for example, if we do not know that a car has been repaired, we now explicitly state that it has not been repaired; if a node $repaired(car_1)$ is not present in some CG, the *completed* form of the CG must include the node $\neg\ repaired(car_1)$.

Given this completed CG, if the permission's normative condition cannot be projected into the CG (because the car has in fact been repaired, for example), the permission derogates the obligation (or rather, that node in the norm tree for which the CG projection is unsuccessful, which will not be coloured black). The permission, and the relevant concepts and relations that derogate the permission, can then be displayed to the user to explain why the norm is not in violation. If, on the other hand, the permission is not relevant to the obligation, then a violation occurs, and the violated norm can again be highlighted in order to show the user its status. Thus, given a norm tree for an (instantiated, unexpired) obligation N, the norm it represents is violated if and only if all of its nodes at the normative condition level are coloured black.

Fig. 10 illustrates the derogation of an obligation by a permission. Dashed lines indicate links between the concepts and relations found in the two nodes, and the normative condition node marked with a grey node with a black centre in the obligation indicates that the node, while evaluating to false, is derogated by a permission. From the figure, it is clear that the obligation is not violated. Note that the permission's activation condition node is black. We assume that while a power failure occurred in the past (instantiating the permission), there is currently no power failure.

4.1. Case study

To illustrate the overall framework, we consider an additional scenario in which rapid response medical units must perform some duties when an emergency situation occurs. These units have the following obligation:

> "If a state of emergency has been declared, a rescue unit is obliged to travel to a casualty, and then collect them, or provide them with medicine until they have no more space and are out of medicines".

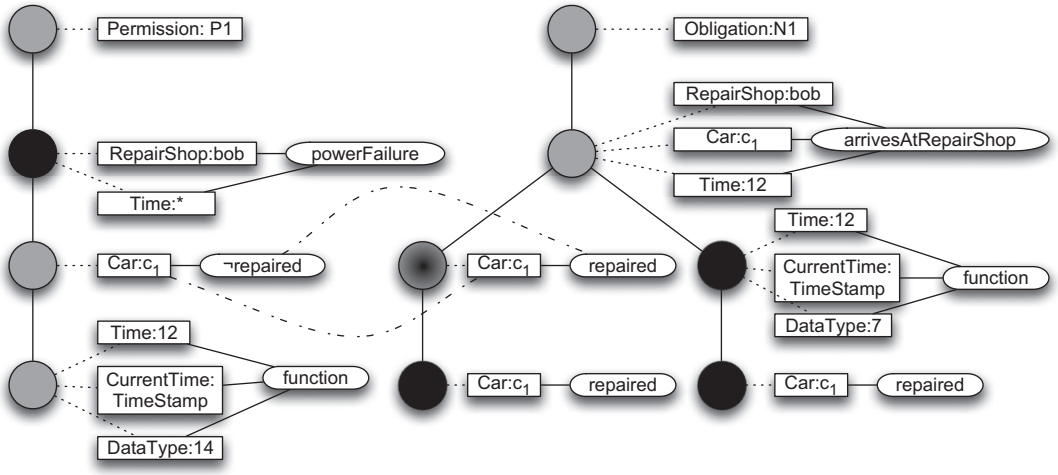

Fig. 10. A norm tree for a permission (left), and obligation (right) evaluated according to some knowledge, showing how the permission derogates the obligation.

Formally, this obligation is represented as follows:

$\langle obligation,$
$stateOfEmergency() \wedge casualty(C),$
$travel(U,C) \wedge (collect(U,C) \vee medicate(U,C)),$
$noSpace(U) \wedge noMedicine(U),$
$rescueUnit(U)\rangle.$

The disjunctive normal form of the obligation's normative condition is:

$(travel(U,C) \wedge collect(U,C)) \vee (travel(U,C) \wedge medicate(U,C)).$

Given this, we assume a very simple permission representing casualty triage: "If the casualty is dead, there is no need to medicate them". Formally, this is as follows:

$\langle permission, dead(C), \neg medicate(U,C), false, rescueUnit(U)\rangle.$

In order to construct the norm tree, we begin by identifying the concepts and relations found in this scenario, where the concepts include *StateOfEmergency*,*Casualty*,*RescueUnit* and *Dead*, and the relations include *travel*,*collect*,*medicate*,*noSpace* and *noMedicine*. These concepts and relations yield the support displayed in Fig. 11, and the abstract norms illustrated in Fig. 12.

