Utilizing Permission Norm in BDI Practical Normative Reasoning

Wagdi Alrawagfeh
Computer Science Department
Memorial University of Newfoundland
St. John’s, NL, Canada
Wagdi.alrawagfeh@mun.ca

Felipe Meneguzzi
School of Computer Science
Pontifical Catholic University of Rio Grande do Sul
Porto Alegre, RS, Brazil
felipe.meneguzzi@pucrs.br

Abstract. Norms have been used in multiagent systems as a standard description of agents’ behaviors. Much effort has been put in formalizing norms and utilizing them in agent decision making with such work often focusing on two types of norms: prohibitions and obligations. However, agents may have incomplete knowledge about norms in a system for several reasons, such as, deficient norms identification techniques or because norms are not fixed and they may change and emerge. In this work we argue that, by assuming that agents do not have complete knowledge of the norms within a system, permission norms are fundamental for modeling unknown normative states. Using Event Calculus, we propose a formal representation of permission norm and show how to use it in agent normative practical reasoning. A simple mineral mining scenario has been used to demonstrate our work, which was implemented in a popular agent programming language.

Keywords: Permission norm, Norm-representation; Normative reasoning

1 Introduction

Open Multiagent Systems often contain agents that are heterogeneous, autonomous, self-interested and which can join and leave the system at any time [1][2]. These features make interaction, coordination or collaboration in the system challenging problems. To address such challenges, systems of social norms have been proposed to provide a standard description of desirable behaviors within a society. There are two major views regarding the integration of norms in multiagent systems. In the regimentation approach where agents must obey norms and do not have choice about violating norms [3]. In this view the agents’ behaviors are more predictable; however agents drastically lose their flexibility and autonomy. In the enforcement approach the agents have the choice to comply or violate norms. In order to keep the system stable and encourage agents to respect norms, agents who violate norms are subject to punishment and those who comply with norms are often rewarded [4][5][6]. In this paper, we take the latter view of enforcement, since our approach focuses on agents that plan in order to achieve greater runtime flexibility. A substantial amount of recent work

© Springer-Verlag Berlin Heidelberg 2011
focuses on practical normative reasoning using a variety of mechanisms. Panagiotidi and Vasquez-Salceda focus on planning based normative reasoning [7], in which agents form goals from norms; Criado et al. develop an agent architecture that reasons about the agent objectives based on norms [8]; while Meneguzzi et al. develop a mechanism to steer existing agent behavior towards norm achievement while executing plans to achieve agent goals [9]. In these efforts, only two types of norms are considered in normative agent decision-making: obligations and prohibitions [10][11][12][13]. Thus, in these systems, agents check whether performing a particular behavior complies with obligations or violate prohibitions, making compromises in order perform norm compliant behaviors. In consequence processing permission norms are often ignored in agent practical reasoning. Such design choice seems to stem from the adoption of the sealing principle: “whatever is not prohibited is permitted” [14]. This principle is sound if agents have complete knowledge about the normative states of a particular system, so they can always determine whether some action violates a norm or not. Such clear-cut division of the state-space is illustrated in Fig1-a, which depicts an agent’s knowledge of a system’s normative states in case of complete knowledge and sealing principle. In this case, explicit reasoning about permission norms is not required since permission norms simply represent the absence of prohibition.

However, when agents do not have complete knowledge, some actions will be known as either prohibited, obliged, or permitted (an thus whether they are norm compliant), whereas others will be unknown. In order to address this challenge, we assume that agents do not have complete knowledge and that such incompleteness is explicitly represented, thus, in normative terms, world states can be either obliged, prohibited, permitted, or unknown. The resulting division of the state space is illustrated in Fig1-b, which depicts the incomplete knowledge of agent about a system’s normative states. Thus, agents do not have complete knowledge and some states will be known for agents as prohibited (F), others as obliged (O), while others are permitted (P); the rest of the state space is thus unknown (U). By adding permission norm to the reasoning mechanism the agent will be able to reason about preferences over behaviors that are known as permitted over unknown ones. For example if the agent needs to navigate from A to B and there are two paths X and Y. If the agent identifies that taking path Y is permitted and taking X is unknown, then a rational agent should take path Y rather than X (assuming X and Y have the same cost).

