SOCS

A COMPUTATIONAL LOGIC MODEL FOR THE DESCRIPTION, ANALYSIS AND VERIFICATION OF GLOBAL AND OPEN SOCIETIES OF HETEROGENEOUS COMPUTEES

IST-2001-32530

Deliverable D5: A logic-based approach to model interaction amongst computees

Project number:	IST-2001-32530
Project acronym:	SOCS
Document type:	DN (discussion note)
Document distribution:	I (internal to SOCS and PO)
CEC Document number:	IST32530/UNIBO/010/DN/I/a23
File name:	5010-a23[modelling-interactions].pdf
Editor:	Paola Mello
Contributing partners:	UNIBO, DIFERRARA, ICSTM, DIPISA, UCY, CITY
Contributing workpackages:	WP2
Estimated person months:	50
Date of completion:	25 June 2003
Date of delivery to the EC:	30 June 2003
Number of pages:	95

ABSTRACT

Computees are Computational Logic-based entities interacting in the context of global and open computing environments. The focus of this work is on the interactions among computees that form a society, and on the definition of a Computational Logic-based architecture for computee interactions. We propose a layered architecture where the society defines the allowed interaction protocols, which determine the "socially" allowed computee interaction patterns. The semantics of protocols and of the computee communication language are defined in a uniform way by means of Social Integrity Constraints. The main advantages of this approach are in the design of societies of computees, in the possibility to detect undesirable behaviour, and in the possibility to formally reason upon the specifications of such a framework, and to understand its own limits and potential. We show our ideas in different scenarios.

Deliverable D5: A logic-based approach to model interaction amongst computees

P. Mello[1], P. Torroni[1], M. Gavanelli[2], M. Alberti[2], A. Ciampolini[2], M. Milano[1], A. Roli[1], E. Lamma[2], F. Riguzzi[2], and N. Maudet[3]

[1] DEIS, University of Bologna
Email: {pmello|ptorroni|aciampolini|mmilano|aroli}@deis.unibo.it

[2] DI, University of Ferrara
Email: {mgavanelli|malberti|elamma|friguzzi}@ing.unife.it

[3] DoC, Imperial College
Email: maudet@doc.ic.ac.uk

ABSTRACT

Computees are Computational Logic-based entities interacting in the context of global and open computing environments. The focus of this work is on the interactions among computees that form a society, and on the definition of a Computational Logic-based architecture for computee interactions. We propose a layered architecture where the society defines the allowed interaction protocols, which determine the "socially" allowed computee interaction patterns. The semantics of protocols and of the computee communication language are defined in a uniform way by means of Social Integrity Constraints. The main advantages of this approach are in the design of societies of computees, in the possibility to detect undesirable behaviour, and in the possibility to formally reason upon the specifications of such a framework, and to understand its own limits and potential. We show our ideas in different scenarios.

Contents

1 Introduction

Computees are Computational Logic-based entities interacting in the context of global and open computing systems [KSST03]. They are abstractions of the entities that populate Global Computing environments [GC]. These entities can form complex organizations, that we call *Societies Of ComputeeS* [SOC]. The main objective of Global Computing, rephrased in terms of SOCS, is to provide a solid scientific foundation for the design of societies of computees, and to lay the groundwork for achieving effective principles for building and analyzing such systems.

Computees are autonomous computational entities, autonomous in the sense that their inner activity is not externally controlled. Computees have their own knowledge, capabilities, resources, objectives and rules of behavior. Each computee typically has only a partial, incomplete and possibly inaccurate view of the environment and of the other computees, and it might have inadequate resources or capabilities to achieve its objectives. We do not make assumptions on the location or movements of computees, and we propose a model of societies which is not based on an a-priori determined rigid configuration.

In our approach, we believe that the knowledge and technologies acquired so far in the area of Computational Logic provide a solid ground to build upon. In the companion document D4, reporting on the activity done during the first year of the project with respect to Workpackage 1 ("A logic-based model for computees") [KSST03], it is possible to see how computees, in order to achieve their objectives and exploit their knowledge, are provided with reasoning functionalities (such as deduction, abduction, reasoning with priorities, etc.) and Computational Logic-based capabilities (such as planning, goal decision, temporal reasoning, and so forth). The role of Computational Logic at the level of the single computee is therefore to provide a formalism and an operational semantics to its capabilities and to its life cycle which activates such capabilities.

At the society level, the role of Computational Logic is to provide both a declarative and an operational semantics to interactions. The advantages of such an approach are to be found:

(i) in the design and specification of societies of computees, based on a formalism which is declarative and easily understandable by the user;

(ii) in the possibility to statically analyze the behavior of the society and of its individuals, based on the properties that such a framework allows to prove;

(iii) in the possibility to detect undesirable behavior, through *on the fly* control of the system based on the computees' observable behavior (communication exchanges) and to dynamically check the conformance of such behaviour with the constraints posed by the society;

(iv) in the possibility to understand its own limits and potential, through the study of verified properties which will help to define the application domains of our results.

This document presents a framework based on Computational Logic for modelling interaction among computees in an open environment, and summarizes part of the work done during the first one and a half year of the project, with respect to Workpackage 2 ("Modelling interaction amongst computees").

The main features of the framework here presented are:

(i) the use of Computational Logic to model and give semantics to interactions; and

(ii) the use of a uniform formalism (Social Integrity Constraints) for expressing both protocols and "social" semantics of communication languages.

1.1 Dealing with an open environment

The society formal model should satisfy the following high-level objectives derived directly from the GC vision of an open and changing environment.

There are several notions of openness that could be adopted for societies of computees. In the following we will take into consideration the two most widely accepted definitions of openness in the agent community. According to [Dav01], in an open (artificial) society "there are no restrictions for agents/processes to join/leave the society": this means that it is possible for any agent to enter the society simply by starting an interaction with a member of it. This is also the acceptation adopted in the *Agentcities* initiative [Age], whose aim is to increase the commercial and research potential of agent-based applications by constructing a worldwide, open network of platforms hosting diverse agent-based services.

Another widely accepted definition of openness in agent societies is that given in [APS02] (derived from [Hew91]) where an agent society is open if three properties hold:

(i) the behavior of members and their interactions are unpredictable (i.e., the execution of the society is non-deterministic);

(ii) the internal architecture of each member is neither publicly known nor observable (i.e., members may have heterogeneous architectures);

(iii) members of the society do not necessarily share common goals, desires or intentions (i.e., each member may conflict with others when trying to reach its own purposes).

The two definitions are based on different notions, but they imply similar effects on the society features. The first definition of openness is more focused on the society membership, and therefore it necessarily implies that we have to take into consideration the way an agent can enter/leave the society: if a society can be entered/left without any restriction, we call it open. In particular, according to [Dav01], in addition to openness, an agent society may need to support (some of) the following properties: *flexibility*: the degree to which agents are restricted in their behavior by the society; *stability*: the predictability of the consequences of actions; *trustfulness*: the extent to which agents trust the society. Typically those properties are not independent of each other. Since entering an open society is completely free, in principle this does not impose any monitoring/constraining infrastructure to the society. For these reason, as observed in [Dav01], such kinds of "open" society are often unstable and untrustful.

The second definition of openness [APS02] is more based on externally observable features within the society. It implies that members could be heterogeneous, and possibly non-cooperative. Thus, in this kind of open society, as in the one proposed by [Dav01], trustfulness of members and of the society itself could be low. Moreover, again as in [Dav01], being the society behavior non-deterministic, this could cause a low stability as an additional property.

With reference to SOCS objectives, we propose a model of the society where:

- the way computees enter/exit the society is not necessarily constrained;
- the presence of heterogeneous computees within societies is emphasized;
- it might not always be guaranteed that the internal structure of computees is observable, and that their social behaviour is predictable.

We are interested in modelling societies by specifying a social knowledge which possibly expresses some constraints on members' social behavior.

We are also interested in expressing a notion of *social goal*, and in supporting both goal-directed and non-goal-directed societies. For all these reasons, the kind of agent society we will consider is more adherent to the definition of openness given in [APS02].

1.2 Modelling interactions amongst computees

SOCS has many aspects in common with the current work on Multi-Agent Systems. One such model is [APS02], that we cited above. Societies of agents and agent interactions have been widely studied, and models have been proposed which offer several advantages in certain domains.

In this report, we carefully put SOCS into relationship with the agent literature, with the purpose to single out the original contribution of our work with respect to such a conspicuous and rich material. In this introduction we briefly motivate our approach towards agent interactions. Our arguments are:

- the use of a declarative and understandable representation,
- a uniform medium to understand aspects of interactions so far treated separately and thus disconnected,
- an operational model to pave the way to an implementation of societies of computees based on the declarative specifications,
- a bridge built to verification and formal proof of properties, and
- a link to the model of individual entities in the society.

Agent Communication Languages (ACL) and Conversation Protocols (CP) are the traditional approaches to support interactions among software agents. The semantics of speech acts in ACL is customarily defined in terms of mental attitudes such as beliefs, desires and intentions. This approach has been criticized as inadequate for open environments [Sin98] since agents cannot verify whether the private beliefs of other agents comply with speech act definitions without pre-established constraints on how agents are internally implemented.

On the other hand, CP are static structures that define the sequences of utterances making a coherent conversation. This approach has been criticized for its lack of flexibility, i.e., the lack of compositional rules governing how protocols are extended or merged. An additional shortcoming is that ACL and CP are defined independently of each other. We have tried to combine them in a way that recognizes the dependencies between the two.

In our approach, we define the semantics of the protocols and the communication language of the computees as integrity constraints over social events (e.g., communicative acts), which caters for heterogeneity of computees and openness of societies, since it makes no assumptions on the internal structure of the computees. Being the present report a companion work of D4 [KSST03], from now on we will call the communication language of the computees "Computee Communication Language" (CCL).

The formalism adopted for protocols and CCL is based on Social Integrity Constraints (IC_S) which express *constraints* on the communication patterns of computees, and therefore determine *expected* communicative acts, on the basis of the history of social events.

Building on previous work on abductive logic-based agents [KS99, STT02a, CLMT02], we define the society's knowledge assimilating it to abductive logic programs [CP86, EK89, KM90]:

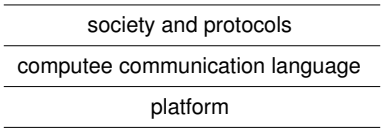

Figure 1: The multi-layered architecture

we define a notion of expected social events, and express them as abducible predicates, while using IC_S to constrain the "socially admissible" communication patterns of computees. The syntax of IC_S and of the society in general are those of an extended logic program, and the operational semantics will draw inspiration from that of abductive frameworks such as the IFF proof-procedure by Fung and Kowalski [FK97]. In this way, we address a foundational aspect and a major engineering problem, allowing for different kinds of verification: static, dynamic, based on outside observation of the computees communication exchanges, based on knowledge on the computee internals. This is, we believe, one of our main innovations in the theory and design of systems of autonomous entities in the Global Computing environment.

1.3 Basic architecture of the society

With this report we want to commit to a model of interactions which the other Workpackages will build upon later on in the project. The objectives of our design are on the one hand the possibility to capture violations and incorrect social behavior of computees, and possibly to recover from a state of failure, on the other hand to give the society the means to be pro-active in helping the computees behaving in a socially correct way, for instance, by suggesting to the computees what their expected social acts are, or by showing them the possible social consequences of their actions. Some of those objectives are met at this stage of the work and some others are subject for current and future investigation.

We propose a multi-layered architecture to model and implement a society of computees. We rely on a functional definition of a society, implemented through a management infrastructure that supports the definition of constraints on communicative acts (protocols), and possibly roles and operations for joining and exiting a society.

We would like to stress here that the architecture we propose is at an abstract level. In particular, when we talk about "society", we do not suggest that there is any global entity that contains all protocols and "social" information of all sorts. Our "society" is instead an abstraction, which could be in practice implemented by one or several computees, or by any other suitable entity (which we will refer to, in the following, as *social infrastructure*): the choice will depend on the underlying algorithms and on the specific application.

The architecture is composed of three layers (see Figure 1):

(*iii*) Society and Protocols

(*ii*) Computee Communication Language

(*i*) Platform

At the bottom level we put the *platform* used to implement the system, which supports low-level communication among computees. The platform layer is beyond the scope of this report, as it is relevant to Workpackage 4 of the project.

On top of the platform, we have the *CCL*, which defines the syntax and semantics of communicative acts, still specified via IC_S. The CCL layer equips computees with means for exchanging information and knowledge, and provides formalisms for communication handling propositions, rules, and actions instead of simple objects with no associated semantics. What is handled by the CCL is related to the single computee formal model (see [KSST03], Section 10, and Section 10 of this document).

The topmost level (*Society and Protocols*) defines the social knowledge, and provides formalisms to rule computees' interactions. It can specify entrance/exiting rules, relationships between roles and the expected behaviour of computees, with respect to the roles that they cover. Protocols formalize social rules of interaction, by posing constraints on the computees' communicative acts and on their sequences.

In defining our model of interaction, and mainly in defining the syntax and the expressive power of IC_S, we started from different case studies. Therefore, along with the "horizontally" layered structure of the interaction architecture, we have a "vertical" notion of *scenario*, which we use to put things into context and explain the ideas by means of running examples. In this report we use three examples.

The first example is inspired by [STT02a], which proposes a negotiation solution to a resource reallocation problem, presenting a framework based on Abductive Logic Programming. In this scenario, computees are the entities holding the resources.

The second example is based on combinatorial auctions. Auctions are an important way of allocating items among autonomous and self-interested entities. In combinatorial auctions [Nis00, San02], bidders can bid on combinations of items and associate a price to each combination. The auctioneer should solve the winner determination problem, i.e., it should choose the best bids that cover all items at the minimum price. In this report, we briefly show how to model the communication protocol of a society implementing the combinatorial auction, where bidders and auctioneers are modeled as computees. We also show how it is possible to capture possible deviations from the proper/expected behavior of a participant (bidder or auctioneer).

The third example, NetBill [CTS95], is a simple security and transaction protocol designed for ruling selling and delivery of low-priced information goods (such as software or journal articles) over the Internet. It manages all the phases of a complete transaction: price negotiation, ordering, delivery of goods, payment and delivery of a receipt of payment. It relies on a NetBill server as an information store and authentication server for sellers and buyers and as an interface to traditional financial entities (such as bank accounts or credit cards).

1.4 Dissemination of results

The present document, D5, reports on both previously published and original work. Among the conferences and workshops where parts of D5 have been or will soon be presented, we cite: the *UK Multi-Agent Systems Annual Conference* (UKMAS) [TMM+02], the *International Workshop on Formal Approaches to Multi-Agent Systems* (FAMAS) [ACG+03a], the *3rd International Central and Eastern European Conference on Multi-Agent Systems* (CEEMAS) [ACG+03b], the *Workshop on Logic and Communication in Multi-Agent Systems* (LCMAS) [AGL+03c], the *First International Workshop on Declarative Agent Languages and Technologies* (DALT) [AGL+03b], the *18th International Joint Conference on Artificial Intelligence* (IJCAI) [STT03], and the *8th National Congress on Artificial Intelligence* (AI*IA) [AGL+03a]. In particular, in [TMM+02] a layered architecture for societies of computees (Section 1.3) has been proposed, where at the bottom level a platform is used to implement the system and give

support to computees' communication, and a communication language layer defines syntax and semantics of communicative acts, while society and protocols are in a higher layer. A social semantics for communicative acts (Section 5.3) has been presented in [ACG+03b], along with a discussion about the advantages and motivation of a social semantics of communication with respect to other approaches, and in [ACG+03a] an implementation of a restricted class of IC_S is proposed, based on the CHR language [Frü98]. [AGL+03a] defines the full syntax of IC_S, the scope of variables, quantification, and gives some results about the conditions for a proper behaviour of the framework, along with a formal declarative semantic characterization of concepts such as coherence and consistency of sets of expectations and their fulfillment (mainly Section 3.3 and Section 4.1). [AGL+03b] shows the practical use of our theoretical framework, by means of a simple though realistic case study of resource sharing (Section 8.1). The protocols are taken from [STT03], where the authors introduce a formalization of dialogues, policies and protocols, and discuss about conformance of policies to protocols. Finally, [AGL+03c] focusses on formal and automatic verification of properties of interactions and protocols (Section 6).

1.5 Organization of the report

This work is structured as follows:

2 Societies of Computees - In this section, after giving some background notions on the modelling of agent societies, we define the social knowledge base and we introduce the notions of *events* and *expectations*;

3 Protocols - In this section, we survey background work on protocols, we introduce the syntax of IC_S, and we show how we can model roles and manage membership;

4 Semantics of the social framework - In this section, we give the declarative semantics of our model of society and its operational counterpart, and with the help of examples we show the "evolution" of expectations during the life of a society of computees. We discuss some aspects of our framework related to temporal reasoning, and we conclude by giving an abductive interpretation of our framework;

5 Computee Communication Language - In this section, after briefly surveying the main proposals for Agent Communication Languages, relevant to our approach, we introduce the basic ideas on the CCL and its semantic characterization via IC_S.

6 Formal verification of properties - In this section, we elaborate on the use of our framework to automatically prove properties of interactions, such as compliance to protocols;

7 Extensions - This section preliminarily studies some extensions of the framework that could be developed in future work. We discuss about violation and recovery from states of violation, protocol learning, trust management, and about our approach to the study of emerging behaviour of societies of computees;

8 Examples - In this section we introduce some simple examples, used as application scenarios: the resource reallocation problem, combinatorial auctions and the NetBill protocol;

9 Related work - In this section, we discuss our work in relationship with the state of the art in agent societies, protocols and ACL;

10 Discussion and evaluations of objectives - In this section, we conclude by evaluating our work with respect to the Technical Annex and to deliverable D3 [LMM$^+$03], and we discuss it in relationship with other Workpackages.

Three appendices complete the deliverable, by giving some additional information and background on Logic Programming:

Appendix A Background - In this section, we give some background on Abductive Logic Programming, as it is used within this report.

Appendix B Comparing IC_S with the IC of the IFF proof-procedure - In this brief section, we elaborate on some differences and similarities between syntax and restrictions of IC_S and those of the integrity constraints handled by Fung and Kowalski's IFF proof-procedure for abductive reasoning [FK97].

Appendix C Learning - In this section, we provide some background on Inductive Logic Programming which can help the reader in better understanding its exploitation to learn social behavior in societies of computees

2 Societies of computees

In this section, we give some background notions on agent society modelling, and we define the syntax of our model of societies of computees.

2.1 Modelling agent societies

Several approaches to agent society modelling can be found in the Multi-Agent Systems literature. The earliest attempts to model a society of agents or interacting computational entities have their roots in the Distributed Problem Solving (DPS) area, further extended towards contract networks and market models. Different perspectives lead to dependency network models, Deontic Logic-based models and models based on modal and/or temporal logics (in particular, extensions of the Belief, Desire, Intention - BDI - architecture [RG92a]). Another interesting approach is that of organizational models which define societies as the actual (and dynamic) composition of organizations. Organizations specify the roles agents will enact and their interactions by means of protocols. Finally, we should mention bottom-up models which focus on the spontaneous emergence of coalitions, cooperation and norms; for instance, economic and social-based multi-agent systems [Axe97] and artificial life [DFCF91]. In the following, we give a brief description of some of the most relevant approaches amongst the ones mentioned above for the purposes of our work.

Distributed Problem Solving - The need for a society model emerges in DPS when problems have to be addressed by coordinating an ensemble of computational entities. It hence follows the requirement for the definition of models, systems and tools for dealing with dynamic coalition formation, cooperation and competition.

In this context, a society is a collection of agents interacting (i.e., cooperating and/or competing) to solve the assigned tasks. The rules of the society define the actions that agents are allowed to take and the allowed interaction patterns. Two typical examples of this kind of societies are those based on market models and contract networks.

Market models - Market models [WW98, Wel93] use the economic metaphor of market, where agents sell and buy goods and services. The society is thus defined as a market and the equilibrium reached after the economic transactions is the result of the computation performed by the society. Market-oriented programming has been mainly applied to tackle resource allocation problems. Market oriented models can be considered as a computational framework rather than an actual model for societies: in fact, they are more focused on the computational process, than on the definition of social interaction.

Contract Net model - The Contract Net model (CN) [IS00, Smi80] is strongly related with market models, and specifies a dynamic organization model based on the notion of negotiation in an auction scenario. In the CN model, contracts define sets of tasks to be accomplished. Agents need to perform some tasks and have some capabilities. Tasks are performed by agents having the required capabilities. The negotiation is aimed at assigning tasks to the agents which have the required capabilities to fulfill them. The structure of the negotiation is the following:

- Announcement: each agent announces the tasks that need to be carried out.
- Bid: agents make bids to perform tasks announced by other agents.
- Award: bids are evaluated and contracts are awarded.

Even though the CN model presents more structured agent interaction patterns, it essentially lacks an organization definition, since it mainly consists of a mechanism for task distribution.

Dependence-based models - The concepts of social power and dependence lead to the formalization of dependence-based models [Sic01, SCDC98]:

> "Briefly, we can say that an agent *depends* on another one if the latter can facilitate/prevent the former to achieve one of his goals. We can also say that in this case the second agent has *power* on the first one."

Dependence among agents has been formalized and different types of dependence have been formally defined (see, for example, [Sic01, SCDC98]). By means of these models, it is possible to define dependence networks (which describe the relations among agents by explicit dependence relations) and social reasoning mechanisms can be provided. A notion of agent consistency based on dependence networks is introduced in [SD95, SD01]. This issue is related to the work being done on the model of the single computee (WP1), since it is based on an individual perspective of beliefs.

BDI architectures - A milestone work in Multi-Agent Systems is that of Belief-Desire-Intention (BDI) architectures [RG92a]. BDI architectures are based on the assumption that agents have an internal representation of the world and of the *mental states* of other agents. This representation is used to enable an agent to *reason* about the other agents. A formal modelling of a society, which extends the BDI architecture to societies, can be found in [PNJ99]. BDI-like models suffer from a main problem which is very hard to overcome in scenarios involving heterogeneous and autonomous computational entities [Sin98]:

> "It appears to be repeating the past mistake of emphasizing *mental agency* – the supposition that agents should be understood primarily in terms of mental concepts, such as beliefs and intentions. It is impossible to make such a semantics work for agents that must be autonomous and heterogeneous. This approach supposes, in essence, that agents can read each other's minds. This supposition has never held for people, and, for the same reason, it will not hold for agents."

Organizational models - Among the various organizational models available in the literature, we briefly describe one which we consider of particular interest for the purposes of this project.

In [DMDW02, DMWD02, DMW02] an organizational model is defined, based on a framework which consists of three interrelated models: organizational, social and interaction. The *organizational model* defines the coordination and normative elements and describes the expected behavior of the society. Its components are roles, constraints, interaction rules, and communicative and ontology framework. The *social model* specifies the contracts that make explicit the commitments regulating the enactment of roles by individual agents. Finally, the *interaction model* describes the possible interactions between agents by specifying contracts in terms of description of agreements, rules, conditions and sanctions. In this organizational model, Deontic Logic is used to specify the society norms and rules.

Institutional approach - We conclude this brief outlook on society modelling by mentioning the *institutional* approach, which deals with the definition of agent interactions. This approach supposes that the situations where computees interact may involve commitments, delegation, repetition of interactions and risk. A suitable model to establish and enforce conventions is that of *institutions* [EdlCS02, NS02]. The basic aim of an institution is to facilitate, oversee and enforce commitments among agents.

In our model of society, we draw inspiration from [DMWD02], and we aim at proposing an architecture that fulfills the following requirements:

- it should explicitly specify the organizational and normative elements of the society since an open society cannot rely on its embedding in the intentions, desires and beliefs of each computee;
- it should include formalisms for the description, construction and control of the organizational and normative elements of a society (roles, norms and goals) instead of computee beliefs and states;
- it should provide mechanisms to describe the environment of the society (e.g., in terms of interactions between computees) and the society, and make it possible to formalize the expected outcome of roles in order to verify the overall animation of the society;
- it should provide building directives concerning the communication capability and ability to conform to the expected behavior of computees in the society.