Now, suppose that a state of emergency exists, and that a dead casualty $c1$ has been detected by a rescue unit $r1$. Furthermore, $r1$ has space and medicine available. Given that the rescue unit has not travelled to the casualty, collected it, or provided medicine, is it in violation of its obligation? In order to determine this, we must compute the completed CG of the instantiated obligation's normative condition. Fig. 13 shows the completed form of the graph for both the left and right hand branches of the instantiated *obligation* norm tree's normative condition nodes. The dotted lines within Fig. 13 illustrate that the *permission*'s normative condition projects into the obligation's right hand branch normative condition.

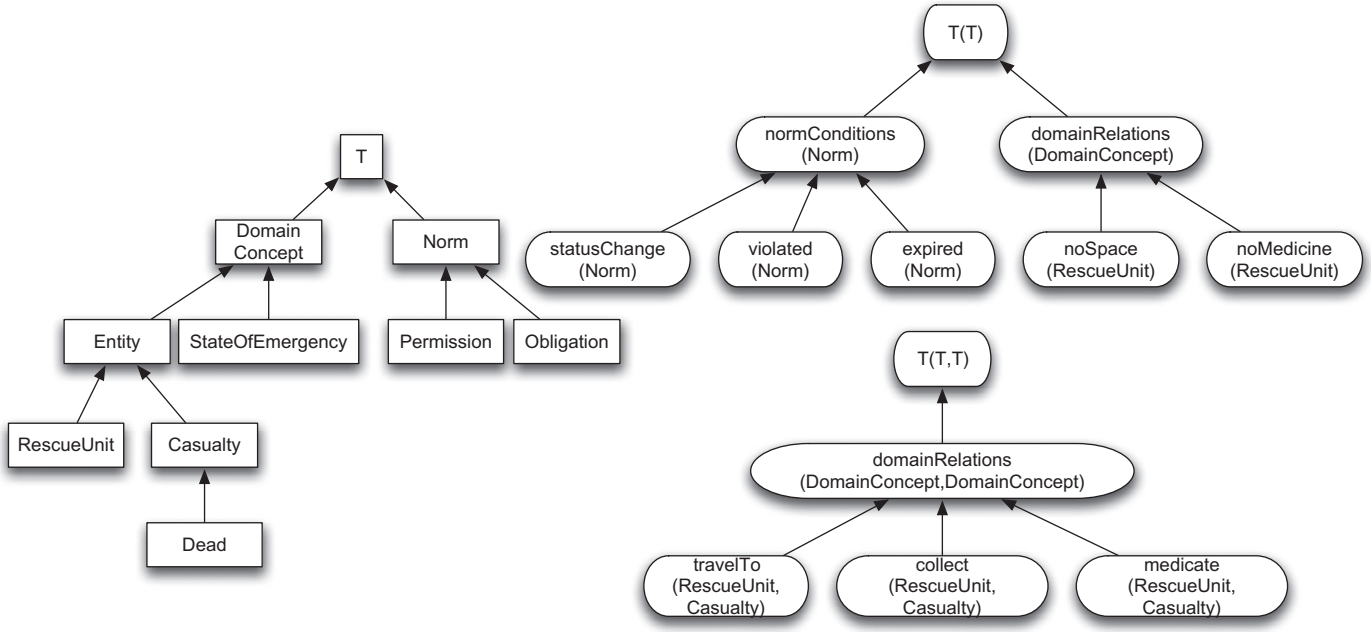

Fig. 11. The CG support composed of the concept hierarchy (left) and relation hierarchies (right).