Our contributions in this paper are the following: we add the idea of utilizing permission norms in the agents’ practical reasoning; we present a formal representation of the permission norm and integrate it in a normative reasoning strategy that also reasons about prohibition and obligation norms. We assume that agents have a mechanism to discover norms as agents explore the state-space (e.g. Alrawagfeh et al. [25], Savarimuthu et al. [26]), thus, we are not concerned about addressing the norm identification problem. The organization of this article is as follows: in section 2 we briefly present a background. Section 3 discusses norm representation. The normative reasoning strategy using permission norms is presented in Section 4. In Section 5 we apply our work on the mineral mining scenario. In the last section, we conclude our work and suggest for future work.
2 Background

2.1 JASON

Beliefs, Desires and Intentions (BDI) [15] is one of the most widely studied architectures to implement practical reasoning in multi-agent systems. The BDI architecture is also widely used in the definition of agent programming languages, such as AgentSpeak(L) [17], arguably, the most widely studied such language. Jason [18,19] is a Java-based interpreter for an extended version of AgentSpeak(L) [16] where agents use a belief-base that represents knowledge using logic programming constructs that, unlike traditional AgentSpeak(L) allow Prolog-like logical rules in the agent definition. There are two types of goals defined in AgentSpeak(L): achievement goal which is represented by a literal prefixed with “!”; and test goal which is represented by a literal prefixed by “?”.

Goals and belief updates serve as triggers to the execution of hierarchical plans contained in a plan library. We now briefly review the Jason version of the AgentSpeak(L) syntax for use in this paper. The most basic syntax elements in Jason are predicates, which are represented by alphanumeric strings starting with a lower case character. Predicates may have any number of terms, with this number referred to as the arity of a predicate. A predicate represents a fact about the world and may be evaluated to either true or false. Predicates with arity greater than 0 have a number of terms. Terms are similar to predicates but they represent objects in the domain and can be either functions (terms with arity greater than 0); constants (representing specific objects in the domain); or variables, which follow the Prolog standard and start with an upper case letter or the underscore sign, representing an unnamed variable. The “+” and “-” symbols are used to represent changes in the belief base, and thus represent belief addition and deletion respectively. A plan is structured as follows: **Triggering-event: Context <- body.** In a plan, the triggering-event part is separated from the context part by the symbol “:”. In rules, the symbol “:−” separates a rule left and right hand sides; the symbols “&” and “|” indicate a conjunction and disjunction operators are indicated by the symbols “&” and “|” respectively. For more details on the semantics of Jason, we refer the reader to [20].
2.2 Event Calculus

Event Calculus (EC) is a logical framework consisting of predicates and axioms to represent and reason about actions and their effects. EC was originally proposed in logic programming [21] with the purpose to affirm that as a result of executing a particular sequence of actions some fluents are initiated to be true in a specific time-point and no action occurred that terminates this fluents. EC is known by its simplicity in describing concepts and straightforward implementation, and is thus used widely [22][23] to represent concepts in multi-agent systems.

A fluent defined as a property whose value is subject to change at different points in time. The basic components of EC are actions A, fluents F and time T.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>happens(A,T)</td>
<td>Action A occurs at time T</td>
</tr>
<tr>
<td>holdsAt(F,T)</td>
<td>Fluent F is true at time T</td>
</tr>
<tr>
<td>terminate(A,F,T)</td>
<td>Occurrence of action A at time T will make fluent F false after time T</td>
</tr>
<tr>
<td>initiates(A,F,T)</td>
<td>Occurrence of action A at time T will make fluent F true after time T</td>
</tr>
<tr>
<td>clipped(T,F,Tn)</td>
<td>Fluent F is terminated between time T and Tn</td>
</tr>
<tr>
<td>&lt;, &gt;, &lt;=, &gt;=</td>
<td>Standard order relations for time</td>
</tr>
</tbody>
</table>

We define the predicate between(A,T1,T2), which means that action A occurred after time T1 and before T2. Sometimes the effect of an action does not occur immediately. Therefore, we modified the event calculus predicates initiates(A,F,T) and terminates(A,F,T) as follows:

- initiatesAt(A,F,T1,T2) : the occurrence of action A at time T1 makes fluent F true at T2.
- terminatesAt(A,F,T1,T2) : the occurrence of action A at time T1 makes fluent F false at time T2.

If the occurrence of action A at time T1 initiates the fluent F at T2 where T1 is before T2 then, the predicate initiates(A,F,T1) is not sufficient to represent the delayed effect. Thus, we replaced the initiates(A,F,T1) predicate by initiatesAt(A,F,T1,T2) predicate, where initiatesAt(A,F,T1,T2), states that the occurrence of action A at T1 will make fluent F true after T2, when T1 ≤ T2. However, initiatesAt(A,F,T1,T2) has the same semantics of initiates(A,F,T1) when T1 = T2.

Below, we summarize the basic event calculus axioms (with the slight modification on the initiates predicate) that are important to our work:

- EC1: clipped (T1, F, T4) :- happens(A, T2) & terminatesAt(A, F, T2,T3)& T1 < T2 & T2≤T3 & T3<T4

This means that fluent F is terminated by the occurrence of action A between time T1 and T4. In other words, after time T1 if an action A occurred at time T2 and if the
effect of this action is reflected at time T3 then the fluent F becomes false between T1 and T4.