2.2 Social expectations

The idea of social expectations stems from our intention to meet the requirements mentioned above (in particular, the possibility to specify normative elements), and the need to meet those of Global Computing (in particular, to provide a model which can be used in presence of incomplete information, or information that becomes available over time).

In our model, the society is time by time aware of social events that dynamically happen in the social environment (*happened* events). Moreover, it encodes the "normative elements" in what we call IC_S, as we will show below. Based on the available history of events, and on its specification of IC_S, the society can define what are the "expected social events" (that have not yet happened) and the social events that are expected *not* to happen. The expected events, from a normative perspective, reflect the "ideal" behaviour of the society. We call these events *social expectations*.

The idea of expected behavior can be considered related to *deontic logic*; however, our claim is that we do not need the full power of the standard Deontic Logic, but only constraints on events that are expected to happen or not to happen. Notice that we do not use deontic operators, but instead we map expectations into first-class predicates (**E** for positive and **NE** for negative expectations, see the next section).

The set of social expectations is adjusted when the society acquires new knowledge from the environment on social events that was not available at the time of generating such expectations. In this perspective, the society should be able to deal with *unexpected* social events from the environment, which violate the expectations, as it can be the case in an open environment where *regimentation* cannot be assumed.[1]

Apart from defining the correct behaviour of computees in a society, there could be other uses of expectations. Indeed, they could be used pro-actively by the society: suitable social policies could make them public in order to try and influence the behaviour of the computees, towards an ideal behaviour. On the other hand, happened events that were expected not to happen can raise mechanisms of recovery from violation (e.g., sanctioning computees that cause them), without preventing the society from continuing its operation.

2.3 Representation of society knowledge.

The knowledge in a society is represented by the following 4-tuple:

$$\langle SOKB, SEKB, \mathcal{IC}_S, \mathcal{G} \rangle$$

where:

- $SOKB$ is the *Social Organization Knowledge Base*,
- $SEKB$ is the *Social Environment Knowledge Base*,
- $\mathcal{IC}_S$ is the set of *Social Integrity Constraints* (IC_S), and
- $\mathcal{G}$ is the set of *Goals* of the society.

[1] Regimentation in Multi-Agent Systems is defined as the principle that actions which an agent is obliged to take are actually executed [Kro95]. This principle reflects the ideal behavior of an agent in a normative system [ESP99].

The **Social Organization Knowledge Base** ($SOKB$) defines structure and properties of the society, and has strong similarities with the organization level in [DMWD02]. Indeed, it defines the structural description of the society, possibly including, for instance, rules for joining/leaving the society, and role assignment.

The $SOKB$ can change from time to time, as, for example, norms may change during the life of the society due to learning or knowledge revision. Nevertheless, this knowledge can be seen as *static* since it describes the organization of a society which is constant in a specific time window, and which, however, changes more slowly than the $SEKB$ does.

The current instantiation of a society is described by the **Social Environment Knowledge Base** ($SEKB$), which takes into account occurred events and (positive or negative) expectations about social events. The $SEKB$ concerns the actual process of the life of the society (e.g., which computees take part to the society, their interactions, etc.). The $SEKB$ is a dynamic environment space for representing the environment as it is "perceived" by the society, and the expectations on the "observable" behavior of computees at society level. Note that the recorded events are only those that are observable and significant for the society.

While the $SOKB$ defines properties of the society, the $SEKB$ concerns with the instantiation of these properties. Indeed, given a society description, many instantiations are possible, each one characterized by a different $SEKB$. In particular, the $SEKB$ dynamically evolves and is composed of:

- *Observable* and *relevant* events for the society (*happened events*: atoms indicated with functor **H**);

- *Expectations* on the future: events that should (but might not) happen in the future (atoms indicated with functor **E**), and events that should not (but might indeed) happen in the future (atoms indicated with functor **NE**).

We often call *history* the set of facts concerning dynamic (happened) events; both the history and current expectations are represented as logical formulae (see the next section).

$\mathcal{IC}_S$ expresses what is expected or *should* happen or not, given some happened events. For example, a IC_S in $\mathcal{IC}_S$ could state that the manager of a resource should give an answer to whomever has made a request for that resource. IC_S can produce expectations on the future. As we will better see in the following sections, a correct (or *conforming*) behaviour of a society requires a match between expectations and history, e.g., **E** expectations have to be matched by **H** events, and **NE** expectations should not have a corresponding **H** event.

In the following, we define the syntax of $SEKB$, $SOKB$, and $\mathcal{G}$. $\mathcal{IC}_S$ will be described in Section 3. *Atom* and *Term* are intended as usual in Logic Programming [Llo87]; *Constraint* is a constraint in the CLP sense [JM94].

2.4 Syntax of the Social Environment Knowledge Base

As we explained earlier, the $SEKB$ contains information about the (actual) behavior of computees and about their expected behavior. It is composed of events and expectations. In particular, the syntax of the $SEKB$ is as follows:

$$
\begin{array}{rcl}
SEKB & ::= & \mathbf{HAP}\ [\wedge \mathbf{EXP}] \\
\mathbf{HAP} & ::= & [\mathbf{H}(Event[,\ Time])]^{\star} \\
\mathbf{EXP} & ::= & Expectation\ [\ \wedge (Expectation|Constraint)]^{\star} \\
Expectation & ::= & [\neg]\mathbf{E}(Event\ [,T]) \mid [\neg]\mathbf{NE}(Event\ [,T]) \\
Event & ::= & Atom
\end{array}
\tag{1}
$$

The syntax of constraints is as in [KSST03]:

$$
\begin{array}{rcl}
Constraint & ::= & AtomicConstraint \mid Constraint \wedge Constraint \mid \\
 & & Constraint \vee Constraint \mid \neg Constraint \\
AtomicConstraint & ::= & Variable\ Relop\ Term \\
Relop & ::= & = \mid \neq \mid < \mid > \mid \leq \mid \geq \\
Term & ::= & Atom \mid Variable \mid Term\ Op\ Term \\
Op & ::= & + \mid - \mid * \mid \div
\end{array}
\tag{2}
$$

Events - The *Events* recorded in the history are the socially significant ones; here, we are mostly concerned with computees' communicative, as well as physical, performed actions. They are expressed as

$$\mathbf{H}(Event[,\ Time]) \tag{3}$$

where *Event* is an atom describing the event occurred and *Time* is the time at which the event occurred. We explicitly consider time as a parameter of an atom **H** as it is often important to consider time as a special variable, with axioms specific to temporal variables (as we will see in Section 2.7). However, the time parameter is optional because in some cases it is not the central issue. In such cases, we use $\mathbf{H}(Event)$ as a syntactic sugar for $\mathbf{H}(Event, \_)$, where the underscore represents an unnamed variable.

Intuitively, a **H** atom represents a socially significant event that happened in the society, i.e., social events are mapped into **H** predicates. Events that happen, such as dialogue moves (social events), are part of the $SEKB$. The history of the society grows monotonically as new events are recorded. In the following, we will use the notation **HAP** for such a history.

A **H** atom is always ground: when an event happens, we suppose to be given all the significant information about it.

Expectations - Expectations (positive and negative) are hypotheses of the society about the (future) behavior of computees; a computee may then fulfill or not the society's expectations: because of the openness of the society and the autonomy and heterogeneity of the computees, there is no guarantee that expectations will be fulfilled.

The syntax of expectations is the following:

$$\mathbf{E}(Event[,\ Time]) \tag{4}$$

$$\mathbf{NE}(Event[,\ Time]) \tag{5}$$

for, respectively, positive and negative expectations. **E** is a positive expectation about an event (the society expects the event to happen) and **NE** is a negative expectation, (i.e., the society expects the event not to happen in order to fulfill the protocols).

Note the difference between $\neg\mathbf{E}(X)$ and $\mathbf{NE}(X)$. The first expresses the fact that the society does not have an expectation about the happening of event X (yet, if the event happens, no

protocol will be violated), while the second expresses the fact that the society expects the event not to happen.

Expectations can have non-ground terms as arguments. Intuitively, if an atom **E** is in the set of expectations, we hope that an atom **H** will unify with it (and one is enough to fulfill the expectation). Thus, variables in an **E** atom are always existentially quantified. For instance,

$$\mathbf{E}(tell(Auctioneer, Bidders, openauction(Item, Dialogue)), T_{open})$$

stands for an expectation about a communicative act *tell*made by a computee ($Auctioneer$), addressed to a (group of) computees ($Bidders$), with subject $openauction(Item, Dialogue)$, at a time T_{open}.

Often we need to share variables between expectations; thus the scope of the existentially quantified variables is the whole set of expectations. On the other hand, **NE** atoms represents disproved behavior: something that hopefully will not (ever) happen. Variables in a **NE** atom are universally quantified if they are not shared with an **E** atom. To sum up

- variables in **E** atoms are always existentially quantified with scope the entire set of expectations
- the other variables, that occur only in **NE** atoms are universally quantified (the scope of universally quantified variables is not important, as $\forall X, p(X), q(X)$ is equivalent to $\forall X p(X)$, $\forall Y q(Y)$).

In the following, we will use the notation **EXP** for the set of expectations.

2.5 Syntax of the Social Organization Knowledge Base

We consider the $SOKB$ as a logic program. The syntax of the $SOKB$ is as follows:

$$\begin{array}{rcl} SOKB & ::= & [Clause]^\star \\ Clause & ::= & Atom \leftarrow Body \\ Body & ::= & ExtLiteral\ [\ \wedge ExtLiteral\ ]^\star \\ ExtLiteral & ::= & Literal \mid Expectation \mid Constraint \\ Expectation & ::= & [\neg]\mathbf{E}(Event\ [, T]) \mid [\neg]\mathbf{NE}(Event\ [, T]) \\ Literal & ::= & Atom \mid \neg Atom \mid true \end{array} \tag{6}$$

In a clause, the variables are quantified as follows:

- Universally, if they occur only literals of kind **NE** (and possibly constraints), with scope the body;
- Otherwise universally, with scope the entire $Clause$.

The following is a clause:

$$sold(Item) \leftarrow \\ \quad \mathbf{E}(tell(Auctioneer, Bidders, openauction(Item, Dialogue)), T_{open})$$

It says that one way to sell an item is to have some computee tell a set of possible bidders that an auction is open for the item.

2.6 Goals of the society

Both goal-directed and not goal-directed behavior for a society is supported. A *Goal* has the same syntax as the *Body*. A Goal can activate a derivation; if the goal is "*true*", then the behavior for the society will be non-goal-directed.

While trying to reach a goal of the society, the society may derive the need for some computee to behave in some way. Thus, the society *expects* some computee to behave accordingly, in order to fulfill the society's needs.

As an example, we can consider a society with the goal of selling items (this example is a simplified version of the scenario in Section 8.2). In order to sell an item, the society might expect some computee to embody the role of auctioneer. The goal of the society could be

$$\leftarrow sold(nail)$$

and the society might have, in the $SOKB$, a rule like the one shown above, having $sold(nail)$ in the head. Of course, there could be more clauses specifying other ways for achieving the same goal, like expecting some computee to advertise it on some public channel, and so on. The protocol of the auction (i.e., the way the auctioneer and the bidders interact) can be then specified by means of IC_S.

2.7 Temporal reasoning

As we hinted earlier, one of the parameters of happened events and expectations is *time*. We conclude this section by introducing some ideas about temporal reasoning in our social framework. Details about temporal information management will be provided in Section 4.2, after we define the declarative semantics of the framework.

Temporal reasoning should be taken into account when defining protocols in an agent society and when checking the computees' protocol compliance. Several frameworks have been proposed to deal with this form of reasoning, most notably those based on Event Calculus [KS86], Interval Algebra [All83], Point Algebra [VK89], Simple Temporal Problems and Temporal Constraint Satisfaction Problems [DMP91] and Time Map Management [DM87]. In addition, some promising approaches have been proposed for integrating qualitative and quantitative temporal reasoning [KL91, Mei91].

Some of these frameworks are based on temporal constraints (relations among time variables), and the problem is represented as a Constraint Satisfaction Problem (CSP) [vH89], in terms of a constraint graph where nodes are the problem variables (points or intervals) and arcs are (binary) constraints (temporal relations between pairs of variables). Given a constraint graph, we are interested in determining whether the graph is consistent, or in determining an equivalent minimal representation by computing its transitive closure, or in finding a feasible temporal scenario. Many algorithms have been proposed for solving these problems (see, for example, [All83, DMP91, PV90, VK89]).

For our purposes, and for efficiency reasons, we have decided to use a simple temporal reasoning framework, based on a quantitative constraint-based approach working on temporal points. Qualitative reasoning is not needed in this application since we suppose that when an event occurs we know the precise time point. In addition, qualitative reasoning is needed if we have to compute a qualitative closure of the network, while our purpose is to detect infeasibility of instantiated events. We decided to use a discrete time domain, instead of a continuous one because it can be handled more efficiently; also "Continuous Time domains have very little

applicability in computer science solutions since in practical terms the domain has to be sampled with some level of frequency" [FIP03]. Therefore, we represent time on integer numbers, and we have a finite domain solver which checks consistency and is able to find a consistent scenario if any. Clearly, being the temporal representation discrete, the minimal interval between two events is a time unit, which can be tuned according to the application. If we are modelling a society with a slow dynamic, we can consider a time unit as a day or an hour, while if we are representing society with a faster dynamic the time tick can be a minute or a second. In addition, using finite domains, we have a temporal horizon T_{Max} of the system which represents the *entire life* of the society.

One advantage of using discrete temporal points is that, if we restrict ourselves to consider only the relations *before*, *after* and *at the same time*, based on the operators $<$, $>$, and $=$, we have a complete polynomial propagation [VK89]. In this way, from a declarative viewpoint the time variables are considered as the other finite domain variables; operationally, we will use specific axioms for time (Section 4.2).

Therefore, as explained earlier, we associate to each event and to each expectation a time variable whose feasibility and propagation are managed by a constraint solver on finite domains of integers. Happened events are always associated with a ground time since we know exactly that the event has occurred at time t

$$\mathbf{H}(Event, t)$$

while expectations may have a (constrained) variable as time parameter, explaining the possible time points in which the event is expected to happen, or not to happen.

3 Protocols

A *protocol* specifies the "rules of encounter" governing a dialogue between agents [RZ94, MPW02]. It specifies which agent is allowed to say what in a given situation. It will usually allow for several alternative utterances in every situation and the agent in question has to choose one according to its private *policy*. Sometimes in the literature, dialogue policies are also referred to as *strategies*. A good protocol will enable fruitful interaction in general. A good policy will benefit the agent using it. The protocol is *public*, while each agent's policy is *private*. Protocols are practically important because they may help to select the adequate answer to an incoming utterance, thus reducing the complexity of this task for an agent (see Section 10 of D4 [KSST03] and [EMST03b]).

In this section, we provide a language (IC_S) for defining protocols and checking the compliance of the behavior of a computee, or, in general, of a society, to protocols. We first survey background work and motivate our choices, then we introduce the syntax of IC_S.

3.1 Agent interaction protocols

Until recently, Agent Communication Languages (ACL) research issues were primarily related to the generation and interpretation of individual ACL messages. They have consequently largely remained disconnected from the large amount of existing work on Interaction Protocols [BHS93, Dem95] (or IP, for short), which was essentially viewed as a practical matter as far as agent communication theory was concerned. It was indeed assumed, more or less explicitly, that conversations structure should emerge as a consequence of the semantics of individual

messages (that is, a question should be followed by an answer because the agent should be able to recognize the other agent's underlying intention, not because it is specified as such in a protocol). This is illustrated by the following quote, taken from the specifications of the Foundation for Intelligent Physical Agents (FIPA) [FIP01b]:

> "A designer of agent systems has the choice to make the agents sufficiently aware of the meanings of the messages and the goals, beliefs and other mental attitudes the agent possesses, and that the agent's planning process causes such IPs to arise spontaneously from the agents' choices. This, however, places a heavy burden of capability and complexity on the agent implementation, though it is not an uncommon choice in the agent community at large. An alternative, and very pragmatic, view is to pre-specify the IPs, so that a simpler agent implementation can nevertheless engage in meaningful conversation with other agents, simply by carefully following the known IPs."

Because this position has raised many critics, especially in the context of open systems as discussed earlier (see Section 1.1), IPs are now considered as structures of theoretical importance when one tries to model agent interactions. As a result, nowadays research on ACLs tries to address the gap between the individual messages and the extended message sequences (dialogues) that arise between agents (witness the fact that the series of workshops on ACL and IPs, held separately until 1999 have now merged into an *"Agent Communication Languages and Conversation Policies"* series).

Let us now introduce the various formalisms proposed in the literature to regulate the interaction and allow the generation of conversations.

Input-output pairs - This is the most basic way of representing interaction patterns: the pairs just specify the appropriate answer(s) (*output*) to a received message (*input*). It is clear that this model does not permit to refer to the history of the dialogue, which makes it only suitable for the simplest interactions.

Finite State Machines - Finite State Machines (FSMs) are arguably the most adequate (and popular) formalism to account for *sequential* interactions. The state of the automaton describes the state of the conversation. Carefully designed FSMs have been implemented in real application,s see for instance COOL [BF95]. However, because it is necessary to specify all the local states of the interaction, it is clear that designers face a practical specification problem and consequently tend to oversimplify the protocols.

Dooley graphs - Dooley graphs have been introduced in the Multi-Agent community by [vP96], as a natural way to represent dialogue "as it happens" (the set of utterances that are related to one another are closer in the graph). A Dooley graph basically consists of a set of nodes (the participants), a set of indexes (the acts), a set of arcs connecting the nodes, *and a set of vertices connecting the arcs* providing a way to represent various forms of relations between the acts (replies, resolves, completes) beyond simple precedence.

AUML Protocol Diagrams - Protocol diagrams rely on an extension of the classical UML formalism specially dedicated to agents [BMO01]. Protocol diagrams introduce a number of new features: most notably, concurrent messages are allowed, and the cardinality of messages is not restricted to the one-to-one case. The notion of role is central: protocol diagrams typically represents the lifelines of agents using defined roles, and the steps in

which the communicative acts are sent between these agents. AUML supports partial or complete reuse of protocols. There is still ongoing research trying to enhance the formalism with useful notions (e.g., synchronization, exception handling, see [Hug02]). However, it should be kept in mind that AUML remains a semi-formal specification.

Coloured Petri Nets - Colored Petri Nets (CPNs) is a well-known formalism for concurrent systems which has recently been proposed to account for interaction protocols [CCF+00]. A CPN basically consists of places, transitions, states, functions and domains. Unlike AUML protocol diagram, CPN is a formal model with a proper semantics, which allows verification and validation of the interaction protocols. Interestingly, it is argued in [MEH02] that AUML protocol diagrams can be translated into CPNs, and some guidelines are offered to help the designer in this task. Roughly speaking, each agent lifeline will correspond to a sequence of places and transitions, while the exchange of a message between two roles is represented by a synchronization place and arcs.

Dialogue games - Recently, a significant trend of research has been influenced by the philosophical work on *informal logic* [Ham70]. The main feature of the dialogue game approach is (*i*) to rely on a notion of conversational store in order to keep track of the relevant part of the history; (*ii*) to allow different compositions of interaction protocols (embedding, sequence, parallelization, etc.), see for instance [MP02, DHvdT00]. A thorough review of current work on dialogue game-based protocols can be found in [MC02].

Event Calculus - In [YS02], a variant of the event calculus [KS86] is applied to commitment-based protocol specification. The semantics of messages (i.e., their effect on commitments) is described by a set of *operations* whose semantics, in turn, is described by *predicates* on *events* and *fluents*; in addition, commitments can evolve, independently of communicative acts, in relation to *events* and *fluents* as prescribed by a set of *postulates*. This way of specifying protocols is more flexible than traditional approaches specifying protocols as action sequences in that it prescribes no initial and final states or transitions explicitly, but allows any possible protocol to run with the only condition that, at the end of a protocol run, no commitment must be pending; agents with reasoning capabilities can themselves plan an execution path suitable for their purposes (which, in that work, is implemented by an abductive event calculus planner).

Process calculi - Process calculi (or process algebras) specify interactions among a set of concurrent processes in a formal way. Since the first process calculus (CSP [Hoa78]), a number of calculi have been designed in order to model mobility, cryptography, access control and other features of distributed systems. Among the many existing calculi one of the most noteworthy is the π-calculus [MPW92]. Process calculi are used in order to specify protocols of interactions among agents in two projects from the Global Computing initiative: DEGAS [DEG01] and MYTHS [MYT01]. In DEGAS interaction protocols are specified in UML and are automatically translated into π-calculus in order to prove qualitative and quantitative properties of the protocol. In MYTHS a process calculus based on types is developed, and it is shown how it can be used in order to prove formal properties of protocols, in particular as regards security. None of these projects employs the protocol specification for checking the conformance of agents to the protocol.

This list is by no means exhaustive, but only reflects what we considered to be the most significant proposals. Others important approaches include for instance open protocols in the

context of Agentis [dKL98], or the framework of Electronic Institutions [ERA+00].

None of the formalisms described above has been definitely accepted as *the* interaction protocol formalism, and some of them are still the subject of active ongoing research. It is worth noting however that, in its 1999 specifications, FIPA used a finite state machine representation of its interaction protocols. And that, as a consequence of the collaboration between FIPA and OMG (Object Management Group), the 2001 specifications has recently adopted the new Agent UML standard [BMO01] and thus uses Protocol Diagrams to describe interaction. Most of the times, however, protocol designers use the simplest formalism which meet their requirement for a given application.

3.2 IC_S to express protocols

This section presents the motivations underlying our proposal for modelling interaction protocols. More specifically, we will motivate the use of sets of constraints on the social behavior (i.e., social events) to specify protocols within our societies of computees. When computees join a society, indeed, they join one or more roles, thereby acquiring restrictions on how they can act and, in particular, communicate (see further section 3.4). Our approach could rely on an "IC_Sbased semantics" for specifying protocols. The motivations for adopting this approach are the same supporting commitments and committed-based semantics in [YS02]. The idea is also in a way similar to that of *conversation policies*, defined as *"general constraints on the sequences of semantically coherent messages leading to a goal"* [GHB00], but with a more flexible approach.

Computees - It is clear that the formalisms described so far do not lend themselves to be used as they stand in a Computational Logic-based framework such as the one that we envisage. Because of the reactive capability of computees (see D4 [KSST03]), integrity constraints are perfectly well suited to our purposes.

Flexibility - Most of the formal approaches to model protocols reviewed in Section 3.1 require that each state of the interaction is described. This can be practically tedious and motivate designers to over-constrain protocols, affecting in turn the flexibility of the interactions and the autonomy of the computees. Instead, *"participants must be constrained in their interactions only to the extent necessary to carry out the given protocol and no more"* [YS02]. Our IC_S have this feature.

Expressiveness - IC_S allow to capture the different features exhibited by the formalisms described in Section 3.1. Sometimes, however, it can be necessary to include extra integrity constraints left implicit in semi-formal models, as shown in [EMST03a]. The explicit representation of the time parameter within the constraints allows to handle synchronization easily. On top of that, we believe that constraints will allow us to extend the expressiveness of classical protocols. For instance, deadlines could be introduced as additional IC_S, as we shall see in Section 4.2.

Properties - Since protocols are expressed in terms of IC_S, verifying that some properties hold for a given protocol will be made easier. For instance, computees can be tested for compliance on the basis of their communications, or even (in some cases) of their specifications, by relying on techniques well-studied in the context of Computational Logic. We also envisage to use the same techniques to formally study the different combinations of protocols as proposed in the dialogue game literature.

Exceptions - In all the classical formalisms described, it is unclear how the society should react in case of violation of the rules described by the protocol. This issue can be addressed in the IC_S-based approach that we propose: for this, IC_S for society protocols express just expectations about social behavior. That is, if the IC_S are violated, the society is not inconsistent, but only in an undesirable state. It is then possible to define how such an undesirable state can be abandoned.