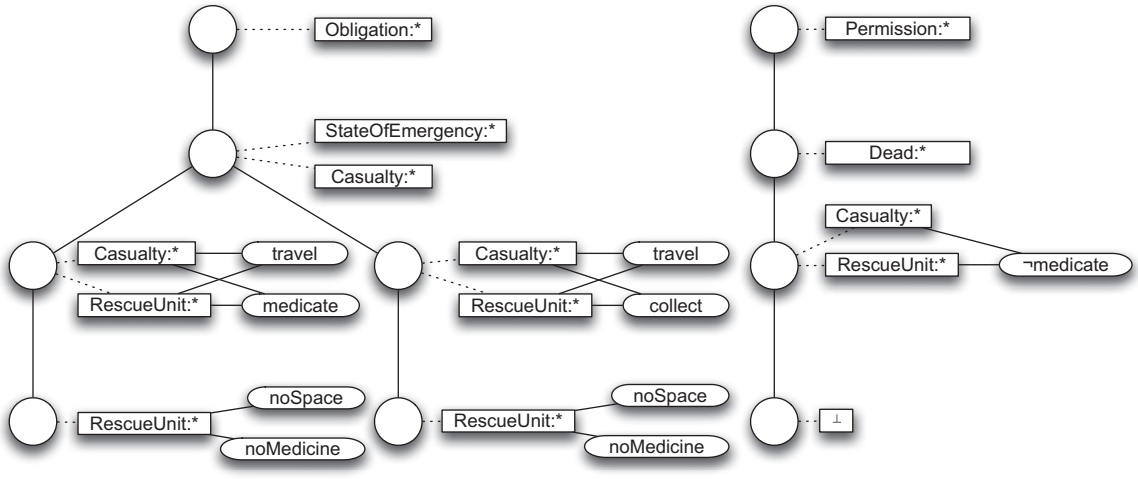

Fig. 12. The abstract norm trees.

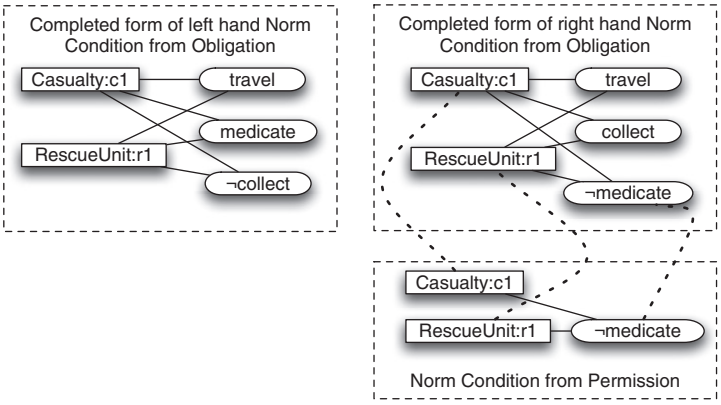

Fig. 13. The completed form of the obligation's normative condition (top left and top right), with the projection of the permission's normative condition.

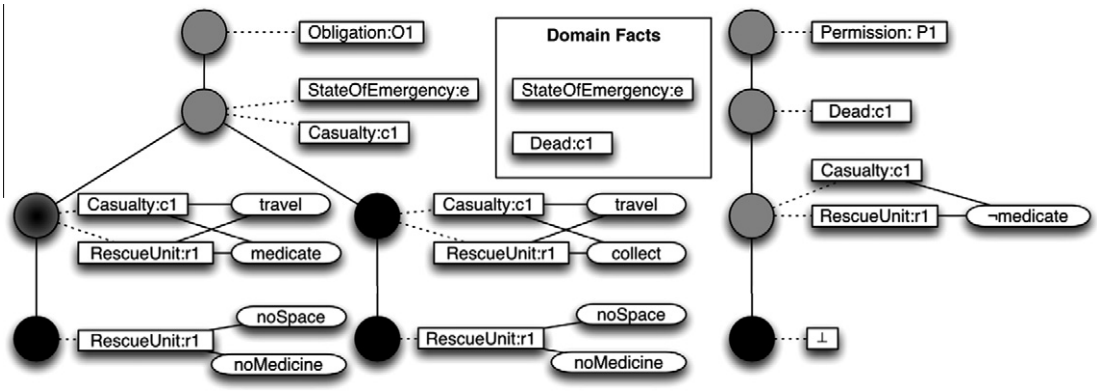

Fig. 14. Norm instantiation according to the domain facts.

However, no such projection is possible into the left hand branch. Therefore, the permission derogates the left hand branch of the obligation's norm condition, and the norm is not violated. This is shown in Fig. 14. In summary, our CG approach to norm explanation makes clear exactly how the permission enables a user to understand how the permission interacts with the obligation.

