Graph-based Norm Explanation

Madalina Croitoru and Nir Oren and Simon Miles and Michael Luck

Abstract Norms impose obligations, permissions and prohibitions on individual agents operating as part of an organisation. Typically, the purpose of such norms is to ensure that an organisation acts in some socially (or mutually) beneficial manner, possibly at the expense of individual agent utility. In this context, agents are *norm-aware* if they are able to reason about which norms are applicable to them, and to decide whether to comply with or ignore them. While much work has focused on the creation of norm-aware agents, much less has been concerned with aiding system designers in understanding the *effects* of norms on a system. The ability to understand such norm effects can aid the designer in avoiding incorrect norm specification, eliminating redundant norms and reducing normative conflict. In this paper, we address the problem of norm understanding by providing *explanations* as to why a norm is applicable, violated, or in some other state. We make use of conceptual graph based semantics to provide a graphical representation of the norms within a system. Given knowledge of the current and historical state of the system, such a representation allows for explanation of the state of norms, showing for example *why* they may have been activated or violated.

Madalina Croitoru
LIRMM, University Montpellier II, France, e-mail: croitoru@lirmm.fr

Nir Oren
Dept. of Computer Science, University of Aberdeen, UK e-mail: n.oren@abdn.ac.uk

Simon Miles
Dept. of Informatics, King's College London, UK e-mail: simon.miles@kcl.ac.uk

Michael Luck
Dept. of Informatics, King's College London, UK e-mail: michael.luck@kcl.ac.uk

1 Introduction

Norm-aware agents make use of concepts such as obligations, permissions, and prohibitions, to represent and reason about socially imposed goals and capabilities. Such agents are able to decide whether to act in a manner consistent with norms, or whether to ignore them. Typically, norms are imposed on a set of agents in order to increase the overall utility of a system or society (often at the cost of individual utility)[9], or to reduce computational or communication overhead [4].

While a norm-aware agent is able to reason about which norms are applicable to it, or to another agent given a particular context, the problem of *explaining* why a norm is applicable, or violated, or in some other similar state, has not been investigated in depth. Yet the ability to provide such an explanation has multiple benefits. For example, system designers would be better able to understand the interactions between different norms, allowing them to avoid creating redundant norms [3], and to specify their norms more precisely. Conversely, users would be able to elicit a more intuitive understanding of the operation of a system by establishing the reasons why certain norms were assigned a particular status in response to system events.

Norms are typically specified within some knowledge-based system using a logic which, for non-technical users, is often difficult to understand. Such knowledge-based systems (KBS) are designed in order to represent knowledge (within a knowledge base) in a such a way that reasoning can be performed over it. In turn, a knowledge base is built on top of some set of *ontologies*. From an epistemological viewpoint, an ontology answers the question "what kinds of things exist in the application domain?" For our normative framework we consider computational ontologies, which provide a symbolic representation of classes of objects, called *concepts*, as well as the possible relationships between objects, called *relations* or *roles*. All other pieces of knowledge in the KBS are expressed by structures built with the ontology terms (concepts and relations). For a KBS to be usable, it is essential that the user understands and controls not only the knowledge base construction process, but also how results are obtained from the running system. It should be easy for this user 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. The last point, namely the ability to understand why the system gives a certain answer, is especially important since the user's computing expertise may vary. Given the difficulty non-specialists have in understanding formal textual explanations of the logical inference process, we propose a graphical norm representation, based on conceptual graphs [10], which have a sound and complete semantics with respect to a subset of first order logic [5]. The benefits of using graphs for representing knowledge stem from the following:

- First, graphs are simple mathematical objects (they only use elementary naive set theory notions such as elements, sets and relations) which have graphical representations (sets of points and lines connecting some pairs of points) and thus can be visualised.

- Second, graphs can be equipped with a logical semantics: the graph-based mechanisms they are provided with are sound and complete with respect to deduction in the assigned logic.