Our approach also guarantees autonomy, in that computees are not constrained in their behaviour but they can act as they planned to do. The outcome of their actions will depend, from a social perspective, from the fact that they obey or not to the IC_S defining the protocols.

3.3 Syntax of Social Integrity Constraints

The IC_S in $\mathcal{IC}_S$ are used to check if a computee inside the society behaves in a permissible way with respect to its "social" behavior. Intuitively, IC_S are rules used to provide information about the *expected* behavior of computees.

The syntax of $\mathcal{IC}_S$ is as follows:

$$
\begin{array}{rcl}
\mathcal{IC}_S & ::= & [ic]^\star \\
ic & ::= & \chi \rightarrow \phi \\
\chi & ::= & (HEvent|Expectation)\ [\wedge BodyLiteral]^\star \\
BodyLiteral & ::= & HEvent|Expectation|Literal|Constraint \\
\phi & ::= & HeadDisjunct\ [\ \vee HeadDisjunct\ ]^\star|\bot \\
HeadDisjunct & ::= & Expectation\ [\ \wedge (Expectation|Constraint)]^\star \\
Expectation & ::= & [\neg]\mathbf{E}(Event\ [,T]) \mid [\neg]\mathbf{NE}(Event\ [,T]) \\
HEvent & ::= & [\neg]\mathbf{H}(Event\ [,T]) \\
Literal & ::= & Atom \mid \neg Atom \mid true \\
Event & ::= & Atom
\end{array}
\tag{7}
$$

The syntax of *Constraints* is the same defined in Section 2.4 below the syntax of the $SEKB$.

Given a IC_S $\chi \rightarrow \phi$, χ is called the *body* (or the *condition*) and ϕ is called the *head* (or the *conclusion*).

Syntactic restrictions, scope and implicit quantification of variables - The rules of scope and quantification are as follows:

1. A variable must occur at least in an *Event* or in an *Expectation*.
2. The variables that occur both in the body and in the head are quantified universally with scope the entire IC_S.
3. The variables that occur only in the head must occur in at least one *Expectation* in Eq. (7), and
   (a) if they occur in literals **E** or $\neg$**E** are quantified existentially and have as scope the disjunct they belong to;
   (b) otherwise they are quantified universally.
4. The variables that occur only in the body have the body as scope and

(a) if they occur only in conjunctions of $\neg\mathbf{H}$, **NE**, $\neg$**NE** or *Constraints* are quantified universally;

(b) otherwise are quantified existentially.

5. the quantifier $\forall$ has higher priority than $\exists$ (e.g., literals will be quantified $\exists\forall$ and not viceversa).

There are several reasons why we decided to adopt this convention. Firstly, we wanted to keep the notation simple. We did not want to load the notation with explicit quantification symbols in the IC_S, and at the same time we wanted IC_S to have a simple reading, and an intuitive meaning associated. Secondly, we had to take into account compatibility issues: As described in the syntax of the $SEKB$ (Section 2.4), **E** atoms in the $SEKB$ are existentially quantified, and **NE** atoms are universally quantified. The syntax of the $SOKB$ is consistent with that of the $SEKB$, so atoms occurring in the *Head* of a IC_S are quantified accordingly (rule 3 above). Integrity Constraints, in ALP, are usually universally quantified with scope the whole Integrity Constraint, and rule 2 above states that the other variables in the IC_S are quantified accordingly, suggesting that IC_S are considered as particular Integrity Constraints. On the other hand, we decided to have exceptions to this rule, stated in rule 4a: $\neg\mathbf{H}$ and **NE** atoms[2]. This is due to the fact that $\neg\mathbf{H}$ atoms, that can only occur in the *Body* of a IC_S, have the intuitive meaning: "if an event did not happen, trigger the rule". Consider the following example:

Example 1. *If I did not say "propose" at* some *time T, you should not say "accept".*

This sentence can be written as

$$\neg\mathbf{H}(tell(A, B, propose), T), T < T1 \rightarrow \mathbf{NE}(tell(B, A, accept), T1).$$

Without rule 4a, variable T would be universally quantified with scope the whole IC_S, meaning that

> *(for all T1,A,B) At* all *times T before T1, if A does not say "propose", then B is not entitled to say "accept".*

We find this meaning is quite counterintuitive: the rule triggers in all times in which A does not say "propose". With the exception in rule 4a, the quantification is

$$\forall T1, A, B\,[\forall T \neg\mathbf{H}(tell(A, B, propose), T), T1 > T] \rightarrow \mathbf{NE}(tell(B, A, accept), T1).$$

thus, $\forall T$ and $\rightarrow$ are exchanged in priority:

> *(for all T1,A,B) If, at all times T before T1 A did not say "propose", then B is not entitled to say "accept".*

[2]Note that rule 4b is equivalent to rule 2, in fact $\forall X(Body \rightarrow Head)$ is equivalent to $(\exists X Body) \rightarrow Head$ if X only occurs in the *Body*.

We conclude by giving an example of IC_S. In this example, if a computee X says "ask" to a computee Y during a conversation D, Y is expected to answer back either "yes" or "no" (but not both of them):

$$\begin{array}{l}\mathbf{H}(tell(X,Y,ask,D),T) \rightarrow \\ \quad \mathbf{E}(tell(Y,X,yes,D),T'),T'>T \\ \quad \vee\ \mathbf{E}(tell(Y,X,no,D),T'),T'>T\end{array}$$

$$\begin{array}{l}\mathbf{H}(tell(X,Y,yes,D),T) \rightarrow \\ \quad \mathbf{NE}(tell(X,Y,no,D),T')\end{array}$$

$$\begin{array}{l}\mathbf{H}(tell(X,Y,no,D),T) \rightarrow \\ \quad \mathbf{NE}(tell(X,Y,yes,D),T') \\ \mathbf{H}(tell(X,Y,S,D),T) \rightarrow \\ \quad \mathbf{NE}(tell(X,Y,S,D),T'),T'>T\end{array}$$

The last IC_S says that a computee is expected not to repeat the same thing twice in the same dialogue.

In the example above, X telling "ask" to Y generates an expectation which can be fulfilled if Y answers back "yes" (Y behaves "properly"). Y violates the protocol, instead, if it says both "yes" and "no", due to the second constraint ("improper" behavior of Y). At all times there can be alternative sets of expectations (e.g., $\mathbf{E}(\ tell(Y,X,yes,D),\ T'),\ T'>T\ OR\ \mathbf{E}(\ tell(Y,X,no,D),\ T'),\ T'>T$).

In Appendix B, we compare the syntax of IC_S with the syntax of integrity constraints in the IFF proof-procedure.

3.4 Roles

In the analysis and design of MAS, *roles* play a fundamental part. Software engineering methodologies have been defined to design MAS, which are based on the concept of roles. In the Gaia methodology [WJK00], a role is associated to a set of *rights* (what an agent embodying the role can do) and *responsibilities* (what an agent embodying the role should do); responsibilities can be of two types: *liveness* (i.e., ensuring that something good will happen) and *safety* (i.e., something bad will not happen). Other approaches propose to associate *capabilities* and *expected behavior* [CLZ02]. Capabilities are the set of actions that an agent (embodying the role) can perform; they are related to the concept of *pro-activeness*, since they concern what an agent may do. The expected behavior in [CLZ02] is the reaction to incoming events, and is thus related to the *reactiveness* of the agent.

The concept of roles can be accommodated in our framework by expressing both capabilities and duties as *preconditions* for expectations to be raised in $\mathcal{IC}_S$. Intuitively, if a computee R is asked to perform a task by a computee S, the expectation about its future behavior is raised only if

1. the role of S (or, one of its roles, if S embodies more than one role) gives it the *capability* to ask for the given service; and

2. the role of R (resp., one of its roles) assigns it the *duty* to perform the given action when requested.

For this purpose, we introduce in the $SOKB$ the predicate

$$role(RoleName, Capabilities, Duties),$$

which is used to define the set of roles that can be assumed by the members of the society. Along with the name of a role, its associated $Capabilities$ and $Duties$ are specified as a list of predicates. One can write the following IC_S:[3]

$$\begin{array}{l}\mathbf{H}(tell(S, R, ask(perform(Action))), T_r) \wedge \\ embodies(S, Role_S) \wedge role(Role_S, Capabilities_S, \_) \wedge ask(perform(Action)) \in Capabilities_S \\ \wedge embodies(R, Role_R) \wedge role(Role_R, \_, Duties_R) \wedge perform(Action) \in Duties_R \\ \rightarrow \mathbf{E}(perform(R, Action), T_a)\end{array} \tag{8}$$

Notice that the IC_S in Eq. (8) can be obtained syntactically, as a preprocessing, from the following, more intuitive, rule written by the user:

$$\mathbf{H}(tell(S, R, ask(perform(Action))), T_r) \rightarrow \mathbf{E}(perform(R, Action), T_a)$$

In this simple example, we have a *static* assignment of roles: we suppose to have a predicate

$$embodies(Computee, Role)$$

defined in the $SOKB$ that associates computees and roles. Of course, one could have a dynamic assignment of roles; in this case the IC_S would be triggered depending on some event where one computee starts embodying the given role (e.g., a computee registers in the given role), or is entitled a given role (e.g., a computee is accepted for the role).

For example, let us consider an auction (this is a simplified version of the example provided in Section 8.2, where *combinatorial* auctions are considered). We can have two roles, namely *auctioneer* and *bidder*. The auctioneer may have the capabilities of opening an auction, reply to bidders if their bid was winning or loosing, and declaring closed the auction. The duties concern taking into consideration bids.

$$role(auctioneer, \underbrace{\{openauction(Item), answer(Reply), closeauction\}}_{\text{(Capabilities)}}, \underbrace{\{bid(Item, Price)\}}_{\text{(Duties)}})$$

The bidders, in this simple example, can place bids and have no duties (of course, in a real auction one should pay for the good, etc.; here we do not model the actual transaction).

$$role(bidder, \underbrace{\{bid(Item, Price)\}}_{\text{(Capabilities)}}, \underbrace{\emptyset}_{\text{(Duties)}})$$

A rule of the auction protocol, as written by the user, can be the following:

$$\begin{array}{l}\mathbf{H}(tell(A, B, openauction(Item)), T_{open}) \wedge \mathbf{H}(tell(B, A, bid(Item, Price)), T_{Bid}), T_{Bid} > T_{Open} \\ \rightarrow \mathbf{E}(tell(A, B, answer(win)), T_{answer}) \vee \mathbf{E}(tell(A, B, answer(lose)), T_{answer}), T_{Answer} > T_{Bid}\end{array}$$

[3]for the sake of brevity, we use the symbol $\in$ instead of using a set membership predicate.

that will be automatically translated, by taking roles into consideration, into

$$\mathbf{H}(tell(A, B, openauction(Item)), T_{open}) \wedge \mathbf{H}(tell(B, A, bid(Item, Price)), T_{Bid}) \wedge$$
$$embodies(B, Role_B) \wedge role(Role_B, Capabilities_B, \_) \wedge bid(Item, Price) \in Capabilities_B$$
$$\wedge embodies(A, Role_A) \wedge role(Role_A, Capabilities_A, Duties_A)$$
$$\wedge openauction(Item) \in Capabilities_A \wedge answer(Reply) \in Duties_A$$
$$\rightarrow \mathbf{E}(tell(A, B, answer(win)), T_{answer}) \vee \mathbf{E}(tell(A, B, answer(loose)), T_{answer})$$

The other rules of the (combinatorial) auction protocol are shown in Section 8.2 and can be syntactically translated in a similar way.

3.5 Entering and exiting the society

As we already mentioned in Section 1.1, societies have been classified, from the openness viewpoint, as *open*, *semi-open*, *semi-closed* and *closed* [Dav01].

In an open society (or an open artificial society) "there are no restrictions for agents/processes to join/leave the society" [Dav01]: this means that it is possible for any agent to enter the society simply by starting an interaction with a member of it. This is also the acceptation adopted in Agentcities initiative [Age] to realize the commercial and research potential of agent-based applications by constructing a worldwide, open network of platforms hosting diverse agent-based services. In open societies, there is no clear frontier of the society: everybody can start interact with a member of the society and is considered a member in its turn. One drawback is that it is not clear which agents are currently members of the society.

In semi-open societies, the agent must also be admitted by the society to enter and, if accepted, it is a member until it leaves or is expelled; typically, there is a gate-keeper that decides whether to accept or not a given admission request. In semi-closed societies there is a representative of the agent inside the society.

In closed societies, one cannot dynamically become a member (the members are decided before the society starts).

All of these types of societies can be accomplished in our framework by exploiting the IC_S. Typically, being member of a society brings benefits and duties. If a given action can only be performed by members of the society, one can forbid the action to all the non members (which is quite meaningless in open societies). This translates, in a semi-open society, as follows:

"If you have never been admitted to the society, you cannot do *Action*."

$$\neg\mathbf{H}(tell(Gatekeeper, A, admit), T_{join}) \rightarrow \mathbf{NE}(do(A, Action), T_{action})$$

"If you have left the society (either because you decided to leave, or because you were expelled) and you have not been admitted again, then you cannot do *Action*."

$$\mathbf{H}(tell(A, Gatekeeper, leave), T_{leave}),$$
$$\neg\mathbf{H}(tell(Gatekeeper, A, admit), T), T \leq T_{currtime}, T > T_{leave}$$
$$\rightarrow \mathbf{NE}(do(A, Action), T_{currtime})$$

$$\mathbf{H}(tell(Gatekeeper, A, expel), T_{leave}),$$
$$\neg\mathbf{H}(tell(Gatekeeper, A, admit), T), T \leq T_{currtime}, T > T_{leave}$$
$$\rightarrow \mathbf{NE}(do(A, Action), T_{currtime})$$

"You should not have done *Action* before entering the society the first time."

$$\begin{array}{l}\mathbf{H}(tell(Gatekeeper, A, admit), T_{join}),\\ \neg\mathbf{H}(tell(Gatekeeper, A, admit), T_{nj}), T_{nj} < T_{join}\\ \rightarrow \mathbf{NE}(do(A, Action), T_{action}), T_{action} < T_{join}\end{array}$$

$$\begin{array}{l}\neg\mathbf{H}(tell(Gatekeeper, A, admit), T_{nj})\\ \rightarrow \mathbf{NE}(do(A, Action), T_{action})\end{array}$$

Symmetrically, members of a society have duties and should behave in a permissible way; again, this can be established by saying

$$\begin{array}{l}\mathbf{H}(tell(Gatekeeper, A, admit), T_{join})\wedge\\ \neg\mathbf{H}(tell(A, Gatekeeper, leave), T_{leave})\wedge\\ \neg\mathbf{H}(tell(Gatekeeper, A, expel), T_{leave}), T_{leave} > T_{join}\\ \rightarrow \mathbf{E}(do(A, GoodBehavior), T_g), T_g > T_{join}\wedge\\ \mathbf{NE}(do(A, Misbehave), T_m), T_m > T_{join}\end{array}$$

For what concerns semi-closed societies, rights and duties belong to the *representative* of the computee inside the society, which is a member of a society; thus we can apply the same rules as in a closed society.

Besides the classification by Davidsson [Dav01], there are also societies in which one cannot leave the society in any moment, but only when it does not have pending expectations.

In this section, we gave a set $\mathcal{IC}_S$ of IC_S defining at the protocol level the actions that agents are expected to do, depending on whether they succeed or fail in obtaining a membership. Another way to approach the problem is discussed in D4 [KSST03], Section 10.

4 Semantics of the social framework

In previous sections, we introduced the model for societies of computees in terms of $SOKB$ and $SEKB$, and the concept of social events, social expectations and $\mathcal{IC}_S$ for expressing protocols, with their syntax.

In the following, we introduce a series of successively more refined declarative semantics. The various declarative semantics offer a range of options for different proof-procedures, and are a ground basis to identify relevant properties of the society and its protocols.

An interesting point is that the declarative semantics is given as clean extension of that given for (extended) Logic Programming, making existing results for Logic Programming and Non-Monotonic Reasoning re-usable in this context.

4.1 Declarative semantics

The semantics of a society is here given, intuitively, by identifying a set of expectations which, together with the society's knowledge base and the happened events, implies an instance of the goal - if any - and *satisfies* the integrity constraints.

Throughout this section, for the sake of simplicity, we always consider the ground version of society's knowledge base and integrity constraints, and do not consider $Constraints$.

We first introduce the concept of *admissible set of social expectations*. Intuitively, given a society, and a set of events **HAP**, an admissible set of social expectations consists of a set of expectations about social events that are compatible with the $SOKB$, the set **HAP**, and $\mathcal{IC}_S$.

More formally, we introduce the following definition.

Definition 2. *Given a society and a set of events* **HAP**, *an* admissible set *of social expectations* **EXP** *is a set of expectations such that:*

$$SOKB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{IC}_S \tag{9}$$

Note that definition above follows, like in [FK97], the theoremhood view of integrity constraint satisfaction.

Furthermore, definition above is non-committal with respect to the notion of $\models$. This is mainly because at this stage of the project we have not yet defined the proof-procedure, and we wish to keep different options open at this time, so to guarantee compatibility with different (model-theoretic) semantics. Nonetheless, we have investigated a number of different definitions for $\models$. All these interpret $P \models C$ as expressing that C is true in all the *intended* models of P. The definitions differ, however, in their understanding of the notion of intended model.

If we interpret social expectations as abducible predicates (see section 4.3) we can rely upon a three-valued model-theoretic semantics as intended meaning, as done, for instance, in a different context, by [FK97, DS98].

Intuitively, many different sets of expectations are admissible, given $\mathcal{IC}_S$, the $SOKB$ and the set **HAP**.

Example 3 (admissible set of expectations). *As an example, let us consider the following protocol*

> *If I tell you "start" then you should pass; if I tell you "stop" then you should not pass.*

$$\mathbf{H}(tell(X, Y, start)) \rightarrow \mathbf{E}(pass(Y)).$$
$$\mathbf{H}(tell(X, Y, stop)) \rightarrow \mathbf{NE}(pass(Y)).$$

If the event $\mathbf{H}(tell(X, Y, start))$ *actually happens, then an expectation will be raised telling the receiving computee that it should pass. Otherwise, if the event* $\mathbf{H}(tell(X, Y, stop))$ *happens, another expectation is raised: the receiving computee is expected not to pass. If none of the actions* $tell(X, Y, start)$ *and* $tell(X, Y, stop)$ *occurs, no expectation is raised, and the computee can perform as it wishes.*

Let us now consider the following situation:

- $SOKB = \emptyset$
- **HAP**$= \{\mathbf{H}(tell(thomas, yves, start)), \mathbf{H}(tell(david, yves, stop))\}$
- $\mathcal{IC}_S = \{\mathbf{H}(tell(X, Y, start)) \rightarrow \mathbf{E}(pass(Y)),$
  $\mathbf{H}(tell(X, Y, stop)) \rightarrow \mathbf{NE}(pass(Y))\}$
- $\mathcal{G} = \emptyset$

$\mathbf{EXP}_1 = \{\mathbf{E}(pass(yves)), \mathbf{NE}(pass(yves))\}$ *is an admissible set of expectation, w.r.t. the* $SOKB$, $\mathbf{HAP}$, *and* $\mathcal{IC}_S$ *above. Notice that any set of expectations larger than* $\mathbf{EXP}_1$ *is also admissible.*

Instead, $\mathbf{EXP}_2 = \{\mathbf{E}(pass(yves))\}$ *is not an admissible set of expectations, because* $\mathbf{H}(tell(david, yves, stop)) \rightarrow \mathbf{NE}(pass(yves)) \in \mathcal{IC}_S$, $\mathbf{H}(tell(david, yves, stop)) \in \mathbf{HAP}$, *and* $\mathbf{NE}(pass(yves)) \notin \mathbf{EXP}_2$.

Admissible set of expectations can be however self-contradictory (e.g., both $\mathbf{E}(p)$ and $\neg\mathbf{E}(p)$ may belong to an admissible set). In this respect, the introduced notion of admissible expectation set is a sort of para-consistent [DP98] semantics.

More refined semantics can be given by identifying a subset of admissible expectation sets as the intended semantics for a society.

In particular, among admissible sets of expectations, we are interested in those which are *coherent* and *consistent* with respect to the following notions.

Definition 4. *A set of social expectations* $\mathbf{EXP}$ *is* coherent *if and only if :*

$$\{\mathbf{E}(p), \mathbf{NE}(p) \not\subseteq \mathbf{EXP}\}$$

Example 5 (coherent set of expectations). *Let us consider the situation presented in Example 3.* $\mathbf{EXP}_2 = \{\mathbf{E}(pass(yves))\}$ *is a coherent set of expectations (although it is not admissible w.r.t. the* $SOKB$, $\mathbf{HAP}$, *and* $\mathcal{IC}_S$ *of Example 3). On the other hand,* $\mathbf{EXP}_1 = \{\mathbf{E}(pass(yves)), \mathbf{NE}(pass(yves))\}$ *is not a coherent set of expectations (although it is admissible).*

Intuitively, we are not interested in sets of social expectations that, at the same time, require that a particular event p should happen and should not happen. In a social context, if computees are aware of social expectations, thus they can plan and act appropriately in order to achieve them, an incoherent situation, of course, has no contribution and information for them.

Definition 6. *A set of social expectations* $\mathbf{EXP}$ *is* consistent *if and only if :*

$$\{\mathbf{E}(p), \neg\mathbf{E}(p)\} \not\subseteq \mathbf{EXP}$$

and

$$\{\mathbf{NE}(p), \neg\mathbf{NE}(p)\} \not\subseteq \mathbf{EXP}$$

Example 7 (consistent set of expectations). *Let us consider a modification of Example 3:*

- $SOKB = \emptyset$
- $\mathbf{HAP} = \{\mathbf{H}(tell(thomas, yves, start)), \mathbf{H}(tell(david, yves, stop))\}$
- $\mathcal{IC}_S = \{\mathbf{H}(tell(X, Y, start)) \rightarrow \mathbf{E}(pass(Y)),$
  $\mathbf{H}(tell(X, Y, stop)) \rightarrow \neg\mathbf{E}(pass(Y))\}$
- $\mathcal{G} = \emptyset$

The intuitive meaning of the second IC_S *in* $\mathcal{IC}_S$ *is:*

> *If I tell you "stop" then one should not expect that you pass.*

$\mathbf{EXP}_2 = \{\mathbf{E}(pass(yves))\}$ *is a consistent set of expectations, although it is not admissible, since* $\mathbf{H}(tell(david, yves, stop)) \rightarrow \neg\mathbf{E}(pass(yves)) \in \mathcal{IC}_S$, $\mathbf{H}(tell(david, yves, stop)) \in \mathbf{HAP}$, *and* $\neg\mathbf{E}(pass(yves)) \notin \mathbf{EXP}_2$.

$\mathbf{EXP}_3 = \{\mathbf{E}(pass(yves)), \neg\mathbf{E}(pass(yves))\}$ *is instead an admissible set of expectation, but it is not consistent.*

Intuitively, we are not interested in sets of social expectations that are intrinsically inconsistent, i.e., that at the same time, expect something and do not expect the same thing.

When no coherent (and consistent) admissible expectation set exists, and therefore an incoherency (or inconsistency) arises, it means that the society has been modelled in a wrong way since a IC_S representing social laws is violated. In this case, it could be necessary to modify the society's $SOKB$ or $\mathcal{IC}_S$ by means of a theory revision process. This will be subject for future work.

We would like to stress that, up to now, we do not assume that expected events actually happen. This is in accordance with an *open* view for society where social expectations are just a suggestion for what should be done (or not done). It can be the case that in a situation an expectation is assumed as true, but the expected event does not happen (which leads to a violation, and possibly to a sanction).

A further refined semantics is then given by identifying, among coherent and consistent admissible expectation sets, those which are *fulfilled* by a set of events $\mathbf{H}$ in a society. This reflects the *ideal* behavior of a society [ESP99].