5. Discussion

5.1. Evaluation

Norms provide a means of regulating system behaviour, yet their structure and operation can often obscure the understanding that is possible, especially by end-users. In particular, it is important to understand not just the structure of norms, but also their status at different points in time, and the ways in which they interact. This latter aspects is critical, for the interaction between norms can affect their status. In order to provide an effective means for supporting user understanding, we have developed a visual model to explain the structure and status of a norm. This ability to provide explanations of a norm's status is especially useful; for example, complex contract disputes may require that some rewards or penalties be assigned by a human mediator, but in order to perform this assignment, the mediator must first understand which norms were violated, and which were complied with. Norm explanation is also important at the system design stage, where an understanding of norm status in different situations is needed to ensure correct system behaviour.

Previous work such as [5] has demonstrated that graphical systems excel in cases where non-technical users must be catered for, and this is exactly the approach we adopt. More specifically, we can identify the following benefits of our graphical normative representation. First, the graphical system can be used to identify which elements of the environment impact on a norm, even when making use of specialisation or generalisation of concepts or relations (as illustrated in Fig. 5), when it is not clear to a user how different concepts may relate to each other. In this way, users can directly track the effects of changes in the environment on a norm. Similarly, through the association of CGs with norms, it is possible to support navigation between norms sharing identical, specialised or generalised relations or nodes, or sharing markers. The set of norms affected by changes to the environment can thus be easily tracked.

Importantly, a graphical system is able to provide the user with an easily understandable snapshot regarding the status of the system. More specifically, by adopting an approach in which the colours associated with the nodes of an instantiated norm's tree indicate their status, we provide a means for users to quickly identify which norms have what status, and why (as illustrated in Fig. 9).

Finally, and as indicated above, the interactions between different parts of a system can be made explicit. Since individual norms can combine in complex ways to give sophisticated structures by virtue of the links between permissions and obligations, for example, providing a visual representation can be argued to be vital to ensure clarity of presentation and understanding. Indeed, identifying obligations that are derogated due to permissions, and in turn identifying these permissions is not trivial, yet as we have shown in Section 4 (and Fig. 10), this becomes relatively straightforward with an appropriate representation.

All these aspects can be seen directly from the work presented in this paper. Clearly however, while this evaluation of our contribution is justified in its own terms, and demonstrates the validity of our approach in providing explanation, the claim of aiding users requires a more substantial (and more challenging) evaluation. In particular, since one of the core advantages of the graphical approach lies in enhancing user *understanding*, the next step in evaluating our framework must be to undertake user studies, comparing the graphical approach and standard, text-based techniques for representing norms, and their impact on and value to users. Current work is concerned with implementation of a software tool for exactly this purpose, providing clear visualisations of the status of norms as described earlier by displaying the norm trees found in a running system, colouring tree nodes as appropriate, displaying the node CG graphs (and enabling further analysis to identify and display projections, graph support and the like). More interesting and valuable functionalities are also anticipated, for example, if an obligation is derogated, selecting an appropriate node will allow a user to visualise the associated derogating permission, and *vice versa*.

Although anecdotal evidence from early trials with users already suggests that our approach has significant merit, this more substantial user evaluation will require a methodical effort with multiple users across different user categories (for example, expert users with an understanding of logic, non-expert users with less formal modelling experience, and the like). Clearly, this is a major undertaking that is beyond the scope of the current work, yet this will be important before an enhanced appreciation of the value of the graphical norm explanation approach can be established.

5.2. Related work

Much of the existing work on norms and normative reasoning originated from the philosophical domain. While recognising the conditional nature of norms, such work emphasised problems such as identifying what state of affairs should hold, or how to resolve normative conflict. However, apart from the work of Governatori et al. [9], few have considered how a normative system evolves when norms are fulfilled. Governatori et al. adopt a defeasible logic based approach to norm representation, with norms expiring when a *defeater* to them is introduced. Within a long lived system, this approach is cumbersome; reinstantiating a norm requires the introduction of a defeater to the defeater. In contrast, the framework presented in this paper is intended to capture the evolution of a norm over time, allowing for its instantiation and expiration, as well as recording the time periods during which a norm was complied with or violated. Since the internal structure of such a norm is somewhat complex, some technique for explaining why a norm is in a certain state is required, and we proposed a visual model for explaining this status of a norm. This ability to provide explanations of a norm's status in such domains is particularly useful; for example, complex contract disputes may require that some rewards or penalties be assigned by a human mediator, but in order to perform this assignment, the mediator must first understand which norms were violated, and which were complied with. Norm explanation is also important at the system design stage, where an understanding of norm status in different situations is needed to ensure correct system behaviour.

