Normative KGP Agents

Fariba Sadri[1], Kostas Stathis[2], Francesca Toni[1]

[1]Dept. of Computing, Imperial College London. {fs,ft}@doc.ic.ac.uk
[2]School of Informatics, City University London. kostas@soi.city.ac.uk

Abstract

We extend the logical model of agency known as the KGP model, to support agents with normative concepts, based on the roles an agent plays and the obligations and prohibitions that result from playing these roles. The proposed framework illustrates how the resulting normative concepts, including the roles, can evolve dynamically during the lifetime of the agent. Furthermore, we illustrate how these concepts can be combined with the existing capabilities of KGP agents in order to plan for their goals, react to changes in the environment, and interact with other agents. Our approach gives an executable specification of normative concepts that can be used directly for prototyping applications.

Keywords: Computational Logic, Normative Agents, Abduction, Priorities, Planning, Roles, Obligations, Prohibitions.

1 Introduction

The development of complex applications based on autonomous agents often need to find methods of organising the functionality of a composite system by distributing responsibilities to the parts, in such a way that interactions of the whole are coordinated in a decentralised and coherent manner. Whether we deal with autonomous robots that plan, software agents that manage networks, a traffic or a file-sharing system, or any other similar application, it is becoming increasingly recognised that the resolution of the underlying problems lies with developing frameworks that are based on the notion of social agency [14].

The basic idea behind the notion of social agency is to use abstractions from such diverse fields as sociology, computing science, organisational theory, and law in order to specify [16] and then implement complex organisations of agents referred to as *artificial societies* [28]. Such an implementation seeks to apply social agents in practical applications where the formation of open societies [4] are envisaged to be regulated by *norms*. The notion of a norm is important here in that member agents of a society [34] must have the capability to reason with norms and they must be capable of communicating norms to other member agents. The problem then reduces to resolving the issue of how to develop

normative autonomous agents [12] for supporting practical applications based on artificial societies.

To develop normative agents programmers often rush into implementing the rules governing an artificial society without properly understanding the subtleties that underlie their specification. Motivated by this observation a number of researchers, e.g. see [10, 17, 9], are seeking to understand the modalities required to specify the norms of an artificial society as a separate issue from their implementation. Deontic concepts such as obligation, prohibition, permission, rights, power, and entitlement, are currently being scrutinised and analysed in detailed formal frameworks that have been produced as a result. Much of the work in this effort, however, can only be used as a guide to implementations, as it abstracts away from the computational characterisation of the resulting specifications and, more importantly, from the way these normative concepts are to be used during the operation of the agent in an artificial social environment.

Motivated by this last observation, this work seeks to complement the specification effort previously described by presenting a framework that illustrates how normative concepts, such as obligations and prohibitions, can be used by an agent while it reasons, reacts, plans, and communicates in the context of an artificial society. The framework builds upon an existing framework called the KGP (Knowledge, Goals, Plan) model of agency [19] which we have successfully implemented in the prototype agent platform PROSOCS [33]. The main aim of this paper is to show how to extend the KGP model with normative concepts, thus providing a framework where we can develop agents that can reason about norms that are expected to govern their own behaviour while pursuing their own personal goals.

The resulting model makes a number of contributions. It allows the agents to become aware of the obligations and prohibitions related to their activities and environment. It allows the programmer to specify different types of agent behaviour by defining a theory of priorities and preferences whereby agents decide which of their goals (personal or norm-driven) to commit to. In this way the programmer can design a variety of agent types, for example, socially responsible, socially irresponsible, selfish, opportunistic, and so on. The behaviour of the agent can be designed to change dynamically depending, for example, on the environment the agent is situated in, on communicative acts received from other agents, and the history of its own and other agents' activities. The approach also allows the integration of the normative concepts and reasoning with the rest of the agent's activities. A distinguishing feature of our approach, that can be considered as a major advantage, is that the normative rules of the kind proposed here can be looked at in many different ways: they are executable specifications, they are directly implementable and, within the declarative and operational model of the KGP agents, they can force agents to exhibit the expected behaviour, conformant with their specification.

The paper is structured as follows. In section 2 we motivate the contribution of the paper by a series of informal examples. We then provide, in section 3, the required background summarising the main features of KGP agents, including the two extensions of computational logic it is based on. Then, in section 4, we

describe in detail how to extend the current model so that we can accommodate normative concepts such as roles, obligations, and prohibitions. Then, in section 5 we revisit one of the examples of section 2 and illustrate how the new model deals with it. In section 6 we show how the new model can be used to design a variety of agent types with different social behaviours. In the final two sections we discuss the related work and conclude.

2 Some motivating examples

To ground the discussion in the paper we outline here three scenaria that abstract away from practical applications in that only human agents are involved. The scenaria still illustrate, however, the kind of interactions that may arise in practical applications amongst an agent's different types of goals and plans, including personal and norm-driven ones, and the impact of the agent's preferences and behaviour characteristics.

Example 1 Consider an agent John who wishes to upgrade his computer and to this end plans to buy a new PC at the cost of £500. Suppose also that in the meantime John parks his car illegally and is sanctioned with a fine of £150.

In our framework the goal of upgrading the computer and the plan of purchasing a new PC are part of the personal goals and plan of John, while paying the fine for illegally parking a car is part of his social (norm-driven) goals and plans. John becomes aware of the fine by receiving a message or other form of input from the environment.

Now suppose that John has only £600 available. Under the circumstances it cannot fulfill both his personal and his social plans. He needs to make a choice between them. If he is socially responsible he abandons the purchase of a new PC and pays the fine instead. In this case John might be able to find another way of fulfilling his goal of upgrading the computer, for example by purchasing a new hard disk at a lower cost. If he is socially irresponsible he ignores the fine and buys a new PC.

Example 2 Suppose our agent John has two personal goals, one of building an extension to his house, and another of improving his garden. To fulfill the second goal he decides to plant a large tree which he intends to bring home in his car. He then notices that the car has broken down and, in reaction, he decides to rent a van instead. Suppose also that John is aware of the social norms stating that:

- anyone who intends to build an extension to his house should first inform the local authority, and
- no one should obstruct his neighbour's view.

Reflecting on these norms John becomes aware that he should inform the local authority about his intention to build an extension. He also realises that if he

were to plant the large tree he would be in violation of the second social norm. John could proceed in a number of different ways depending on his priorities by deciding to:

1. Fulfill all his social obligations, thus informing the local authority about his proposed extension, and deciding not to plant the large tree and consequently deciding not to rent a van.
2. Ignore all his social obligations, thus not informing the local authority about the extension, and also renting the van and ultimately planting the tree.
3. Tread a middle ground, fulfilling the first social obligation, namely the informing of the local authority, which is not too detrimental to his personal goals, and ignoring the second one, namely not blocking the neighbour's view, which would prevent his personal plan.

Example 3 Consider example 2 slightly modified as follows. John wishes to build an extension to his house and the normative rule related to this activity is not that he should inform the local authority but that he should seek approval from it. If John is a responsible agent he will make "seeking the approval from the local authority" a social (norm-driven) goal for himself in addition to his (personal) goal of building an extension. He can then proceed to plan for all his goals in a similar way. For example, to fulfill the goal of seeking approval from the local authority for his extension he may plan to write a specification for the building work, get it signed by an authorised architect and then submit it to the local authority. To fulfill his goal of building the extension he may plan to get a bank loan, choose a builder and so on, and wherever necessary he will impose temporal constraints on the actions across all his plans.

3 Background

3.1 Extensions of logic programming for the KGP model

3.1.1 Abductive Logic Programming with constraints

An *abductive logic program* is a tuple $\langle P, A, I \rangle$ where:

- P is a *normal logic program*, namely a set of rules of the form $H \leftarrow L_1 \wedge \ldots \wedge L_n$, with H atom, $L_1, \ldots, L_n$ literals, and $n \geq 0$. Literals can be positive, namely atoms, negative, namely of the form $not\, B$, where B is an atom, or constraint atoms over some given structure $\Re$ equipped with a notion of constraint satisfaction $\models_{\Re}$ (as in constraint logic programming [15]). All variables in rules are implicitly universally quantified from the outside.
- A is a set of *abducible predicates* in the language of P, but not in $\Re$. Atoms whose predicate is abducible are referred to as *abducible atoms* or simply as *abducibles*.