Our goal in this paper is to provide a graph-based semantics to the normative framework found in [8]. This normative framework was designed with a number of purposes in mind, namely to allow for the monitoring of the changing status of norms, and to support agent reasoning regarding norms. The semantics we describe allows one to graphically represent the changes in norms, and to determine their status using graph based operations such as projection. Thus, we are able to provide a visual explanation of certain aspects of normative reasoning.

2 Background

2.1 The Normative Model

Due to space constraints, we do not provide a complete formal description of the normative model. Instead, we describe the model by examining how it may be applied to a small example. Consider a situation where an agent *Alice* takes her car (an *Audi*) 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.

The repair shop 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).

Given this example, we observe that a norm may be defined in terms of five components. First, a norm has a *type*, for example, 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 agent; if this normative condition does not hold, the norm is not being followed (i.e. in the case of our obligation, it is *violated*). Fourth, norms have a termination, or *expiration condition*, identifying the situations after which the norm no longer affects the agent. Finally, the norm must identify those agents to which it applies, known as the *norm targets*.

Note that the requirement on the repair shop to repair 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*. Then, when a customer brings in a car, the norm is instantiated, imposing a normative requirement upon 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.

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 $\mathscr{L}$, such as first order logic. A norm is then a tuple of the form:

$$\begin{aligned}&\langle NormType,\\ &NormActivation,\\ &NormCondition,\\ &NormExpiration,\\ &NormTarget\rangle\end{aligned}$$

where

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

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]:

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

The predicate labels in this example refer to both events and states. This was done for ease of presentation; the use of a more complex underlying language would disambiguate these concepts, and provide us with a richer typology of temporal concepts.

In [8], a logical semantics for the instantiation and processing of norms represented using the tuple representation of norms is described. The tuple's attributes map directly onto the five components of a norm detailed above.

Temporal notions play a major role in this model of norms. A norm is instantiated at some time when the norm's activation conditions hold. The instantiated norm then persists, regardless of the valuation of the activation condition, until the norm's expiration conditions hold. Finally, we identify agents by constants of $\mathscr{L}$. Therefore, if we assume that some car car_1 arrives at Bob's repair shop at time 12, we would instantiate the abstract norm and obtain the following instantiated norm:

$$\langle obligation,$$

[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.

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

One issue we have not yet addressed is where, conceptually, norms are stored. It is clear that norms are not predicates (though they may be represented as such). We thus assume the existence of a separate *normative environment*, which is used to keep track of the abstract and instantiated norms within the system. Since norms may be instantiated and expire as time passes, the normative environment must, at each time point, identify which norms exist. One possible implementation of the normative environment is described in [8].

One of the main purposes of this normative model is to identify the changing status of norms over time. A norm's status may include the fact that it is instantiated or abstract, whether it is being complied with or violated, and whether it has expired. This status may be referred to by other norms. For example, a norm stating that "if a car is in the shop, and must be repaired within 7 days, and seven days have not yet passed, it is possible to request an extension for this repair work" could be written as follows (given that the norm above is labelled $n1$ and that the action of requesting a delay is written using the *requestDelay* predicate):

$$\begin{aligned} &\langle permission,\\ &active(n1) \wedge \neg violated(n1),\\ &requestDelay(X, Car),\\ &expired(n1) \vee violated(n1),\\ &repairShop(X)\rangle \end{aligned}$$

The $violated(n1)$ predicate makes use of 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. The $active(n1)$ predicate returns true if norm $n1$ is active, and the $expired(n1)$ predicate returns true if $n1$ has expired. These, and other such predicates are formally defined in [8].