- **EC3**: holdsAt \((F, T3)\) \(-\) happens\((A, T1) \& \text{initiatesAt}(A, F, T1, T2) \& T1 \leq T2 \& T2<T3 \& \text{not clipped (}T2, F, T3)\)

This means that the fluent F holds at Time T3 if the action A occurs at time T1 and the fluent F holds after time T2 and F has not been terminated between T2 and T3.

### 3 Norm Representation

In this section, we review the norm representation presented in [23] before we extend it with permissions. In [23] three fluents are defined to represent prohibition and obligation norms. These fluents work like flags that are raised if a prohibition is violated, or an obligation is either fulfilled or violated. Here we add another fluent for permissions that is true only if a particular permitted sequence of actions is performed.

#### 3.1 Norms

A norm is defined in [24] as a tuple \( N = <D, C, Seq, S, R> \) where:

- **D** \(\in\{F,O,P\}\) is the deontic type of the norm, F for prohibition, O for obligation and P for permission.
- **C** is the norm’s context. The specified sequence of actions is obliged, prohibited or permitted if C is a logical consequence of the agent’s belief base. C is composed of (possibly empty) \(\beta\), \(\alpha\) or both. \(\beta\) is composed of holdsAt predicates which describe a particular world state. \(\alpha\) is an event calculus formula to represent a sequence of actions.
- **Seq** is a sequence of action(s) that agents are not supposed to perform, have to perform or may perform in case of prohibition, obligation\(^1\), or permission respectively. Note that Seq is different than \(\alpha\), since \(\alpha\) is part of the context condition (actions that trigger norm activation) whereas Seq is the object of the norm’s deontic type (actions that are forbidden, obliged or permitted).
- **S** is the sanction that will be applied if the norm has been violated or not fulfilled.
- **R** is the reward that agents may get if they fulfill an obligation norm.

We define the deontic types of norm as follows:

- **Prohibition norms**: In a particular context, if the occurrence of a sequence of action(s) is subject to punishment then this sequence of actions is prohibited in that context.
- **Obligation norms**: If the nonoccurrence of prescribed sequence of action(s) in a particular context is followed by punishment then this sequence of action(s) is

\(^1\) If the order of actions was not important in a norm then in the norm representation we omit the dependencies among T1,T2..Tn-1. However T1,T2..Tn-1 should be less than Tn.
obligated in that context. Also, the fulfillment of this sequence might be subjected to rewards.

- **Permission norms:** In a particular context, if the occurrence of a sequence of action(s) is not subject to punishment then this sequence of actions is permitted in that context.

The punishment and reward in the prohibition and obligation represent an incentive for agents to change their behaviors. As we see in the next section, a permission norm gives the agents the possibility of preferring the known permitted actions to the unknown actions (by unknown actions we mean the actions that are not known whether prohibited, obliged or permitted). Fluent $\text{fPun(Nid,S)}$ has been defined for prohibition norms, which, if true that means a prohibition norm has been violated. Similarly, if $\text{oPun(Nid,S)}$ fluent becomes true that means an obligation norm has been violated. But if the $\text{oRew(Nid,R)}$ becomes true this means an obligation norm has been fulfilled. For permission norm the story is different, regardless of whether the agents act according to a permission norm or not there is no reward or sanction. However, to prefer the behavior that acts according to permission norms over other behaviors, we define a fluent $\text{pRew(Nid)}$. When this fluent becomes true that means a permitted sequence of action(s) has been performed. Plan X is preferred over plan Y if X has more permitted actions than Y.

For the purpose of representing the three deontic of norm we adopt the prohibition and obligation definition in [23] and introduce the permission definition:

1. $\text{fPun(Nid,S)}$ fluent becomes true if the prohibition norm Nid is violated. The sanction of the violation is S. Nid is a unique number of prohibition norm, where S is an integer number representing the sanction value. The prohibition norm is represented as follows:

   $\text{initiatesAt(An, fPun(Nid,S), Tn, Tn+1):- C, happens(A1, T1) \& \ldots \& happens(An, Tn) \& T1 < T2 < \ldots < Tn.}$

   - $D = F$, // $\text{fPun}$ refers to prohibition norm’s violation
   - $C$: the norm’s context.
   - $\text{Seq} = A1,..An$. // the prohibited sequence of actions.
   - $S$: the sanction value which will be applied on the violator agent.
   - $R$ is empty.
   
   This means that, if the actions $A1,..An$ occurred at time $T1,..Tn$ respectively and the context $C$ was a logical consequence from the agents belief base, then the sanction that might be applied on the actor after $Tn$ is $S$.

2. $\text{oPun(Nid,S)}$ fluent becomes true if the obligation norm Nid has not been fulfilled. The punishment issued for this violation is S. Nid is a norm identification number and S is the punishment value.

3. $\text{oRew(Nid,R)}$ represents the rewards that may be granted by complying with the obligation norm Nid, where R is the reward value.