As described in Section 3, instantiated norms are created by copying abstract norms and modifying the labels within the norm's basic graph. Recent work on CGs [21] has examined the possibility of adding a special *evolves into* relation to capture the notion of transformation over time, and it is tempting to utilise this relation to formally represent the instantiation of a norm. However, this relation is currently only useful when the objects being represented will transform into the evolved object in a predictable manner, and can therefore not be directly applied to our work. Nevertheless, identifying a more formal approach to creating instantiated norms from abstract norms is worth pursuing, as this would allow us to answer questions about possible norm instantiations.

Our graphical representation highlights the link between permissions and obligations, and borrows some ideas from [7], wherein CGs were used to express and manage the interdependencies between security policy rules. Since norms can be used to express such rules [12], many issues identified there (such as the detection of redundant policies) map directly to the domain of norms.

More generally, however, we are aware of very little work dealing with the explanation of norms to users. This may be due to an implicit assumption that normative systems are fully automated, and that explanation is thus not necessary, or perhaps due to an assumption regarding the technical expertise of a system's users. However, even if a user is able to understand a norm representation, graphical explanations may still be advantageous when reasoning about complex interactions between large groups of norms. One exception to this is the recent work of Miles et al. [14], which touches on the concept of norm explanation. Here, a causal graph is used to analyse and explain norm violation, and then to identify whether there were mitigating circumstances for the violation.

6. Conclusions and future work

Norms have a complex lifecycle, becoming instantiated, and thus placing an expectation on an agent's behaviour at certain points in time, following which they may expire and cease to influence an agent. Within a long lived system, norms may be instantiated and expire multiple times; at any point in time, only a certain subset of norms may be relevant to identifying what behaviour should take place. Furthermore, examining a single norm in isolation does not provide enough information to determine whether an agent is acting in compliance with the norm. For example, as shown in Section 4, permissions may derogate norms, and multiple norms must be considered when reasoning about their effects. Critically, while existing norm representations are sufficient for automated reasoning, their form is not ideal for explaining the behaviour of the system to end-users. In order to provide a user with an effective understanding of a normative system, all of these issues must be taken into consideration.

The goal of our approach is to provide an effective tool for system understanding to end-users. Our underlying norm formalism is able to model the norm's lifecycle, while our conceptual graph based representation enables a user to consider the interactions between obligations and permissions, and understand them in an intuitive manner. Many avenues remain open for future investigation. Other studies have shown that graphical representations are more easily understood than logic-based ones [5] by non-experts, and though we have proceed on this legitimate assumption, we have yet to undertake the user studies that will confirm this empirically, but aim to do so in the short term. We also intend to leverage the formal power of our model, by investigating the use of graph theoretical operations to identify redundant norms [2]. Similarly, we believe that graph-based operations can be used to detect, and help resolve, normative conflict. Both of these applications effectively validate the structure of the norms, and we thus aim to apply existing work on CG validation [8] to aid us in this task. Furthermore, projection can also act as a *similarity* measure, and can thus be applied to determining the trustworthiness of contracts (as encoded by groups of norms) along the lines suggested by Groth et al. [10]. Finally, we have focused on the status of norms

at a single point in time; we plan to investigate how our approach can aid in explaining interactions not only between simultaneously active norms, but also how they can be used to identify and explain temporally distributed normative interactions.

References

[1] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook, Cambridge University Press, 2003.

[2] G. Boella, L. van der Torre, Permissions and obligations in hierarchical normative systems, in: Proceedings of the Nineth International Conference on Artificial Intelligence and Law (ICAIL-03), ACM, New York, NY, USA, 2003, pp. 109–118.

[3] G. Boella, L. van der Torre, Institutions with a hierarchy of authorities in distributed dynamic environments, Artificial Intelligence and Law 16 (2008) 53–71.