As seen in the norm above, norms can often refer to other norms and the variables found within them (e.g. *Car* in the example above). Determining the status of a norm thus requires examining the interactions between multiple norms, and given a system containing many norms, it can be difficult for a user or designer to identify why some norm is assigned a certain state. A graphical model for norms would allow for such links to be made explicit. More generally, humans are able to assimilate large amounts of graphical information, and thus, by modelling norms graphically, the norm system can be more easily understood. Our chosen graphical formalism is based on conceptual graphs, due to their well understood nature and

formal semantics. Having provided an overview of our normative model, we now proceed to describe conceptual graphs in more detail.

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. They all share the basic idea of representing domain knowledge using a graph, but there are differences concerning notation, as well as representational and reasoning power supported by the language. In semantic networks, *diagrammatical reasoning* is mainly based on path construction in the network. We can distinguish two major families of logically founded languages born from semantic networks: KL-ONE and conceptual graphs. KL-ONE [12] is considered to be the ancestor of description logics (DLs) ([1]), which form the most prominent family of knowledge representation languages dedicated to reasoning on ontologies. However, DLs have lost their graphical origins. In contrast, conceptual graphs (CGs) were introduced by Sowa (cf. [10, 11]) as a diagrammatic system of logic with the purpose "to express meaning in a form that is logically precise, humanly readable, and computationally tractable" (cf. [11]). Throughout the remainder of this paper we use the term *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]).

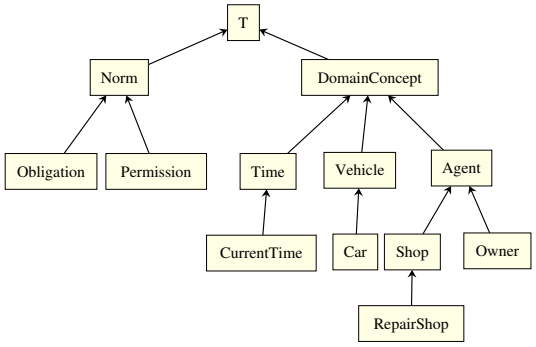

Fig. 1 Conceptual graph support: the concept hierarchy.

Within the conceptual graph approach, all types of knowledge are encoded as graphs and can thus be visualised in a natural way. A CG partitions knowledge into two types. The first type identifies the CG's *vocabulary* and can be seen as a basic ontology. This vocabulary is composed of hierarchies of concepts and relations, which are referred to as the CG's *support*. Since both the concept support and relation support are partial orders, they can be visualised by their Hasse diagram. The

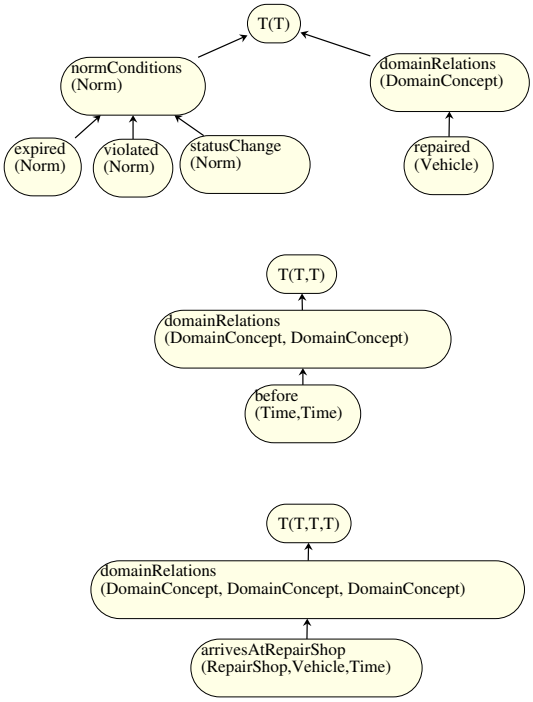

Fig. 2 Conceptual graph support: the relation hierarchy.

partial order represents a specialisation relation. If $t' \leq t$ within the partial order, then t' should be interpreted as a specialisation of t. Figures 1 and 2 respectively illustrate the concept and relation hierarchies that are used to represent norms and domain concepts in the repair shop example throughout this paper.