   Let $\delta$ be a sequence of actions, possibly empty, and $\delta'$ is a sequence of prescribed non-occurred action(s). An obligation norm violation occurs if in a particular context,
\( \delta \) occurred but \( \delta' \) did not occur. The obligation norm fulfillment occurs when both \( \delta \) and \( \delta' \) occur. Fluent \( oPun(Nid,S) \) becomes true if the obligation norm \( Nid \) is violated. The \( \text{initiates}(A,F,T1) \) predicate is not suitable to represent the violation of obligation norms, because it does not represent the delayed effects of actions. Therefore, we replace the \( \text{initiates}(A,F,T1) \) predicate by the \( \text{initiatesAt}(A,F,T1,T2) \) predicate. Two rules are needed to represent the obligation norm:

\[
\text{initiatesAt}(A_i,oPun(Nid,S),T_i,T_{n+1}) :- C \land \text{happens}(A_i,T_i) \land \ldots \land \text{happens}(A_j,T_j) \land \ldots \land \text{not happens}(A_j,T_j) \mid \ldots \mid \text{not happens}(A_n,T_n) \land T_1 < \ldots < T_i < \ldots < T_j < \ldots < T_n.
\]

- \( D = O \). // \( oPun \) refers to obligation norm’s violation
- \( C \): the context. (as specified before)
- \( \text{Seq} = A_j,\ldots, A_n. \) // the obligatory sequence of actions
- \( S \): the sanction. (as mentioned before)
- \( R \) is empty.

Which means that, if the context \( C \) was a logical consequence of the agent belief base and a sequence of actions (possibly empty) \( A_1,\ldots,A_i \) occurred at time \( T_1,\ldots,T_i \) respectively, and a sequence of actions \( A_j,\ldots,A_n \) did not occur at \( T_j,\ldots,T_n \), then after \( T_n \) the sanction that may be applied is \( S \).

\[
\text{initiatesAt}(A_n,oRew(Nid,R),T,T_{n+1}) :- C \land \text{happens}(A_1,T_1) \land \ldots \land \text{happens}(A_i,T_i) \land \ldots \land \text{happens}(A_j,T_j) \land \ldots \land \text{happens}(A_n,T_n) \land T_1 < \ldots < T_i < \ldots < T_j < \ldots < T_n.
\]

This means that, if \( C \) is entailed from agent’s belief base and a (possibly empty) sequence of actions \( A_1,\ldots,A_i \) occurs at time \( T_1,\ldots,T_i \), and a sequence of actions \( A_j,\ldots,A_n \) occurs at \( T_j,\ldots,T_n \), then after \( T_n \) the reward that may be granted is \( R \).

4. \( pRew(Nid) \) fluent becomes true if a permitted sequence of actions has been performed. Where \( Nid \) is the norm identification number (unique number for each permission norm). the permission norm is represented as follows:

\[
\text{initiatesAt}(A_n,pRew(Nid),T,T_{n+1}) :- C \land \text{happens}(A_1,T_1) \land \ldots \land \text{happens}(A_i,T_i) \land T_1 < T_2 < \ldots < T_n.
\]

This means that, if \( C \) entailed from agent belief base and the sequence of actions \( A_1,\ldots,A_i \) occurred at time \( T_1,\ldots,T_i \) respectively, then after time \( T_n \) the fluent \( pRew(Nid,1) \) becomes true. We illustrate this representation using the blocks world scenario in the example below.

**Example 1.**

Suppose we have three blocks: red, blue, and green, and that we have the following situation: \( \text{on(red,blue)}, \text{on(blue,table)} \) and \( \text{on(green,table)} \). Now, if we have a permission norm that states “it is permitted to put green on red if red was not on the table”. This permission norm is represented as follows:

\[
\text{initiatesAt}(\text{on(green,red)},pRew(Nid),T_1,T_2) :- \text{not holdsAt(on(red,table),T_1)} \land \text{happens(on(green,red),T_1)} \land T_1 < T_2.
\]


4 BDI agent normative reasoning

In order to develop our normative reasoning strategy, we leverage the normative reasoning strategy presented by Alrawagfeh [24], which utilizes prohibition and obligation norms to find the best plan (in the sense of highest utility). In the context of BDI agents, the best plan is found among the applicable plans. Our extension in this section is to utilize permission norms in order to find the safest plan among the best plans. We argue that using permission norms in practical reasoning within a normative system is important for at least two reasons when agents have incomplete knowledge of normative states; First, if it is the agents duty to identify norms, the norms identification mechanism may miss some norms; and second, norms are not fixed; they may change, emerge or vanish. Hence, presuming that “whatever is not prohibited is permitted” is not adequate since it does not account for such missing norms.