- I is a set of *integrity constraints*, that is, a set of sentences in the language of P. All the integrity constraints in the KGP model have the implicative form $L_1 \wedge \ldots \wedge L_n \Rightarrow A_1 \wedge \ldots \wedge A_m$ $(n \geq 0, m > 0)$ where L_i are literals (as in the case of rules) [1], and A_j are atoms as in the case of rules, but also possibly including the special atom $false$. All variables in an integrity constraint are implicitly universally quantified from the outside, except for variables occurring only in the *head* $A_1 \wedge \ldots \wedge A_m$, which are implicitly existentially quantified with scope the head itself.

Given an abductive logic program $\langle P, A, I \rangle$ and a formula (*query*) Q, which is an (implicitly existentially quantified) conjunction of literals in the language of the abductive logic program, the purpose of abduction is to find a (possibly minimal) set of (ground) abducible atoms Γ which, together with P, "entails" (an appropriate ground instantiation of) Q, with respect to some notion of "entailment" $\models_{LP}$ that the language of P is equipped with, and such that the extension of P "satisfies" I (see [20] for possible notions of integrity constraint "satisfaction"). Here, the notion of "entailment" is a combined semantics $\models_{LP(\Re)}$, as presented in [15], also taking into account satisfaction of the constraint atoms occurring in the abductive logic program.

Formally, given a query Q, a set Δ of (possibly non-ground) abducible atoms, and a set C of (possibly non-ground) constraints, the pair (Δ, C) is an *abductive answer* for Q, with respect to an abductive logic program $\langle P, A, I \rangle$, iff for all groundings σ for the variables in Q, Δ, C such that $\sigma \models_{\Re} C$, it holds that

(i) $P \cup \Delta\sigma \models_{LP(\Re)} Q\sigma$, and

(ii) $P \cup \Delta\sigma \models_{LP(\Re)} I$.

Here, $\Delta\sigma$ plays the role of Γ in the earlier informal description. Note that, if the query is simply $true$, the abducibles in Δ, along with the constraints in C, guarantee the overall consistency with respect to the integrity constraints given in I.

Such notion can be extended to take into account an initial set of (possibly non-ground) abducible atoms Δ_0 and an initial set of (possibly non-ground) constraint atoms C_0. In this extension, an abductive answer for Q, with respect to $\langle P, A, I \rangle$, and Δ_0, C_0 is a pair (Δ, C) such that

(i) $\Delta \cap \Delta_0 = \{\}$

(ii) $C \cap C_0 = \{\}$, and

(iii) $(\Delta \cup \Delta_0, C \cup C_0)$ is an abductive answer for Q with respect to $\langle P, A, I \rangle$ (in the earlier sense).

[1] If $n = 0$, then $L_1, \ldots, L_n$ represents the special atom $true$.

3.1.2 Logic Programming with Priorities

For the purposes of this paper, a *logic program with priorities*, referred to as $\mathcal{T}$, consists of four parts:

(i) a low-level part P, consisting of a logic program; each rule in P is assigned a name, which is a term; e.g., one such rule could be

$$n(X) : p(X) \leftarrow q(X, Y), r(Y)$$

with name $n(X)$;

(ii) a high-level part H, specifying conditional, dynamic priorities amongst rules in P; e.g., one such priority could be

$$h(X) : m(X) \succ n(X) \leftarrow c(X)$$

to be read: if (some instance of) the condition $c(X)$ holds, then (the corresponding instance of) the rule named by $m(X)$ should be given higher priority than (the corresponding instance of) the rule named by $n(X)$. The rule itself is named $h(X)$;

(iii) an auxiliary part A, defining predicates occurring in the conditions of rules in P, H and not in the conclusions of any rule in P;

(iv) a notion of incompatibility which, for our purposes, can be assumed to be given as a set of rules defining the predicate *incompatible*, e.g.

$$incompatible(p(X), p'(X))$$

to be read: any instance of the literal $p(X)$ is incompatible with the corresponding instance of the literal $p'(X)$. We assume that incompatibility is symmetric, and refer to the set of all incompatibility rules as I.

Any concrete LPP framework is equipped with a notion of entailment, that we denote by $\models_{pr}$, defined differently by different approaches to LPP, wrt some given underlying logic programming semantics $\models_{LP}$. Intuitively, $\mathcal{T} \models_{pr} \alpha$ iff α is the conclusion (wrt $\models_{LP}$) of a sub-theory of $P \cup A$ which is "preferred" wrt $H \cup A$ in $\mathcal{T}$ over any other sub-theory of $P \cup A$ that derives a conclusion incompatible with α (wrt I). For example, in [29, 23, 21], $\models_{pr}$ is defined via argumentation. The framework of LPP can be usefully extended with constraints in the same way that CLP extends logic programming: all logic programs with priorities in this paper are assumed to be written in such an extension. We will assume that $\mathcal{T} \models_{pr} X$ iff X is the set of all preferred conclusions from $\mathcal{T}$ (in the earlier sense).

Note that our approach does not depend crucially on the use of the framework of LPP: other frameworks for the declarative specification of preference policies, e.g. Default Logic with Priorities [7], could be used instead.

3.2 KGP model overview

We summarise the KGP model of agency [19, 6] by focusing on the components relevant to this paper. We refer the reader to [19, 6] for any additional details.

KGP relies upon an internal (or mental) state and a set of reasoning capabilities, supporting planning, temporal reasoning, identification of preconditions of actions, reactivity, goal decision and a sensing capability. Based on the above capabilities, the state is operated on by a set of transitions, defining how the state of the agent changes. Transitions also require, apart from the capabilities, a set of selection functions, to provide appropriate inputs to them. A cycle theory then decides which transitions should be applied when, using the output of the selection functions in order to take this decision.

Internal state. This is a tuple $\langle KB, Goals, Plan, TCs \rangle$, where:

- KB describes what the agent knows of itself and the environment and consists of several modules supporting the different reasoning capabilities, including
  - KB_{plan}, for Planning,
  - KB_{pre}, for the Identification of Preconditions of actions,
  - KB_{react}, for Reactivity,
  - KB_{TR}, for Temporal Reasoning,
  - KB_{GD}, for Goal Decision,
  - KB_0, for holding the (dynamic) knowledge of the agent about the external world in which it is situated (including past communications), and perceived through its sensing capability. KB_0 is contained in all other modules of KB.

  Syntactically, KB_{plan}, KB_{TR} and KB_{react} are abductive logic programs (see [6]), KB_{GD} is a logic program with priorities (see [6]), and KB_{pre} is a logic program. KB_0 contains assertions recording the actions which have been executed (by the agent or by others) and their time of execution as well as the properties (i.e. fluents and their negation) which have been observed, possibly concerning other agents, and the time of the observation. For example, KB_0 of some agent a may contain assertions $executed(park(W129FGC, t), 5)$, namely a has parked a car with number plate $W129FGC$ at time $t = 5$, $observed(b, issue\_fine(W129FGC, 10), 12)$, namely a has observed at time 12 that agent b has issued a fine to car with number plate $W129FGC$ at time 10, $observed(\neg parked(W129FGC, t'), 7)$, namely a has observed that car with number plate $W129FGC$ is not parked where it should at time $t' = 7$.

- $Goals$ is a set of properties that the agent wants to achieve, each one equipped with a time variable. Goals may also be equipped with temporal constraints (given in the TCs component of the state) binding the time variable and constraining when the goals are expected to hold. Goals may be *mental* or *sensing*. Both can be observed to hold (or not to hold) via

the Sensing capability, updating KB_0. In addition, mental goals can be brought about actively by the agent by its Planning capability and its actions.

