Detecting Normative Indirect Conflicts: dealing with actions and states

Jean de Oliveira Zahn and Viviane Torres da Silva

Computer Science Department – Universidade Federal Fluminense (IC/UFF)
24.210-240 – Niterói – RJ – Brazil

{jzahn, viviane.silva}@ic.uff.br

Abstract. *Norms have been used in Multi-agent Systems to describe what agents can do, must do and cannot do. It intends to regulate the behavior of the autonomous and heterogeneous entities. One of the main issues on the specification of norms is the detection of normative conflicts. Two norms are in conflict when the fulfillment of one norm violates the other and vice-versa. Although several works have been proposed to deal with normative conflicts, the majority focuses on direct normative conflicts that occur when the norms apply to the same entity and govern the same behavior. We propose a mechanism to check indirect conflicts between norms that do not regulate exactly the same behavior but related ones. In order to do so, it is important to analyze the characteristics of the multi-agent system domain while checking for the normative conflicts. In this paper, we extend our preliminary work by identifying the relationships between actions and states, presenting the relationships that can be used to link the states of the multi-agent system and discussing the checking of indirect normative conflicts that can only be found when considering these relationships. The conflict checker is able to find out conflicts between norms that do not regulate the achievement of the same states and conflicts between norms when one regulates the achievement of a state and the other the execution of an action.*

1. Introduction

In open multi-agent systems, norms are being used to regulate the behavior of the involved entities. Norms state what can be performed (permission), cannot be performed (forbidden) and must be performed (obligation) in the system. One of the main challenges on developing normative systems is that norms may conflict with each other. The conflict between two norms arises when the compliance with one norm results on the violation of the other, and vice-versa.

Several approaches propose methods for detecting direct and indirect conflicts. Direct conflicts occur between norms that govern the same behavior executed by the same entity and indirect conflicts are identified when we consider the relationships between the behaviors being regulated and the relationships between the entities. Thus, the detection of indirect conflicts is only possible when we analyze the domain-dependent relationships between the states and actions being regulated and the relationships between the involved entities.

In our preliminary work [Zahn and Silva 2014] we describe the relationships used relate the entities and to relate the actions of a multi-agent system. The conflict checker algorithm stated in [Zahn and Silva 2014] focus on the identification of indirect conflicts by considering these relationships. It is able to identify conflicts between norms that are not applied to the same entity and between norms that regulate the execution of different actions.

In our current work, we extend such approach by presenting three relationships that can be used to relate states (dependency, composition and orthogonality). We also present the normative conflict checker that considers such relationships and the definition of the actions of the domain when checking for conflicts. The conflict checker is able to check for conflicts between two norms that regulate the achievement of different (but related) states and also conflicts between two norms when one norm regulates the execution of an action and the other the achievement of a state. The normative structure used in our current work extends the structure represented in our previous work [Zahn and Silva 2014]. However, besides to actions, this paper considers states as a behavior regulated by the norm.

The remainder of this paper is divided in 5 sections. Section 2 presents the definition of action and state being used in this paper. Section 3 describes the three relationships between states being considered in this paper. This Section also discusses about the rules to be followed by the conflict checker algorithm that faces two norms that regulate different but related states or that faces two norms when one regulates the execution of an action and the other the achievement of a state. Section 4 presents the algorithm to detect the indirect normative conflicts. Section 5 states some related works and, finally, Section 6 concludes and presents future work.

2. State and Action Definition

In this paper, we follow the definitions presented in [Russell and Norvig 2010] for states and actions. Each state is represented as a conjunction of *fluents* that are ground, functionless atoms. An action is described by defining the precondition for its execution and the effect (or post-condition) of its execution. The precondition of an action represents the state that must be true before the execution of the action. The precondition of an action must hold on the current state in order to be executed. The effect of an action represents the state that will be true if the action is executed. Since most actions leave most things unchanged [Russell and Norvig 2010], the effect of an action defines only what is changed after its execution. Therefore, an action is defined by a state representing its precondition, its name and its effect, as follows:

```
Action Scheme:
        (preconditionState; actionName; effect)
```