We illustrate this argument with the following scenario. Suppose that an agent wants to achieve a goal G and there are two plans P1 and P2, of equal and highest utility based on prohibition and obligation norms, to achieve G. Suppose P1 has some prohibited action(s) but because of incomplete knowledge of agents, the agent does not know that these actions are prohibited, then mistakenly the agent will presume them as permitted. However, if the agent maintains the permission norms as it does with prohibition and obligation then it can prefer between P1 and P2, the plan that has more permitted actions than other will be the safest. In Fig.2, the basic BDI interpreter overview is illustrated as white boxes, while our proposed additions are presented as gray boxes. To deal with dynamic norms, the norm identification process needs to be integrated with normative reasoning strategy. The agent discovers emerged and abrogated norms by the norm identification process. However, the norm identification process is beyond the scope of this paper [25][26].

![Fig. 2. BDI Reasoning processes flow](image)

The execution of plan Plan makes fluent help(Plan) true if it results in more rewards than punishments (based on the helpful-rule below). The punishments value comes as a result of violating prohibition norms or not fulfilling obligation norms. The rewards value comes as a result of achieving the goal that is associated with Plan and fulfilling obligation norms. Predicate goalpreference(G,Points) here is used to describe the importance of achieving goal G, where Points is an integer number referring to the importance of G and the goal importance is determined according to the agent’s designed objectives.
To utilize permission norms in the normative reasoning strategy we define
\text{safe(Plan)} fluent which becomes true if the execution of \text{Plan} conforms with one or
more permitted norms (based on the safe-rule below). If we have two plans of equal
utility (based on prohibition and obligation norms) then the plan that conforms to
more permitted norms is the safest, since the other actions that are unknown might be
prohibited. Our agent uses the following application-independent axioms in its norma-
tive reasoning strategy:

EC1 & EC3' from Section 3

\text{Ax1: between(A,T1,T2) :- happens(A, T) & T1<T & T<T2}
\text{Ax2: terminatesAt(*,help(P),T1,T2):- happens(*,T1)}
\text{Ax3: terminatesAt(*,safe(P),T1,T2):- happens(*,T1)}
\text{Ax4: terminatesAt(*,pPun(I,S),T1,T2):- happens(*,T1)}
\text{Ax5: terminatesAt(*,oPun(I,S),T1,T2):- happens(*,T1)}
\text{Ax6: terminatesAt(*,oRew(I,S),T1,T2):- happens(*,T1)}

In order to terminate fluents, an application-independent special event * is used. It
refers to the fact that the associated fluents becomes false after T. These axioms help
agents find a potential norm violation/fulfillment that may result from executing one
plan. If we want the agent to find the potential norm’s violation/fulfillment that may
result not only from executing the current plan, but also the one that may result by
actions interleaving of current plan and the previous plan, then we need to add to the
right hand side of Ax2…Ax6 another \text{happens(*,T2)} predicate. Here, the * action
monitors the end point of those fluents that are mentioned in the axioms above. With
these changes Ax2 will be as follows

\text{Ax2b: terminatesAt(*,help(P),T1,T2):- happens(*,T1) & happens(*,T2).}

Same thing applies for \text{Ax3, Ax4, Ax5 and Ax6.}

using the above application-independent axioms in addition to the following rules (we
call them helpful-rule and safe-rule) the agent is able to find the set of best plans
among the applicable plans (using helpful-rule) and out of the best plans find the safest
plan (using safe-rule).

helpful-rule:

\text{initiatesAt(K,help(Plan_i),T1,T2):-}
\quad \text{.findall(V1,holdsAt(oRew(_,V1),T2+1), Wins) &}
\quad \text{.findall(V2,holdsAt(pPun(_,V2),T2+1), Loses1) &}
\quad \text{.findall(V3,holdsAt(oPun(_,V3),T2+1), Loses2) &}
\quad \text{goalpreference(G,Points) &}
\quad \text{Points + \sum(Wins) - \sum(Loses1) - \sum(Loses2)> 0.}

safe-rule:

\text{initiatesAt(K,safe(Plan_i),T1,T2):-}
\quad \text{.findall(V1,holdsAt(pRew(_,V1),T2+1), Count).}

In the helpful-rule, by asserting action K (see Algorithm 1, lines 3,4 and 5), we simu-
late that K has occurred), and by executing .findall(V,P,S) function we obtain all the
values of V where predicate P is true and variable S instantiates with the sum of values
V. Finally, \( \text{sum(Wins)} \) obtains the rewards that may be granted if \( \text{Plan}_i \) is executed. \( \text{sum(Loses1)} \) and \( \text{sum(Loses2)} \) will have the sanctions that may result if the plan was executed. Conversely, in the safe-rule \( \text{findall(V,P,S)} \) function we obtain all the values of \( V \) where predicate \( / \) is true and variable \( S \) instantiates with the sum of values \( V \). \( \text{Count} \) will have the number of permission norms that are complied with if \( \text{Plan}_i \) is executed. We can now define how our agent finds the set of best plans and then the safest plan. We need the following additional definitions:

- Let \( \text{act}(\Pi) \) be a function that returns the sequence of actions of the plan \( \Pi \), \( (A_1,A_2,...,A_n) \). Before the agent takes the decision of intending/committing plan\( \Pi \), it will use the Algorithm 1 to find the most profitable and safe plan.
- Let \( \Gamma \) be the set of applicable plans.
- Let \( \text{Bel} \) be a belief base represents the agent’s knowledge about the society along with the society’s norms represented in Event Calculus.
- Let \( \Omega = \text{EC1, EC3', Ax1, Ax2, Ax3, Ax4, Ax5,Ax6, helpful-rule, safe-rule} \) and \( \text{Bel} \).