The second type of knowledge encoded within a conceptual graph is based on the representation of entities and their relationships, encoded by a labelled graph with two kinds of nodes (corresponding to entities and relations). Edges link an entity node to a relation node, and such nodes are labelled by types of the vocabulary. Concept nodes are normally drawn as rectangles and relation nodes as ovals, while the edges incidental to a k-ary relation node are numbered from 1 to k. Figure 3 presents an example of this type of graph, which encodes the fact that a car arrived at the *repairShop* at some time. This second type of graph is called a *basic graph* (abbreviated BG) in the CG literature.

Having described the (graphical) syntax of a CG, we now proceed to look at its semantics. These semantics are in fact defined using first order logic, as defined by a mapping classically denoted by Φ in the conceptual graphs literature [11].

More specifically, let G and H be two 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, which preserves edges and may decrease concept and relation labels, that is: *(i)* for any edge labelled i between the

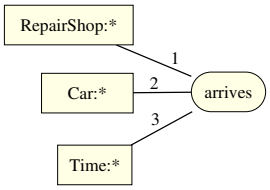

Fig. 3 A generic basic conceptual graph fact.

nodes c and r in G, there is an edge labelled i between the nodes $\pi(c)$ and $\pi(r)$ in H; *(ii)* for any (concept or relation) node x in G, the label of its image $\pi(x)$ in H is less than or equal to the label of x.

The fundamental 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, i.e. $\Phi(\mathscr{V}), \Phi(H) \models \Phi(G)$ (i.e., this is a soundness and completeness theorem of BG homomorphism with respect to first order logic entailment). It should be noted that BGs are in fact equivalent to the positive, conjunctive and existential fragment of first order logic.

Having described our the normative model and introduced conceptual graphs, we now 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

We will represent both abstract and instantiated norms using a tree structure, referred to as the *norm tree*. The root of the node tree represents the entire norm (by capturing its type and target), while lower levels of the tree represent different portions of the norm. Nodes in the second level of the norm tree are associated with the activation condition, while nodes in the third level are associated with the normative condition, and in the fourth level with the expiration condition. Figure 4 depicts a norm tree. Each of the nodes in the norm tree has associated a conceptual graph representation of their content.

The nodes of the norm tree are used to represent disjunctive conditions of the appropriate portion of the norm. More formally, we assume that the norm target parameter consists of a conjunctive combination of predicates, and that all other parameters (except for norm type) may contain disjunctions. In order to map a norm into a norm tree, we represent the norm using the disjunctive normal form of its elements, i.e. a norm tuple $\langle Type, AC, NC, EC, NT \rangle$ can be rewritten as

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

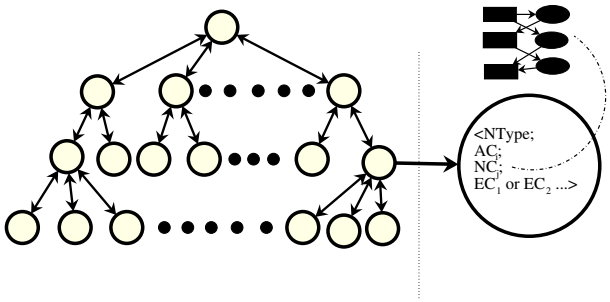

Fig. 4 A conceptual representation of a norm tree

Where $Type$, AC_i,NC_j, EC_k and NT are all conjunctive positive existential first order logic formulae. It should be noted that this requires the assumption of negation as failure. We may then represent each of these formulae as a conceptual graph, defined on some given support (i.e. domain ontology).