Let´s consider for instance an action to load a plane P with a cargo C. The precondition of this action states that the plane must be free and the effect of this action states that the cargo is in the plane.

```
Example:
        (free(p); load(c,p); in(c,p))
```

3. Relationships Between States

The aim of this section is to demonstrate that two norms apparently not in conflict are in fact in conflict when we figure out that the behaviors regulated by these norms are related. In order to give an example, please consider the following norms below. Norm 1 obliges *agent 1* to execute *action A* and norm 2 forbids the same agent to achieve state

S1. In this paper we are assuming that entities whose behaviors are being regulated are the same (or are related entities [Zahn and Silva 2014]), the context where the norms apply are the same and the period during while they are active intersect. Therefore, we are using a very simple template to describe our norms that does only mention the deontic concept (permission, obligation or prohibition) and the behavior being regulated.

```
Norm Template : {deonticConcept, actionState}
       Norm1 = {Obligation, execute A}
       Norm2 = {Prohibition, achieve S1}
```

Since these norms are not regulating exactly the same behavior, one could conclude that they are not in conflict. But, let's now suppose that *action A* is defined as follows:
`(S1, A, S2)` where S1 is the **precondition** for the execution of *A* and S2 is its **effect**.

If *agent1* decides to fulfill the prohibition of *norm2* stating that it cannot achieve state S1, it will never be able to execute *A*. Consequently, agent1 will violate norm1 that obliges it to execute *A*. Since the fulfillment of one norm implies on the violation of the other, we must conclude that these norms are in fact in conflict. Note that this conclusion is only possible because we have information about the domain where the norms are immersed, i.e., we know the definition of *action A* and the dependency between *action A* and state S1. Thus, if these norms are used in another domain that changes the definition of *action A*, it is not possible to affirm before hand that there will be a conflict between the norms.

The next subsections present three relationships between states and between states (or actions) that our approach is able to analyze when checking for conflicts between norms.

3.1. Dependency

The dependency relationship between states and between states and actions is originated from the definition of the actions themselves. As states in Section 2, an action is defined by its precondition and its effect. Since an action can only be executed if its precondition is achieved, there is a dependency between the action and the precondition. Moreover, the effect is only achieved it the action is executed. Therefore, there is a dependency between the effect and the action. In addition, the effect is only achieved if the precondition is achieved (and the action is executed). The definition of an action implicitly states the dependencies between precondition, action and effect that can be explicitly represented as follows: `(client, dependent, dependency)`

```
(precondition, action, dependency)
(action, effect, dependency)
(precondition, effect, dependency)
```

In order to explain the dependency relationship, let's consider the example of loading a plan presented in Section 2. The action *load(c,p)* implicitly defines the dependencies below. The dependence relationships defined implicitly should be used when checking for normative conflicts, as described in the next sections.

```
(free(p), load(c,p), dependency)
(load(c,p), in(c,p), dependency)
(free(p), in(c,p), dependency)
```

3.1.1. Obligation or Permission to Achieve the Dependent State

When a norm obliges (or permits) the achievement of a state, it means that the entity which behavior is being regulated must (or can) execute an action which **effect** is such state. For instance, in order to achieve the state *in(c,p)* it is necessary to execute the action *load(c,p)*. The state *in(c,p)* depends on the action *load(c,p)*.

If there is only one action whose **effect** is the state that must (or can) be achieved, there must not be a norm prohibiting the execution of such action. If the execution of the action is prohibited, this norm will be in conflict with the one that obliges (or permits) the achievement of the state. It is impossible to achieve the state without executing the action.

3.1.2. Obligation or Permission to Execute the Dependent Action

When a norm obliges (or permits) the execution of an action, it means that the **precondition** of such action must be achieved in order to the entity be able to execute the action. For instance, in order to execute the action *load(c,p)* it is necessary to achieve the state *free(p)*. The action *load(c,p)* depends on the state *free(p)*. Thus, if there is a norm prohibiting the achievement of the **precondition**, this norm will be in conflict with the one obliging (or permitting) the execution of the action. In any other case, there will not be a conflict.