These are used in the following algorithm.

---

Algorithm 1 //find best plan

Input: applicable plans.
Output: safest plan. // the plan with the highest utility and the one that conforms with more permission norms.

1: function FindBestPlan(\( \Gamma \))
2: for all \( \Pi \) in \( \Gamma \) do
3: \( T \leftarrow \) current time.
4: for all \( \theta \in \text{act}(\Pi) \) do
5: \( \text{Bel} \leftarrow \text{Bel} \cup \text{happens}(\theta,T++) \).
6: end for
7: \( T \leftarrow \) current time.
8: if \( \Omega \vdash \text{holdsAt(help}(\Pi),T) \) then
9: \( \text{utility}(\Pi) \leftarrow \text{Points} + \text{sum(Wins)} - \text{sum(Loses1)} - \text{sum(Loses2)} \). // see helpful-rule.
10: end if
11: delete the asserted happens predicates at line 5.
12: end for
13: \( \text{BestPl} \leftarrow \{ \max \{ \text{utility}(\Pi) \} \} \).
14: for all \( \Pi \) in \( \text{BestPl} \) do
15: \( T \leftarrow \) current time.
16: for all \( \theta \in \text{act}(\Pi) \) do
17: \( \text{Bel} \leftarrow \text{Bel} \cup \text{happens}(\theta,T++) \).
18: end for
19: \( T \leftarrow \) current time.
20: if \( \Omega \vdash \text{holdsAt(safe}(\Pi),T) \) then
21: end if
As shown in Lines 4-6, the predicates `happens` (see Table 1) are added starting from time `T1` (the current time). The actions specified in predicate `happens` have not occurred yet. By adding the plan’s actions, the agent simulates that it has executed them in order to reason about whether the current plan `Π` is helpful or not. Plan `Π` is helpful if the predicate `holdsAt(help(Π),T)` is deduced from current belief base; if that is the case, then rewards outweigh losses. The plan of maximum utility is then added to the best plan set `BestPl`. The set `BestPl` will have the plans of highest equal utilities. Lines 14-25 find the safest plan out of `BestPl`. At Line 20 if the predicate `holdsAt(safe(Π),T)` is deduced for current belief base then this means that there is at least one permission norm being complied with as a result of executing plan `Π`. The number of times permission norms are complied with results from executing plan `Π` are saved in variable `Count`. The plan of maximum count is thus the safest plan. The safest plan will be ready for execution by adding it to the intentions. The `happens` predicate for each action of an executed plan will be added to the belief base (same as in Lines 4-6). Predicate `happens(*,Tn+1)` is added to the belief base after executing the last action of the chosen plan. The purpose of adding the special event `*` is to terminate the fluents (`help`, `safe`, `fPun`, `oPun` and `oRew`) after `Tn+1`. Thus, if plans executed in the past were causing norm violation/fulfillment then we do not want this violation/fulfillment to be revealed again in the future. However, our strategy is able to reveal the violation/fulfillment that may result from the interleaving of the current plan (under investigation) and the previous executed plan.

5 Experiments

For experiment purposes, a mineral mining society which has been adapted from the Gold Miners scenario\(^2\) has been used. We adopt a similar scenario as the one presented in [23], that scenario is composed of; a grid-like territory with gold and silver pieces which are scattered in; and agents to collect the scattered pieces to their respective depot (one for silver and one for gold). In the territory there is a monitor agent who plays the role of police in the scenario. The monitor agent is able to observe other agents actions and also able to issue sanctions or rewards. There are set of norms which govern the society. We assume that agents do not know all the norms. The society has three agents, one that uses prohibition and obligation norms in its practical reasoning. The other one uses prohibition, obligation and permission norms in its practical reasoning. Both of them have the same prohibition and obligation norms.

\(^2\) http://jason.sourceforge.net/Jason/Examples
The third agent is the monitor agent. We call the first agent best-agent and the second agent best-safest-agent, and call the third agent monitor-agent. The best-agent uses in its practical reasoning the axioms Ax1...Ax6 and the helpful-rule. The best-safest-agent uses the axioms Ax1...Ax6, helpful-rule and safe-rule. The norms are represented in the monitor-agent using EC, monitor-agent uses the Ax1...Ax6 to check if a violation/fulfillment occurred.

