Verifying Conflicts among Multiple Norms in Multi-agent Systems

(Extended Abstract)

Eduardo Augusto Silvestre
Federal Fluminense University (UFF)
Av. Gal. Milton Tavares de Souza, sem nro
24210-346, Niterói, Brazil
eduardosilvestre@iftm.edu.br

Viviane Torres da Silva
IBM Research, Brazil (on leave from UFF)
Av. Gal. Milton Tavares de Souza, sem nro
24210-346, Niterói, Brazil
vivianetorressilva@gmail.com

ABSTRACT

Norms represent the expected behavior of an agent in a multi-agent system. Generally, norms describe the actions that can be performed, must be performed, and cannot be performed in the system. Due to the number of norms defined to govern a multi-agent systems (MAS), norms may conflict with each other. Norms are in conflict when the fulfillment of one norm violates the other and vice-versa. There are several works that analyze the conflicting norms. However, to the best of our knowledge, these works only analyze conflicts between pairs of norms. There are situations that conflict can only be detected when we analyze several norms together. This work presents an approach to check for conflicts among multiple norms and a strategy to minimize the complexity of this NP-hard problem.

Keywords

Multi-agent systems; normative multi-agent systems; normative conflict

1. INTRODUCTION

Multi-agent systems (MAS) have been gaining great importance in the development of various applications. MAS are autonomous, and heterogeneous societies that can work to achieve common or different goals [5].

In order to deal with the autonomy and diversity of interests among different members, the behavior of agents is governed by a set of norms specified to regulate their actions [1]. The norms govern the behavior of agents by defining obligations (stating the actions that the agents must perform), prohibitions (stating the actions that the agents must not perform) or permissions (stating the actions that the agents can perform). An important challenge when implementing normative MAS is that the set of norms can be in conflict. Conflicts occur when norms regulating the same behavior are activated and are inconsistent [3]. In such cases, the agent is unable to fulfill all the activated norms.

Although there are several works that deal with normative conflicts, to the best of our knowledge, all those approaches check for conflicts by analyzing the norms in pairs. However, there are conflicts that can only be detected when we consider several norms together. For instance, let's consider a conflict that can only be detected if norms N1, N2 and N3 are analyzed together. N1 obliges agent A to dress red shirt. N2 forbids agent A to dress red pant. N3 obliges agent A to dress pant and shirt of the same color. There are no conflicts between the pairs N1-N2, N2-N3 and N1-N3, but when the three norms are analyzed together, we can figure out the conflict.

Since several authors in literature have proved that the analysis of multiple norms is a NP-complete problem [3], we have developed an strategy to minimize the complexity of such problem. Our conflict checker starts by filtering the set of norms and grouping the norms in subsets of norms that may be in conflict. The analysis of conflicts is in fact applied to such subsets. The main contributions of this work are: (i) a more expressive representation of norms; (ii) a normative conflict checker algorithm able to check for conflicts among multiple norms; and (iii) a Java application.

2. BACKGROUND

The norm definition is based on [2] but our representation is more complex and expressive. A norm n $\in N$ is a tuple of the form {deoC, c, e, a, ac, dc} where deoC is a deontic concept from the set {O(obligation), F(prohibition) or P(permission)}, c $\in C$ is the context where the norm is defined, e $\in E$ is the entity whose action is being regulated, a $\in B$ is the action being regulated, ac $\in Cd$ indicates the condition that activates the norm and dc $\in Cd$ is the condition that deactivates the norm.

An action is defined by the name of the action and, optionally, its attributes and their values, an object where the action will be executed and a list of attributes (with their values). Thus, in this paper we define four different ways to represent the action: (i) {action}; (ii) {action object}; (iii) {action (attr1=value1,...,attrN=valueN)}; and (iv) {action object (attr1=value1,...,attrN=valueN)}. In order to exemplify these four ways to describe a action, let's consider the following four prohibition norms: Na = forbids agent A to get dressed, Nb = forbids agent A to dress pant, Nc = forbids agent A to dress red clothes and Nd forbids agent A to dress red pants. The action of the norms can be represented as: (i) Na: {to dress}; (ii) Nb: {to dress pant}; (iii) Nc: {dress (color={red})}; and (iv) Nd : {dress pant (color={red})}.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

3. CONFLICT CHECKER

Our conflict checker algorithm is divided in three steps. First, all norms are transformed into permissions (details in [4]) in order to facilitate the analysis. Since all norms are permissions, the analysis made by the conflict checker is very simple, it checks if the norms intersects. Two or more norms intersects if there is at least one possible situation where the agent is able to fulfill all norms being analyzed.