3.1.3. Prohibition to Achieve the Dependent State

When a norm prohibits the achievement of a state, it means that the entity whose behavior is being regulated should not execute actions whose **effect** is such state. For instance, if the state *in(c,p)* is prohibited, the entity cannot execute the action *load(c,p)*.

If there is an action, which **effect** is the state being prohibited, there must not be a norm permitting or obligating the execution of such action. Note that such verification must be done for every norm applied to the actions, which **effect** is the state being prohibited. For instance, if there is a norm prohibiting an entity to achieve *in(c,p)*, the conflict arises when another norm permits (or obliges) the execution of an action whose effect is *in(c,p)*.

3.1.4. Prohibition to Execute the Dependent Action

When a norm prohibits the execution of an action, it means that the **effect** of such action will not be achieved if the agent fulfills the norm. If there is a permission or an obligation applied to such **effect** and there is not any action with the same **effect**, the norms are in conflict. If is not possible to achieve the state without violating the prohibition of executing the action.

3.2. Composition

We define this relationship since it is particular important when there is a need to abstract the details of simple behaviors and to group all behavior in a more generic one. This generic behavior aggregates the others that are implicitly represented. For instance, let's suppose that a plane can only fly from a place to another if it is loaded and fueled. We can define the state *ready(p)* being a composition of the states *in(c,p)* and *fueled(p)*.

```
(in(c,p), ready(p), composition)
```

```
(fueled(c,p), ready(p), composition)
```

So, the composition between behaviors indicates that there is one behavior that represents the whole (in our example *ready(p)*) and others that represent parts of the whole (in our example *in(c,p)* and *fueled(p)*). In [Silva 2013] this relationship is applied in the scope of actions. In the current paper we apply in the context of states and define the *wholeState* composed of *partStates*. The *wholeState* is achieved when all *partStates* are achieved. In this Section we detail the checking for conflicts between norms applied to different states/actions related by the composition relationship.

3.2.1. Obligation or Permission to Achieve the WholeState

State-State: When a norm obliges (or permits) the achievement of a *wholeState*, it means that all its *partStates* must be achieved in order to fulfill the norm. Therefore, there must not be another norm that prohibits the achievement of any *partState*. If there is a norm prohibiting the achievement of at least one *partState*, this norm will be in conflict with the other that obliges (or permits) the achievement of the *wholeState*. The fulfillment of the prohibition will violate the obligation (or permission) and vice-versa.

State-Action: Let's now consider the problems related to the obligation (or permission) applied to *wholeState* and norms applied to actions. If there is a norm that prohibits the execution of an action which effect is the *wholeState* (or a *partState*), such norm will conflict with the obligation (or permission) of achieving the *wholeState*/*partState* if this action is the only action with effect equal to the *wholeState*/*partState*. If there are other actions, there is a need to check if all other actions are also being prohibited. If the action is not executed by fulfilling the prohibition, the agent will not achieve the *wholeState*/*partState* and the obligation (or permission) of achieving the *wholeState* will be violated. However, if there are other actions that can be executed and that will achieve the *wholeState*/*partState*, there will not be a conflict between these norms.

3.2.2. Prohibition to Achieve WholeState

State-State: When a norm prohibits the achievement of a *wholeState*, it does not prevent the achievement of the *partStates* separately. The violation of the prohibition will only occur if all *partStates* are achieved in the same period of the prohibition. Thus, a conflict between the prohibition applied to the *wholeState* and obligations (or permissions) applied to *partStates* will only occur if all *partStates* are being obliged (or permitted) to be achieved in the same period of the prohibition applied to the *wholeState*. If there is at least one *partState* that is not being obliged to be achieved at the same period of the prohibition, the conflict is not characterized.