Then a norm tree T_{n_1} for norm n_1 is defined as follows:

1. The root of the tree is a node containing n_1. The node is labelled with $Type$ and NT.
2. Each child of the root (i.e. each node at level one) contains a norm n_1^i for node $i = 1 \ldots a$ of the form

$$\langle Type, AC_i, \bigvee_{j=1,c} NC_j, \bigvee_{k=1,e} EC_k, NT \rangle$$

   The node containing n_1^i is labelled with AC_i.
3. Each node at level two which is a child of n_1^i contains a norm n_1^{ij} for $j = 1 \ldots c$ of the form

$$\langle Type, AC_i, NC_j, \bigvee_{k=1,e} EC_k, NT \rangle$$

   The node containing n_1^{ij} is labelled with NC_j.
4. Each node at level three which is a child of n_1^{ij} contains a norm n_1^{ijk} for $k = 1 \ldots e$ of the form

$$\langle Type, AC_i, NC_j, EC_k, NT \rangle$$

   The node containing n_1^{ijk} is labelled with EC_k.

Each node in the tree is associated with a conceptual graph.

Let us consider the norm presented in Section 2.1 stating that a repair shop has an obligation imposed upon it, to repair a car within seven days of its arrival. Figure 5 illustrates the simplified norm tree[2] that is associated with this norm. The top node

[2] In the remainder of this paper, and unless otherwise stated, we ignore the norm target parameter, assuming it is present in the root node.

represents the type of the norm. The second level of the tree depicts the norm's activation condition, while the third level represents the normative condition.

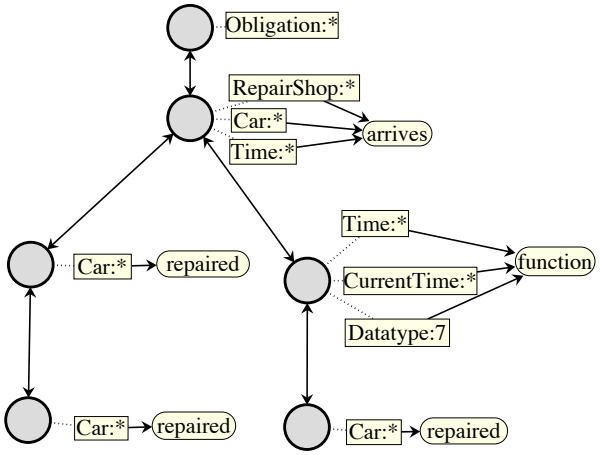

Fig. 5 The Norm Tree for the repairshop example norm.

There is a separation between the semantics of the normative model and its norms, and those of the knowledge base 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 base system is kept separate from these norm model semantics by means of conceptual graph annotations of the nodes in the normative tree. Thus, the knowledge base system will identify *which* of the normative conditions disjunctions actually evaluated to true in the case where the normative condition is true. This is done by means of colouring; black nodes indicate unsatisfied conditions, while grey nodes indicate satisfied normative conditions. If at least one node is satisfied at some level of the norm tree, that condition is deemed to have been satisfied (thus, for example, if at least one node at the third level of the tree is not black, the normative condition for the norm is met).

Finally the last level of the tree depicts the expiration condition. It is assumed that the norm target attribute is used to retrieve the literal corresponding to the agent name upon whom the norm is imposed, and we thus assume it forms part of the tree's root node. However, future work will look into retrieving agents by their type (e.g. which obligations are imposed on agents of type *shop*, which may include more specific agents of type *repairShop*, *groceryStore* and so on). In this case the graph based representation of the support will also provide useful feedback to the user (for example, identifying that an agent was selected as its type is a descendant of the *repairShop* node in the ontology).

The conceptual graph representation provides us with two advantages over a textual representation of the norm. First, the conceptual graph representation makes visually explicit the types of the concepts linked up by predicates (*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 (e.g. a variable label such as *RepairShopNumber* could imply a certain ordering of the variables etc.). 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 let us assume we are trying to represent the fact that a car arrives at a repair shop that accepts only Volvos, and the time when the car arrives at the repair shop has to be later than 9 (the shop's opening time). This norm requires reasoning about different ontological levels as well as the logical formulae representing the norms, and a textual logic based representation of this norm can be difficult to follow. The conceptual graph depiction of this type of norm is visual, and thus more intuitive.