Syntactically, properties are *timed fluent literals* $l[t]$, where l refers to a (positive or negative) property that the agent wants to hold and t is the *time* of the goal, namely a variable, implicitly existentially quantified within the overall state of the agent. An example is $has\_driving\_licence(ag_1, t_1)$, indicating that agent ag_1 wants to have a driving license at time t_1.

- $Plan$ is a set of actions scheduled in order to satisfy goals. Each is equipped with a time variable possibly bound by temporal constraints (given in TCs), similarly to $Goals$, but constraining when the action should be executed. Actions are partially ordered, via their temporal constraints. Actions may be *physical*, *communicative*, or *sensing*.

  Syntactically, actions are *timed action literals* $a[t]$, where a refers to the operator of the action, and t is the *execution time* of the action, namely a variable, implicitly existentially quantified within the overall state of the agent. An example is $pay\_fine(ag_1, t_2)$, indicating that agent ag_1 wants to perform an act of paying a fine at time t_2.

- TCs is a set of constraint atoms (referred to as *temporal constraints*) in some given underlying constraint language. We assume that the constraint predicates include $<, \leq, >, \leq, =, \neq$. These constraints specify when goals are to hold and when actions are to be executed, and they are extended and instantiated as the agent operates. For example, TCs may contain $10 < t_1 < 20$, $t_2 < t_1$.

Within a state, goals in $Goals$ and actions in $Plan$ are organised within two tree structures, with roots $\bot^{nr}$ and $\bot^{r}$, by associating with each goal and action its parent, which is either a goal or one of the two roots. Actions have no children. Actions and goals are referred to as *reactive* if they belong to the tree with root $\bot^{r}$, and *non-reactive*, if they belong to the tree with root $\bot^{nr}$. The children of the roots are referred to as *top-level* goals or actions. Top-level goals are generated by calls to the Goal Decision (with root $\bot^{nr}$) or Reactivity (with root $\bot^{r}$) capabilities. Children of goals are introduced by calls to the Planning capability.

In this paper, we will refer to *non-reactive* goals and actions as *personal*, to distinguish them from *social* goals and actions that will be introduced to fulfill social norms. We will also adopt an alternative (and equivalent) representation of the state of agents, emphasising the different types of goals and actions, via a tuple $\langle KB, GP_p, GP_r, TCs \rangle$ where GP_p is the set of all personal goals and actions and GP_r is the set of all reactive goals and actions in the earlier state representation. Goals and actions can be distinguished as they are built from ontologically distinguished sets of fluent and action literals.

Reasoning capabilities. These include:

- Goal Decision, which decides new (possibly temporally constrained) top-level goals in GP_p, adapting its own preferences on the basis of changes in the environment, as recorded in KB_0.
- Reactivity, which reacts to perceived changes in the environment, by identifying new (possibly temporally constrained) top-level goals and/or actions in GP_r on the basis of the contents of GP_p and KB_0. Typically, these goals and actions are generated to repair a plan in GP_p in the light of changes in the environment or to respond to communications from other agents.
- Planning, which generates partial plans for sets of goals. It provides (temporally constrained) sub-goals and actions designed for achieving the input goals.
- Identification of Preconditions, which identifies the preconditions for action execution.
- Temporal Reasoning, represented by $\models_{TR}$, which reasons about the evolving environment, and makes predictions about properties (fluents) holding in the environment, based on the partial information the agent acquires within KB_0.

Sensing capability. This links the agent to its environment, by allowing to observe that properties hold or do not hold, and that other agents have executed actions. It also allows agents to receive communication from other agents.

Transitions. The state of an agent evolves by applying transition rules, which employ capabilities. The transitions include:

- *Passive Observation Introduction* (POI) changes KB_0 by introducing unsolicited information coming from the environment or communications received from other agents. It calls the Sensing capability.
- *Plan Introduction* (PI) changes part of the GP_p, GP_r and TCs of a state, according to the output of the Planning capability.
- *Goal Introduction* (GI) changes the GP_p and TCs of a state by replacing GP_p with goals that the Goal Decision capability decides to be most preferred, and by adding any corresponding temporal constraints to TCs.
- *Reactivity* (RE) changes the GP_r and TCs of a state by replacing GP_r with the goals and actions returned by the Reactivity capability, and by adding any corresponding temporal constraints to TCs.

- *State Revision* (SR) revises GP_p and GP_r e.g. by dropping goals that have already been achieved, actions that have already been executed, and goals and actions that have run out of time. It calls the Temporal Reasoning capability and also checks satisfaction of the temporal constraints.

- *Action Execution* (AE) is responsible for executing all types of actions, thus changing the KB_0 part of KB by recording that actions have been executed. It calls the Sensing capability for the execution of sensing actions.

Transitions T are represented as

$$(\mathbf{T}) \qquad \frac{S}{S'}\,\tau$$

where S is the agent state prior to the transition, τ is the time of application of the transition, and S' is the state resulting from applying the transition. We give here the formal definition of SR as it will be used and extended later in the paper:

$$(\mathbf{SR}) \qquad \frac{\langle KB, GP_p, GP_r, TCs\rangle}{\langle KB, GP'_p, GP'_r, TCs\rangle}\,\tau$$

where $GP'_p \cup GP'_r$ is the biggest subset of $GP_p \cup GP_r$ consisting of all items (goals or actions) $X = \langle x[t], Y\rangle$ such that:

(i) $Y \in GP'_p \cup GP'_r \cup \{\bot^{nr}, \bot^{r}\}$, and

(ii) there exists a total valuation σ such that $\sigma \models_{\Re} TCs' \wedge t > \tau$, and

(iii) if $x[t]$ is an action literal then it is not the case that $executed(x[t], \tau') \in KB_0$, and

(iv) if $x[t]$ is a fluent literal then there is no total valuation σ such that $\sigma \models TCs \wedge t \leq \tau$ and $KB \models_{TR} x[t]\sigma$, and

(v) for every sibling $Z = \langle x'[t'], Y\rangle$ of X in $GP_p \cup GP_r$, either Z is a sibling of X in $GP'_p \cup GP'_r$ or, if x' is a fluent, there is a total evaluation σ such that $\sigma \models TCs \wedge t' \leq \tau$ and $KB \models_{TR} x'[t']\sigma$, and, if x' is an action, $executed(x'[t'], \tau') \in KB_0$.

Condition (ii) removes timed-out goals and actions, and (iv) removes goals that are already achieved, (iii) removes actions that have already been executed, (v) removes goals and actions whose siblings are already timed out (and thus deleted, by condition (ii)), and (i) removes actions and goals with ancestors which are got rid off (recursively).

Cycle. The behaviour of an agent is given by the application of transitions in sequences, repeatedly changing the state of the agent. These sequences are not determined by fixed cycles of behaviour, as in conventional agent architectures, but rather by reasoning with cycle theories (see [18]). These are logic programs with priorities defining preference policies over the order of application of transitions, which may depend on the environment and the internal state of the agent. Examples of cycle theories, giving a *normal* behavioural profile as well as other profiles, can be found in [32].

The Event Calculus. KB_{plan}, KB_{react}, KB_{TR} and KB_{pre} are all specified within the framework of the event calculus (EC) for reasoning about actions, events and changes [22]. Moreover, KB_{GD} makes use of an EC theory for its auxiliary part. Finally, EC theories will also be used to define the new features of N-KGP agents. Here, we give a high-level description of the EC, see [22] for any further details and [6] for the specific form of EC we are adopting.