State-Action: If there are norms prohibiting the execution of actions, which precondition or effect are *partStates* (or wholeState), these norms will never conflict with the prohibition applied to the *wholeState*. However, on the other hand, if the obligation/permission is applied to an action which precondition is a *partState*, the conflict between such obligation/permission and the prohibition applied to the *wholeState* does not occurs. The entity is prohibited to achieve the *wholeState* but is not prohibited to achieve the *partStates* insolated. The entity will be able to fulfill the prohibition of achieving the *wholeState* and the obligation/permission of execution the action. For instance, if there is a norm permitting/obliging the execution of *unload(c,p)* and another prohibiting the achievement of *ready(p)*, these norms are not in conflict. In order to execute *unload(c,p)* the precondition *in(c,p)* must be achieved but it does not mean that *read(p)* is achieved *ready(p)* is achieved only if *in(c,p)* and *fueled(p)* are achieved.

3.3. Orthogonality

There are states that can be achieved at the same time by the same entity. For instance, *in(c,p)* and *fueled(p)* can be achieved at the same time since the actions to *load(c,p)* and *fuel(p)* can be executed at the same time. However, there are states that cannot be achieved at the same time by the same entity because they are orthogonal. For instance, *flying(p)* and *taxing(p)* cannot be achieved at the same time. The plane cannot be flying and taxing at the same time. In this Section, we focus on the checking of conflicts between norms applied to different states/actions related by the orthogonal relationships.

```
        flying(p), taxing(p), orthogonal)
Actions definitions:
        (flying(p), land(p), taxing(p))
        (taxing(p), takingOff(p), flying(p))
```

3.3.1. Obligation or Permission to Achieve State

State-State: If there is a norm obligating the achievement of a state and another obligating (or permitting) the achievement of another state that is orthogonal to the previous one, these norms are in conflict. Both states cannot be achieved by the same entity at the same time. For example, if there are obligations to achieve *flying(p)* and to achieve *taxing(p)*, these norms are in conflict.

State-Action: If there is a norm obligating the achievement of a state and another obligating (or permitting) the execution of an action which **effect** is a state orthogonal to the previous one, these norms are in conflict. Since both states cannot be achieved by the same entity at the same time, the entity cannot fulfill the obligation to achieve the state and to execute the action. For instance, if there is a norm obligating the execution of *land(p)* which **effect** is *taxing(p)* and another norm obligating the achievement of *flying(p)*, there norms are in conflict.

3.3.2. Prohibition to Achieve State

State-State: If there is a prohibition to an entity to achieve a state and a permission (or an obligation) to this entity to achieve another state that is orthogonal to the first one, there is not a conflict. The entity can achieve the state being permitted (or obliged) without violating the prohibition since they are orthogonal. On the other hand, if there is a prohibition to achieve a state and another prohibition to achieve another state that is orthogonal to the first one, there is a potential inconsistence if the only two states available to the entity are the ones being prohibited. For instance, let's suppose that a plane is flying or taxing. Therefore, if there is a norm prohibiting a plane to achieve *flying(p)* and another prohibiting the plane to achieve *taxing(p)*, these two norms are in conflict. Note that we are not dealing with such restriction since it would be necessary to know all possible states of the system.

State-Action: If there is a norm prohibition the achievement of a state and another obligating (or permitting) the execution of an action which **effect** is a state orthogonal to the previous one, these is not a conflict. However, if there is a norm prohibiting the achievement of a state and another prohibiting the execution of an action which effect is a state orthogonal to the first one, there is a potential conflict since these are the only states that can be achieve. A similar case was discussed previously.