3.2 Instantiating Norms

Figure 5 represents an abstract norm. Now assume that a number of new facts are added to the knowledge base, namely that some car, c_1 arrived at Bob's repair shop at time 12. In predicate form, we write $arrivesAtRepairShop(bob, c_1, 12)$.

This piece of knowledge will be projected to all the norm conditions in the system. The mapping will instantiate a number of generic nodes in the conceptual graphs annotating the norm tree nodes. In this way we obtain the instantiated norm shown in Figure 6. For clarity, we differentiate between norm trees for instantiated and abstract norms by colouring the nodes of the latter in white, and of the former in grey or black (depending on whether they are satisfied or not).

3.3 Computing the Status of Norms

So far, we have shown how abstract and instantiated norms may be represented as norm trees. The main focus of the framework presented in [8] revolved around norm status monitoring (i.e. identifying when a norm has a specific status such as complied with or expired), and we now discuss how a norm's status may be identified using the norm tree structure.

As new facts appear and disappear within the knowledge base, the status of norms will change. Computing these statuses is done by checking for the existence of projections between the facts in the environment and the conceptual graph annotations of the norm tree. The norm tree on the left of Figure 7 contains a mixture of black and white nodes. The white node corresponds to the fact that the node is satisfied,

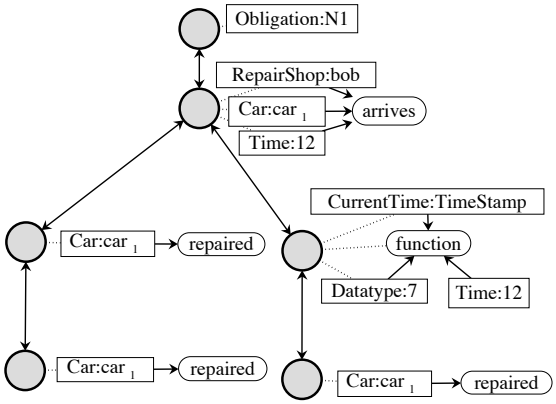

Fig. 6 An instantiated norm for the repairshop example.

e.g. there is a projection between the environment (on the right hand side) and the corresponding CG annotation. The other nodes are black: they are not satisfied. Thus, for 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 Figure 7. Similarly, there is a projection (and thus the node is white) between the CG on the right, and the CG captured by 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 white.

During its lifecycle, an abstract norm becomes instantiated. While instantiated, its normative condition may evaluate to true or false at different times. Finally, 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 met, i.e. evaluates to true, if any of the nodes at the norm condition level are white. Similarly, a norm expires if any of the nodes at the expiration condition level are white.

A norm's status includes whether it is activated or expiring, and whether it is being met, and it is easy to determine this from the norm tree. It is also possible to identify more complex norm statuses. For example, a norm is said to be *violated* if it is an obligation which has been instantiated, and whose normative condition evaluates to false. It is possible to construct this condition as a query to the knowledge base, and from this, visually determine whether the norm is violated or not.

4 Discussion

Much of the existing work on norms and normative reasoning originated from the philosophical domain. Such work, while recognising the conditional nature of

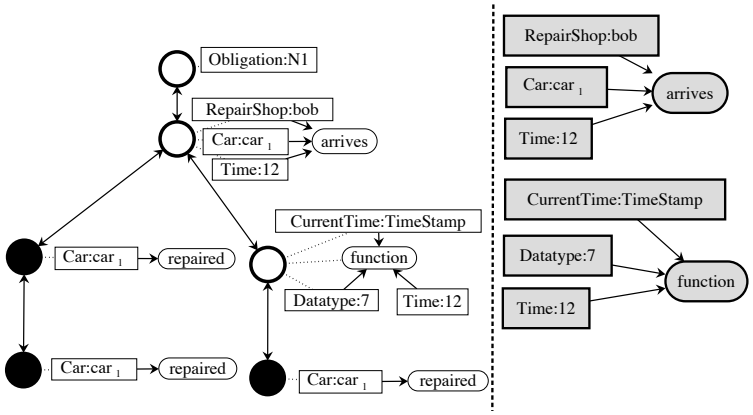

Fig. 7 A norm tree whose nodes are evaluated according to the knowledge base shown on the right.