Definition 8. *Given a society and a set of events* $\mathbf{HAP}$*, a coherent and consistent admissible set of social expectations* $\mathbf{EXP}$ *is* fulfilled *if and only if:*

$$\mathbf{HAP} \cup \mathbf{EXP} \models \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{NE}(p) \rightarrow \neg\mathbf{H}(p)\} \tag{10}$$

Example 9 (fulfilled set of expectations). *Let us consider the following situation:*

- $SOKB = \emptyset$
- $\mathbf{HAP}_1 = \{\mathbf{H}(tell(thomas, yves, start)), \mathbf{H}(pass(yves))\}$
- $\mathcal{IC}_S = \{\mathbf{H}(tell(X, Y, start)) \rightarrow \mathbf{E}(pass(Y))\}$
- $\mathcal{G} = \emptyset$

$\mathbf{EXP}_2 = \{\mathbf{E}(pass(yves))\}$ *is a coherent, consistent and fulfilled admissible set of expectations, w.r.t* $SOKB$, $\mathbf{HAP}_1$, *and* $\mathcal{IC}_S$.

But if we consider a different history: $\mathbf{HAP}_2 = \{\mathbf{H}(tell(thomas, yves, start))\}$, *then,* $\mathbf{EXP}_2$ *is not a fulfilled set of expectations w.r.t.* $SOKB$, $\mathbf{HAP}_2$, *and* $\mathcal{IC}_S$ *(still, it is admissible, coherent, and consistent).*

Intuitively, many different formulas are admissible with respect to $\mathcal{IC}_S$, the $SOKB$ and the history $\mathbf{HAP}$. By Definition 8, we select, among them, those where the happened events cover all the events that should happen, and none of the events that should not happen.

Notice that such a fulfilled admissible formula might not exist, even if a coherent and consistent admissible expectation set exists. The reason is the *violation* of the protocol: some computee did not behave as expected, and some action need be taken to recover from this violation (see Section 7.1).

Definition 10. *Given a society and a set* **HAP** *of events, if each coherent and consistent admissible set of expectations is not fulfilled (i.e., if the constraints* $\mathbf{E}(p) \rightarrow \mathbf{H}(p)$ *or* $\mathbf{NE}(p) \rightarrow \neg\mathbf{H}(p)$ *are violated), then we say that* **HAP** *produces a violation in the society.*

Also notice that in the previous definitions, we did not deal with a possible *Goal* of the society. If we want to consider a goal-directed society, then we introduce the following definition.

Definition 11. *Given a society, a goal* G *and a set of events* **HAP**, *we say that* G *is* achievable *iff there exists a coherent and consistent admissible set of social expectations* **EXP** *such that:*

$$SOKB \cup \mathbf{HAP} \cup \mathbf{EXP} \vDash G \tag{11}$$

Example 12 (achievable goal of a society). *Let us consider the following situation:*

- $SOKB = \{G_1 \leftarrow \mathbf{E}(pass(yves)),$
  $G_2 \leftarrow \mathbf{E}(pass(david))\}$
- $\mathbf{HAP} = \{\mathbf{H}(tell(yves, david, stop))\}$
- $\mathcal{IC}_S = \{\mathbf{H}(tell(X, Y, start)) \rightarrow \mathbf{E}(pass(Y)),$
  $\mathbf{H}(tell(X, Y, stop)) \rightarrow \mathbf{NE}(pass(Y))\}$
- $\mathcal{G} = \{G_1, G_2\}$

G_1 is an achievable goal w.r.t. $SOKB$, **HAP**, *and $\mathcal{IC}_S$, thanks to the coherent and consistent admissible set of social expectations* $\mathbf{EXP} = \{\mathbf{E}(pass(yves)), \mathbf{NE}(pass(david))\}$.

On the other hand, there exists no coherent and consistent admissible set of social expectations to make G_2 achievable, given history **HAP**.

Note that the notion of goal achievability does not guarantee that the goal is really achieved, since expectations may not be fulfilled, i.e., the corresponding events that should happen can be possibly not generated, and vice-versa. This introduces the following definition:

Definition 13. *Given a society, a goal* G *and a set of events* **HAP**, G *is* achieved *iff there exists a fulfilled coherent and consistent admissible set of social expectations* **EXP** *such that:*

$$SOKB \cup \mathbf{HAP} \cup \mathbf{EXP} \vDash G \tag{12}$$

Example 14 (achieved goal of a society). *Let us consider the following situation:*

- $SOKB = \{G_1 \leftarrow \mathbf{E}(pass(yves)),$
  $G_2 \leftarrow \mathbf{E}(pass(thomas))\}$
- $\mathbf{HAP} = \{\mathbf{H}(tell(yves, david, stop)), \mathbf{H}(pass(yves))\}$
- $\mathcal{IC}_S = \{\mathbf{H}(tell(X, Y, start)) \rightarrow \mathbf{E}(pass(Y)),$
  $\mathbf{H}(tell(X, Y, stop)) \rightarrow \mathbf{NE}(pass(Y))\}$
- $\mathcal{G} = \{G_1, G_2\}$

G_1 is an achieved goal of the society. G_2 is achievable, but it has not yet been achieved.

The set of admissible expectations can be further refined by introducing further constraints (as done, for instance, in order to remove incoherent and inconsistent expectation sets). Any further refinement then possibly augment the potentiality for violation, since it introduces further constraints. In the following section, for instance, we introduce a constraint pertaining time, which allows us to deal smoothly with deadlines.

The declarative semantics here given is a clean extension of that given for (Extended) Logic Programming, making existing results for logic programming and monotonic reasoning re-usable in this context. In particular, a model-theoretic semantics for various non-monotonic extensions of Logic Programming (such as abduction, default and explicit or classical negation) has been proposed in [BLMM97], and shown equivalent to well-known three-valued semantics for negation, such as preferential and 3-valued stable semantics.

In this way, the main advantage is that we can exploit well-known (model-theoretic) approaches in order to provide a basis for relevant properties, even if we have to deal with dynamic events.

4.2 Temporal information management

It is now time to give some details about the temporal information management of our framework.

We have a special event, i.e., the society *clock* represented by the atom *current_time* that happens at each time tick. Clearly, if we have

$$\mathbf{H}(current\_time, t_{current})$$

the next event *current_time* will happen at $t_{current} + 1$. A more interesting case concerns the expectations:

$$\mathbf{E}(Event, T).$$

In this case, unless otherwise specified, T is an existentially quantified variable which is not subject to any constraint, and its domain represents the *entire life* $[0..T_{Max}]$ of the system (i.e., the society). However, T can occur in one or more constraints. In this case its domain is reduced accordingly by removing unfeasible values.

As an example, let us consider the problem of modelling a protocol that specifies deadlines for events to happen. If a given event should happen within a fixed deadline d, the temporal variable associated T whose initial domain is $[0..T_{Max}]$, is now pruned to $[0..d]$. The IC_S stating that given an event at time T an event $Event_1$ is expected at time T_{answer} within the deadline is the following:

$$\mathbf{H}(Event, T) \rightarrow \mathbf{E}(Event_1, T_{answer}), T_{answer} \geq T, T_{answer} \leq T + 10$$

When the *Event* happens, its temporal variable is linked to a ground value, say 5. The corresponding domain of T_{answer} becomes [5..15]. The semantics of a domain variable T_{answer} ranging on [5..15] is simply given by the disjunction $T_{answer} = 5 \vee T_{answer} = 6 \vee \ldots \vee T_{answer} = 15$. Therefore the IC_S results in a disjunction:

$$\begin{array}{lll} \mathbf{H}(Event, 5) & \rightarrow & \mathbf{E}(Event_1, T_{answer}), T_{answer} = 5 \\ & \vee & \mathbf{E}(Event_1, T_{answer}), T_{answer} = 6 \\ & \vee & \\ & \vdots & \\ & \vee & \mathbf{E}(Event_1, T_{answer}), T_{answer} = 15 \end{array}$$

This IC_S is respected, i.e., no violation arises, if $Event_1$ happens in one of the time ticks belonging to the domain of T_{answer}. For that time tick we have that the expected event is fulfilled by the corresponding happened event.

A violation occurs if for all time points belonging to the domain of T_{answer} we have an expectation which is not fulfilled, i.e., we have that for all $t \in [5..15]$ $\mathbf{E}(Event_1, t)$ and $\neg\mathbf{H}(Event_1, t)$.

Now a problem arises: how can we monitor events that have not happened? Let us consider the following hypothesis that ensures that the declarative semantics is respected: we assume that the society is able to monitor all the events that happen in the society itself. Thus, for each time tick, we assume a sort of *closed world assumption* on the happened events for that time tick. Given the set $\mathbf{U}$ of all possible events that can happen in the society, and given $\mathbf{HAP}(t)$ the set of all happened event at a given time tick t, we can assume that $\mathbf{U} \setminus \mathbf{HAP}(t)$ is the set of not happened events. Thus for each $E \in \mathbf{U} \setminus \mathbf{HAP}(t)$ we assume $\neg\mathbf{H}(E, t)$.

Thus, if the event $Event_1$ does not happen within the deadline at each time tick t in the domain of T_{answer} we have $\neg\mathbf{H}(Event_1, t)$. For definition 9, the set of expectation is not fulfilled. As a consequence the operational semantics produces a violation.

Clearly, operationally, for each time tick we can restrict our attention to those events that are expected to happen. If they do not happen, i.e., if they do not belong to $\mathbf{HAP}(t)$, a violation arises.

More formally, let us consider the set $\mathbf{EXP(t)}$ that contains the set of expectations in the time tick t. For each event $E \in \mathbf{EXP}(t)$ such that $E \notin \mathbf{HAP}(t)$, we assume $\neg\mathbf{H}(E, t)$.

Let us consider now negative expectations. For each event which is expected not to happen, we consider the associated temporal variable T representing the time of the negative expectation. If a given action is expected not to happen in the society, then we have

$$\mathbf{NE}(Event, T).$$

If T is not subject to constraints, it means that the event is expected *never* to happen. On the contrary, if we have some event that is expected not to happen for 10 time ticks after a given event happens, we can write:

$$\mathbf{H}(Event, T) \rightarrow \mathbf{NE}(Event_1, T1), T1 \leq T + 10, T1 > T$$

In this case, $Event_1$ is expected not to occur for the 10 time ticks after the occurrence of $Event$.

When the $Event$ occurs, its temporal variable is linked to a ground value, say 5. The corresponding domain of $T1$ becomes [5..15]. Note that in this case, occurring $T1$ in a $\mathbf{NE}$, $T1$ is universally quantified, therefore the IC_S results in a conjunction of IC_S

$$\begin{aligned}
&\mathbf{H}(Event, 5) \rightarrow \mathbf{NE}(Event_1, T1), T1 = 5 \\
&\wedge \\
&\mathbf{H}(Event, 5) \rightarrow \mathbf{NE}(Event_1, T1), T1 = 6 \\
&\wedge \\
&\ldots \\
&\wedge \\
&\mathbf{H}(Event, 5) \rightarrow \mathbf{NE}(Event_1, T1), T1 = 15
\end{aligned}$$

This IC_S is respected, i.e., no violation arises, if $Event_1$ does not happen in all the time ticks belonging to the domain of $T1$.

A violation occurs if $Event_1$ happens between 5 and 15. In this case, we have a negative expectation for $Event_1$ which is not fulfilled. For one $t \in [5..15]$ $\mathbf{NE}(Event_1, t)$ and $\mathbf{H}(Event_1, t)$.

Our framework has different features from those of frameworks based on Event Calculus, see [APS02, YS02] to name a few. Our choice has the following features: first a temporal point representation is enough for our purposes, second we can rely upon an efficient and complete propagation of temporal constraints. Therefore, we think that the way we model and reason upon time is sufficiently powerful for our purposes, and it is a good tradeoff between expressiveness and efficiency.

Our constraint-based framework for temporal information management can be easily integrated by each single computee, employing extended Abductive Event Calculus (see Section "Temporal reasoning" in D4 [KSST03]). In this way, using a constraint solver, we can enhance the efficiency of the temporal information management both at the society level and in each computee.

With respect to flexibility, in our approach, extension and merging of protocols can be accommodated by a suitable composition of constraints representing individual or partial interaction patterns.

Let us consider the following example where a "combination" of two individual interaction patterns is obtained just by the union of their individual IC_S:

$$\begin{aligned}&\mathbf{H}(tell(X, Y, request(A), D), T) \rightarrow \\ &\quad \mathbf{E}(tell(Y, X, accept(A), D), T'), T' > T + 2, T' < T + 10\end{aligned} \tag{13}$$

"Y is expected to accept a request within a time between 2 and 10 minutes after the request has been made"

$$\begin{aligned}&\mathbf{H}(tell(X, Y, request(A), D), T) \rightarrow \\ &\quad \mathbf{NE}(tell(Y, X, accept(A), D), T'), T' > T, T' < T + 5\end{aligned} \tag{14}$$

"Y is expected not to accept a request in the next 5 minutes of the request".

The union of IC_S (14) and (13), if only *coherent* sets of expectations (Definition 4) are required, restricts the time points in which the reply is expected to happen to the interval [6..10], i.e.:

"Y is expected to accept a request within a time between 6 and 10 minutes after the request"

4.3 The society knowledge as an Abductive Logic Program

The semantics of our social framework can be smoothly given an abductive interpretation. This is interesting in its own right and we foresee will play a role in Workpackages 3, so as to exploit well-assessed proof-theoretic techniques in order to check compliance of the overall computation within a society, with respect to the expected social behavior. This is indeed one of the main motivation of our approach.

In the following, we interpret the knowledge available at the social level as an Abductive Logic Program (ALP, see Appendix A). Imposing that all computees in a society share the same abducible predicates is a minimal requirement. The idea is derived from work by Kowalski and Sadri on abductive agents [KS99], where the abducibles are produced within an agent cycle, and represent actions in the external world.

The abducible atoms are the positive and negative expectations, **E** and **NE**. Finally, we have negative expectations (¬**E** and ¬**NE**), also represented as abducible atoms, in accordance with the usual way abduction can be used to deal with negation [EK89]. The set **EXP** can be seen as a set of hypotheses (possibly, a set of disjunctions of atomic hypotheses [Poo88, FK97]).

At the society level, knowledge can therefore be represented as an ALP, i.e., a triple: $\langle KB, \mathcal{E}, IC \rangle$ where:

- KB is the $SOKB$ and the history of events **HAP**;
- $\mathcal{E}$ is a set of *abducible predicates*, standing for positive and negative expectations (or their negation);
- IC is the set $\mathcal{IC}_S$ of IC_S.

Declaratively, the sets of social expectations discussed in Section 4.1 correspond to abduced set of hypotheses when the society is interpreted as and ALP.

Operationally, the idea is to exploit abduction for checking the compliance of the computation at a social level. Abduction captures relevant events (or hypotheses about future events), and a suitably extended abductive proof-procedure can be used for integrity constraint checking. If we consider that there exists a goal G at the society level (see also Section 4), then G is achieved when some expectations **EXP** are abduced, i.e.:[4]

$$KB \vdash_{\mathbf{EXP}} G \tag{15}$$

and **EXP** is a set of abducibles (coherent and consistent, see Def. 2 and 4) such that

$$KB \cup \mathbf{EXP} \cup \vdash IC \tag{16}$$

and conformance to rules and protocols is guaranteed by:

$$KB \cup \mathbf{EXP} \vdash \{\mathbf{E}(X) \rightarrow \mathbf{H}(X)\} \cup \{\mathbf{NE}(X) \rightarrow \neg\mathbf{H}(X)\} \tag{17}$$

if this last condition is not verified, then a *violation* occurs (see Def. 10).

The set **HAP** of happened events is included in the KB and therefore does not occur explicitly in Eq. (16) and Eq. (17).

Notice that Eq. (16) is the operational counterpart of Eq. (9) and Eq. (17) is that of Eq. (10). It should be noted that, if the society does not have a goal, i.e., G is *true*, Eq. (15) is always true for any set of **EXP** and, therefore, only equations (16) and (17) are significant (as in the declarative semantics, Section 4).[5]

Give this abductive re-interpretation of out social framework, in the later stages of the project a suitable proof-procedure should be defined in order to efficiently deal with such a semantics. In particular, the fulfillment check should be incremental (in order to detect violations as soon as possible), the operational phases of Equations (15)–(17) should be interleaved properly, and complexity issues should be taken into account [EM02].

[4] We do not commit at this stage for any particular semantics for the $\vdash$ symbol. Many semantics indeed could be given, such as for instance the classical SLDNF derivation as usual in Logic Programming. $KB \nvdash G$ is a shorthand for $not(KB \vdash G)$. The symbol $\vdash_\Delta$ stands for an abductive derivation with set of abduced atoms Δ.

[5] Notice that the condition established by Eq. (17) is new with respect to the abductive frameworks defined in Computational Logics.

Therefore, the adoption of this kind of representation for protocols and communicative acts requires suitable (extended abductive) proof-procedures (see previous work of CYPRUS on integration of abduction and constraint processing [Kak00, KvND01], previous work of UNIBO and DIFERRARA on the multi-agent ALIAS architecture [CLM+03], work on speculative computation of Satoh [Sat02], etc.).

From a first analysis, that will be developed in Workpackage 3, we understand that an extension of the IFF proof-procedure [FK97] will be appropriate. In Appendix B we present a comparison between the integrity constraints of the IFF proof-procedure and the IC_S of our framework.

5 Computee Communication Language

In this section, we describe our approach to the definition of a logic-based semantics for a CCL.

The section begins with a brief overview of existing work on the subject of Agent Communication Languages (ACLs), especially aimed at clarifying the differences between the most common approaches to definition of semantics, namely *mentalistic*, *conversational* and *social*.

We then explain why a *social* semantics appears the most appropriate for the purposes of the SOCS project, and show how the $\mathcal{IC}_S$ can be used to define CCL semantics; we also provide several example definitions of CCL communicative acts. What follows is not meant to define a substitute for the sets of communication primitives defined in existing ACLs; rather, it explains how the $\mathcal{IC}_S$ can be used to define the semantics of a given set of primitives.

Drawing inspiration from existing social ACL approaches, we build an independent model, based instead on a social constraint-based semantics where constraints can be used for defining semantics of communicative acts in terms of their social effects. The approach here presented is also documented in [ACG+03a, ACG+03b].

5.1 Agent Communication Languages: state of the art

In most proposals for agent systems, the formalism used to express knowledge exchange between agents is an ACL.

An ACL provides language primitives (also called *communicative acts*, *utterances*, and in certain contexts *performatives*, *illocutions*, or *dialogue moves*) which, mostly taking inspiration from the Speech Act theory [Aus62], implement the agent communication model. Each performative is associated with an attitude which represents its *meaning* (e.g., why the communicative act is performed, and what the expected effects are). Therefore, ACLs (as, for instance, KQML [FLM97] and FIPA ACL [FIP01a]) have much higher expressive power than traditional inter-process communication primitives.

Moreover, since ACL primitives are intended as means to support knowledge exchange, the content of each communicative act is usually expressed by means of a proper *content language*, whose features can express in an accurate way the information to be transferred.

More formally, an ACL is defined by two languages:

- the *communication* language, represented by a set of communication performatives, each corresponding to a different illocution;
- the *content* language which expresses the information to be transferred.

In recent years, much effort has been devoted to the definition of a standard for ACL [FIP01a]; however, the definition of ACL semantics is still an open issue.

Currently, in the international agent community, the main approaches to define ACL semantics are the *mentalistic* [FLM97, FIP01a], the *conversational* [GHB00], and the *social* approach (see [Sin98]).

Mentalistic approaches define ACL semantics in terms of agents mental states. The KQML language [LFP99] is divided into three layers:

- the *content* layer, where the actual content of the language is. KQML does not specify a particular content language;
- the *message* layer, which identifies the network protocol to be used to deliver the message, and describes optional features, such as the content language or the ontology;
- the *communication* layer, which describes the low-level communication parameters such as the identity of the sender and receiver and the identifier associated with the communication.

The semantics of KQML is given in terms of:

- *pre-conditions*, which define the necessary conditions for an agent to send or to successfully receive a message;
- *post-conditions*, which define the state of both interlocutors after a successful utterance or receipt of a performative;
- *completion* conditions, which define the final state after a conversation has taken place.

For instance, the semantics of $tell(A, B, X)$ is [LFP99]:

- **Pre(A)**: $\textsc{Bel}(A, X) \wedge \textsc{Know}(A, \textsc{Want}(B, \textsc{Know}(B, S)))$
- **Pre(B)**: $\textsc{Int}(B, \textsc{Know}(B, S))$, where S may be any of $\textsc{Bel}(B, X)$ or $\neg\textsc{Bel}(B, X)$
- **Post(A)**: $\textsc{Know}(A, \textsc{Know}(B, \textsc{Bel}(A, X)))$
- **Post(B)**: $\textsc{Know}(B, \textsc{Bel}(A, X))$
- **Completion**: $\textsc{Know}(B, \textsc{Bel}(A, X))$

It can be noticed that all the semantics are given in terms of mental states of sender and receiver; thus, in order to verify that an agent is communicating according to such semantics, it is necessary to access its mental state.

KQML+ [BM98] is an attempt to overcome the limitations of KQML that the authors detect, namely:

- out of the five categories of performatives that the speech act theory recognizes (*directive*, *assertive*, *commissive*, *expressive* and *declarative*), KQML performatives are limited to the assertive and directive categories;
- the choice of names of performatives is inappropriate, as it does not properly reflect their meaning;

- different authors use different performatives for the same communication task.

To overcome these limitations, and in particular to ground their ACL more consistently onto the speech act theory than in the case of KQML, the authors propose new performatives and new slots in a KQML+ message. In particular, a *primitive-performative* slot is added, to represent which of the five categories of speech acts the KQML+ message belongs to.

A KQML+ message is used by an agent to express its positioning with respect to a communicative state. The choice of the primitive and expressed performative follow the positioning: for instance, a propose positioning can be expressed by a message with *direct* in the primitive-performative slot and *ask* in the expressed-performative slot (which corresponds to the performative of KQML).

FIPA ACL assumes a BDI (Belief, Desire, Intention) model for the agents [RG92b], and relies on it for defining the semantics of communicative acts in terms of Feasibility Preconditions (i.e., the conditions that have to be satisfied for the communicative act to be planned) and Rational Effects (i.e., the expected effect that the communicative act would have).

As an example, the definition of the FIPA ACL performative *request* is as follows:
`<Sender, REQUEST (Receiver,a)>`

$$\begin{aligned}
\texttt{FP}: \;& \texttt{FP}(\texttt{a})[\texttt{Sender}\backslash\texttt{Receiver}] \wedge \\
& \texttt{B}_{\texttt{Sender}}\texttt{Agent}(\texttt{Receiver}, \texttt{a}) \wedge \\
& \texttt{B}_{\texttt{Sender}} \neg \texttt{PG}_{\texttt{Receiver}} \texttt{Done}(\texttt{a}) \\
\texttt{RE}: \;& \texttt{Done}(\texttt{a})
\end{aligned}$$

where

- FP denotes the *feasibility preconditions* of the act;
- RE denotes the *rational effect* of the act;
- $\texttt{FP}(\texttt{a})[\texttt{Sender} \setminus \texttt{Receiver}]$ denotes the part of the FPs of a, which are mental attitudes of the Sender;
- $\texttt{B}_{\texttt{Sender}}\texttt{Agent}(\texttt{Receiver}, \texttt{a})$ means that Sender believes that Receiver can perform a;
- $\texttt{B}_{\texttt{Sender}} \neg \texttt{PG}_{\texttt{Receiver}}$ Done(a) means that Sender believes that Receiver does not (yet) intend to perform a.
- Done(a) means that a has "just" taken place.

It is worth noticing that, according to this definition, the agent Sender should not only be aware of its own mental state, but also have beliefs about the agent Receiver's mental state.