4. Conflict Checker

The conflict checker algorithm checks for conflicts by considering normative pairs. The algorithm that is based on the relationships between states described in Section 3 is divided in three small algorithms, each one associated with one relationship kind.

```
Algorithm 1 Verifying Dependency Relationship
Require: $n_1$ and $n_2$ as parameter
Function: dependencyRelationship($n_1$, $n_2$)
if ((checkStateAndActionRelationship($n_1$.as, $n_2$.as) = dependency)) then
  if (( ($n_1$.deoC = O or $n_1$.deoC = P) and $n_2$.deoC = F) and
     ($n_1$.as is 'state' and $n_2$.as is 'action')) then
      if (selectActionsHaveEffect($n_1$.as)=$n_1$.as) then
         return true
      else if (checkAllBehavior(selectActionsHaveEffect(n1.as), F)) then
            return true
          endif
      endif
  endif
  if (( ($n_1$.deoC = O or $n_1$.deoC = P) and $n_2$.deoC = F) and
     ($n_1$.as is 'action' and $n_2$.as is 'state') and
     ($n_1$.as ∈ selectActionsHavePrecondition($n_2$.as)) then
       return true
  endif
  if (($n_1$.deoC = F and $n_1$.as is 'state') and
      checkAnyBehavior(selectActionsHaveEffect($n_1$.as), F)) then
       return true
  endif
  if (($n_1$.deoC = F and ($n_2$.deoC = P or $n_2$.deoC = O) and
     ($n_1$.as is 'action' and $n_2$.as is 'state')) then
     if (selectActionsHaveEffect($n_2$.as) = $n_1$.as) then
        return true
    else if (checkAllBehavior(selectActionsHaveEffect($n_1$.as), F)) then
           return true
         endif
     endif
  endif
endif
return false
end function
```

Figure 1. Verifying dependency relationship.

Algorithm 1 is responsible to check if the behaviors being regulated by the two norms are related by the dependency relationship (by executing *checkStateAndActionRelationship* function) and, if it is the case, if the norms are in conflict or not. The algorithm follows the strategies described in Section 3.1:

- (Sections 3.1.1) If one norm is obliging (or permitting) the achievement of a state and the other the execution of an action, there is a potential conflict. If the state is the effect of the action and there is not any other action with such **effect**, there is a conflict. In addition, if there is other actions with the same **effect** but all of them are being prohibited, there is also a conflict.
- (Sections 3.1.2) If one norm is obligating (or permitting) the execution of an action and the other is prohibiting the achievement of a state that is the **precondition** of such action, the norms will be in conflict.

- (Section 3.1.3) If one norm is prohibiting the achievement of a state, it is necessary to check if any action which **effect** is the state is being permitted or obligated. If it is the case, the norms will be in conflict.
- (Section 3.1.4) If one norm is prohibiting the execution of an action and the other is permitting or obligating the achievement of a state, it is important to check if there are other actions with the same **effect**. If not, the norms are in conflict. If there are other actions, we must check if they are also being prohibited. In this case, there is a conflict.

```
Algorithm 2 Verifying Composition Relationship
Require: $n_1$ and $n_2$ as parameter
Function: compositionRelationship($n_1$, $n_2$)
if (checkStateRelationship($n_1$.as, $n_2$.as) = composition) then
  if (($n_1$.deoC = P or $n_1$.deoC = O) and ($n_2$.deoC = F)) and
      $n_1$.as is 'wholeState' ) then
        return true
  endif
endif
if (checkStateRelationship($n_1$.as, $n_2$.as.effect) = composition) then
  if (($n_1$.deoC = P or $n_1$.deoC = O) and ($n_2$.deoC = F)) and
      $n_1$.as is 'wholeState' ) then
       if (checkAllBehavior(selectActionsHaveEffect($n_1$.as, F)) then
          return true
       endif
  endif
endif
if (checkStateRelationship($n_1$.as, $n_2$.as) = composition) then
  if (($n_1$.deoC = F) and $n_1$.as is 'wholeState' ) then
     if (checkAllPartStates($n_1$.as, P) or checkAllPartStates($n_1$.as, O)) then
        return true
     endif
  endif
endif
if (checkStateRelationship($n_1$.as, $n_2$.as.effect) = composition) then
  if ( (($n_1$.deoC = F) and $n_1$.as is 'wholeState') and
      (($n_2$.deoC = O) or ($n_2$.deoC = P)) then
       if (checkAllBehavior(selectAllPartStates($n_1$.as, O)) or
           checkAllBehavior(selectAllPartStates($n_1$.as, P)) ) then
             return true
       endif
  endif
endif
return false
end function
```

Figure 2. Verifying composition relationship.