norms, emphasised problems such as identifying what state of affairs should hold, or how to resolve normative conflict. However, apart from the work of Governatori [6], few have considered how a normative system evolves when norms are fulfilled. Governatori adopts 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. 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 the status of a norm. The ability to provide explanations for 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 statuses in different situations is needed to ensure correct system behaviour.

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 a presumption regarding the technical expertise of the system's users. However, even if a user is able to understand a norm representation, when reasoning about complex interactions between large groups of norms, graphical explanations may be advantages. The work described in [7] touches on the concept of norm explanation. Here, norm violation is analysed and explained by means of a causal graph. The causal graph was then further processed to identify whether mitigating circumstances existed for the norm's violation, and norm explanation was thus not the focus of that work.

In this paper we described how a rich model for tracking and determining the status norms may be represented graphically. As a norm's status changes, so does its graphical representation. This allows the normative system to be understood visually. The use of conceptual graphs to provide the formal underpinnings of our representation will allow us to extend this work in a number of interesting directions. While other studies have shown that graphical representations are more easily understood by non-experts than logic based ones [5], we have not yet evaluated our model in this way, and intend 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. Finally, we intend to investigate more complex norm statuses than the ones described in this paper. For example, a more complete model of obligation violation requires determining whether a permission, acting as an exception to the obligation, exists. Here, complex interactions between more than one norm must be considered, and graphical models are ideal for reasoning about, and explaining such interactions.

Acknowledgements The authors would like to thank the EU Agreement Technologies COST action for providing a STSM grant which made this work possible.

References

1. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
2. G. Boella and L. van der Torre. Permissions and obligations in hierarchical normative systems. In *Proc. of ICAIL 03*, Edinburgh, Scotland, 2003.
3. G. Boella and L. van der Torre. Institutions with a hierarchy of authorities in distributed dynamic environments. *Artificial Intelligence Law*, 16:53–71, 2008.
4. W. Briggs and D. Cook. Flexible social laws. In C. Mellish, editor, *Proc. of the Fourteenth Int. Joint Conf. on Artificial Intelligence*, pages 688–693, San Francisco, 1995. Morgan Kaufmann.
5. M. Chein and M. Mugnier. *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*. Springer, 2009.
6. G. Governatori, J. Hulstijn, R. Riveret, and A. Rotolo. Characterising deadlines in temporal modal defeasible logic. In *Proc. of AI-2007*, volume 4830 of *Lecture Notes in Artificial Intelligence*, pages 486–496, 2007.
7. S. Miles, P. Groth, and M. Luck. Handling mitigating circumstances for electronic contracts. In *AISB 2008 Symp. on Behaviour Regulation in Multi-agent Systems*, pages 37–42, 2008.
8. N. Oren, S. Panagiotidi, J. Vazquez-Salceda, S. Modgil, M. Luck, and S. Miles. Towards a formalisation of electronic contracting environments. In *Proc. of Coordination, Organization, Institutions and Norms in Agent Systems, the International Workshop at AAAI 2008*, pages 61–68, Chicago, Illinois, USA, 2008.
9. Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1–2):231–252, 1995.
10. J. F. Sowa. Conceptual Graphs. *IBM Journal of Research and Development*, 20(4):336–375, 1976.
11. J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.
12. W. Woods and J. Schmolze. The kl-one family. *Computers Math. Applic.*, 23:133–177, 1992.