In a nutshell, the EC allows to write meta-logic programs which "talk" about object-level concepts of *fluents*, *events* (that we interpret as *action operations*), and *time points*. The main meta-predicates of the formalism are: $holds\_at(F, T)$ (a fluent F holds at a time T), $holds\_at(\neg F, T)$ (the negation $\neg F$ of a fluent F holds at a time T), $clipped(T_1, F, T_2)$ (a fluent F is clipped - from holding to not holding - between times T_1 and T_2), $declipped(T_1, F, T_2)$ (a fluent F is declipped - from not holding to holding - between times T_1 and T_2), $initially(F)$ (a fluent F holds from the initial time, say time 0), $initially(\neg F)$ (the negation $\neg F$ of a fluent F holds from the initial time, say time 0), $happens(O, T)$ (an operation O happens at a time T), $initiates(O, T, F)$ (a fluent F starts to hold after an operation O at time T) and $terminates(O, T, F)$ (a fluent F ceases to hold after an operation O at time T). Roughly speaking, the last two predicates represent the cause-effects links between operations and fluents in the modelled world. We also use a meta-predicate $precondition(O, L)$ (the fluent or negation of fluent L is one of the preconditions for the executability of the operation O). Finally, we use meta-predicates $assume\_happens(O, T)$ and $assume\_holds(L, T)$ in order to model planning and reactivity. Intuitively, an action can be planned for, via abduction of an *assume_happens* atom (similarly for fluents).

The EC includes axioms to model persistence of fluents and their negation from the time they are initiated and terminated by events, respectively, from the initial time (if they hold initially) and from the time they are observed (as recorded in KB_0), unless their value is clipped by other events or observations. Events can be actions planned for or already executed.

The representation of fluent literals and action literals in the state of KGP agents is in the following correspondence with EC literals: $l[t]$ corresponds to $assume\_holds(l, t)$ and $a[t]$ corresponds to $assume\_happens(a, t)$.

4 N-KGP: the Normative KGP Model

4.1 Motivation

The normative KGP agent model (N-KGP agent model) is to extend the KGP model to allow the integration of agents in societies. To this end we require agents first to become aware (possibly partially) of the norms of the society, then to decide which of these norms are relevant to their particular role in the society and other current personal and environmental circumstances. Equipped with this awareness they then have to weigh up their own personal goals against the obligations and prohibitions that the society imposes on them via the norms, and make choices according to their preferences.

We would like to model different types of agents, for example responsible ones who may give higher priority to what the society expects of them compared with their own personal goals, completely irresponsible ones who would always disregard the society expectations, and selfish ones who may respect the society's expectations and commit to fulfilling their social obligations so long as these do not hinder them in perusing their own personal goals.

We extend and adapt the KGP model by incorporating into it a new knowledge base to cater for social norms, new capabilities and transitions to allow agents to make choices based on their preferences and give guidelines for defining appropriate cycle theories for normative agents.

These modifications to the KGP model allow agents

- to make choices about their personal goals, for example on the basis of their own profiles and notions of urgency and priority and environmental factors,
- to be reactive and adaptive to changes in their environment,
- to be able to analyse the norms of their societies to determine how they apply to them individually based on their roles, environment and history,
- to set themselves goals and constraints on the basis of the norms,
- to maintain a clear representation of their different types of goals and actions, personal, reactive, social,
- to weigh up their own personal goals and plans against their social goals and plans and make choices amongst them,
- to integrate their normative reasoning with the rest of their functionalities such as planning for their goals, adapting to their environment and interacting with other agents.

In the remainder of this section we give details of the normative KGP model.

4.2 Normative Concepts in KGP

Our approach in adopting normative concepts (and reasoning) in the KGP model is motivated by the need for artificial societies of agents that require an organisation and division of tasks. In this context, we interpret responsibility for tasks of an agent in terms of the roles an agent has been assigned to play in a social environment. Roles are associated with obligations and prohibitions, used to define what is expected of the agent that plays a role. Rather than developing or deploying any fully-fledged normative theory, our intention is to concentrate on a simple theory but explore in detail its interaction with the existing operations of KGP agents and the provable conformance of (certain types of) agents to this theory.

Roles are assigned to agents, possibly with associated temporal constraints, by authorised agents. Agents are equipped with knowledge of their roles and of information gained through observations in the environment. Agents also have self-knowledge of their own goals and plans. On the basis of all this information, agents become aware of their obligations and prohibitions.

4.2.1 Actions and Fluents

In general obligations and prohibitions will be on performing actions or bringing about fluents. Within obligations and prohibitions we will represent actions as terms of the form

$$act(Act, Actor, Parameters).$$

Act names the action, *Actor* is the agent to carry out the act, and *Parameters* is the set of attributes that further specify the action. For instance, the term:

$$act(pay\_fine, ag_1, ''W129FGC'', 60)$$

represents the action of agent ag_1 having to pay a fine of 60 pounds for a car with registration number W129FGC.

We shall represent fluents within obligations and prohibitions as terms of the form:

$$fluent(Fluent, Actor, Parameters).$$

Fluent is the name of the fluent, *Actor* is the agent to bring about *Fluent*, and, as before, *Parameters* represents the rest of the attributes describing the *Fluent*. For instance, the term:

$$fluent(has\_driving\_licence, ag_1, ''ST340578KS'')$$

represents the fluent that agent ag_1 has a driving license with number ST340578KS.

4.2.2 Roles

In our model the agents' responsibility for tasks and assignment of obligations and prohibitions is determined by the *roles* the agents play in their social environment. Therefore, we incorporate within the KGP model roles assigned to agents, by using an EC-based representation. We use the fluent:

$$role(Agent, Role)$$

to state when a specific *Role* is assigned to a specific *Agent*. $role/2$ fluents are initiated and terminated by means of event calculus $initiates/3$ and $terminates/3$ predicates. For instance, the addition of the following domain-specific rule for $initiates/3$:

$$initiates(assign(Y, X, t\_w(W, F, T), T', role(X, t\_w(W, F, T)))$$

allows us to conclude that after the occurrence of an action of the form

$$assign(ag_1, ag_2, t\_w(chelsea, 9, 17))$$

ag_2 plays the role of traffic warden (t_w) in *chelsea* between times 9 and 17. Depending on the ontology of an application, we can qualify the assignment of roles, e.g. by means of a rule such as:

$$initiates(assign(Y, X, t\_w(W, F, T), T', role(X, t\_w(W, F, T)) \leftarrow \\ authorised(act(assign(Y, X, t\_w(W, F, T), Y), T')$$

which relies upon a notion of "authorisation". We will not address this notion further in this paper.

Note that roles of agents might change as time progresses and will be initiated or terminated as a result of events happening in the social environment in which an agent is situated.

4.2.3 Obligations

We represent obligations as atoms of the form:

$$obliged(act(Act, Actor, Parameters), T)$$
$$obliged(fluent(Fluent, Actor, Parameters), T)$$

We read atoms of the first kind as follows: there is an obligation on the agent *Actor* to bring about the action specified by *Act* (with the appropriate *Parameters*) at a time *T*. This time may be required to satisfy some temporal constraints specified in *TCs*. We read atoms of the second kind as follows: there is an obligation on the agent *Actor* to bring about the fluent *Fluent* to hold at a time *T* (which may also need to satisfy some temporal constraints).

For example, an agent *a* may be obliged to pay a fine, as follows:

$$obliged(act(pay\_fine, a, W129FGC), t)$$

where $t < 20$ might belong to TCs.

4.2.4 Prohibitions

Prohibitions can be specified in a similar way to obligations. We use atoms of the form:

$$prohibited(act(Act, Actor, Parameters), T, TC)$$
$$prohibited(fluent(Fluent, Actor, Parameters), T, TC)$$

to indicate that *Actor* is prohibited from performing *Act* (with the appropriate *Parameters*) at *all* times T within the time constraints specified in TC, and *Actor* is prohibited from bringing about *Fluent* (with the appropriate *Parameters*) to hold at *all* times T within time constraints TC, respectively.

For example, an agent a may be prohibited to park in the city centre between 10 and 18, as follows:

$$prohibited(act(park, a, city\_centre), T, 10 < T < 17)$$

In the remainder of the paper, for simplicity, we will drop the $Parameters$ argument from *obliged* and *prohibited* atoms.