The mentalistic approach to the ACL semantics has been much criticized mainly because its underlying assumptions regarding agents' internals are not realistic in open societies of heterogeneous agents. As Singh stated [Sin98], emphasizing mental agency leads to the supposition that agents should be primarily understood in terms of mental concepts, such as beliefs and intentions: this approach supposes, in essence, that agents can read each other's minds. Whenever agents' mental states are not accessible, which is reasonably the case if agents operate in open and heterogeneous environments, it is impossible to verify semantic compliance of communicative acts.

In addition, as observed in [GP01], the meaning of each communicative act is strictly linked to a specific mental state (e.g., the feasibility precondition), so communicative acts are not

flexible for use in different contexts. In this perspective, an ACL's formal semantics should better emphasize social agency. This approach recognizes that communication is inherently public and thus depends on the agent's social context.

A different approach relies on Conversation Policies (CP) [GHB00], which express meaning and composition of speech acts for agents by means of static structures which define the sequences of communicative acts making a coherent conversation. The main advantage of this approach is that in this perspective a communicative act does not have an absolute meaning, but its semantic rather depends on the conversation (i.e., the context in which it is uttered), thus overcoming one of the main drawbacks of the mentalistic approach. However the conversational approach has been criticized for its lack of flexibility, i.e., the lack of compositional rules governing how protocols are extended or merged (see [MC02]).

The *social* approach defines ACL semantics in terms of the effects of the communicative acts on the society. Following this approach, even if the agents' mental state cannot be accessed, it is possible to verify whether communicating agents in a society comply to some social laws which regulate the interactions.

In this setting, a commitment-based semantics [Sin00, FC02] could fully reflect this, by adopting the notion of social commitment in a multi-agent setting. A social commitment is an obligation which binds an agent (usually the speaker in a communicative act) to the society. So, each social commitment refers to a *content* (i.e., the action, or the proposition to be made true), a *debtor* (i.e., the agent engaged to make the content true) and a *creditor* (i.e., the agent relative to which the commitment is made).

In particular, in Singh's work [Sin00] three levels of semantics for each communication performative are defined: the objective claim (that the subject of the communication is true), the subjective claim (that the communication is sincere) and the practical claim (that the speaker is justified in making the communication).

For instance, the semantics of the performative `inform(s, h, p)` is:

- `s` is committed towards `h` that `p` holds (objective claim);
- `s` is committed towards `h` that `s` believes `p` (subjective claim);
- `s` is committed towards the society that he has reasons to believe `p` (practical claim).

In this way, the mentalistic approach is adopted only at the subjective level, while at the practical level a commitment towards the agent society is used.

The social approach is applied to the definition of ACL semantics in [FC02], where an operational specification of an ACL is given in an object-oriented framework by means of the *commitment* class. A commitment represents an obligation for its *debtor* towards its *creditor*. A commitment is described by a finite state automaton, whose states (which can take the values of *empty*, *pre-commitment*, *cancelled*, *conditional*, *active*, *fulfilled* and *violated*) can change by application of methods of the commitment class, or of rules triggered by external conditions. Semantics of communicative acts is specified in terms of methods to be applied to a commitment when a communicative act is issued.

Within this framework, for instance, the semantics of the assertive `inform` performative is given as follows:

$$\texttt{inform}(\texttt{Sender}, \texttt{Receiver}, \texttt{P}) = \texttt{make C}(\texttt{Sender}, \texttt{Receiver}, \texttt{P}, \texttt{true}, \texttt{CC})$$

where `C(Sender,Receiver,P,true,CC)` is a (*conditional*) commitment (with the condition already *true*) made by `Sender`, to agent `Receiver` (the "creditor" of the commitment) that `P`

(the content) will be satisfied. The effect of this utterance will define a commitment initially in the transitory state `CC`, that will immediately move (due to the *true* condition) in the state `A`. It might be later either fulfilled (if `P` becomes *true*), or violated (if `P` becomes *false*).

As a further example, let us report also the commitment-based semantics of the directive performative `request`:

$$\texttt{request}(\texttt{Sender}, \texttt{Receiver}, \texttt{F}) = \texttt{make C}(\texttt{Receiver}, \texttt{Sender}, \texttt{F}, \texttt{true}, \texttt{P}),$$

which means: a (*conditional*) pre-commitment with `Receiver` as the debtor to agent `Sender` (the creditor) that `F` will be performed (the content in this case is an action to be possibly performed by `Receiver`) will be executed. In case of an explicit acceptance by the `Receiver` (with an *accept* act), the commitment will move in the transitory state `CC` to reach (due to the *true* condition) the state `A`. It might be later either fulfilled (if the `Receiver` performs `F`), or violated (if the `Receiver` does not perform `F` within a given deadline). If, instead, the `Receiver` refuses the request, the commitment will move in the cancelled state (`C`).

5.2 Constraint-based semantics for CCL

Since the SOCS project is aimed at modelling open societies of heterogeneous computees, among the approaches to the definition of semantics of communication, the *social* approach appears more appropriate, because it allows for verification of compliance of communicative acts to protocols, regardless of the internal structure of individual computees.

The semantics of communicative acts from the point of view of a single computee is, in our opinion, not to be constrained, since it does not *directly* affect the correct functioning of the society. In human societies, from a social point of view it is not necessary that contractors fully understand a contract they are signing for the contract to be effective, although it is obviously in their own interest, and it will make easier for them to fulfill the expectations on their behavior: we think a similar approach should be adopted in modelling interaction among computees. Thus, although in WP1 we show how to accommodate communication within the model of the single computee, we do not impose this as a requirement.

We rather distinguish between

(a) communicative acts as observable facts (from an external perspective);

(b) communicative acts as actions planned (from an internal perspective) in order to make a goal feasible.

From an external perspective, communicative acts may be subject to the rules (in a broad sense) of the society. From an internal perspective, communicative acts are *motivated* by their intended effect (see D4, [KSST03]). This approach smoothly integrates in the framework adopted to specify interaction protocols (see Section 3.3). The constraint-based semantics also guarantees the requirements of flexibility, declarativeness, readability and verifiability.

We express semantics of communicative acts in a CCL, exploiting the same IC_Sbased framework described in Section 3.3.

In the following, unless otherwise specified, we will express a computee's communicative act as follows:

$$\mathit{CommunicativeActId}(X, Y, \mathit{Content}, \mathit{Context})$$

where X is the speaker computee, Y is the intended recipient(s), *Content* is the content of the message and *Context* is an identifier of the interaction context between X and Y. In the case of

dialogues, *Context* can be a dialogue identifier, set by the computee who initiates the dialogue. A possible event is:

$$\mathbf{H}(request(thomas, david, give(scooter), evening\_dialog), 21)$$

5.3 Social semantics of CCL performatives

In this section, we map into our framework some of the linguistic primitives defined in [FC02]. What follows should be intended neither as a final choice of communicative acts for a CCL nor as a final choice on how to the define the chosen ones, but as an example of how some of the most common communicative acts might be defined in our semantic framework.

It will be shown that the formalism of $\mathcal{IC}_S$ can express both social semantics of communicative acts and communication protocols, which yields the additional advantage that the same proof-procedures can be used to verify both.

5.3.1 Social semantics as expectations raised

In the following, we express the social semantics of communicative acts in terms of the expectations raised by a given set of events on the subsequent behavior of computees, by means of IC_S.

Assertives: inform - Intuitively, an *inform* communicative act is used by a computee to assert the truth of the content to another computee. In a commitment-based setting like that of [FC02], this equates to the speaker computee to *commit* to the truth of the content to the hearer computee.

In our framework, a possible definition of the semantics of *inform* is as follows:

$$\begin{aligned} &\mathbf{H}(inform(A, B, P, D), T) \\ \rightarrow &\mathbf{E}(true(A, B, P)) \end{aligned} \tag{18}$$

where, with $\mathbf{E}(true(A, B, P))$, we mean that A is responsible towards B with respect to the truth of P; in other words, if P is proved false, then A has violated a commitment towards B.

We are aware that verifiability is a problem here: who is supposed to verify the truth of P? According to our approach, built on the principle that it is not acceptable to make assumptions about computees (and, therefore, about their truthfulness) there should be a *super partes* entity in the society, equipped with a knowledge base allowing it to decide the truth value of the content of a message. If this is not the case, the only way is probably to associate no expectations to an *inform* act, and let the hearer computee decide about the trustworthiness of the speaker.

Commissives: promise and conditionalPromise - A *promise*, like an *inform*, commits the speaker to the truth of the content, but for the former the speaker is responsible for fulfilling it by a physical action.

$$\begin{aligned} &\mathbf{H}(promise(A, B, P, D), T_p) \\ \rightarrow &\mathbf{E}(do(A, B, P, D), T_d) : T_d \leq T_p + \tau \end{aligned} \tag{19}$$

where *do* is the action that should make P true. The constraint $T_d \leq T_p + \tau$, where τ is some constant, expresses that the expectation will be fulfilled only if the *do* event happens by the prescribed deadline $T_p + \tau$.

The expectation in a *conditionalPromise* becomes effective only when an event (which plays the role of a condition[6] that is external to the dialogue and thus, intuitively, is supposed not to be an action performed by the speaker) happens:

$$\begin{aligned} &\mathbf{H}(\mathit{conditionalPromise}(A, B, \mathit{cond}(P, Q), D), T_c) \\ \wedge&\mathbf{H}(Q, T_Q) \\ \rightarrow&\mathbf{E}(\mathit{do}(A, B, P, D), T_d) : T_d \leq \mathit{max}(T_c, T_Q) + \tau \end{aligned} \tag{20}$$

where Q is a term describing an event and, as usual, $\mathbf{H}(Q, T_Q)$ expresses that Q happens at time T_Q.

Directives: request and conditionalRequest - A *request* does not, by itself, generate any expectation. The hearer computee can either *accept* or *reject* the content of the *request*, by the corresponding communicative acts. Only in case of an *accept* the content of the *request* becomes expected:

$$\begin{aligned} &\mathbf{H}(\mathit{request}(A, B, P, D), T_r) \\ \wedge&\mathbf{H}(\mathit{accept}(B, A, P, D), T_a) \\ \wedge&T_r < T_a \\ \rightarrow&\mathbf{E}(\mathit{do}(B, A, P, D), T_d) : T_d \leq T_a + \tau \end{aligned} \tag{21}$$

where $T_r < T_a$ means that the expectation will be raised only if the *request* happens before the *accept*.

A *conditionalRequest* is different from a *request* in that its content becomes the content of an expectation only once the hearer has *accept*ed it *and* an event, specified as a condition in the content of the *conditionalRequest*, has happened.

$$\begin{aligned} &\mathbf{H}(\mathit{conditionalRequest}(A, B, \mathit{cond}(P, Q), D), T_r) \\ \wedge&\mathbf{H}(\mathit{accept}(B, A, \mathit{cond}(P, Q), D), T_a) \\ \wedge&T_r < T_a \\ \wedge&\mathbf{H}(Q, T_Q) \\ \rightarrow&\mathbf{E}(\mathit{do}(B, A, P, D), T_d) : T_d \leq \mathit{max}(T_a, T_Q) + \tau \end{aligned} \tag{22}$$

There is no need to express the semantics of a *reject* by a IC_S, because a *reject*ed *request* (or *conditionalRequest*) generates no expectations.

Proposals: propose - A *propose* is similar to a *conditionalRequest*, with the difference that for the former the speaker is able by itself to fulfill the condition by a *do* action.

As *conditionalRequest*, however, *propose* does not, by itself, generate any expectation. It is with *accept* that both the speaker and the hearer become committed to their respective expectations. We assume that the hearer and the speaker can have different time limits for fulfilling the expectations on their behavior.

[6] In [FC02], the condition is expressed as a *temporal proposition* object. Temporal propositions express the truth value (true, false or undefined) of a statement (about some state of affairs holding, or some action having been performed, or commitment having been created) in a given time interval, with existential or universal temporal quantification. Thus, Fornara and Colombetti's framework can express a broader set of conditions than ours.

$$\begin{aligned}
&\mathbf{H}(propose(A,B,prop(P_A,P_B),D),T_p)\\
\wedge&\mathbf{H}(accept(B,A,prop(P_A,P_B),D),T_a)\\
\wedge&T_p<T_a\\
\rightarrow&\mathbf{E}(do(A,B,P_A,D),T_{d_A}):T_{d_A}\leq T_a+\tau_A\\
\wedge&\mathbf{E}(do(B,A,P_B,D),T_{d_B}):T_{d_B}\leq T_a+\tau_B
\end{aligned}\tag{23}$$

5.3.2 Permissible sequences of communicative acts

In the following, we will show how to express permissible sequences of communicative acts (i.e., communication protocols) by means of the $\mathcal{IC}_S$.

For instance, we may want to express that a *request*, *conditionalRequest* or *propose* is expected to be followed by an answer (either *accept* or *reject*) by a specified amount of time. This is achieved by the following IC_S:

$$\begin{aligned}
&\mathbf{H}(request(A,B,P,D),T_r)\\
\rightarrow&\mathbf{E}(accept(B,A,P,D),T_a):T_a\leq T_r+\tau\\
\vee&\mathbf{E}(reject(B,A,P,D),T_e):T_e\leq T_r+\tau
\end{aligned}\tag{24}$$

$$\begin{aligned}
&\mathbf{H}(conditionalRequest(A,B,P,D),T_c)\\
\rightarrow&\mathbf{E}(accept(B,A,P,D),T_a):T_a\leq T_c+\tau\\
\vee&\mathbf{E}(reject(B,A,P,D),T_e):T_e\leq T_c+\tau
\end{aligned}\tag{25}$$

$$\begin{aligned}
&\mathbf{H}(propose(A,B,P,D),T_p)\\
\rightarrow&\mathbf{E}(accept(B,A,P,D),T_a):T_a\leq T_p+\tau\\
\vee&\mathbf{E}(reject(B,A,P,D),T_e):T_e\leq T_p+\tau
\end{aligned}\tag{26}$$

It is also possible to specify that a computee cannot perform both an *accept* and a *reject* in the same dialogue, by means of IC_S (27) and (28):

$$\begin{aligned}
&\mathbf{H}(accept(A,B,P,D),T_a)\\
\rightarrow&\mathbf{NE}(reject(A,B,P,D),T_r):T_r\geq T_a
\end{aligned}\tag{27}$$

$$\begin{aligned}
&\mathbf{H}(reject(A,B,P,D),T_r)\\
\rightarrow&\mathbf{NE}(accept(A,B,P,D),T_a):T_a\geq T_r
\end{aligned}\tag{28}$$

On the other hand, we may want to express that an *accept* or a *reject* are expected not to happen, unless a *request*, *conditionalRequest* or *propose* have happened before.

A possible way to express this is that if no *request*, *conditionalRequest* or *propose* has happened before a given time, no *accept* or *reject* should happen at that time:

$$\begin{aligned}
&\neg\mathbf{H}(request(A,B,P,D),T_r):T_r<T_0\\
\wedge&\neg\mathbf{H}(conditionalRequest(A,B,P,D),T_c):T_c<T_0\\
\wedge&\neg\mathbf{H}(propose(A,B,P,D),T_c):T_p<T_0\\
\rightarrow&\mathbf{NE}(accept(B,A,P,D),T_0)\\
\wedge&\mathbf{NE}(reject(B,A,P,D),T_0)
\end{aligned}\tag{29}$$

Another way to achieve the same result[7] is to state that if an *accept* or *reject* has happened, then a *request*, *conditionalRequest* or *propose* is expected to have happened before, as in the following two IC_S:

$$\begin{aligned} &\mathbf{H}(accept(A,B,P,D),T_a) \\ \rightarrow&\mathbf{E}(request(B,A,P,D),T_r) : T_r < T_a \\ \vee&\mathbf{E}(conditionalRequest(B,A,P,D),T_c) : T_c < T_a \\ \vee&\mathbf{E}(propose(B,A,P,D),T_p) : T_p < T_a \end{aligned} \tag{30}$$

$$\begin{aligned} &\mathbf{H}(reject(A,B,P,D),T_e) \\ \rightarrow&\mathbf{E}(request(B,A,P,D),T_r) : T_r < T_e \\ \vee&\mathbf{E}(conditionalRequest(B,A,P,D),T_c) : T_c < T_e \\ \vee&\mathbf{E}(propose(B,A,P,D),T_p) : T_p < T_e \end{aligned} \tag{31}$$

We call the expectations in IC_S (30) and (31) *backward* expectations, because they regard events that should have happened before the event that has caused them.

As shown by the previous examples, the $\mathcal{IC}_S$ represent a uniform formalism that is expressive enough for both social semantics of communicative acts and communication protocols, two aspects which are usually specified (and checked for compliance) at different levels. For instance, an *accept* communicative acts will raise expectations about communicative acts that are expected to have been issued before in order to make the *accept* permissible (because of IC_S (30)), about communicative acts that are expected not to happen because they would contradict the *accept* (due to IC_S (27)) and about physical actions that are expected to happen as consequence (due to IC_S (21), (22), or (23)). A uniform formalism and verification method for all these different social aspects of communication is, in our opinion, a valuable advantage of our framework.

6 Formal verification of properties

In this section, we make a point about proving and verifying properties of societies of computees. We anticipate that the work that we are going to present here is very preliminary, since it is planned to be matter of study for the third year of the project. The purpose of this section then is mainly to provide a concrete support to our claim that a formal Computational Logic-based approach to modelling agent societies is very promising, since it bridges the gap between specification and verification.

In [GP02, PG02], Guerin and Pitt propose a classification of properties that are relevant for e-commerce systems, with particular emphasis on properties of protocols and interactions. This is only a part of the properties that we wish to study along the project, and it is especially focussed on properties of interactions. A more elaborated document about properties is [EMS+03].

In this setting, they propose a formal framework for verification of properties of "low level computing theories required to implement a mechanism for agents" in an *open* environment, where by open the authors mean that the internals of agents are not public, as discussed in Section 1.1.

[7] The result is the same in that, in both cases, a non-conforming sequence of communicative acts will be recognized as a violation; however, the set of expectations raised in the two cases is different.

Verification of properties is classified in three types, depending on the information available, and whether the verification is done at design time or at run time:

Type 1: verify that an agent will always comply;

Type 2: verify compliance by observation;

Type 3: verify protocols' properties.

As for *Type 1* verification, the authors propose using a model checking algorithm for agents implemented by a finite state program. As for *Type 2* verification, the authors refer to work done by Singh [Sin00], where "agents can be tested for compliance on the basis of their communications". As for verification of *Type 3*, the authors show how it is possible to prove properties of protocols by using only the ACL specification. They construct a fair transition system representing all possible observable sequences of states and prove by hand that the desired properties hold over all computations of the multi-agent system. This type of verification is demonstrated by an auction example.

In this section, we sketch a possible methodology that can be followed to achieve in our setting all the above three types of verification, in an *automatic* way. Such a methodology is based on the fact that both (public) protocols and (internal) computee policies are expressed in the same formalism. This is still preliminary work, but we include it here in order to back up some claims made in Section 1.1.

Let us briefly describe an example adapted from [PG02]. An auction is held to sell a *lot* of items. The system that implements the auction is composed of four computees: an auctioneer (*auct*), two bidders (*bidder1* and *bidder2*), and a computee *rand* who will decide the winner randomly in case both *bidder1* and *bidder2* declare the same value. The auction mechanism is designed so to show properties such as: feasibility, incentive compatibility, individual rationality, optimal expected value for the seller, symmetry.

In particular, the auction mechanism is a modified Vickrey auction, where a bidder can either bid 'low' (if its valuation for the item being sold is 3) or 'high' (if its valuation is 4), and in case a high bidder wins the price to pay is $3 + \frac{2}{3}$, otherwise it is 3. The winner determination is as follows: if there is exactly one high bidder, it wins, otherwise *rand* will generate a random value in $\{1, 2\}$, and the winner will be determined accordingly (*bidder1* if *rand* produces 1 and vice versa).

In Figure 2, we define the protocol by giving the $\mathcal{IC}_S$. In Figure 3, we define the semantics of the communicative acts for this example.

6.1 Verification for open systems

Let us now consider the three different types of verification for open systems, and sketch how they could be achieved in our framework.

6.1.1 Type 1 verification (a computee will always comply)

Verification that a computee will always comply cannot be done by externally monitoring its behavior. Quoting Hume, "we are in a natural state of ignorance with regard to the powers and influence of all objects when we consider them a priori" ([Hum48], IV.ii.32). For this kind of verification, we need to have access to the computees' internals. We can express the computee's program and policies by means of a Logic Programming-based formalism, such as, for instance,

$\mathbf{H}(tell(bidder1, auct, h)) \ \wedge \ \mathbf{H}(tell(bidder2, auct, l))$
$\quad \rightarrow \mathbf{E}(award(auct, bidder1, 3 + \frac{2}{3}))$

$\mathbf{H}(tell(bidder1, auct, l)) \ \wedge \ \mathbf{H}(tell(bidder2, auct, h))$
$\quad \rightarrow \mathbf{E}(award(auct, bidder2, 3 + \frac{2}{3}))$

$\mathbf{H}(tell(bidder1, auct, h)) \ \wedge \ \mathbf{H}(tell(bidder2, auct, h))$
$\quad \rightarrow \mathbf{E}(request(auct, rand, number))$

$\mathbf{H}(tell(bidder1, auct, l)) \ \wedge \ \mathbf{H}(tell(bidder2, auct, l))$
$\quad \rightarrow \mathbf{E}(request(auct, rand, number))$

$\mathbf{H}(tell(bidder1, auct, h)) \ \wedge \ \mathbf{H}(tell(bidder2, auct, h)) \ \wedge \ \mathbf{H}(tell(rand, auct, 1))$
$\quad \rightarrow \mathbf{E}(award(auct, bidder1, 3 + \frac{2}{3}))$

$\mathbf{H}(tell(bidder1, auct, h)) \ \wedge \ \mathbf{H}(tell(bidder2, auct, h)) \ \wedge \ \mathbf{H}(tell(rand, auct, 2))$
$\quad \rightarrow \mathbf{E}(award(auct, bidder2, 3 + \frac{2}{3}))$

$\mathbf{H}(tell(bidder1, auct, l)) \ \wedge \ \mathbf{H}(tell(bidder2, auct, l)) \ \wedge \ \mathbf{H}(tell(rand, auct, 1))$
$\quad \rightarrow \mathbf{E}(award(auct, bidder1, 3))$

$\mathbf{H}(tell(bidder1, auct, l)) \ \wedge \ \mathbf{H}(tell(bidder2, auct, l)) \ \wedge \ \mathbf{H}(tell(rand, auct, 2))$
$\quad \rightarrow \mathbf{E}(award(auct, bidder2, 3))$

Figure 2: Definition of the auction protocol.

$\mathbf{H}(award(auct, Bidder, P))$
$\quad \rightarrow \mathbf{E}(buy(Bidder, lot, P))$

$\mathbf{H}(request(auct, rand, number))$
$\quad \rightarrow \mathbf{E}(tell(rand, auct, 1)) \ \vee \ \mathbf{E}(tell(rand, auct, 2))$

Figure 3: Semantics of the speech acts.

$tell(auct, bidder1, start) \ \wedge \ highbidder$
$\Rightarrow tell(bidder1, auct, h)$

$tell(auct, bidder1, start) \ \wedge \ \neg highbidder$
$\Rightarrow tell(bidder1, auct, l)$

Figure 4: *bidder1*'s policy (time is left implicit).

a set of reactive constraints in KB_{react}, which will be evaluated by the computee using its $\models_{react}$ capability, as it is shown in D4 [KSST03]. We give an example of a policy for *bidder1* in Figure 4, where we assume that the auctioneer agent *auct* starts the auction by sending a *start* message to all participants. We also assume, for the sake of simplicity, that *bidder1* is interested in buying the item sold in the auction, and that it is a high bidder, by considering the predicate *highbidder* to be true in *bidder1*'s knowledge base.

A way to prove that *bidder1* will always comply is to show that for all the IC_S expressing the protocol, if in the head of a IC_S there is a social event which is expected from *bidder1* (e.g., $\mathbf{E}(buy(bidder1, lot, 3 + \frac{2}{3}))$), then, the history of events that by social viewpoint leads to such expectation, leads by *bidder1*'s viewpoint to producing such event.