Algorithm 2 checks if the states being regulated by the norms are related by the composition relationship. If it is the case, it follows the strategies presented in Section 3.2:

- (Sections3.2.1) If a norm obliges (or permits) the achievement of the *wholeState* and the other prohibits the achievement of a *partState*, these norms are in conflict. In addition, if a norm obliges (or permits) the achievement of the *wholeState* and the other prohibits the execution of an action which effect is the *wholeState/partState*, there is a potential conflict. It is important to check if it is the only action that can achieve the *wholeState/partState*. If it is, there is a conflict. If not, there will be a conflict if all actions are being prohibited.
- (Section 3.2.2) If a norm prohibits the achievement of a *wholeState*, there will be a conflict if all *partStates* are being permitted or obliged. In addition, if a norm prohibits the achievement of a *wholeState* and the other obligates or permits the execution of an action which effects is a *partState* and, there will be a potential

conflict. If there are obligations or permissions applied to all other *partStates*, there will conflicts.

Algorithm 3 is responsible to check for conflicts between behaviors that are orthogonal, following the strategies described in Section 3.3:

```
Algorithm 3 Verifying Orthogonal Relationship
Require: n_1 and n_2 as parameter
Function: orthogonalRelationship(n_1, n_2)
if ((checkStateRelationship(n_1.as, n_2.as) = orthogonal) and
   (n_1.deoC = O or n_1.deoC = P) and (n_2.deoC = O or n_1.deoC = P)) then
     return true
endif
if ((checkStateRelationship(n_1.as, n_2.as.effect) = orthogonal) and
   (n_1.deoC = O or n_1.deoC = P) and (n_2.deoC = O or n_1.deoC = P)) then
     return true
endif
if ((checkStateRelationship(n_1.as, n_2.as.precondition) = orthogonal) and
   (n_1.deoC = O or n_1.deoC = P) and (n_2.deoC = F) then
   if (checkAllBehavior(selectAllActionHavePrecondition(n_1.as, F))) then
      return true
   endif
endif
if ((checkStateRelationship(n_1.as, n_2.as.precondition) = orthogonal) and
   (n_1.deoC = F) and (n_2.deoC = O or n_2.deoC = P)  then
   if (n_1.as = n_2.as.effect) then
      return true
   endif
endif
return false
end function
```

Figure 3. Verifying orthogonal relationship.

- (Sections 3.3.1) If a norm obligates (or permits) the achievement of a state and the other obligates (or permits) the achievement of another state orthogonal to the first one, they are in conflict. In addition, if there is a norm obligating (or permitting) the achievement of a state and the other obligating (or permitting) the execution of an action which **effect** is orthogonal to such state, there is a conflict. Moreover, if a norm obligates (or permits) the achievement of a state and the other prohibits the execution of an action which precondition is orthogonal to such state, there is a potential conflict. It is important to check if all actions which precondition is such state are being prohibited. If it is the case, there are conflicts.

```
Algorithm 4 Verifying Relationships – Main
Require: n_1 and n_2 from the norm base
if (n_1.as = n_2.as and
   ((n_1.deoC = O and n_2.deoC = F) or (n_1.deoC = P and n_2.deoC = F))) then
      return norms are in conflict!
else
  if (dependencyRelationship(n_1, n_2) or
      dependencyRelationship(n_2, n_1) or
      compositionRelationship(n_1, n_2) or
      compositionRelationship(n_2, n_1) or
      orthogonalRelationship(n_1, n_2) or
      orthogonalRelationship(n_2, n_1)) then
         return norms are in conflict!
  endif
endif
return Norms are not in conflict!
end function
```

Figure 4. Verifying Relationships – Main.

- (Sections 3.3.2) If there is a prohibition to achieve a state and an obligating (or permitting) to execute an action which precondition is orthogonal to the state, there is a potential conflict. It is important to check if the effect of such action is the one being prohibited.