4.3 State of N-KGP Agents

The state of N-KGP agents at any given time is represented as a tuple of the form: $\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs \rangle$, where

- KB consists of all the knowledge bases of KGP agents, namely KB_{plan}, KB_{react}, KB_{pre}, KB_{GD}, KB_{TR} and KB_0, together with
  - KB_{soc}, to cater for the normative reasoning which determines the social goals and actions of the agent,
  - KB_{rev}, to cater for the overall state revision reasoning, resolving conflicts amongst goals and actions of different types, belonging to different parts of the state.

  From now on, we will refer to KB_{GD} as KB_{PGD}, as it allows to reason about *personal* goals and actions.
- GP_p and GP_r are as before, for the KGP agent model
- $GP_{\mathcal{O}}$ has a the same tree structure as GP_p and GP_r, but with a root $\perp^{\mathcal{O}}$. As in the KGP model, the nodes in $GP_{\mathcal{O}}$ may be goals or actions. $GP_{\mathcal{O}}$ contains the social obligations of the agent and any plans (actions and sub-goals) the agent may have generated for fulfilling them.

- $GP_{\mathcal{A}}$ is a set of assertions, specifying the actions and fluents to be avoided, and when they should be avoided. Syntactically, these are of the form $avoid(x[all], TC)$, where $x[all]$ is a *universally quantified* timed fluent literal or timed action literal, with the time *all* constrained by TC. Logically, these assertions can be seen as representing integrity constraints in abductive logic programming, of the form $x[T] \wedge TC' \Rightarrow false$ (with T implicitly universally quantified over the integrity constraint), where TC' is obtained from TC by replacing all occurrences of *all* by T. In effect $GP_{\mathcal{A}}$ contains (a representation of) the social prohibitions of the agent. The constant *all* is used in order to represent universally quantified variables in abductive answers (with respect to KB_{soc}, see below in section 4.3.1).

- TCs is as before, for the KGP model.

Below we give details of KB_{soc} and KB_{rev}.

4.3.1 KB_{soc}

KB_{soc} is an abductive logic program (see section 3.1.1) representing social norms.

The logic program in KB_{soc} is the one in KB_{plan}, possibly extended with some auxiliary definitions.

The integrity constraints in KB_{soc} are those in KB_{plan} plus instances of the integrity constraints of the general forms:

$$holds\_at(role(Actor, Role), T) \wedge Conditions(Actor, T)$$
$$\Rightarrow obliged(act(Act, Actor), T') \wedge TC$$

$$holds\_at(role(Actor, Role), T) \wedge Conditions(Actor, T)$$
$$\Rightarrow obliged(fluent(Fluent, Actor), T') \wedge TC$$

$$holds\_at(role(Actor, Role), T) \wedge Conditions(Actor, T)$$
$$\Rightarrow prohibited(act(Act, Actor), all, TC)$$

$$holds\_at(role(Actor, Role), T) \wedge Conditions(Actor, T)$$
$$\Rightarrow prohibited(fluent(Fluent, Actor), all, TC)$$

Here, *Conditions* may be any conjunction of literals in the language of KB_{plan}. The first two rules above can be read as follows: if an *Actor* is playing a *Role* at time T and the *Conditions* hold in the knowledge base of the *Actor* at T, the *Actor* must fulfill the obligation of making the *Act* happen at T' or bring about the property specified by *Fluent* (with the appropriate *Parameters*) at T', respectively, with T' satisfying the temporal constraints *TC*. The last two rules are read as follows: if an *Actor* is playing a *Role* at time T and the *Conditions* hold in the knowledge base of the *Actor* at T, the *Actor* is prohibited from

performing the *Act* or bringing about the property specified by *Fluent* (with the appropriate *Parameters*) at *all times* satisfying the temporal constraints *TC*.

Furthermore, the set of integrity constraints in KB_{soc} include:

$$
\begin{aligned}
&prohibited(fluent(Fluent, Actor), all, TC) \wedge \\
&obliged(fluent(Fluent, Actor), T) \wedge TC[all/T] \\
&\Rightarrow false
\end{aligned}
$$

$$
\begin{aligned}
&prohibited(act(Act, Actor), all, TC) \wedge \\
&obliged(act(Act, Actor), T) \wedge TC[all/T] \\
&\Rightarrow false
\end{aligned}
$$

where $TC[all/T]$ stands for TC where all occurrences of *all* are replaced by $T2$. These constraints guarantee that the set of obligations and prohibitions generated by the $\models_{sGD}$ capability is "consistent". It basically guarantees that the underlying theory of normative reasoning is coherent.

Atoms of the form $obliged(X, T)$ and $prohibited(X, all, TC)$ are the only abducibles in KB_{soc}.

Note that in defining prohibitions we have somewhat extended the syntax of abductive logic programming by using the constant *all* to represent universally quantified variables in the head of integrity constraints defining prohibitions.

Concrete examples of integrity constraints in KB_{soc} are:

$$
\begin{aligned}
&holds\_at(role(X, t\_w(W, F, T), T') \wedge \\
&observed(parked(C, W), T') \wedge \\
&F < T' < T \\
&\Rightarrow obliged(act(issue\_fine(C), X), T' + 1)
\end{aligned}
$$

$$
\begin{aligned}
&holds\_at(working\_day, T) \wedge \\
&holds\_at(role(X, driver(C)), T) \wedge \\
&holds\_at(in(W, city\_centre), T) \\
&\Rightarrow prohibited(act(park(C, W), X), all, 10 < all < 17)
\end{aligned}
$$

The parameter *Actor* inside the functors *act*, $fluent$ and *role* allow the agent to reason about other agents' obligations and prohibitions. This information can then, in turn, be exploited in the agent's own plans and activities. The syntax of *act*, $fluent$ and *role* can be made even richer to represent, for example, an obligation or commitment of one agent towards another to perform an action or to establish a fluent. Further considerations on these issues, however, are outside the scope of this paper. Therefore, henceforth, for ease of reading, we

assume that the parameter *Actor* is always instantiated to the name of the agent itself and thus drop it from inside the functors *act*, $fluent$ and *role*.

4.3.2 KB_{rev}

KB_{rev} is a logic program with priorities (as defined in section 3.1.2) representing the agent's preference policy towards revising incompatible personal, reactive, and social goals and actions. Here we give some examples of the contents of KB_{rev}, and in particular definitions of the preference ordering $\succ$ in the high-level part and of the *incompatible* predicate. KB_{rev} will also contain an appropriate auxiliary part including the logic program in KB_{plan}.

$$A1 \succ A2 \leftarrow A1 \in GP_{\mathcal{O}} \wedge A2 \in GP_p$$

This rule, which for example could be in the KB_{soc} of a responsible agent, says that any social action (in the tree $GP_{\mathcal{O}}$) has higher priority than any personal action (in the tree GP_p).

$$avoid(A[all], TC) \succ A[T] \quad \leftarrow \quad TC[all/T]$$

This rule, which again could be in the KB_{soc} of a responsible agent, says that any action has lower priority than the record that it is to be avoided (as prohibited). Note that *avoid* is used to represent within the state of an N-KGP agent actions and goals that are prohibited, as we will see later on in sections 4.4.1 and 4.5.1.

$$\begin{aligned} incompatible(A1[T1], A2[T2]) \quad \leftarrow \quad & funds(A1, M1) \wedge funds(A2, M2) \wedge \\ & holds\_at(balance(M), T) \wedge earlier(T1, T2, T3) \wedge \\ & T < T3 \wedge M1 + M2 > M \end{aligned}$$

This rule states that two actions $A1$ and $A2$ at respective times $T1$ and $T2$ are incompatible if they, respectively, need funds to the value of $M1$ and $M2$ and before the earlier action the agent does not have enough funds to cover them.

$$incompatible(A1[T], A2[T]) \leftarrow not\, concurrently\_executable(A1, A2)$$

This rule states that two actions $A1$ and $A2$ are incompatible if they are to be executed at the same time and that is not possible.

$$incompatible(p[T], not\, p[T])$$

This states that a goal and its negation are incompatible.

$$incompatible(A[T], avoid(A[all], TC)) \quad \leftarrow \quad TC[all/T]$$

This states that the record that an action A is to be avoided is incompatible with A.

$$incompatible(A[T], avoid(F[all], TC)) \quad \leftarrow \quad initiates(A, T, F) \wedge T \leq T' \wedge TC[all/T'] \wedge not\, clipped(T, F, T')$$

This states that an action A and the prohibition of a fluent F are incompatible if the action establishes the fluent at the time it is prohibited.