Our aim is to define a methodology to automatically obtain such proof (or its failure). The idea is to define a mapping of IC_S into normal logic programs. A possible such mapping, for a restricted set of IC_S is presented in [AGL$^+$03c].

For instance, using the mapping defined in [AGL$^+$03c], the first IC_S of Figure 2 is transformed into the clause:
$award(auct, bidder1, 3 + \frac{2}{3}) \leftarrow$
$tell(bidder1, auct, h) \wedge tell(bidder2, auct, l)$

Given the normal logic program P into which the protocol is mapped, we consider the abductive logic program $\langle P, IC, \mathcal{A} \rangle$, with $IC = \emptyset$ and $\mathcal{A}$ (the set of abducible predicates) equal to the undefined predicates of $H(P)$ (the Herbrand universe of P).

In order to prove that *bidder1* will always comply to the protocol, we consider both the protocol and the computee reactive rules. For all defined predicates $p \in P$, that represent actions to be taken by *bidder1* we execute the following two steps:

1. Firstly, we consider p as a goal to prove in P, and we obtain a collection of sets Δ_i of social events which entail p (possibly a minimal set?). These represent the history that generated some expectation.

2. Secondly, we consider p as a goal to prove within *bidder1*'s policies, and we obtain again a collection of sets Δ'_i of social events that would lead to the generation of p. If for all Δ_i there is a Δ'_i which is a subset of Δ_i, then the computee will always comply (because the course of events that leads to an "expectation about p" also leads to the generation of the action p according to *bidder1*'s policies.

6.1.2 Type 2 verification (compliance by observation)

For this kind of verification we need to be able to observe the computee's social actions, i.e., the communicative acts that they exchange. As in [PG02], we can assume that this can be

$tell(bidder1, auct, h) \ \wedge \ tell(bidder2, auct, h) \ \wedge \ tell(rand, auct, 1)$
$\rightarrow buy(bidder1, lot, 3 + \frac{2}{3}))$

Figure 5: Protocol property $\mathcal{M}'$.

achieved by policing the society. In particular, "police" computees will be able to "snoop" the communicative acts exchanged by computees and check at run time if they comply with the protocol specifications.

This kind of verification is the one which is most easily accommodated in our social framework. The purpose of its mapping to an ALP framework, and the operational counterpart introduced in Section 4.3 are mainly motivated by the possibility to check the compliance of the overall computation of a society of computees, with respect to the society protocols.

In [ACG+03a] we already propose a possible implementation of the verification of compliance to $\mathcal{IC}_S$ by observation based on the CHR language [Frü98].

6.1.3 Type 3 verification (protocol properties)

$\mathcal{M}'$, defined in Figure 5, is the property that we want this auction mechanism to exhibit. $\mathcal{M}'$ is the first conjunct of the property $\mathcal{M}$ defined in [PG02]. The proof of $\mathcal{M}$ is obtained by repeating the proof that we show for $\mathcal{M}'$ to all conjuncts in $\mathcal{M}$. We assume that all properties that we want to show about protocols can be expressed in the form of integrity constraints. $\mathcal{M}'$ says: if *bidder1* and *bidder2* are both high bidders, and *rand* generates 1, then *bidder1* will buy the *lot*.

In order to prove the protocol properties we do not need to access the computees' internals, nor to know anything about the communication acts of the system, because it is a verification which is statically done at design time. In particular, it can be done by a simple top-down derivation in Logic Programming.

We could define the algorithm for the proof of a property π expressed as an integrity constraint: $Body \rightarrow Head$, for the case of a protocol which is expressed by Social ICs with no disjunctions in the head. Again, we restrict ourselves to the case of normal logic programs.

As we did with Type 1 verification, in order to prove that π holds given the protocol, we could consider the normal logic program P that we obtain from the protocol through the mapping $\mathcal{T}$. Then, we consider $Head$ as a goal to prove in $P \cup Body$ (the preconditions of the property are put together with the logic program). π holds if $P \cup Body \models \pi$.

7 Extensions

In this section, we discuss some extensions that we have been partially studied. This is work in progress, and describes our viewpoint about very challenging issues. We show different ideas that we tried to develop, with advantages and drawbacks for each one.

7.1 Recovery from violations

We have studied different types of problems related to recovery and violations, and found some partially satisfactory solutions. We discuss the problems and the possible solutions.

7.1.1 Problems with violations

The problem of recovering from a violation can be divided in the following steps:

Unexpected events should not start protocols - As we have seen, actions can trigger new expectations. On the other hand, computees are free to perform actions that were expected not to happen. In many cases, actions that were expected not to happen should not trigger interaction protocols, but, possibly, recovery or sanctioning protocols.

Symmetrically, if a protocol can be started by the non-happening of an event, then the non-happening should trigger new expectations only if the event was not expected to happen.

Deciding the culprit - When an expectation **E** (or a **NE**) is violated, the culprit could be the computee that should have performed (resp. should not have performed) the expected (forbidden) action or it may be a different computee. The recovery procedure should decide which computee is responsible for the violation and start a corresponding sanctioning protocol.

Recovery from violation - After a violation has been detected, an action should be performed to lead the society to a consistent (non-violation) state. The society can acknowledge the violation, and remove the inconsistency. We do not yet address this problem.

In the following, we address the first two issues.

7.1.2 Wrong trigger

Events expected not to happen, that actually happen, in many cases should not generate further expectations within that protocol, but should activate instead a different, recovery protocol. For example, let us consider this simple example.

Example 15.

$$\begin{array}{l}\mathbf{H}(tell(Seller, Buyer, offer(Item), Price), T_{offer}) \\ \rightarrow \mathbf{E}(tell(Buyer, Seller, accept(Item), Price), T_{Accept}) \\ \vee \mathbf{E}(tell(Buyer, Seller, refuse(Item), Price), T_{Refuse})\end{array}$$

$$\begin{array}{l}\neg\mathbf{H}(tell(Seller, Buyer, offer(Item), Price), T_{offer}) \\ \rightarrow \mathbf{NE}(tell(Buyer, Seller, accept(Item), Price), T)\end{array} \quad (32)$$

$$\begin{array}{l}\mathbf{H}(tell(Buyer, Seller, accept(Item), Price), T_{Accept}) \\ \rightarrow \mathbf{E}(deliver(Seller, Buyer, Item), T_{Deliver})\end{array}$$

Intuitively, the protocol says that a seller can offer an item to a buyer, and the buyer is expected either to accept or to refuse. If the buyer accepts, then the seller is expected to deliver the good. However, the buyer should not accept a good that was not offered.

Suppose now that a buyer sends an acccept message for a good that was not offered:

$$\mathbf{H}(tell(buyer_1, seller_1, accept(ferrari\_car), 1\$), 1)$$

What is the expected behavior of $seller_1$? Is it supposed to deliver a Ferrari car taking 1$ as payment?

Intuitively, we do not want the seller to be expected to deliver a good if the corresponding accept was expected not to be raised.

Indeed, one could argue that, as in commitments, the expectation about delivery does not depend on the single *accept* of the buyer, but on the couple *offer-accept*, thus one should rewrite the protocol as:

$$\begin{array}{l}\mathbf{H}(tell(Seller, Buyer, offer(Item), Price), T_{offer}) \\ \wedge\mathbf{H}(tell(Buyer, Seller, accept(Item), Price), T_{Accept}) \\ \rightarrow \mathbf{E}(deliver(Seller, Buyer, Item), T_{Deliver})\end{array}$$

However, if the protocol is rather complex, there may be many ways to reach a state in the protocol, so the user has to write very complex implications.

Another possible idea is to avoid triggering (non-recovery) protocols if the action gave rise to violation. For example, we may write:

$$\begin{array}{l}\mathbf{H}(tell(Buyer, Seller, accept(Item), Price), T_{Accept}) \\ \wedge\neg\mathbf{NE}(tell(Buyer, Seller, accept(Item), Price), T_{Accept}) \\ \rightarrow \mathbf{E}(deliver(Seller, Buyer, Item), T_{Deliver})\end{array}$$

Thus, the expectation about delivery is raised if the event *accept* was not expected not to happen (in a sense, if the action was not forbidden, and thus possible, by the protocol).

Symmetrically, one might have a protocol triggered by the non-happening of an event, like

$$\neg\mathbf{H}(p(X)) \rightarrow \mathbf{E}(q(Y))$$

Again, we might think that the action should trigger only if the non happening of event X did not give violation:

$$\neg\mathbf{H}(p(X)) \wedge \neg\mathbf{E}(p(X)) \rightarrow \mathbf{E}(q(Y))$$

Let us finally consider the following examples:

Example 16. *"If I do not* ask *you something, you should not reply."*

$$\neg\mathbf{H}(ask) \rightarrow \mathbf{NE}(reply)$$

If I did not ask you something but I was supposed to, can you reply?

Example 17. *"If I do not send you corrections at 7, submit the paper within the deadline."*

$$\neg\mathbf{H}(corrections, 7) \rightarrow \mathbf{E}(submit, T) : T < T_{deadline}$$

If I send you corrections, but I was not supposed to, are you not expected to send the paper within the deadline?

This issue is more intuitive with expectations about negative behavior. Our current understanding is that the wrong trigger of protocols because of (violating) happened events is an important issue, while its symmetric is not; however, in our framework it is possible to address both of the issues.

To recap, the solution proposed in this section for recovery from violations divides the protocol into two layers: a *normal* and a *recovery* protocol. The normal protocol is triggered only by events that do not raise violation, while the recovery protocol is triggered by events that gave violation. In this way we address the problems identified earlier as follows:

1. Events that violate the protocols can be easily made harmless, in the sense that normal protocols are not triggered by events expected not to happen.
2. The user writes the recovery protocol, thus (s)he decides both the culprit and the punishment.

It is worth noticing that this solution fits perfectly in the declarative semantics (Section 4.1). A proof-procedure designed to find an admissible, coherent, consistent and fulfilled set of expectations will never trigger a recovery protocol (which is triggered only in case of violation, i.e., if the set of expectation is not fulfilled). On the other hand, a proof that accepts also a (admissible, coherent, consistent but) non fulfilled set of expectations will punish the culprit decided by the user by means of the recovery protocol. In the implementation, the proof-procedure will strive to find a fulfilled set of expectations, but if such a set does not exist, will take (as a second choice) a violation state, and trigger the recovery protocol.

7.2 Trust and reputation in societies of computees

In recent times, *trust* has become an important notion in Multi-Agent Systems, especially in electronic commerce related applications and in the design and modelling of institutions [ALF99]. Trust can be defined as a subjective expectation an agent has about another's future behavior based on the history of their encounters [MM02]. According to Dellarocas [Del02],

> "the production of trust has three prerequisites:
>
> - an agent should know its utility function;
> - an agent should set a minimum threshold of satisfaction relative to a transaction;
> - an agent should estimate the trustworthiness of its prospective trading partners.
>
> Of the three elements of trust computation the first is usually internal and private to an agent. The second is either internal or the explicit result of a negotiation process that precedes a transaction. The last one, trustworthiness, is the trickiest one to assess."

In order to estimate the trustworthiness of an agent or institutions, many models have been proposed, for which the role of external information is very important. Some of these are based on the notion of *reputation*.

There are several definitions of reputation: "Reputation is a characteristic or attribute ascribed to one person by another. Operationally, this is usually represented as a prediction about likely future behavior. It is, however, primarily an empirical statement. Its predictive power depends on the supposition that past behavior is indicative of future behavior." [Wil85] "The reputation of an agent s as perceived by agent b in the context of transaction $t_i \in T$ with critical attribute set $\mathbf{R}$ is its trustworthiness distribution $\tau_b^s(\mathbf{R}, t_i)$ in the special case where the estimation of $\tau_b^s(\mathbf{R}, t_i)$ is based on information about the past behavior of s in transactions of class T" [Del02]. Finally, reputation is defined by [MM02] as the "perception that an agent has of another's intentions and norms".[8] Other definitions of reputation can be found in [CF98, Mar94, ZM99].

[8] The notion of *norm* adopted by the authors is taken by Ostrom, 1998 [Ost98]: "heuristics that individuals adopt from a moral perspective, in that these are the kinds of actions they wish to follow in living their life."

7.2.1 Qualitative trust and reputation management in societies of computees.

Most approaches to reputation management in multi-agent systems adopt numeric methods to compute trust, to define satisfaction thresholds, to estimate the reputation of an agent. This results in methods that may be very efficient, but little informed and often depending on subjective estimates of other agents. Predicting the future behaviour of an individual based on numerical data might be convenient when a yes/no evaluation and a threshold are all is needed, and when a large amount of data is enough to grant stochastic value to numbers, but it might be limiting in other scenarios.

In global and open computation environments, and in the SOCS scenario in particular, we would rather prefer to provide the computees of a society with the means to evaluate the behaviour of the others based on their own notion (subjective) of trust.

Given this, it would be difficult if not impossible for a computee to look back at the history of the other computees (provided it is public, which is not always the case), and to evaluate it case by case every time it wants to estimate their trustfulness. It would require knowledge about protocols, which might have changed over time, about the society itself, and so on.

In SOCS, the notions of expectations, fulfillment, and violations can be considered a way to code the behaviour of a computee with respect to the social protocols and norms of a society. They are not numerical values, but pieces of knowledge that can be used to give a qualitative estimate of a computee's trustfulness.

Although trust and reputation are not a scenario of SOCS, we are considering them as an interesting extension to our current work.

7.2.2 Example of trust reputation

We conclude this section by briefly sketching an example of reputation management in societies of computees.

Let us consider three members of a society of computees: a, b, and c. Members of this society buy and sell items among each other. a needs to buy an item which is sold by b and c at equivalent prices.

The decision whether to buy the item from b or from c may rely on the degree of *trust* (however we define it) that a has about the other computees. We assume that a did not have any past encounter with either of them, therefore its notion of trust can be based upon the reputation that b and c have in the community.

a can collect information about b's and c's past behaviour in the society. In particular, the infrastructure – or a representative computee, d – could reply to a's questions about critical attributes in past transactions. Such attributes can be objective, such as b's and c's fulfillment of past expectations, violations that they generated in the past, sanctions, etc. Also, such attributes are not the whole history of their interactions in the society, but represent "filtered" information about their "good" or "bad" behaviour. While being able to provide objective information about trustworthiness, on the other hand, this model allows to take into account also attributes that are subjective (depending on a, in this example). In this approach not only the idea of reputation is subjective (as in [Del02]) but also the interpretation of the information used to model a computee's reputation is subjective.

Some possibilities that open up in this setting are, for instance:

- trust can be individually defined in the computee's knowledge base, as any other predicate. For instance, a computee could define "trustworthy" an individual that never left a

specific expectation unfulfilled (forgetting about other expectations that it may consider irrelevant);

- a computee can use some private integrity constraints to prevent any interaction with others if their "reputation" violates them;

- by looking at the pending expectations in the society, a computee could estimate the likelihood of some computees to take some actions rather than others;

- when violations and sanctions are considered private information, not to be disclosed outside of the infrastructure, the society infrastructure itself (or a computee like d in the example above) could process such information, in order to provide computees with "objective" evaluation of its members' trustfulness, based on criteria which could be made public.

7.3 Learning protocols

Learning and inductive capabilities are very useful functionalities for agents and computees too, as it has been pointed out in [SV00].

Learning is applicable at two different levels inside the project:

1. At level of the single computee, when the computee wants to refine his knowledge and behavior. This inductive capability mainly concerns with Workpackage 1. Among other knowledge, the computee can learn, for instance, also the protocols ruling its communication. In the inductive process, it works by exploiting its own knowledge $SOKB$ as background knowledge (and eventually, its own knowledge on the society and environment).

2. At the society level, learning is applied for refining knowledge and behavior (e.g., protocols) of the whole society. This inductive capability concerns Workpackage 2 since it mainly involves the social behavior emerging from performed communicative acts (as we discuss in the following).

The process of acquiring new or refined behavior patterns in a social context, is a task which pertains to members of a society, but that can be exploited from the society itself or by one of its members, covering a special role.

As pertaining learning and refinement of social rules, the proposed model allows for the easy use of well known learning techniques, to be exploited from single computees, but also from the society itself. The society can decide to learn (by some of its members having a special role) social rules from outcomes of interactions (possibly judged by some oracle), or refine its social rules when, for instance, it is observed that they are always not respected.

In the following, we discuss how to learn social rules from performed communicative acts (as judged from an oracle, for the moment).

A lot of work has been done in Multi-Agent Systems which exploit learning techniques. Before introducing our approach, we briefly survey the most recent approaches to learning in agent systems, and among them those most related with our purposes.

7.3.1 Social rules, protocols and learning

In several contexts, social rules minimize conflict and optimize global efficiency. Social rules are a natural part of any society, and the acquisition of those rules (by computees) is an important aspect of adaptive behavior which pertains Workpackage 1. From past experience, learning can be exploited in order to *behave socially* as, for instance, proposed in [Mat94].

Social learning is the process of acquiring new behavior patterns in a social context, by learning from other individuals of the society [Mat94]. In [Mat94], Mataric discusses how to learn social rules in homogeneous societies where social rules are shared by all individuals. In particular, through social reinforcement learning, she learns (robot) behaviors that do not produce immediate payoff for the single agent but benefit the group as a whole. In that context (which is that of a group of robots moving around a room and interested in getting food), learned social rules are those that minimize interference among agents to direct behavior away from individual greediness and toward global efficiency. She considers three types of reinforcement, one related with the individual perception of progress relative to the current goal, the second coming from observing the behavior of *conspecifics*, and the third received by conspecifics. In particular, this latter form of reinforcement do not require the agent to model the another agent's internal state. Since the agents belong to a homogeneous society in which they obey consistent social rules, any reward or punishment received by a conspecific would have been received by the agent itself in a similar situation. In this way, a society can develop social rules based on individual learning, i.e., without some centrally imposed arbiter. This is accomplished by agents which are able to estimate other agents' reinforcement, and their individual reinforcement is positively correlated with their conspecifics (see also [Mat98]).

Learned behaviors are condition-action pairs, and goal-driven control laws in particular, which couple sensory inputs and effector outputs (e.g., if a robot is near a stopped agent, then proceed).

The proposed approach is interesting since social rules are learned from individuals not only on the basis of their own experience, but taking into account information and experience coming from conspecifics too.

In our society model, any interaction relevant at social level is mapped into an event (stored in the $SEKB$) occurring in the social environment. Any computee can be aware, through passive observation (see D4, [KSST03]) of social acts amongst the rest of computees and, possibly, of sanctions to some of them, thus learning can exploit information from conspecifics too.

Other approaches have exploited learning in multi-agent systems to learn cooperative behavior, but possibly with a less distributed approach.

In [DE02] the authors propose an approach for learning cooperative behavior, where an agent in a team of cooperative agents is substituted by a new agent. The new agent learns how to cooperate with the other agents by means of on-line learning, i.e., by learning while solving the problem at hand. The learning starts from the strategy of the old agent, and is achieved by means of a genetic algorithm. The approach is tested on various versions of the pursuit game, in which a number of agents have to catch a "prey" by surrounding it.

In [LR02] the authors present a toolkit for the development of multi-agent systems based on learning. The toolkit allows the agent to use learning techniques such as reinforcement learning, Q-learning and neural networks. In order to coordinate the agents, each agent is assigned an utility function that is computed by a central authority so that by maximizing their local utility function the agents also maximize a global utility function. The agents apply learning in order

to maximize their local utility function.

Finally, some further approaches in Multi-Agent Systems have exploited learning techniques with the purpose of learning the strategy to adopt in negotiation protocols, or learning communication protocols.

In particular, in [ZS97, ZS96] the authors propose an approach for the modification of agent's strategies during negotiation. The modification is achieved by means of Bayesian learning. The authors illustrate their approach by means of an example. Consider a negotiation scenario that comprehends a buyer and a supplier. They have to negotiate the price of an item to be exchanged. Consider the negotiation process from the buyer point of view. The buyer has a belief about the reservation price of the supplier $RP_{supplier}$. The reservation price of the supplier is the threshold of offer acceptability, i.e., the price above which the supplier is willing to sell the item to the buyer.

The real value of $RP_{supplier}$ is unknown to the buyer. However, the buyer can update his belief (learn) about $RP_{supplier}$ based on the interactions with the supplier and on his domain knowledge. As a result of learning, the buyer is expected to gain more accurate expectation of the supplier's payoff structure, and therefore make more advantageous offers.

In [MOB03] the authors describe an approach for the inference of communication protocols from the conversations between the agents of a multi-agent system. Conversations are made up of sequences of messages exchanged inside the system. Stochastic grammatical inference is used in order to infer a Stochastic Deterministic Finite Automaton that represents a grammar describing the sequence of messages.

In the following, we concentrate ourselves on learning the social behavior emerging from performed communicative acts, as this is relevant to Workpackage 2.

7.3.2 Extended ILP for learning protocols

When protocols that rule the society are represented as integrity constraints and expectations interpreted as abducibles, as proposed above, two issues arise. The former concerns the operational method to ensure that the integrity constraints are satisfied, and it will be faced by adopting proper proof-procedures (see Section 4.3). The latter concerns the capability of automatically identifying/learning these protocols ($\mathcal{IC}_S$).

In particular, when protocols are declaratively stated as constraints (or constraining clauses as for the IFF proof-procedure [FK97]), it is worth exploring the use of machine learning techniques in order to equip computees with the capability of learning, via induction, the protocols ruling a society/institution.

We would like to equip societies of computees with inductive capabilities by exploiting for instance Inductive Logic Programming (ILP) techniques, already experimented by members of the SOCS Consortium in the past. The aim is to learn, incrementally, protocols when expressed as constraints, i.e., learn the integrity constraints. This might be useful whenever societies/institutions do not make public their own protocols or they are not yet assessed (but they will be in the future, as emerging behavior that can be learned).

We require, of course, to consider a proper abductive proof-procedure integrated with the inductive process. If an abductive proof-procedure is taken into account to address the operational support for protocols expressed as constraints, then we can take advantage, in the inductive process, of the integration between ILP and Abductive Logic Programming (ALP), already explored and faced by some members of the Consortium (UNIBO, DIFERRARA, CYPRUS). See also Appendix C.

We can assume that examples for the inductive process are the performed communicative acts, labelled by an evaluation (positive or negative, for the sake of simplicity) given by a sort of oracle. A more realistic evaluation can be obtained by "sanctioned events".

In the inductive process, background knowledge is represented by $SOKB$ and $SEKB$ (apart from examples), examples are **H** events which have taken place, properly recorded in their symbolic representation (split into positive examples, $E+$, and negative examples, $E-$, depending on their evaluation), and a language bias determines the syntax of the integrity constraints to be learned.

We can take advantage from previous integration of ILP and ALP and adopt Abductive Concept Learning introduced in Appendix C as the definition of the learning problem.

In the learning process applied to protocols, background knowledge is represented by the (known) society infrastructure which corresponds to the ALP triple:

$$\langle KB, \mathcal{E}, IC \rangle$$

Examples are abducible atoms themselves, i.e., set of positive events (E^+, which have got a positive evaluation since considered mandatory) and a set of negative events (E^-, which have got a negative evaluation since considered forbidden).

Given E^+, E^-, the ALP triple:

$$\langle KB, \mathcal{E}, IC \rangle$$

and a set S of possible society infrastructures (ALP programs), the problem is to find a new society infrastructure:

$$T' = \langle KB', \mathcal{E}, IC \cup IC' \rangle$$

such that T' belongs to S, and T' is complete and consistent with respect to E^+ and E^-.

Notice that, as result of the learning process, both the static knowledge base and IC can be enlarged. Furthermore, with the notion of correctness introduced above, the compliance of the learned knowledge and protocols to the given (positive and negative) examples can be obtained by also possibly abducing further hypotheses. These abducibles can be, in their turn, new input for a further inductive cycle.