5.1 Gold and Silver mining society

In this society the possible actions which agents can perform are: pick(-,-), drop(-,-) and moveto(-,-). The two competitive agents have one continuous goal which is \( \text{!collect(gold)} \), the importance of achieving this goal is obtained by the predicate \( \text{goalpreference(collect(gold), 10)} \), so the importance of values of achieving the goal is 10. The grid has two depots one for gold and one for silver. The grid has 10 pieces of gold and 10 pieces of silver. The two agents compete to collect those pieces and the game ends when all the pieces are collected. In this experiment the potential violations/fulfillments that may result from the current plan and the previously executed plan are taken into consideration. The two agents have the same plans below:

\[
\begin{align*}
\text{@plan1-1} & \quad /\!/ \text{the agent will collect gold to the silver depot.} \\
& +\text{!collect(gold): free} & < & + \text{!find(gold,X,Y)} ; \text{moveto(X,Y)} ; \\
& \quad \text{pick(gold); moveto(silver_depotX,silver_depotY)} ; \\
& \quad \text{drop(gold,silver_depot).} \\
\text{@plan1-2} & \quad /\!/ \text{the agent will collect gold to the gold depot.} \\
& +\text{!collect(gold): free} & < & + \text{!find(gold,X,Y)} ; \text{moveto(X,Y)} ; \\
& \quad \text{pick(gold); moveto(gold_depotX,gold_depotY)} ; \\
& \quad \text{drop(gold,gold_depot).} \\
\text{@plan1-3} & \quad /\!/ \text{the agent will collect gold to the gold depot and} \\
& \quad \text{collect silver to gold depot.} \\
& +\text{!collect(gold): free} & < & + \text{!find(gold,X,Y)} ; \text{moveto(X,Y)} ; \\
& \quad \text{pick(gold); moveto(gold_depotX,gold_depotY)} ; \\
& \quad \text{drop(gold,gold_depot); !find(silver,X1,Y1)} ; \text{pick(silver)} ; \\
& \quad \text{moveto(gold_depotX,gold_depotY); drop(silver,gold_depot).} \\
\text{@plan1-4} & \quad /\!/ \text{the agent will collect gold to the gold depot and} \\
& \quad \text{collect another gold to the gold depot.} \\
& +\text{!collect(gold): free} & < & + \text{!find(gold,X,Y)} ; \text{moveto(X,Y)} ; \\
& \quad \text{pick(gold); moveto(gold_depotX,gold_depotY)} ; \\
& \quad \text{drop(gold,gold_depot); !find(gold,X1,Y1)} ; \text{moveto(X1,Y1)} ; \\
& \quad \text{pick(gold); moveto(gold_depotX,gold_depotY)} ; \\
& \quad \text{drop(gold,gold_depot).} \\
\text{@plan1-5} & \quad /\!/ \text{the agent will collect gold to the gold depot and} \\
& \quad \text{collect silver to silver depot.} \\
& +\text{!collect(gold): free} & < & + \text{!find(gold,X,Y)} ; \text{moveto(X,Y)} ; \\
& \quad \text{pick(gold); moveto(gold_depotX,gold_depotY)} ; \\
& \quad \text{drop(gold,gold_depot).} \\
\end{align*}
\]
There are set of norms that govern this society; some of them are unknown to best-agent and best-safest-agent. The common norms among the three agents as follows:

**Prohibition norms**

- It is prohibited to drop gold in the silver depot if the gold depot is not full, the sanction value is 5.

  \[
  \text{initiatesAt}(\text{drop(gold,silver_depot)}, \text{fPun}(1, 5), T1, T2):\quad \text{not holdsAt(full(gold_depot), T1)} \quad \text{happens(drop(gold,silver_depot), T1)} \quad \text{&} \quad T1 \leq T2.
  \]

- It is prohibited to carry more than one gold piece at the same time, the sanction value is 10

  \[
  \text{initiatesAt}(\text{pick(gold)}, \text{fPun}(3, 10), T1, T3):\quad \text{happens(pick(gold), T1)} \quad \text{&} \quad \text{happens(pick(gold), T2)} \quad \text{&} \quad \text{not between(drop(gold, _), T1, T2)} \quad \text{&} \quad T1 < T2 \quad \text{&} \quad T2 < T3 \quad \text{&} \quad T3 \leq T3.
  \]

**Obligation norms**

- It is obligatory to collect silver immediately after collecting gold, the sanction value is 10. The reward for adhering is 10.