4.4 New Capabilities of N-KGP agents

In addition to the reasoning capabilities and the sensing capability of the original KGP agents, N-KGP agents have the following reasoning capabilities:

- Social Decision of Goal/Action/Avoid assertions ($\models_{sGD}$)
- State Revision ($\models_{rev}$)

4.4.1 $\models_{sGD}$ capability

Informally this capability takes as input the current state of the agent and returns as output all the social obligations and prohibitions (with their associated temporal constraints) that are applicable to the current state of the agent. The capability computes this output by performing abductive reasoning to analyse the normative rules in the KB_{soc} of the agent in the context of the agent's state.

Formally: given a state $\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle$ and a time instant τ, $\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle \models_{sGD} GP, \mathcal{A}s, C$ iff

- (Δ, C) is an abductive answer for $true$ wrt $KB_{soc} \cup \{time\_now(\tau)\} \cup GP_0$ and $(\{\}, C_0)$ where
  - $GP_0 = \bigcup_{\langle a[t'], \_\rangle \in GP_p \cup GP_r} \{assume\_happens(a, t')\} \cup \bigcup_{\langle l'[t'], \_\rangle \in GP_p \cup GP_r} \{assume\_holds(l', t')\}$
  - $C_0 = TCs \wedge \Sigma$
- $GP = \{x[t] \mid obliged(act(x), t) \in \Delta \, or \, obliged(fluent(x), t) \in \Delta\}$
- $\mathcal{A}s = \{avoid(x[all], Y) \mid prohibited(act(x), all, Y) \in \Delta \ or \ prohibited(fluent(x), all, Y) \in \Delta\}$

Given a state $\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle$ and a time instant τ, $\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle \models_{sGD} \bot$ iff there exists no abductive answer for $true$ wrt $KB_{soc} \cup \{time\_now(\tau)\} \cup GP_0$ and $(\{\}, C_0)$ as defined earlier.

Note that the predicate $time\_now$ belongs to the language of KB_{plan} and thus KB_{soc}, and it can occur in the $Conditions$ of normative rules.

4.4.2 $\models_{rev}$ capability

This is the capability that weighs up the different types (personal, reactive, social) of the agent's goals and actions, and determines, according to the current state of the agent and the dynamic preferences of the agent as specified in its KB_{rev}, which actions and goals the agent will commit to.

Formally: given a state $\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle$ and a time instant τ, $\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle \models_{rev} GP, \mathcal{A}s$ iff

- $KB_{rev} \cup \{time\_now(\tau)\} \models_{pr} X$
- $GP = X \setminus \{avoid(x[T], Y) | avoid(x[T], Y) \in X\}$
- $\mathcal{A}s = \{avoid(x[T], Y) | avoid(x[T], Y) \in X\}$

Intuitively, X consists of the set of all preferred goals and actions in GP_p, GP_r, $GP_{\mathcal{O}}$ in the given state, as well as goals and actions to be avoided, from $GP_{\mathcal{A}}$ in the given state.

4.5 New Transitions of N-KGP agents

N-KGP agents are equipped with all transitions of KGP-agents, but have a revised form of State Revision, that we refer to as SR^+, and an additional transition sGI, for the introduction of social goals, actions, and actions and goals to be avoided. These are detailed below.

4.5.1 The sGI transition

Informally this transition updates the agent's state by updating the part ($GP_{\mathcal{O}}$, $GP_{\mathcal{A}}$) related the agent's social goals and actions. It obtains the required information about how the state is to be updated from the $\models_{sGD}$ capability.

Formally:

$$(\textbf{sGI}) \qquad \frac{\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle}{\langle KB, GP_p, GP_r, GP'_{\mathcal{O}}, GP'_{\mathcal{A}}, TCs'\rangle}\ \tau$$

where $\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle \models_{sGD} GP, \mathcal{A}s, C$ and

- $GP'_{\mathcal{O}} = \{\langle x[t], \perp^{\mathcal{O}}\rangle | x[t] \in GP\}$,
- $GP'_{\mathcal{A}} = \mathcal{A}s$,
- $TCs' = TCs \cup C$,

or $\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle \models_{sGD} \perp$ and

- $GP'_{\mathcal{O}} = GP_{\mathcal{O}}$,
- $GP'_{\mathcal{A}} = GP_{\mathcal{A}}$,
- $TCs' = TCs$.

Note that, for ease of presentation, in the case in which the underlying $\models_{sGD}$ capability is successful, we are assuming that the transition completely rewrites $GP_{\mathcal{O}}$ and $GP_{\mathcal{A}}$ in the original state. This is obviously wasteful as, in case some goals and actions (to be aimed for or avoided) remain the same, any prior planning effort for them is lost.

4.5.2 The SR^+ transition

Informally this transition revises the agent's state by revising all types (personal, reactive, social) of the agent's goals and actions and the record of what the agent should socially avoid. It ensures that the resulting state does not contain incompatible goals and or actions. It also ensures that the state does not contain unnecessary or unachievable goals, or unnecessary or unexecutable actions. In effect it revises the agent's commitments by deleting from the state:

- all goals and actions that have low priority for the agent according to its preferences, and
- all goals and actions that are timed out, and
- all goals and actions that are no longer necessary, because they have been achieved or executed respectively, or they are children or siblings of goals or actions that have low priority or are no longer necessary or are timed out.

Formally:

$$(\mathbf{SR}^+) \qquad \frac{\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle}{\langle KB, GP'_p, GP'_r, GP'_{\mathcal{O}}, GP'_{\mathcal{A}}, TCs\rangle}\,\tau$$

where $\langle KB, GP_p, GP_r, GP_{\mathcal{O}}, GP_{\mathcal{A}}, TCs\rangle \models_{rev} GP, \mathcal{A}s$, and

- $GP'_{\mathcal{A}} = GP_{\mathcal{A}} \cap \mathcal{A}s$
- $GP'_p \cup GP'_r \cup GP'_{\mathcal{O}}$ is the biggest subset of $GP_p \cup GP_r \cup GP_{\mathcal{O}}$ consisting of all items (goals or actions) $X = \langle x[t], Y\rangle$ such that:

(i) $Y \in GP'_p \cup GP'_r \cup\ GP'_{\mathcal{O}} \cup \{\bot^{nr}, \bot^{r}, \bot^{\mathcal{O}}\}$, and

(ii)-(v) as for SR in the KGP model (see section 3.2)

(vi) $X \in GP$.

4.6 Normal Cycle Theory of N-KGP agents

The normal cycle theory of N-KGP agents needs to guarantee that certain high-level constraints are satisfied, and in particular that

- the transition AE is executed only after SR^+ has been executed
- the transition PI is executed only after SR^+ has been executed

- GI (introducing personal goals), RE (introducing reactive goals and actions) and *sGI* (introducing social goals and actions as well as goals and actions to be avoided) are executed in sequence, in this order, with no PI or AE interrupting the sequence.

These constraints on the order of transitions to be executed by N-KGP agents can be achieved by extending the normal cycle theory for ordinary KGP agents given in [32]. They ensure that:

- the agent does not spend time unnecessarily planning for low priority goals and executing low priority actions,
- when it generates its social goals and actions, the agent takes into account an up-to-date view of its personal and reactive goals and actions.