In order to apply the transformation, we assume that if an agent is obliged to execute an action, it needs to be permitted. Therefore, the transformation from an obligation norm to a permission norm is direct. A prohibition is converted into a permission by (i) negating the execution of the action being prohibited, (ii) negating the execution of the action over the object, (iii) and (iv) by inverting the values of the attributes. The second step of our conflict checker is responsible to filter the norms by including them into bags of similar norms. In order to do so, such step uses 3 filters. The filters separate into bags the norms that apply in the same context, govern the same entity and regulate the same action. After applying all filters, only the norms stored in the same bag are the ones that may be in conflict. Norms stored in different bags apply in different contexts, govern different entities or actions. The analyzes of the conflict is executed in the third step of the algorithm. The algorithm checks if the norms in each bag are in conflict. It starts by checking the norms by pairs of norms and then consider all possible group of k-norms until k be equal to the number of norms in the bag. At the end, the algorithm is checking for conflicts among all the norms of the bag at the same time.

For instance, let's consider the three norms described in Section 1. Since norm N3 is a complex norm (a norm applied over two objects), we have splited it in two norms: N3a (obliges agent A to dress pants of color X) and N3b (obliges agent A to dress shirts of color X), where X is a generic color. In the first step of the algorithm, it transforms the three obligations into permissions. Then, in the second step, the algorithm groups all norms in the same bag since they are applied in the same context (we are omitting the context for simplicity), govern the same entity (agent A) and regulate the same action (to dress). In the third and last step, it verifies that there is a conflict among these three norm. The conflict exists because there is not an intersection among the norms, i.e., agent A is unable to dress a red shirt, a pant that is not red, and a shirt and a pant of the same color.

Analysing the most expensive operation of the algorithm, the cost of the algorithm in the best case is O(1). The best case occur when each bag stores exactly one norm, i.e., the norms apply in different contexts and regulate different entities of different actions, and, therefore, are never in conflict. In such case, there is not a need to execute the third step. The worst case occurs when all norms are stored in one bag. It can happen if all norms apply in the same context, govern the same entity and regulate the same action. The cost of the algorithm in the worst case is $O(2^n)$, where n is the number of norms. The cost of the medium case, $O(2^k)$, where k is the number of norms in the bigger bag, we are unable to calculate. Such evaluation depends on the application domain, i.e., it depends on the number of different contexts, entities and actions found in the norms. Although it is not possible to calculate the medium case, we strongly believe (Section 4) that the use of filters can drastically reduce the cost of the conflict checker.

4. VALIDATION

The main purpose of the validation section is to ensure that our approach is able to detect all conflicting norms and ensure that the technique is computationally feasible. We have developed a tool to randomly generate norms based on the norms of type (iv) in order to focus on the checking of multiple conflicts. This test explores the worst scenario, i.e., all 14 norms generated have the same context, entity and action and, therefore, are stored in the same set. In this case, the time spent by the algorithm to check a small set of 14 norms is tremendous. On the other hand, a more feasible scenario where a set of 14 norms is also generated, but now considering different contexts, entities and actions have a execution time extremely smaller than the time spent when all norms are in the same bag. These test cases also demonstrate the applicability of the filter we have defined.

5. CONCLUSIONS AND FUTURE WORK

Despite significant research in the area, there are still many challenges to be considered. We presented an approach able to checker for conflicts among multiple norms that uses transformations and filters to minimize the computational cost. We have implemented our strategy in Java; the source can be accessed in https://goo.gl/CqXR4Z.

A direct consequence of this work is the investigation of how the conflicts among multiple norms should be solved. Can the techniques used to solve conflicts between pairs of norms be used to solve conflicts among multiple norms? An initial approach should investigate the applicability of famous techniques found in literature used to solve conflicts between pairs of norms, such as, lex posterior, lex superior and lex specialis.

In this work, we have not considered indirect conflicts. The algorithm presented in this paper does only check for direct conflicts, i.e., conflicts among norms that have the same entities, contexts and actions. An important and necessary extension of our work is the identification of indirect conflict among multiple norms

REFERENCES

[1] V. T. da Silva. From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. *Autonomous Agents and Multi-Agent Systems*, 17(1):113–155, 2008.

[2] K. da Silva Figueiredo, V. T. da Silva, and C. de O. Braga. Modeling norms in multi-agent systems with normml. In *Coordination, Organizations, Institutions, and Norms in Agent Systems VI - COIN 2010 International Workshops, COIN@AAMAS 2010, Toronto, Canada, May 2010, COIN@MALLOW 2010, Lyon, France, August 2010, Revised Selected Papers*, pages 39–57, 2010.

[3] W. Vasconcelos, M. Kollingbaum, and T. Norman. Normative conflict resolution in multi-agent systems. volume 19, pages 124–152. Springer US, 2009.

[4] G. H. von Wright. Deontic logic. *Mind*, 60(237):1–15, 1951.

[5] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.