Verifying Conflicts Between Multiple Norms in Multi-agent Systems

(Doctoral Consortium)

Eduardo Augusto Silvestre
Supervisor: Viviane Torres da Silva
Federal Fluminense University
Rua Passos da Pátria 156, Bloco E, 24210-240
Niterói - RJ, Brazil
eduardosilvestre@iftm.edu.br

ABSTRACT

In open multi-agent systems norms describe the behavior that can be performed, that must be performed, and that cannot be performed. One of the main challenges on developing normative systems is that norms may conflict with each other. Norms are in conflict when the fulfillment of one norm violates the other and vice-versa. Although there are many approaches to check for conflicts among norms, such approaches only analyze the norms in pairs. However, there are conflicts that can only be detected when the conflict checker analyzes multiple norms together. In this work, we present an algorithm able to check for conflicts between multiple norms that overcomes the complexity of considering multiple norms by using filters that subdivide the norms into small groups. The conflict checker executes over the norms of each group what reduces the time for executing the algorithm.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: {Distributed Artificial Intelligence}{Multiagent systems}[normative conflicts]

General Terms

Algorithms, Performance, Languages

Keywords

Normative multi-agent systems, normative conflict

1. INTRODUCTION

The multi-agent systems (MAS) has been gaining greater importance in research and practice in the development of various applications in recent years. MAS are autonomous, and heterogeneous societies that can work to achieve common or different goals [5]. In order to deal with the autonomy, the behavior of agents is governed by a set of norms what regulate their actions [2]. The norms governing the behavior of agents by defining permissions, obligations and prohibitions. In this context, it is worth mentioning the possibility of existing conflicts between norms. Conflicts occur when norms regulating the same behavior are activated and are inconsistent [4]. If two norms are in conflict, the agent is unable to fulfil both norms. The fulfilment of one norm violates the other, and vice-versa.

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 white shirt. N2 obliges agent A to dress black pants. N3 obliges agent A to dress pants and shirt of the same color. There are no conflicts between N1 and N2 and between N2 and 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 [4], 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 between multiple norms; and (iii) a Java application.

2. BACKGROUND

The norm definition is based on [3] but our representation is more complex and expressive. A norm n $\in N$ is a tuple of the form {deoC, c, e, b, ic, dc} where d is a deontic concept from the set {ob(obligation), pr(prohibition) or pe(permission)}, c $\in C$ is the context where the norm is defined, e $\in E$ is the entity whose behavior is being regulated, b $\in B$ is the behavior being regulated, ic $\in Cd$ indicates the condition that activates the norm and dc $\in Cd$ is the condition that deactivates the norm.

A behavior b is defined by the name of the action and, optionally, an object where the action will be executed and a list of parameters (together with their values) to execute such action. We define behavior as tuple: {action, object (param1=value1,...,paramN=valueN)}, where we can omit the object and its parameters and values; only the pa-

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

rameters and their values; or only the values. Such definition, let's us describe four different ways to represent the behavior: (i) {action}; (ii) {action, object}; (iii) {action, object (param1,. . . ,paramN)}; and (iv) {action, object (param1=value1,. . . ,paramN=valueN)}. In order to exemplify these four ways to describe a behavior, let's consider the following four prohibition norms: Na = forbids agent A to dress, Nb = forbids agent A dress pants, Nc = forbids agent A dress ironing shirt and Nd forbids agent A to dress white shirt. The behavior of the norms can be represented as: (i) Na: {to dress}; (ii) Nb: {to dress, pants}; (iii) Nc: {to dress, pants(ironing)}; and (iv) Nd : {to dress, pants(color=white)}.

3. CONFLICT CHECKER

Our conflict checker algorithm is divided in three steps. First, all norms are transformed into permissions 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 that object, (iii) negating the execution of the action over that object with that parameters, or (iv) by inverting the values of the parameters. 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, regulate the same behavior, and govern the same entity. 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, regulate different behaviors or govern the behavior of different entities. 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 stars 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 between 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, 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 behavior (to dress). In the third and last step, it verifies that there is a conflict between these three norm. The conflict exists because there is not an intersection between the norms, i.e., agent A is unable to dress a white shirt, a pant that is not white, and a shirt and a pant of the same color.

The computation 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 behavior of different entities, 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 behavior. The cost of the algorithm in the worst case is $O(2^n)$, where n is the number of norm. 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 behaviors found in the norms. Although it is not possible to calculate the medium case, we strongly believe that the use of filters can drastically reduce the cots of the conflict checker.

4. 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 between multiple norms that uses transformations and filters to minimize the computational cost. We are in the process of implementing the algorithm in Java, the fist version can be accessed in `http://goo.gl/t4oJk6`. In order to evaluate our approach, we need to apply it in real normative multi-agent systems. It will be important to help us on estimating the cost of executing the algorithm in real cases.

In this work, we have not considered conflicts between related norm, i.e., norms applied in different but related context, to different but related entities that regulate different but related behaviors [1]. The algorithm must consider information about the domain while investigating the conflicts. In addition, we also intend to our approach the resolution of the conflicts among several norms.

REFERENCES

[1] V. da Silva and J. Zahn. Normative conflicts that depend on the domain. In T. Balke, F. Dignum, M. B. van Riemsdijk, and A. K. Chopra, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems IX*, Lecture Notes in Computer Science, pages 311–326. Springer International Publishing, 2014.

[2] 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.

[3] 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.

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

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