5 An Example

In this section we illustrate the use of the proposed formalisation of normative concepts for example 1 discussed in section 2. To formulate this example we assume that the cycle theory of agent *John* first selects the GI transition, generating the goal $upgraded\_computer(t)$, with t possibly constrained. This goal is planned for by calling the PI transition and using appropriate definitions for the EC *initiates* and *terminates* meta-predicates in KB_{plan}, e.g.

$$
\begin{aligned}
&initiates(buy(PC), T, upgraded\_computer) \leftarrow \\
&\qquad holds\_at(pc(PC), T) \\
&initiates(buy(HD), T, upgraded\_computer) \leftarrow \\
&\qquad holds\_at(hard\_drive(PC), T) \wedge holds\_at(have(PC), T) \wedge \\
&\qquad holds\_at(compatible(HD, PC), T) \wedge \\
&\qquad holds\_at(pc(PC), T) \wedge holds\_at(hard\_drive(HD), T)
\end{aligned}
$$

and assume *pc* is known to be a PC. Assume PI introduces in the state a plan *buy*(*pc*) for the initial goal. Then, GP_p would consist of $\{G = \langle upgraded\_computer(t), \bot^{nr}\rangle, \langle buy(pc), G\rangle\}$.

Suppose the cycle theory next selects the POI transition, adding to KB_0

$$observed(issued\_fine(''W129FGC'', £150), 11)$$

and then *sGI*. Assume that KB_{soc} contains the following rule:

$$
\begin{aligned}
&holds\_at(role(owner(C)), T) \wedge \\
&observed(issued\_fine(C, A), T) \\
&\Rightarrow obliged(act(pay\_fine, A), T') \wedge T' = T + 5
\end{aligned}
$$

Then, *sGI* will introduce a social action into $GP_{\mathcal{O}}$, which will become $\{\langle pay\_fine(£150, t'), \perp^{\mathcal{O}}\rangle\}$.

If KB_{rev} contains the rules given in section 4.3.2, and assuming that *John* only has £600 and that the *pc* costs £500, there is a conflict between personal and social goals, and, assuming *John* is responsible (as in section 4.3.2), the SR^+ transition would eliminate the plan for the original personal goal G from GP_p.

Then, PI could generate the alternative plan to buy a new hard disk.

6 N-KGP Agent Varieties

The KGP model allowed us to design heterogeneous agents with different profiles of behaviour and which could behave differently under different circumstances. This was enabled in the KGP model by the use of cycle theories [18, 32] that allowed the designer to specify dynamic preferences amongst the transitions. For example, punctual agents can be specified, preferring to execute actions whose times were running out in preference to their other activities. Also, focussed agents can be specified, preferring to focus on achieving one goal at a time thus maximising their chances of achieving at least some of their goals when it was not possible for them to achieve all of them.

The N-KGP model inherits this design advantage and broadens its scope by allowing us, in addition, to specify a variety of agents who differ in their social behaviours. Many different varieties can be captured by formalising appropriate rules in the KB_{rev} of agents. These rules can be designed so that the behaviour of the agent changes as the agent evolves and progressively perceives its environment and the other agents.

Let us call a *Socially Responsible* agent one which commits to fulfilling all its social obligations while avoiding its prohibitions (provided the two are consistent). For such an agent its KB_{rev} will contain the rules:

$$X1 \succ X2 \leftarrow X1 \in GP_{\mathcal{O}} \wedge X2 \in GP_p$$

$$X1 \succ X2 \leftarrow X1 \in GP_{\mathcal{O}} \wedge X2 \in GP_r$$

$$avoid(X[all], TC) \succ X[T] \leftarrow TC[all/T]$$

Let us call a *Selfish* agent one that respects its social obligations and prohibitions only when they do not interfere with its own personal and reactive goals and actions. Such an agent will have in its KB_{rev} the rules:

$$X1 \succ X2 \leftarrow X1 \in GP_p \wedge X2 \in GP_{\mathcal{O}}$$

$$X1 \succ X2 \leftarrow X1 \in GP_r \wedge X2 \in GP_{\mathcal{O}}$$

$$X[T] \succ avoid(X[all], TC) \leftarrow TC[all/T]$$

A completely *Socially Irresponsible* agent, namely one which completely disregards any social obligations and prohibitions, is very easy to capture. This

can be done, for example, by giving the agent an empty knowledge base for its KB_{soc} module of KB.

The rules in KB_{rev} can be enriched by additional conditions. For example, following [25, 27, 26, 24], we can attach a utility value to each action of the agent and the relation $\succ$ can take such utility values into account. Also, again following the approach of [25, 27, 26, 24], it is straightforward to associate with each obligation a reward for fulfilling it and with each prohibition a sanction for breaking it and use these in the definition of $\succ$.

7 Related Work

Computational logic approaches to specify the rules of the interaction amongst social agents is an active subject of research. The existing society model [2] of PROSOCS, for example, is concerned with the specification of public protocols for communicating agents. That model provides a general language for specifying agent-protocols using the notion of expectations as well as a tool that supports verification of compliance for these protocols. Our work presented here seeks to complement this by allowing agents to become aware of the society's norms and analyse and reason with them according to their own individual circumstances, as normally happens in human societies and would be useful to incorporate in artificial societies.

Kakas et. al [1] also describe a modification of the KGP model that allows preference policies in some of the knowledge components of the agent. They concentrate primarily on the planning component and a new component, not unlike reactivity in the KGP model, which they call the negotiation component. This latter component caters for negotiation in a 2-agent setting. They propose logic programming with priorities [30] for the preference reasoning, but do not give much detail on, for example, how this is used to choose amongst multiple plans. We share with them the aim of incorporating preference policies in various components of the agent model, however one of our primary aims in our work is to extend the KGP model to integrate social norms and normative reasoning within the agent model.

The work described in [5] is also based on computational logic and consists of a theoretical framework for providing executable specifications of particular kinds of multi-agent systems, called open computational societies. Three key components are introduced: a social state, social roles, and social constraints. The specification of these concepts is based on and motivated by the study of legal and social systems, and a representation of deontic concepts in a version of the event calculus is presented.

Our framework differs from that of [5] in that we focus on how to reason about social constraints and roles within a single agent using an abductive interpretation of the event calculus. In other words, we do not take an abstract "bird's eye view" of the interaction between agents but a concrete "agent's view", trying to interpret social constraints from the standpoint of a single agent (using the agent architecture resulting from the KGP model) and further

showing how such constraints can be used during agent deliberation. The advantage of using KGP is that we inherit all the existing reasoning tools and proof procedures supported by the model.

The IMPACT system [3] incorporates obligations and other deontic notions such as permission and prohibition. In common with us they use abductive logic programs as the representation language. But IMPACT incorporates a more extensive theory of deontic concepts, as well as rules for allowing the utilisation of legacy systems. The KGP model, and its extension described here, have different aims compared to IMPACT. We aim at building agents that can plan partially, interleave planning, acting and observations, weigh up their social goals against the personal ones, and use flexible cycle theories that can be designed to provide specific profiles.

Our work shares some of the objectives of Lopez et al (see for instance [24, 26, 25, 27]). Part of their work, like ours concerns the modelling of how agents weigh up their social obligations and personal goals to decide their social behaviour. They base their work on the SMART agent model of [13] and specify the extensions, including norms and normative multi-agent systems using the specification language Z.

They introduce a broad range of concepts, such as motivation, punishment, reward, and give brief Z specifications. Like us they use a form of preferences on goals which they call importance values, which are associations of numerical values to goals on the basis of the agent's motivation. They aim at modelling agents which evaluate the positive and negative effects of norms on the achievement of their private goals in order to decide whether or not to join a society and whether or not to adopt the norms of the society after joining.

Our work is less broad in scope than theirs but we go deeper in the specification of norms and their integration with the capabilities of the KGP agents. Our agents reason with abductive logic programs and the event calculus to decide their roles in society and accordingly their obligations and prohibitions. They then make decisions about whether or not to adopt these amongst their goals in a system that fully integrates norm adoption with planning, temporal reasoning, reactivity and adaptation to the environment sensed through observations and agent-to-agent interaction.