7.3.3 Learning the *deadline* protocol

Let us see an example of learning of the IC in the case of a simple request-response protocol with deadline, involving positive expectations with deadline. Suppose that the protocol to be learned is represented by a single constraint stating that a request should be followed by an answer within two time ticks:

$$\begin{aligned} &\mathbf{H}(tell(S, R, request(G), D), T) \rightarrow \\ &\quad \mathbf{E}(tell(R, S, answer(G), D), T') : T' > T, T' \leq T + 2 \end{aligned}$$

This protocol could be learned by adopting a system that implements Abductive Concept Learning. Such a system is, for example, ICL [DRL95].

In this case, the background theory is empty (assuming that $>$, $<$, $\leq$, $\geq$ are built in the language). We may have, for example, the following positive interpretations (positive communication acts):

$$\begin{aligned} e_1^+ &= \{\mathbf{H}(tell(a, b, request(give(nail)), 1), 0), \\ &\quad\ \mathbf{H}(tell(b, a, answer(give(nail)), 1), 1)\} \\ e_2^+ &= \{\mathbf{H}(tell(a, b, request(send(info)), 5), 10), \\ &\quad\ \mathbf{H}(tell(b, a, answer(send(info)), 5), 12)\} \end{aligned}$$

and the following negative interpretations (negative communication acts):

$$\begin{aligned} e_1^- &= \{\mathbf{H}(tell(a,b,request(give(nail)),2),5), \\ &\quad \mathbf{H}(tell(b,a,answer(give(nail)),2),8)\} \\ e_2^- &= \{\mathbf{H}(tell(a,b,request(send(info)),7),15), \\ &\quad \mathbf{H}(tell(b,a,answer(give(nail)),7),16)\} \\ e_3^- &= \{\mathbf{H}(tell(a,b,request(give(nail)),9),20), \\ &\quad \mathbf{H}(tell(b,a,answer(give(nail)),9),19)\} \end{aligned}$$

From these examples, Abductive Concept Learning is able to learn the desired constraint

$$\begin{aligned} &\mathbf{H}(tell(S,R,request(G),D),T) \\ \rightarrow &\mathbf{H}(tell(R,S,answer(G),D),T') \end{aligned} \tag{33}$$

which is the previous one, apart from the presence of the **E** predicate in the head. Moreover, by adopting techniques similar to those described in [AF97] for learning built-in constraints, we are able to infer the relationship between T and T':

$$T' > T, T' \leq T + 2$$

At this point, in order to have a IC_S, we can turn the **H** literal in the head into a conditional **E** literal having as the condition the constraint shown above.

The application of ILP techniques for learning protocols involving both positive and negative expectations is subject for future work. In Section 8.3.1 we sketch the learning of the NetBill protocol.

7.4 Emerging behaviour

The concept of *emergence* has its roots in the field of complex systems and it roughly refers to "something new appearing at a different scale w.r.t. parts of the system under observation" [Cru94]. There is no universal agreement on the definition of emergence (often even contradictory definitions are given). In the following we report some observations collected in [Cor02]:

- Emergence arises when an observer recognizes a *pattern*.
- Emergence is like a dynamic attractor, or the product of a "deep structure" – a pre-existing potentiality.
- Emergence does not have logical properties; it cannot be deduced (predicted);
- Emergence represents rule-governed creativity based on finite sets of elements and rules of combination.

In the field of Multi-Agent Systems, the concept of emergence is usually referred to emerging behavior, society, conventions, etc. [WW95, LL02, ST97, Axe97, Dav00]. In this context, the term expresses the arising of coordinated behaviors, the autonomous formation of groups or societies and the reaching of agreements on norms regulating interactions among agents. All these phenomena involve the presence of stochasticity. Indeed, something can be generally

defined as emergent when it is not originated from a centralized control, nor the result of deterministic interaction patterns.

Another relevant issue related with emergence is the introduction of adaptive mechanisms in the multi-agent system. Adaptive mechanisms range from simple self-tuning of agent parameters to the application of evolutionary techniques.

One of the main aims of this project is to investigate the composition of (possibly) different reasoning mechanisms, which are embedded in computees. This does not necessarily involve stochastic nor adaptive components. However, we can consider as emerging a characteristic of the system which can be dynamically observed. This characteristic might be also very simple, such as the composition of sequences of interactions among computees. In the experimental setting of Workpackage 6, we will focus mainly on the experimental observation of simple characteristics, originated by interaction patterns among agents. In particular, the goals are the following:

- Observe whether a particular interaction pattern emerges;
- Observe which patterns emerge, given a defined initial setting;
- Try to prove the falsity of a given conjecture on a property, by means of a counter-example[9].

8 Examples

In this section, we present three concrete examples where we show the use of IC_S to express protocols. The first is a resource exchange scenario, first introduced in [STT02b] and then presented in the context of SOCS in [TMM+02, AGL+03b]; the second is a combinatorial auction scenario, and the third is a protocol for the selling and delivery of information goods [CTS95].

8.1 The resource reallocation problem

In this section we briefly recall the resource reallocation scenario studied by [STT02b, STT02a, STT03]. Let us consider a society of computees where individuals have *goals* to achieve, and in order to achieve them they identify plans. Plans are (partially) ordered sequences of actions. In order to execute the actions, computees may need some resources. An action that requires a resource r is said to be *unfeasible* if r is not available to the computee that intends to execute it. Similarly, a *plan* is unfeasible if it contains an action which is unfeasible, and so is the *intention* of a computee, containing such plan[10]. The resources that computees *need* in order to perform an action in a plan but that they do not possess are called *missing* resources. The *resource reallocation problem* is the problem of reducing a possibly non-empty set of missing resources of a computee to the empty set (*resource reallocation problem* of a computee), and, in a society, it is the problem of solving all the *resource reallocation problems of all the computees* (*resource reallocation problem of a society*).

In [STT02b, STT02a, STT03], the exchanges made to solve the resource reallocation problem are determined by means of negotiation dialogues.

[9] Here we refer to the concept of *falsifiable* scientific conjecture described by Popper [Pop68].

[10] In [STT02a] *plans* are modelled as part of the computee *intentions*.

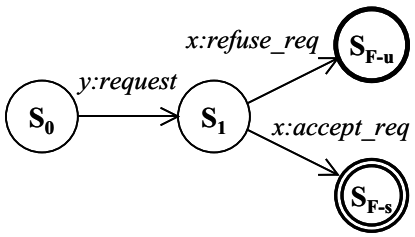

Figure 6: Request dialogue protocol

They do not model the physical counterpart that they potentially represent (e.g., they do not model the physical exchange of resources). In their formulation, what is meant by *resource* is in fact only an abstract entity, identified by its *name*, which possibly represents a physical resource such as a nail or a hammer. The actual delivery of physical resources is not modelled either. We follow a similar approach, but we also assume that communicative acts may come together with *commitments* that at some point computees must fulfill once the act is made. Based on the idea of commitments, we "institutionalize" the computee interactions, and we associate computee societies with institutions (see Section 2.1).

As in 5, we define the CCL for resource reallocation by defining the *communication* language, and the *content* language, and the format of communication acts is as described in Section 5.

8.1.1 Protocols in the resource exchange scenario

The only communication protocol allowed in this sample society is the *request dialogue protocol*, that we define by a state machine. A request dialogue is initiated by a computee, say x, that needs a resource r, if there exists a computee y to whom x still has not requested r.

The protocol is defined in Figure 6, as a finite state machine, consisting of states and arcs, which has as its states an initial state S_0, two final states, S_{F-s} (successful termination) and S_{F-u} (unsuccessful termination), and an intermediate state S_1.

The arcs can be viewed as allowed transitions mapping one state to another given a label. These labels correspond to the content of utterances. In the protocols, we use some abbreviations:

request for $request(give(R, (T_{start}, T_{end}))$,
refuse_req for $refuse(request(give(R, (T_{start}, T_{end}))))$,
and so forth.

We can express this protocol in terms of $\mathcal{IC}_S$, as it is shown in Figure 7.

The first two groups of IC_S express the expected behavior of computees following the request dialogue protocol, while the last two groups of IC_S consider an illicit behavior.

The advantages of using $\mathcal{IC}_S$ to express protocols have been already discussed in the general case. In the particular setting of resource reallocation, an obvious advantage of this approach is that it paves the way to prove formally some properties of the society, such as the class of resource reallocation problems that it can solve, dialogue termination and maximum length, and the conformance of computees to protocols. Some work in this respect has already been done by Sadri et al. [STT02a, STT02c].

conditions that allow to *accept* or *refuse*:

$$\begin{array}{l}\mathbf{H}(tell(X,Y,accept(give(R)),D),T) \rightarrow \\ \quad \mathbf{E}(tell(Y,X,request(give(R)),D),T'),T' < T \\ \mathbf{H}(tell(X,Y,refuse(give(R)),D),T) \rightarrow \\ \quad \mathbf{E}(tell(Y,X,request(give(R)),D),T'),T' < T\end{array}$$

expected moves from S_0 after a deadline of 2 time units:

$$\begin{array}{l}\mathbf{H}(tell(X,Y,request(give(R)),D),T) \rightarrow \\ \quad \mathbf{E}(tell(Y,X,accept(give(R)),D),T'),T' > T,T' \leq T+2 \\ \quad \vee\mathbf{E}(tell(Y,X,refuse(give(R)),D),T'),T' > T,T' \leq T+2\end{array}$$

S_{F-u} and S_{F-s} are final states:

$$\begin{array}{l}\mathbf{H}(tell(X,Y,accept(give(R)),D),T) \rightarrow \\ \quad \mathbf{NE}(tell(Y,X,Move,D),T'),T' > T \\ \mathbf{H}(tell(X,Y,accept(give(R)),D),T) \rightarrow \\ \quad \mathbf{NE}(tell(X,Y,Move,D),T'),T' > T \\ \mathbf{H}(tell(X,Y,refuse(give(R)),D),T) \rightarrow \\ \quad \mathbf{NE}(tell(Y,X,Move,D),T'),T' > T \\ \mathbf{H}(tell(X,Y,refuse(give(R)),D),T) \rightarrow \\ \quad \mathbf{NE}(tell(X,Y,Move,D),T'),T' > T\end{array}$$

S_0 is an initial state:

$$\begin{array}{l}\mathbf{H}(tell(X,Y,request(give(R)),D),T) \rightarrow \\ \quad \mathbf{NE}(tell(Y,X,Move,D),T'),T' < T \\ \mathbf{H}(tell(X,Y,request(give(R)),D),T) \rightarrow \\ \quad \mathbf{NE}(tell(X,Y,Move,D),T'),T' < T\end{array}$$

Figure 7: IC_S for the *request dialogue protocol*

8.2 Modelling a combinatorial auction

As the rise of the Internet and electronic commerce continues, dynamic automated markets will be an increasingly important domain for computees and software agents. Auctions are an important way of allocating items among autonomous and self-interested agents. The software agent technology seems an attractive paradigm to support auction applications, because the introduction of software agents acting on behalf of end-users could reduce the effort required to complete auction activities. Agents are intrinsically autonomous and can be easily personalized to embody end-user preferences. In addition, they are adaptive and capable of learning from both past experience and their environment, in order to cope with changing operating conditions and evolving user requirements [GMM98]. While, in the past, bidders were only humans, recently Internet auction servers have been built and software agents can participate in the auction on behalf of end-users [WWW98]. Electronic auctions have already demonstrated the potential of software agents [CMK96]. Some of these auction servers even have a built-in support for mobile agents [San00].

In this setting, each agent can be viewed as a computee with knowledge base and reasoning capabilities required to perform auction activities on behalf of the users. Various learning techniques have been also applied to auction scenarios and the (bilateral) call market scenario (see, for instance, [HW98, WWT99, CHH00]), e.g., in order to learn the strategy on which to base further bids. Participants can be humans or computees.

Items are not limited to goods, but can also represent resources and services, possibly associated to time windows. Traditionally, auctions are aimed at selling or buying a single item; the auctioneer tries to maximize his/her profit if selling an item, or minimize his/her costs if buying an item. Since bidders make bids on a single item, it is easy to choose the best bid, i.e., the one providing the highest revenue. This kind of auction follows the *sequential auction* mechanism. However, it is difficult to bid in these auctions when more than one item is needed since one bidder can have preferences on bunches of items. In this case, a bidder should make hypotheses on what the other bidders will bid.

To partially solve the problem, the *parallel auction* mechanism has been proposed. Bidders can bid on a set of items simultaneously, bids are visible to all the participants and should be performed in a limited time window. Again, it is easy to choose the best bid by simply choosing the best one. A problem in parallel auctions can arise: it can happen that no bidding would start since all bidders wait for other bids to perform the best offer.

Recently, a third kind of auction mechanism has been proposed, the so called *combinatorial auctions* [Nis00, San02, San96]. Bidders can bid on combinations of items, and associate a price for each combination. The auctioneer should solve the winner determination problem, i.e., it should choose the best bids that cover *all* items. Clearly, the problem becomes combinatorial and deciding the best bids is now difficult. Thus, for a long time combinatorial auctions have not been considered a viable alternative to single or parallel auctions. Recently, however, effective solving methods have been propose that make such auctions a viable alternative.

Depending on the kind of auction, the auctioneer either sells goods/services or buys goods/services. Bidders have the goal to obtain/sell their goods/services under convenient conditions as far as price is concerned. The auctioneer has the goal to sell/obtain a set of goods/services maximizing the profit (or minimizing the cost) at the minimum risk. In a combinatorial exchange, the goal of the administrator is to select bids so as to maximize the surplus.

We are here particularly interested in the communication protocol of a society representing a

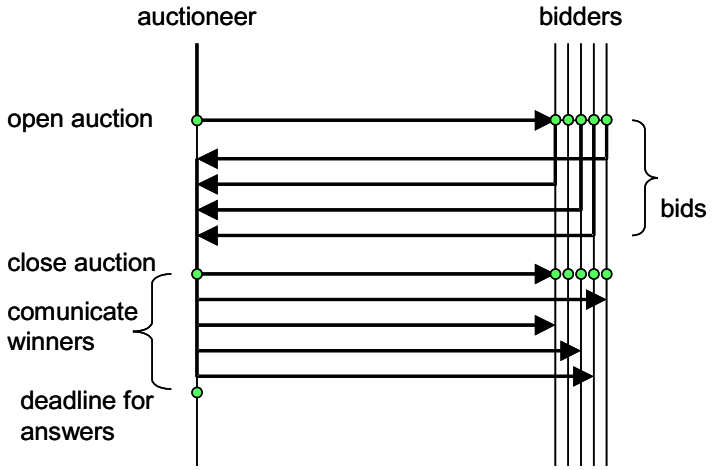

Figure 8: A protocol for Combinatorial Auctions

Combinatorial Auction. We have two roles: the *auctioneer* and the *bidder*. The auctioneer has the task of opening the auction by informing the bidders about the items it wants to sell/buy. The bidders may then start bidding (they are not obliged to). Each bid contains a set of items and a price. The bidding ends when the auction is closed by the auctioneer. The auctioneer then has to communicate within a deadline to each bidder if its bid wins or looses.

The protocol is depicted in Figure 8.

8.2.1 The $SOKB$

The goal of the society might be, for instance, selling a given item. In order to sell an item, the society might expect some auctioneer to open an auction containing the given item. Stated otherwise, the society expects one of the computees to take the role of the auctioneer in order to reach the goal of the society. The goal of the society could be

$$sold(nail)$$

and the society might have, in the $SOKB$, a rule

$$\begin{array}{l} sold(Item) \leftarrow \\ \mathbf{E}(tell(Auct, Bidders, openauction(Items, T_{end}, T_{deadline}, A_{number})), T_{open}) \\ \wedge \mathbf{E}(tell(Bidder, Auct, bid(ItemList', P), A_{number}), T_{bid}) \\ \wedge Item \in Items \wedge Item \in Items' \end{array}$$

saying that an item is sold if some computee is expected to open an auction where the item is for sale and somebody offers to buy it. The parameters of the *openauction* represent

- the whole set of items in the auction, *Items*,
- the closing time of the auction, T_{end}, and

- the deadline within which the auctioneer should answer, $T_{deadline}$.

Instead, each bid has a parameter $ItemList'$ that represents the bid items and should be a subset of the total list of *Items*. The second parameter of the bid is the *price* P.

The details about the exchanged messages and IC_S are explained in the following section.

8.2.2 Rules of interaction (protocols)

In a combinatorial auction, the protocol can be tested by the society (which is the auction in this case) or by a computee delegated by the society to check the protocols.

The first rule is the following: each time a bidding event happens, the auctioneer should have sent before an *openauction* event (to all bidders).

IC auc- 1.

$$\begin{array}{l}\mathbf{H}(current\_time, T_c),\\ \neg\mathbf{H}(tell(R, Bidders, openauction(Items, T_{end}, T_{deadline}), A_{number}), T_{open}),\\ T_{open} \leq T_c\\ \rightarrow \mathbf{NE}(tell(S, R, bid(ItemList, P), A_{number}), T_{bid}), T_{bid} < T_c\end{array}$$

This IC_S imposes that a bidder cannot start placing bids if an auctioneer has not opened an auction.

Placing incorrect bids (a bid for items not for sale, or after time T_{end}) will be forbidden by the following rule:[11]

IC auc- 2.

$$\begin{array}{l}\mathbf{H}(tell(R, Bidders, openauction(Items, T_{end}, T_{deadline}), A_{number}), T_1)\\ \rightarrow \mathbf{NE}(tell(S, R, bid(ItemList, P), A_{number}), \_), ItemList \not\subseteq Items\\ \wedge\mathbf{NE}(tell(S, R, bid(\_, P), A_{number}), T_2), T_2 > T_{end}\\ \wedge\mathbf{NE}(tell(S_{wrong}, R, bid(\_, P), A_{number}), \_), S_{wrong} \notin Bidders\end{array}$$

The auctioneer should answer to each bid if it wins or looses. The answer should be sent after the auction is closed within the deadline $T_{deadline}$.

IC auc- 3.

$$\begin{array}{l}\mathbf{H}(tell(S, R, bid(ItemList, P), A_{number}), T_{bid})\\ \wedge\mathbf{H}(tell(R, Bidders, openauction(Items, T_{end}, T_{deadline}), A_{number}), T_{open}) \rightarrow\\ \quad \mathbf{E}(tell(R, S, answer(win, S, ItemList, P), A_{number}), T_{answer}),\\ \quad\quad T_{answer} > T_{end} \wedge T_{answer} < T_{deadline}\\ \quad \vee \mathbf{E}(tell(R, S, answer(loose, S, ItemList, P), A_{number}), T'),\\ \quad\quad T' > T_{end} \wedge T' < T_{deadline}\end{array}$$

Note that this IC_S is fired each time a bid is generated and checks that the event *answer* will hopefully happen.

Another law is the following: it is not possible that a bidder receives for the same auction on the same bid two conflicting answers. Therefore, each bidder either wins or looses, but not both:

[11] For the sake of brevity, we use the symbol *not* $\subseteq$ and $\notin$ instead of using the corresponding predicates.

IC auc- 4.

$$\mathbf{H}(tell(R, S, answer(loose, S, ItemList, P), A_{number}), \_) \rightarrow \\ \mathbf{NE}(tell(R, S, answer(win, S, ItemList, P), A_{number}), \_)$$

IC auc- 5.

$$\mathbf{H}(tell(R, S, answer(win, S, ItemList, P), A_{number}), \_) \rightarrow \\ \mathbf{NE}(tell(R, S, answer(loose, S, ItemList, P), A_{number}), \_)$$

In addition, the society should ensure that each item should be covered by only one winning bid, i.e., two winning bids of two different bidders cannot contain the same item:

IC auc- 6.

$$\mathbf{H}(tell(R, Bidder_1, answer(win, Bidder_1, ItemList, P), A_{number}), \_) \rightarrow \\ \mathbf{NE}(tell(R, Bidder_2, answer(win, Bidder_2, ItemList', P'), A_{number}), \_), \\ Bidder_1 \neq Bidder_2 \wedge ItemList \cap Itemlist' \neq \emptyset$$

Finally, there is a law that says that it is not possible that the whole set of winning bids does not *cover* the whole set of items in the auction. This rule should be enforced if free disposal is not allowed. Free disposal is the assumption that the auctioneer can obtain/give away one item for free. Since we check elsewhere that the auctioneer has answered to all bidders, the society simply has to collect the whole set of winning bids and check that it covers the whole set of items. This check should be performed after the auction is closed and after the whole set of answers has been computed:

IC auc- 7.

$$\mathbf{H}(tell(R, Bidders, closeauction, A_{number}), T_{end}) \wedge \mathbf{H}(current\_time, T_{current}), \\ \mathbf{H}(tell(R, Bidders, openauction(Items, T_{end}, T_{deadline}), A_{number}), \_) \\ \wedge T_{current} > T_{end}, auctioneer(R) \wedge I \in Items \\ \rightarrow \mathbf{E}(tell(R, S, answer(win, S, ItemList', P), A_{number}), T_{answer}), I \in ItemList' \\ \vee \mathbf{NE}(tell(R, S, answer(win, S, ItemList', P), A_{number}), T_{answer})$$

Notice that I is universally quantified, thus, the IC_S is checked for each $I \in Items$.

If free disposal is not allowed, all the bids are declared as losing. We can simply declare, as an alternative, that no bid can win, thus, the interaction with rule IC auc-3 will ensure that all the bids are losing.

8.3 Modelling the NetBill protocol

In recent years, electronic commerce has gained more and more popularity among both sellers and buyers, because of the ease and fastness with which it allows business transactions. However, there is still little confidence in the Internet as a reliable marketplace, especially for questions regarding security and privacy. Electronic commerce protocols can serve as a remedy to the lack of standard security and privacy mechanisms of the Internet.

NetBill (see [CTS95]) is a security and transaction protocol optimized for the selling and delivery of low-priced information goods, like software or journal articles. The protocol rules transactions between two agents: *merchant* and *customer*.

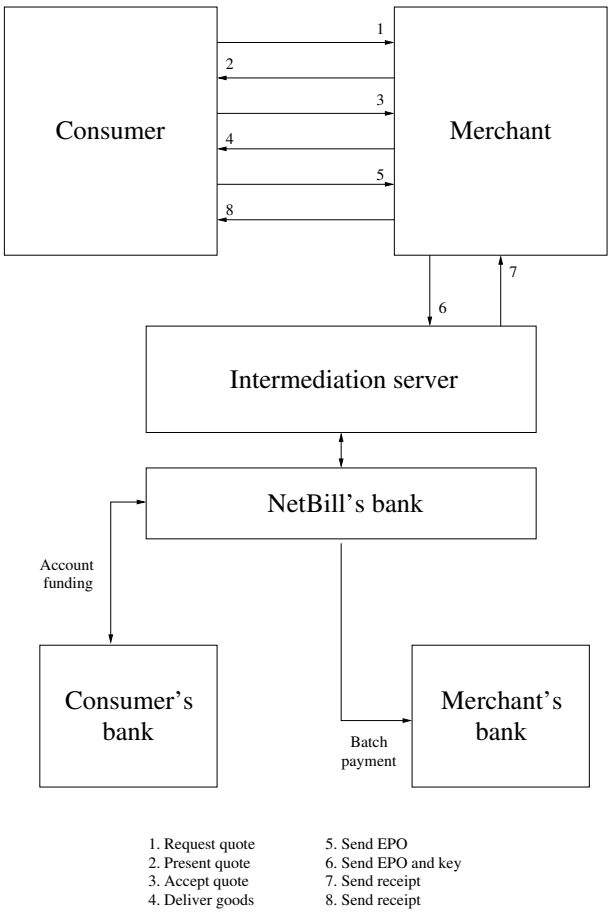

Figure 9: The NetBill protocol

Accounts for merchants and customers, linked to traditional financial accounts (like credit cards), are maintained by a NetBill server.