Algorithm 4 is responsible to call the others algorithms. It is the main algorithm and what returns the finally result about the existence of conflict (or not) among the norms. In order to exemplify our approach, let's consider the compositions relationships described in Section 3.2 and the norms N1 and N2 below. N1 prohibits the execution of the action to load the plan and N2 obligates the plan to be ready. We assume that the contexts are the same, the entities are the same (or are related) and the norms are applied in period of time that intersects.

```
N1 = {Obligation, ready(p)}
N2 = {Prohibition, load(c,p)}
```

By analyzing the domain, we know that the state *ready(p)* is a *wholeState* and *in(c,p)* and *fueled(c,p)* are *partStates*. *Ready(p)* can only be achieved if *in(c,p)* and *fueled(c,p)* are achieved. Besides, in order to achieve *in(c,p)* it is necessary to execute *load(c,p)*. **Algorithm 2** identifies that *n1.as* (i.e., *ready(p)*) is composed of *n2.as*.effect (i.e., of *in(c,p)*) and that n1 is an obligation and n2 a prohibition. Since *loaf(c,p)* is the only action that can achieve *in(c,p)* and this action is being prohibited, the algorithm concludes that N1 and N2 are in conflict.

5. Related Work

Normative conflicts in Multi-Agentes Systems have been a motivator to several works and researchers. We can divide the checking of normative conflicts in two groups: checking of direct normative conflicts and checking of indirect normative conflicts.

The majority of works is concentrated in direct conflicts and it not is able to check indirect conflicts (that is domain-dependent). The authors in [Kollingbaum et al. 2008], [Vasconcelos et al. 2007], [Vasconcelos et al. 2009] and [Vasconcelos and Norman 2009] presents approaches for the checking of norms applied to the same action. This works are not able to detect indirect conflicts.

Some papers focus on the checking of indirect conflicts, such as [Dung and Sartor 2011], [Figueiredo et al. 2011], [Garcia-Camino et al. 2006], [Kollingbaum et al. 2008] and [Vasconcelos et al. 2007]. The approaches in [Gaertner et al. 2007] and [Garcia-Camino et al. 2006] take into account the normative position, which describes activities that are propagated to other activities. In [Gaertner et al. 2007] the approach considers that multiple, concurrent and related activities are executed by agents and present a conflict checker that considers those. The authors in [Garcia-Camino et al. 2006] consider the composition relationship between activities. In addition, they state that the conflict-free normative positions of an activity propagate to its sub-activities. A normative conflict occurs when the normative positions coming from the super-activity contradicts the normative position of a sub-activity.

In [Kollingbaum et al. 2008] and [Vasconcelos et al. 2007] the normative conflict checker considers indirect conflicts by taking into account the domain specific relationships among actions. The composition and delegation relationship are defined between actions and they use unification to find out the norms that overlap. This approach is incomplete, due not consider the entities relationships. In [Aphale et al. 2012] the authors present a model of conflict identification and resolution that focuses attention on conflicts that are most critical to the goals of the organization.

The work presented in [Dung and Sartor 2011] focuses on conflicts between norms defined in different contexts. According to the authors, a particular situation can be judged by different legal systems and the norms of those systems can conflict. Similar to such approach, the works presented in [Li et al. 2013a] and [Li et al. 2013b] are able to detect conflict among different laws defined in different jurisdictions. In this paper, we focus on conflicts between norms defined in the same legal systems.

In [Silva 2013] the author presented an algorithm to detect conflicts between two norms. The algorithm focuses on detecting conflicts between a prohibition and an obligation that do not govern the behavior of the same entity, but entities that are somehow related. In addition, this first version of the conflict checker algorithm can also identify conflicts between a prohibition and an obligation that are applied to different actions related by the refinement and composition relationships. In this current paper, we focus on describing the relationships between states and on checking for conflicts between norms that regulate different states and between norms that regulate an action and a state

6. Conclusion and Future Work

This paper presents an extension of our preliminary work [Zahn and Silva 2014] on the detection of indirect normative conflicts. The first version of the conflict checker [Zahn and Silva 2014] is able to check for conflicts between norms addressed to different but related entities and that regulate the execution of different but related actions. This conflict checker algorithm was extended in this current paper to be able to check for conflicts between norms that regulate the achievement of different but related states and between norms when one regulates the achievement of a state and the other the execution of an action. In other to do so, the algorithm verifies the domain-dependent relationships between the states of the multi-agent systems and also the definitions of the actions. The paper defines (only) three different relationships that can be used to relate states but note that others can be created. The conflict checker algorithm as implemented in Jess < *http://goo.gl/CAxIay*> and is available at <*http://goo.gl/h4hwoz*>.