[4] W. Briggs, D. Cook, Flexible social laws, in: C. Mellish (Ed.), Proceedings of the 14th International Joint Conference on Artificial Intelligence, Morgan Kaufman, San Francisco, 1995, pp. 688–693.

[5] M. Chein, M. Mugnier, Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs, Springer, 2009.

[6] M. Croitoru, N. Oren, S. Miles, M. Luck, Graph-based norm explanation, in: M. Bramer, M. Petridis, A. Hopgood (Eds.), Research and Development in Intelligent Systems XXVII, Proceedings of AI-2010: The Thirtieth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, pp. 35–48.

[7] M. Croitoru, L. Xiao, D. Dupplaw, P. Lewis, Expressive security policy rules using layered conceptual graphs, Knowledge Based Systems 21 (2008) 209–216.

[8] J. Dibie-Barthélemy, O. Haemmerlé, E. Salvat, A semantic validation of conceptual graphs, Knowledge-Based Systems 19 (2006) 498–510.

[9] G. Governatori, J. Hulstijn, R. Riveret, A. Rotolo, Characterising deadlines in temporal modal defeasible logic, in: Proceedings of the 28th International Conference on Artificial Intelligence (AI-2007), Lecture Notes in Artificial Intelligence, vol. 4830, pp. 486–496.

[10] P. Groth, S. Miles, S. Modgil, N. Oren, M. Luck, Y. Gil, Determining the trustworthiness of new electronic contracts, in: Proceedings of the 10th Annual International Workshop on Engineering Societies in the Agents' World (ESAW 2009), Springer, 2009, pp. 132–147.

[11] R.A. Kowalski, M.J. Sergot, A logic-based calculus of events, New Generation Computing 4 (1986) 67–95.

[12] C. Krogh, The rights of agents, in: M. Wooldridge, J.P. Müller, M. Tambe (Eds.), Proceedings of the IJCAI Workshop on Intelligent Agents II: Agent Theories, Architectures, and Languages, Lecture Notes in Computer Science, vol. 1037, Springer-Verlag: Heidelberg, Germany, 1996, pp. 1–16.

[13] P. McNamara, Deontic logic, in: E.N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, 2010, Fall 2010 edition.

[14] S. Miles, P. Groth, M. Luck, Handling mitigating circumstances for electronic contracts, in: Proceedings of the AISB 2008 Symposium on Behaviour Regulation in Multi-agent Systems, pp. 37–42.

[15] M.L. Mugnier, M. Leclère, On querying simple conceptual graphs with negation, Data Knowledge Engineering 60 (2007) 468–493.

[16] N. Oren, M. Croitoru, S. Miles, M. Luck, Understanding permissions through graphical norms, in: J. Leite, P. Torroni, T. Agotnes, G. Boella, L. van der Torre (Eds.), Declarative Agent Languages and Technologies VIII, 8th International Workshop, DALT 2010, Toronto, Canada, May 10, 2010, Revised, Selected and Invited Papers, Lecture Notes in Computer Science, vol. 6814, Springer, 2011, pp. 167–184.

[17] N. Oren, S. Panagiotidi, J. Vazquez-Salceda, S. Modgil, M. Luck, S. Miles, Towards a formalisation of electronic contracting environments, in: J.F. Hubner, E.T. Matson, O. Boissier, V. Dignum (Eds.), Coordination, Organizations, Institutions and Norms in Agent Systems IV, COIN@AAMAS 2008/COIN@AAAI 2008, Lecture Notes in Artificial Intelligence, 5428, Springer, 2008, pp. 156–171.

[18] Y. Shoham, M. Tennenholtz, On social laws for artificial agent societies: Off-line design, Artificial Intelligence 73 (1995) 231–252.

[19] J.F. Sowa, Conceptual graphs, IBM Journal of Research and Development 20 (1976) 336–375.

[20] J.F. Sowa, Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984.

[21] R. Thomopoulos, J.R. Bourguet, B. Cuq, A. Ndiaye, Short communication: answering queries that may have results in the future: a case study in food science, Knowledge-Based Systems 23 (2010) 491–495.

[22] W. Woods, J. Schmolze, The KL-ONE family, Computers and Mathematics with Applications 23 (1992) 133–177.

[23] G.H. von Wright, Deontic logic, Mind 60 (1951) 1–15.