  \[
  \text{initiatesAt}(\text{pick(gold)}, \text{oPun}(1, 10), T1, T4):\quad \text{happens(pick(gold), T1)} \quad \text{&} \quad \text{happens(drop(gold, _), T2)} \quad \text{&} \quad \text{happens(pick(gold), T3)} \quad \text{&} \quad \text{not between(pick(silver), T2, T3)} \quad \text{&} \quad T1 < T2 \quad \text{&} \quad T2 < T3 \quad \text{&} \quad T3 < T4.
  \]

  \[
  \text{initiatesAt}(\text{pick(gold)}, \text{oRew}(1, 10), T1, T4):\quad \text{happens(pick(gold), T1)} \quad \text{&} \quad \text{happens(drop(gold, _), T2)} \quad \text{&} \quad \text{happens(pick(gold), T3)} \quad \text{&} \quad \text{between(pick(silver), T2, T3)} \quad \text{&} \quad T1 < T2 \quad \text{&} \quad T2 < T3 \quad \text{&} \quad T3 < T4.
  \]

In addition to those norms best-safest-agent has the following permission norms:

- It is permitted to drop gold in gold depot

  \[
  \text{initiatesAt}(\text{drop(gold,g_depot)}, \text{pRew}(1), T1, T2):\quad \text{happens(drop(gold,g_depot), T1)} \quad \text{&} \quad T1 \leq T2.
  \]

- It is permitted to drop silver in silver depot

  \[
  \text{initiatesAt}(\text{drop(silver,s_depot)}, \text{pRew}(2), T1, T2):\quad \text{happens(drop(silver,s_depot), T1)} \quad \text{&} \quad T1 \leq T2.
  \]

The monitor agent has the above norms in addition to the following prohibition norm:
It is prohibited to drop silver in the gold depot if the silver depot is not full, the sanction value is 10.

\[
\text{initiatesAt}(\text{drop(silver,gold_depot)}, \text{fPun(1,10)}, T1, T2):- \quad \text{not holdsAt(full(silver_depot),T1) happens(drop(silver,gold_depot),T1) } \quad \& \quad T1 <= T2.
\]

The experiment was executed 10 times and the average taken. Two values for each agent were recorded: the calculated-utility which results from the agent’s prediction in case of a particular plan is chosen; real-utility which results from the execution of a particular plan. These two values could be different, if an agent does not know that a particular act is prohibited then the sanction value of performing this act will not be calculated in the calculated-utility but it will be included in the real-utility (the sanction value will be issued by the monitor agent).

Based on the norms and the 5 plans above, the best-agent finds the two best plans (plan_3 and plan_5) with utility equals to 20, then it will choose one of them randomly. However, plan_3 violates a prohibition that is not known for the two agents under the comparison. The best-safest-agent finds the same best plans (plan_3 and plan_5), then, using safe-rule, it chooses the one that conforms with more permissions, which is plan_5. In Fig.3 the results show that the average utility for goals achieved for best-safest-agent is greater than the utility of the best-agent. That is because best-safest-agent integrates permissions into its normative reasoning and using our preference relation over the best plans. We noticed that the best-agent has collected more gold and silver pieces than the best-safest-agent, that could be due to best-safest-agent spending more time reasoning about plan selection than the best-plan. The results in Fig.4 shows that the real utility (after plan execution) of the best-agent is lower than the predicted/calculated utility (before plan execution), this result goes back to best-agent using only prohibitions and obligations in its practical reasoning, in addition to its incomplete knowledge about the system norms. For the best-safest-agent, the real and calculated utilities were identical, hence, in Fig.4, the line for the best-safest agent calculated utility is not appeared, it is below the best-safest agent real utility, (though this is not necessarily always true). However, the ultimate utility of the best-safest agent should be the same or better than the ultimate utility of the best-agent.

**Fig. 3.** Shows the average collected gold and silver pieces and the ultimate achievement utilities for best-agent and best-safest-agent.

**Fig. 4.** Shows the calculated/predicted utility for the best-safest-agent and best-agent.
6 Conclusion

In this paper we presented a formal representation of norms and a normative reasoning strategy based on event calculus that, in addition to prohibition and obligation norms, takes permission norms into consideration. We demonstrate empirically that when agents have incomplete knowledge about the norms of a system then permissions have a significant role in practical normative reasoning. Using permission norms gives agents the ability to have preference over plans, i.e, plans containing actions that are known to be permitted are preferable over plans that contain actions whose normative status is unknown. The experiments compare between two agents: one (best-agent) that uses prohibition and obligation norms into its practical reasoning; the other one (best-safest-agent) that uses prohibition, obligation and permission norms into its practical reasoning. The results demonstrate that adding permission norms in the reasoning process may have fundamental benefits. However, by best-agent, the average number of collected gold and silver pieces is more than the average collected pieces by best-safest-agent. This may indicate that the normative reasoning in the best-safest-agent needs more time than the case with best-agent. As future work, we plan to do further experiments to study the time efficiency of our practical normative reasoning mechanism. It will also be interesting to compare our best-safest-agent with other BDI norm aware agents in the literature.

Acknowledgments. We thank the anonymous reviewers from COIN for their constructive reviews, which helped to improve this paper substantially.

References