The BOID architecture presented in [8] extends the well known BDI model [31] with obligations, thus giving rise to four main components in representing an agent: beliefs, obligations, intentions and desires. The idea of BOID is to find ways of resolving amongst these components. In order to do so they define agent types including some well known types in agent theories such as realistic, selfish, social and simple minded agents. The agent types differ in that they give different priorities to the rules for each of the four components. For instance, the simple minded agent gives higher priority to intentions, compared to desires and obligations, whereas a social agent gives higher priority to obligations than desires. They use priorities with propositional logic formulae to specify the four components and the agent types.

What we have in common with BOID is that we want to extend our model with the addition of obligations. The existing KGP model already resolves some

of the conflicts that they address. For example, if there is a conflict between a belief and a prior intention, which means that an intended action can no longer be executed due to the changes in the environment, the KGP agent will notice this and will give higher priority to the belief than the prior intention, allowing the agent in effect to retract the intended action and, time permitting, to replan for its goals. The KGP model also includes a notion of priority used in the goal decision capability and the cycle theory that controls the behaviour of the agent. The N-KGP model extends the notion of priorities by incorporating them amongst different types of goals and actions.

One of the aims of the KGP model and the extension discussed in this paper, and one of the differences with the work of [8], is to allow the interleaving of reasoning about obligations, planning to achieve them and other goals and recording and utilising observations. This makes our work closer to the more recent work on BOID, see for instance [11], by providing an executable specification of the obligations in an abductive logic programming setting.

8 Concluding Remarks

We have shown how to extend the logical model of agency, known as the KGP model, to support agents with normative concepts. By using obligations and prohibitions as an example, the proposed framework has shown how to specify an agent that can reason using normative concepts and combine them with its own personal goals in order to plan, adapt to its environment and decide which actions to perform next.

Unlike approaches that are based on a monolithic tool for checking social interactions, one advantage of our work is that obligations and prohibitions can be represented and utilised within individual social agents, thus making the extended model suitable for building multi-agent systems applications whose organisation is based on artificial and open societies of agents.

This work opens up many promising areas of future work. These include incorporation of other norm-related concepts, such as permissions, rewards and sanctions, and generally richer normative theories, and implementation of the extended model in the PROSOCS platform to experiment with concrete applications.

Acknowledgments

We are grateful to the anonymous referees, as well as to all participants of NorMAS 2005, for useful comments on an earlier version of this paper.

References

[1] N. Demetriou A.C. Kakas, P. Torroni. Agent planning, negotiation and control of operation. In *European Conference on Artificial Intelligence*

(ECAI04), 2004.

[2] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Compliance verification of agent interaction: a logic-based tool. In Robert Trappl, editor, *Proceedings of the 17th European Meeting on Cybernetics and Systems Research, Vol. II, Symposium "From Agent Theory to Agent Implementation" (AT2AI-4)*, pages 570–575, Vienna, Austria, April 13-16 2004. Austrian Society for Cybernetic Studies.

[3] K. A. Arisha, F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus. IMPACT: a Platform for Collaborating Agents. *IEEE Intelligent Systems*, 14(2):64–72, March/April 1999.

[4] A. Artikis and J. Pitt. A formal model of open agent societies. In J. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of Conference on Autonomous Agents (AA)*, pages 192–193. ACM Press, 2001.

[5] A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In C. Castelfranchi and L. Johnson, editors, *Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1053–1062. ACM Press, 2002.

[6] A. Bracciali, N. Demetriou, U. Endriss, A. Kakas, W. Lu, P. Mancarella, F. Sadri, K. Stathis, G. Terreni, and F. Toni. The KGP Model for Global Computing: Computational Model and Prototype implementation. In *Global Computing*, LNCS. Springer-Verlag, 2005.

[7] G. Brewka. Reasoning with priorities in default logic. In *Proceedings of AAAI-94, pp. 940-945*, 1994.

[8] J. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. van der Torre. The BOID architecture: conflicts between beliefs, obligations, intentions and desires. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 9–16, Montreal, Canada, 2001. ACM Press.

[9] C. Carabelea, O. Boissier, and C. Castelfranchi. Using social power to enable agents to reason about being part of a group. In *Pre-proceedings of ESAW'04*, Toulouse, October 2004.

[10] C. Castelfranchi, F. Dignum, C. M. Jonker, and J. Treur. Deliberative normative agents: Principles and architecture. In *Agent Theories, Architectures, and Languages*, pages 364–378, 1999.

[11] M. Dastani and L. van der Torre. Programming boid agents: a deliberation language for conflicts between mental attitudes and plans. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'04)*, 2004.

[12] F. Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999.

[13] M. d'Inverno and M. Luck. *Understanding agent systems*. Springer-Verlag, 2nd edition, 2003.

[14] Michael N. Huhns and Munindar P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, San Francisco, CA, USA, 1998.

[15] J. Jaffar and M.J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20:503–582, 1994.

[16] A.J.I. Jones and M.J. Sergot. On the characterisation of law and computer systems: the normative systems perspective, 1993. Deontic Logic in Computer Science: Normative System Specification. John Wiley and Sons, Chicester (1993) 275–307.

[17] A.J.I. Jones and M.J. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, June 1996.

[18] A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. Declarative agent control. In Leite J. and Torroni P., editors, *Proceedings CLIMA'04, 5th International Workshop on Computational Logic in Multi-Agent Systems*, Lisbon, Portugal, Sep. 2004.

[19] A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *European Conference on Artificial Intelligence (ECAI04)*, pages 33–39, 2004.

[20] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press, 1998.

[21] A. C. Kakas and P. Moraitis. Argumentation based decision making for autonomous agents. In J. S. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo, editors, *AAMAS 2003*, pages 883–890, Melbourne, Victoria, July 14–18 2003. ACM.

[22] R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[23] R.A. Kowalski and F. Toni. Abstract argumentation. *Artificial Intelligence and Law Journal, Special Issue on Logical Models of Argumentation*, 4:275–296, 1996.

[24] F. Lopez y Lopez and M. Luck. Towards a model of the dynamics of normative multi-agent systems. In *International workshop on Regulated agent based social systems: theories and applications (RASTA '02)*, pages 175–193, 2002.

[25] F. Lopez y Lopez and M. Luck. A model of normative multi-agent systems and dynamic relationships. In M. Paolucci G. Lindemann, D. Moldt, editor, *Regulated agent-based social systems*, Lecture notes in AI, 2934, pages 259–280. Springer, 2004.

[26] F. Lopez y Lopez, M. Luck, and M. d'Inverno. Constraining autonomy through norms. In *Proceedings of the 1st Conference on Autonomous Agents and Multiagent Systems (AAMAS'01)*, pages 674–681, 2002.

[27] F. Lopez y Lopez, M. Luck, and M d'Inverno. Normative agent reasoning in dynamic societies. In *Proceedings of the 3rd Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, pages 259–280, New York, 2004.

[28] Julian A. Padget, editor. *Collaboration between human and Artificial Societies: Coordination and Agent-based Distributed Computing.* Springer, LNAI 1624, 2001.

[29] H. Prakken and G. Sartor. A system for defeasible argumentation, with defeasible priorities. In *Proc. International Conference on Formal and Applied Practical Reasoning*, volume 1085 of *LNAI*, pages 510–524. Springer Verlag, 1996.

[30] Henry Prakken and Giovanni Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 7(1), 1997.

[31] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[32] F. Sadri and F. Toni. Profiles of behaviour for logic-based agents. In *Proceedings of CLIMA VI*, 2005.

[33] K. Stathis, A. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: a platform for programming software agents in computational logic. In J. Müller and P. Petta, editors, *Proceedings of the Fourth International Symposium "From Agent Theory to Agent Implementation"*, Vienna, Austria, April 13-16 2004.

[34] F. Toni and K. Stathis. Access-as-you-need: a computational logic framework for flexible resource access in artificial societies. In *Proceedings of the Third International Workshop on Engineering Societies in the Agents World (ESAW'02)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2002.