We are in the process of defining other relationships that can be used to relate actions (dependency and orthogonality) that were not described in [Zahn and Silva 2014] and extending the conflict checker to follow these relationships. In addition, we have noticed that there are some conflicts between norms that can only be detected when we analyze more than two norms. We are defining a strategy to check for conflicts that can analyze multiple norms at the same time without configure an NP problem.

References

Aphale, M. S., Norman, T. J., Sensoy, S., "Goal Directed Conflict Resolution and Policy Refinement," in International Workshop on Coordination, Organisations, Institutions and Norms, 2012, pp. 87-104.

Dung, P., Sartor, G., “The modular logic of private international law,” in Artificil Intelligence and Law. Springer, 19(2-3), 2011, pp. 233-261.

Figueiredo, K., Silva, S., Braga, C., “Modeling Norms in Multi-agent Systems with Norm-ML,” in International Workshop on Coordination, Organisations, Institutions and Norms VI. LNAI 6541, Springer, 2011, pp. 39-57.

Gaertner, D., Garcia-Camino, A., Noriega, P., Vasconcelos, W., “Distributed Norm Management in Regulated Multi-agent Systems,” in International Conference on Autonomous Agents and Multiagent Systems. ACM, 2007, pp. 624-631.

Garcia-Camino, A., Noriega, P., Rodrigues-Aguilar, J., “An Algorithm for Conflict Resolution in Regulated Compound Activities,” in Engineering Societies in the Agents World VII, LNCS 4457, Springer, 2006, pp. 193-208.

Kollingbaum, M., Vasconcelos, W., Garcia-Camino, A., Norman, T., “Managing Conflict Resolution in Norm-Regulated Environments,” in Engineering Societies in the Agents World VIII, LNCS 4995, Springer, 2008, pp. 55-71.

Kollingbaum, M., Vasconcelos, W., Garcia-Camino, A., Norman, T., “Conflict Resolution in Norm regulated Environments via Unification and Constraints,” in Declarative Agent Languages and Technologies V, LNCS 4897, Springer, 2008, pp. 158-174.

Li, T., Balke, T., De Vos, M., Satoh, K., Padget, J., ”Detecting Conflicts in Legal Systems. In New Frontiers in Artificial Intelligence,” LNCS 7856, Springer, pp. 174-189, 2013a.

Li, T., Balke, T., De Vos, M., Padget, J., Satoh, K., “Legal Conflict Detection in Interacting Legal Systems.” in The 26th International Conference on Legal Knowledge and Information Systems (JURIX), 2013.

Russell, Stuart J., Norvig, Peter., Artificial Intelligence: a modern approach. 3rd ed., Pearson, Upper Saddle River, New Jersey, 2010.

Silva, V., “Normative Conflicts that Depend on the Application Domain,” in International Workshop on Coordination, Organisations, Institutions and Norms, 2013, pp. 119-130.

Vasconcelos, W., Kollingbaum, M., Norman, T., “Resolving conflict and inconsistency in norm regulated virtual organizations,” in International Conference on Autonomous Agents and Multiagent Systems. ACM, 2007, pp. 632-639.

Vasconcelos, W., Kollingbaum, M., Norman, T., “Normative conflict resolution in multi-agent systems,” in Journal of Autonomous Agents and Multi-Agent Systems. ACM, 19(2), 2009, pp. 124-152.

Vasconcelos, W., Norman, T., “Contract Formation through Preemptive Normative Conflict Resolution,” in International Conference of the Catalan Association for Artificial Intelligence. ACM, 2009, pp. 179-188.

Zahn, J. O., Silva, V. T., “On the Checking of Indirect Normative Conflicts,” in: Workshop-Escola de Sistemas de Agentes, seus Ambientes e aplicações, 2014, Porto Alegre. Anais do Workshop-Escola de Sistemas de Agentes, seus Ambientes e aplicações, 2014. p. 13-24.