The protocol prescribes a price negotiation phase, in which the customer presents evidence of his/her identity, and requests the quote for a good, and the merchant answers with the price. Then, the customer accepts or declines the offer; acceptance constitutes an order for delivery of the goods. The merchant then delivers the goods under encryption (remind they are information goods), withholding the key. To fulfill payment, the customer constructs and digitally signs an Electronic Payment Order (EPO) and sends it to the merchant, who will append the encryption key to it, digitally sign it and forward it to the NetBill server. The NetBill server takes care of the actual money transfer and returns a digitally signed receipt, including the key, to the merchant. The merchant forwards receipt and key to the customer, who will so be able to decrypt the good.

In the following, we will suppose the agents in a NetBill-ruled transaction are computees. We study how IC_S can be used to express (a simplified version of) the protocol as described in [YS02]. The protocol is depicted in Figure 9.

Two computees are involved in the protocol: *customer* (*consumer*) and *merchant*.

We suppose the following performatives, with the associated intuitive meaning described in the following, are available:

- *request*$(C, M, good(G, Q), D)$: consumer C requests to merchant M the quote Q for a good G (in a dialogue D);
- *present*$(M, C, good(G, Q), D)$: merchant M tells consumer C that its quote for good G is Q;
- *accept*$(C, M, good(G, Q), D)$: consumer C tells merchant M that it is willing to pay Q for good G;
- *deliver*$(M, C, good(G, Q), D)$: merchant M delivers good G to consumer C, expecting that a quote Q will be paid for it;
- *epo*$(C, M, good(G, Q), D)$: consumer C sends an Electronic Payment Order for an amount of Q to merchant M, in payment for good G[12];
- *receipt*$(M, C, good(G, Q))$: merchant M sends a receipt of payment for good G for an amount of Q to consumer C.

By using IC_S, we can specify what sequences of communicative and physical actions are acceptable, in the sense that they do not violate the protocol[13]. The protocol does not deal with deadlines, so neither will we: no constraints are imposed on time variables in expectations.

We will follow the general principle that an event brought about by a computee should not, by itself, commit another computee to anything[14]. This can be achieved in (at least) two ways.

Expectations from bilateral actions - One possibility is to generate an expectation only as consequence of two or more communicative or physical actions (at least one of one computee and one of the other), or as consequence of an action which fulfills an expectation. In this way, we prevent expectations for a computee from being raised by unilateral actions of another computee.

A possible specification of the protocol by means of IC_S follows (IC_S (34)–(36)):

$$\begin{aligned} &\mathbf{H}(present(M, C, good(G, Q), D), T_p) \\ \wedge&\mathbf{H}(accept(C, M, good(G, Q), D), T_a), \\ &T_p < T_a \\ \rightarrow&\mathbf{E}(deliver(M, C, good(G, Q), D), T_d) \end{aligned} \tag{34}$$

[12]Here we consider an interaction between two computees only; therefore, we do not consider interaction between the merchant and the intermediation server (steps 6 and 7 in Figure 9) and assume that an *epo* performative implies that the corresponding payment is always fulfilled correctly. We will also assume that certification of identity and digital signs are handled by the underlying infrastructure

[13]We will not consider IC_S linking *request* and *present* acts, because we think negotiation should not be constrained: however, if so desired, this can be easily added.

[14]Obviously, this may not be the case if we consider a hierarchical structure of roles in the society; but for an electronic commerce setting, it would not make much sense if the merchant or the consumer had some kind of power over the other.

An expectation for merchant M to *deliver* the good is generated by IC_S (34) only if *both* a *present* and an *accept* event have occurred. If consumer C *accept*s a quote which merchant M has not *present*ed, no expectation is raised.

$$\begin{aligned}&\mathbf{H}(\mathit{deliver}(M, C, \mathit{good}(G, Q), D), T_d)\\ \wedge&\mathbf{E}(\mathit{deliver}(M, C, \mathit{good}(G, Q), D), T_d)\\ \rightarrow&\mathbf{E}(\mathit{epo}(C, M, \mathit{good}(G, Q), D), T_e)\end{aligned} \tag{35}$$

IC_S (35) is effective (i.e. generates an *epo* expectation) only if the *deliver* event corresponds to an existent *deliver* expectation. Since a *deliver* expectation can only be raised for IC_S (34), this implies that corresponding *present* and *accept* events have occurred before; in other words, this prevents consumer C from being obliged to pay for unrequested goods that have been delivered to it. The same mechanism is used in IC_S (36).

$$\begin{aligned}&\mathbf{H}(\mathit{epo}(C, M, \mathit{good}(G, Q), D), T_e)\\ \wedge&\mathbf{E}(\mathit{epo}(C, M, \mathit{good}(G, Q), D), T_e)\\ \rightarrow&\mathbf{E}(\mathit{receipt}(M, C, \mathit{good}(G, Q), D), T_r)\end{aligned} \tag{36}$$

Backward expectations - The second way is to use *backward* expectations, i.e. expectations concerning past events (see Section 5.3.2); in this way, it is possible to ensure the desired sequentiality of actions.

$$\begin{aligned}&\mathbf{H}(\mathit{accept}(C, M, \mathit{good}(G, Q), D), T_2),\\ \rightarrow&\mathbf{E}(\mathit{present}(M, C, \mathit{good}(G, Q), D), T_1), T_1 < T_2\end{aligned} \tag{37}$$

$$\begin{aligned}&\mathbf{H}(\mathit{accept}(C, M, \mathit{good}(G, Q), D), T_1)\\ \rightarrow&\mathbf{E}(\mathit{deliver}(M, C, \mathit{good}(G, Q), D), T_2).\end{aligned} \tag{38}$$

An *accept* event generates two expectations: one backward, to ensure it follows a corresponding *present* event, and one ordinary, which commits merchant M to *deliver*. It is worth noticing that, in case of an *accept* without a previous *present*, although the *accept* event is a violation, the *deliver* expectation would be generated anyway, if we do not prevent it by a recovery procedure (see Section 7.1).

The same applies to the pairs of IC_S(39)-(40) and (41)-(42); instead, for a *receipt* event, it is only needed to verify that it follows a corresponding *epo* event (IC_S (43)).

$$\begin{aligned}&\mathbf{H}(\mathit{deliver}(M, C, \mathit{good}(G, Q), D), T_2),\\ \rightarrow&\mathbf{E}(\mathit{accept}(C, M, \mathit{good}(G, Q), D), T_1), T_1 < T_2\end{aligned} \tag{39}$$

$$\begin{aligned}&\mathbf{H}(\mathit{deliver}(M, C, \mathit{good}(G, Q), D), T_1)\\ \rightarrow&\mathbf{E}(\mathit{epo}(C, M, \mathit{good}(G, Q), D), T_2)\end{aligned} \tag{40}$$

$$\begin{aligned}&\mathbf{H}(\mathit{epo}(C, M, \mathit{good}(G, Q), D), T_2),\\ \rightarrow&\mathbf{E}(\mathit{deliver}(M, C, \mathit{good}(G, Q), D), T_1), T_1 < T_2\end{aligned} \tag{41}$$

$$\begin{aligned}&\mathbf{H}(epo(M,C,good(G,Q),D),T_1)\\ \rightarrow&\mathbf{E}(receipt(M,C,good(G,Q),D),T_2)\end{aligned} \tag{42}$$

$$\begin{aligned}&\mathbf{H}(receipt(M,C,good(G,Q),D),T_2),\\ \rightarrow&\mathbf{E}(epo(C,M,good(G,Q),D),T_1),T_1<T_2\end{aligned} \tag{43}$$

Note that, in both cases, the protocol is fulfilled when no more expectations exist (which holds true once the last *receipt* act has been properly performed).

Both protocol specifications are fulfilled by a complete sequence of actions as the one described in [YS02]; however, the second is stricter in that it generates a violation if a non-initial action is performed and its predecessor has not been (for instance, a *deliver* without a previous *accept*), whereas in the first case this is allowed, but is not effective in generating expectations. Obviously, which of the two possibilities is better depends on the designer's choice; however, there are many more possibilities to make the protocol specification stricter or looser.

8.3.1 Learning the NetBill protocol

(For background material on machine learning, see Appendix C).

Let us see how the constraints describing the NetBill protocol can be learned. Let us consider first the constraints belonging to the case of expectations from bilateral actions and, in particular, IC_S (34) that we reproduce here for the ease of reading:

$$\begin{aligned}&\mathbf{H}(present(M,C,good(G,Q),D),T_1)\\ &\wedge\mathbf{H}(accept(C,M,good(G,Q),D),T_2)\wedge T_1<T_2\\ \rightarrow&\mathbf{E}(deliver(M,C,good(G,Q),D),T_3)\end{aligned} \tag{44}$$

Suppose we have the following set of positive interpretations (positive communication acts, as judged by an oracle):

$e_1^+ = \{\mathbf{H}(present(a,b,good(nail,10),1),2),$
$\mathbf{H}(accept(b,a,good(nail,10),1),5),$
$\mathbf{H}(deliver(a,b,good(nail,10),1),7),\ldots\}$
$e_2^+ = \{\mathbf{H}(present(c,d,good(hammer,20),2),10),$
$\mathbf{H}(accept(d,c,good(hammer,20),2),13),$
$\mathbf{H}(deliver(c,d,good(hammer,20),2),15),\ldots\}$

and the following set of negative interpretations (negative communication acts):

$e_1^- = \{\mathbf{H}(present(a,b,good(hammer,10),3),17),$
$\mathbf{H}(accept(b,a,good(hammer,10),3),20),$
$\mathbf{H}(deliver(a,b,good(nail,10),1),23),\ldots\}$
$e_2^- = \{\mathbf{H}(present(a,b,good(nail,10),4),25),$
$\mathbf{H}(accept(b,a,good(nail,10),4),26),$
$\mathbf{H}(deliver(a,b,good(nail,20),4),28),\ldots\}$
$e_3^- = \{\mathbf{H}(present(a,b,good(nail,10),5),30),$
$\mathbf{H}(accept(b,a,good(nail,10),5),32),\ldots\}$
$e_4^- = \{\mathbf{H}(present(c,d,good(hammer,10),6),40),$
$\mathbf{H}(accept(d,c,good(hammer,10),6),37),$
$\mathbf{H}(deliver(c,dgood(hammer,10),6),42),\ldots\}$

In order to learn back IC_S (34) we can exploit ICL [DRL95]. The background theory is empty and $>$, $<$, $\leq$, $\geq$ are built in the language.

From the previous sets of positive and negative interpretations, ICL learns the following IC_S:

$$\begin{aligned} &\mathbf{H}(\mathit{present}(M, C, \mathit{good}(G, Q), D), T_1) \\ &\wedge \mathbf{H}(\mathit{accept}(C, M, \mathit{good}(G, Q), D), T_2) \\ \rightarrow &\mathbf{H}(\mathit{deliver}(M, C, \mathit{good}(G, Q), D), T_3) \end{aligned} \tag{45}$$

Moreover, by adopting the system described in [AF97] the following relation among the time variables can be learned

$$T_1 < T_2$$

that, since it involves only variables in the body, can be added to the body of the IC_S. Finally, in order to have a IC_S the **H** literal in the head is turned into an **E** literal.

The other IC_S of the "expectation from bilateral actions" encoding of the NetBill protocol can be learned in a similar way. However, note that for learning IC_S (35)–(36), we need not only a record of the interactions between computees but also a record of the various expectations that have raised inside the individual computees.

9 Related work

The main contribution of this deliverable is in providing a declarative representation of the social knowledge, and in a *logic formalism* based on social expectations and $\mathcal{IC}_S$, for *specifying* social rules and easily verifiable protocols, and for defining the semantics of communicative acts in an open system scenario. Our work relates to several aspects of Multi-Agent Systems research.

In previous sections, we already gave some background on societies (Section 2.1), protocols (Section 3.1), and communication languages (Section 5.1) for multi-agent systems. In this section we discuss the contribution of our work in relationship with the state of the art in these areas. For the sake of conciseness, we will keep this section partial and will discuss only the most important relationships with our work.

We consider societies that are open in the sense that computees are free to enter or leave the society. However, even for open societies, we need to verify that the identity that the computee communicates is indeed its real identity. To this purpose, we could use techniques developed in the project SECURE [SEC01], such as APER (A Peer Entity Recognition scheme). In such a recognition scheme, a small initial measure of trust is given to any entity so that it can enter the society and begin collaborating with other entities. Through continued collaboration, trust between entities increases and evolves. Some hints about how to estimate trustworthiness based on our social framework have been given in Section 7.2.

Several researchers have studied the concepts of norms, commitments and social relations in the context of Multi-Agent Systems [CFS99]. Furthemore, a lot of research has been devoted in proposing architectures for developing agents with social awareness (see, for instance [CDJT99]). Our approach, can be conceived as complementary to these efforts, since instead of proposing a specific architecture for designing computees, our work in mainly focused on the definition of a society infrastructure based on Computational Logic for regulating and improving robustness

of interaction in an open environment, where the internal architecture of the computees might be unknown.

Our work is very close, as far as the objectives and methodology, to the work on computational societies presented and developed in the context of the ALFEBIITE project [ALF99], and the work by Singh [YS02] where a social semantics is exemplified by using a commitment-based approach. With these works we share the same view of an open society as that of [APS02], which we introduced in Section 1.1. In turn, our work is especially oriented to computational aspects, and it was developed with the purpose of providing a computational framework that can be directly used for automatic verification of properties such as compliance (see Section 6.1.2).

Most of the formal approaches to model protocols that we briefly reviewed in Section 3.1 specify protocols as legal sequences of actions. In this way, protocols can be over-constrained, and this affects autonomy, heterogeneity, and ability to exploit opportunities and exceptions [YS02]. Moreover, the mentalistic approach to protocol definition has been much criticized mainly because its assumptions regarding agents' internals are not realistic in open societies of heterogeneous agents, as, e.g., stated by [Sin98]: "Whenever agents' mental states are not accessible, which is reasonably the case if agents operate in open and heterogeneous environments, it is impossible to verify semantic compliance of communicative acts." Therefore we advocated a *social* approach, where the semantics of interactions is defined in terms of the effects of the computees interactions on the society. Following this approach, even if the computees mental state cannot be accessed, it is possible to verify whether communicating computees in a society comply to some social laws and norms which regulate the interactions.

As regards the approaches based on a process calculus, namely developed in the projects DEGAS [DEG01] and MYTHS [MYT01], both of them employ the protocol specification only for proving property of the system and not, as we also do, for checking the conformance of agents to the protocol.

Another expressive advantage of our framework is that it can express with the same formalism both protocols and social semantics of communicative acts.

A piece of work that inspired ours, as concerning the semantics of communication language, is that by Fornara and Colombetti [FC02] (see also Section 5.1). The approach there presented is similar to ours in that it is *social-based*: it makes no assumptions on the nature of agents, specifies semantics of actions as their social effects and presupposes a social framework (which is called *institution* in [CFV02]) for assigning agents with roles, verifying their social behavior and, possibly, recovering from violation conditions. There are, however, some significant differences with [CFV02], mainly originating from the different paradigm we have chosen to express semantics (*logic-based* instead of *object-oriented*). The main difference is that a *commitment* can have several states in [CFV02]. Although an *expectation* has no state, we can map to our framework the states of *empty* (no expectation), *active* (an expectation which has been generated), *fulfilled* (an expectation which has been fulfilled) and *violated* (an expectation which has been violated); since we cannot map the states of *pre-commitment*, *canceled* and *conditional*, our approach might, at a first look, seem less expressive. However, if we think of the social infrastructure/institution as a black box with computees/agents' actions as inputs and fulfillment or violation of *expectations/commitments* as outputs, it is possible to obtain an equivalent behavior with our approach to one specified by the approach of [FC02]. We give a thorough comparison of our work and that of [FC02] in [ACG$^+$03a].

Furthermore, our notion of *expectation* is more general than that of *commitment* in [FC02]: it represents the necessity of a (past or future) event, and is not bound to have a debtor or a creditor, or to be brought about by a computee.

A different, and interesting, way of linking social semantics of communicative acts and protocol specification is presented in [YS02]. However, in that work it is the single agent which, by exploiting its reasoning/planning capabilities, must find a communication path leaving no pending commitments (the alternative, to be applied when agents lack reasoning capabilities, is to compile a protocol specification to a Finite State Machine). Our approach ensures protocol compliance regardless of computees' reasoning capabilities, since it lets us express explicitly constraints between communicative acts, if so desired; however, equipping the communication model of single computees with sufficient knowledge to reason about social expectations is certainly an interesting option. This topic is discussed in D4 [KSST03].

In [APS02], Artikis et al. present a theoretical framework for providing executable specifications of particular kinds of multi-agent systems, called open computational societies, and present a formal framework for specifying, animating and ultimately reasoning about and verifying the properties of systems where the behaviour of the members and their interactions cannot be predicted in advance. Three key components of computational systems are specified, namely the social constraints, social roles and social states. The specifications of these concepts is based on and motivated by the formal study of legal and social systems (a goal of the ALFEBIITE project), and therefore operators of Deontic Logic are used for expressing legal social behaviour of agents [Wri51, vdT03]. ALFEBIITE has investigated the application of formal models of norm-governed activity to the definition, management and regulation of interactions between info-habitants in the information society. Their logical framework comprises a set of building blocks (including doxastic, deontic and praxeologic notions) as well as composite notions (including deontic right, power, trust, role and signalling acts).

Deontic Logic enables to address the issue of explicitly and formally defining norms and dealing with their possible violations. It represents norms, obligations, prohibitions and permissions. Deontic Logic enables to deal with predicates like "p ought to be done", "p is forbidden to be done", "p is permitted to be done". In our context, we only have expectations that can be respected by computees or violated, depending on their internal structure and behaviour. Taking this view, the **E** predicate is a suggestion for what should be done. As discussed also in [Sin98] at social level we do not have to constrain the autonomy of individuals, therefore we decide of expressing just expectations: $\mathbf{E}(h)$ expresses the expectation that h happens, while $\mathbf{NE}(h)$ expresses the expectation that h does not happen.

Our operator **E** could be compared with the $\mathcal{O}$ operator of Deontic Logic. But, while in a deontic context, $\mathbf{E}(h)$ ($\mathbf{NE}(h)$), could be interpreted as h is obliged (forbidden), in our context they are interpreted as it is expected that h happens (does not happen). For these reasons we do not rely on Deontic Logic for the semantics of this operator, but rather on abduction.

We believe that with our proposed formalism based on $\mathcal{IC}_S$ constraints we can capture (in a computational setting) the concept of (conditional) obligation with deadline presented in [DMDW02]. We map time explicitly. They write: `Oa(r<d|p)` to state that if the precondition `p` becomes valid, the Obligation becomes active. The obligation expresses the fact that `a` is expected to bring about the truth of `r` before a certain condition `d` holds.

For instance, if we have:

$$p = \mathbf{H}(tell(S, a, request(G), D, T))$$
$$r = \mathbf{H}(tell(a, S, answer(G), D, T')), T' > T$$
$$d = T' > T + 2$$

we can map `Oa(r<d|p)` into a IC_S:

$\mathbf{H}(tell(S, a, request(G), D), T) \rightarrow$
$\mathbf{E}(tell(a, S, answer(G), D), T'), T' > T, T' \leq T + 2.$

A notable difference with [APS02] is that we do not explicitly represent the institutional power of the members and the concept of valid action. Permitted are all social events that do not determine a violation, i.e., all events that are not explicitly forbidden are allowed, and this implements a sort of "open world assumption" at a society level. Differently, permission, when it needs to be explicitly expressed, is mapped the negation of a negative expectation: $\neg\mathbf{NE}(\ldots)$.

With respect to social constraints and roles, in [APS02] the framework is based on (Logical) Constraints, specified by using a temporal formalism based on the Event Calculus [KS86]. We, instead, represent social constraints by Integrity Constraints on happened events and expectations. We do not map our framework on the Event Calculus but, instead, on an abductive framework, arguing that in an open society, knowledge on computees behaviour is incomplete, and we need to do a sort of guess and expectation on the social behaviour of the computees. Therefore the semantics of our system can be directly mapped onto an abductive framework, where expectations can be confirmed (fulfilled) or disconfirmed (violated) by the history of the happened social events. Moreover, we deal with time by using suitable constraints on finite domains (see section 4.2 for a more elaborated discussion), while they use a temporal formalism based on the Event Calculus.

We are aware that the temporal framework we use is less expressive than that generally used in these cases [YS02, APS02], but we think that it is powerful enough for our goals and is a good trade off between expressiveness and efficiency.

Moreover, differently from [APS02], also in defining CCL semantics, we have followed a *social* approach which, differently from the *mentalistic*, seems to fulfill the requirements of communication in *open* societies of *heterogeneous* computees. Therefore, we link social semantics of communicative acts and protocol specification in a uniform way.

Another notable related work is IMPACT [AOR$^+$99, ESP99]. In IMPACT, agent programs may be used to specify what an agent is obliged to do, what an agent may do, and what an agent cannot do on the basis of deontic operators of Permission, Obligation and Prohibition, whose semantics doe not rely on a Deontic Logic semantics. In this respect, IMPACT and our work have similarities even if their purpose and expressivity are different. The main difference is that the goal of agent programs in IMPACT is to express and determine by its application the behavior of a single agent. In particular, "whenever the agent's state changes, the agent **must** take actions in accordance with some clearly specified operating principle so as to ensure that the resulting state satisfies some integrity constraints" [ESP99]. Our goal is to express rules of interaction, that instead cannot really determine and constrain the behavior of the single computees participating to the interaction protocols, since computees are autonomous. Constraints in IMPACT are to be satisfied, and their violation has the principal meaning of a corrupted state. In this case, agents must be able to recover from being corrupted to uncorrupted [EMS02]. In our setting, violation does not necessarily imply a "corrupted" state of the society, but just a "violation" of a protocol by a particular computee. Also with different purposes, we share with IMPACT the same effort of giving semantics in terms of (refined) restrictions of the possible sets of behaviors. As in [ESP99], we do not rely on Deontic Logic, but on abduction and well known semantics for Extended Logic Programming, with many advantages from both the declarative and operational points of view.

Our work is not only directly related to social aspects of MAS, but also to extensions of Logic Programming for MAS. In particular, the syntax of IC_S is strictly related to that of

integrity constraints in the IFF proof-procedure [FK97]. In Appendix B we compare our work done about IC_S with some syntactic aspects of the integrity constraints handled by the IFF proof-procedure.

10 Discussion and evaluations of objectives

In this work, we defined a Computational Logic-based framework for modelling societies of computees and their interactions. This work reports on both published and original work done within SOCS Workpackage 2 (WP2): *"Modelling interactions between computees"*. In Section 1.4 we listed the workshops and conferences where parts of this deliverable have been submitted and presented, with the aim of both disseminating our results and get feedback from peer review. The objectives of WP2 are specified in the Technical Annex included in the project documentation [SOC]. In particular, in the Technical Annex, page 6, it reads:

> "**Objective O2** will deliver a formal logic-based framework to characterize the interactions between computees in a rule-based manner, either by relying on protocols shared and agreed upon by all computees in a given society, or by interaction patterns that are specific to individual computees and possibly different for different computees."

In deliverable D3 [LMM+03], we set down the evaluation criteria that we would follow to evaluate our work. In particular, with respect to WP2, we stated the following:

> "in order to achieve this vision, the model should satisfy the following specific objectives:
>
> - societies should be able to function in the presence of incomplete information, due to the open and dynamically evolving environment;
> - societies should be able to handle changes in a dynamic environment;
> - societies should be able to adapt their behaviour, as they assimilate new knowledge about the social events;
> - societies should include high-level communication mechanisms.
>
> In addition, we have set down two major objectives that relate more to the nature of the model rather than to the resulting behaviour of the society that the model achieves:
>
> - the model should lend itself easily to a computational realisation (during Phase 2);
> - the model should be precise and amenable to formal verification of properties pertaining to the interactions of the computees belonging to a particular societies (during Phase 3)."

For this purpose, evaluation criteria have been classified into two groups, the first one related to the GC base requirements (adaptability, openness, heterogeneity, amenability to dealing with partial information), the second one related to properties of the model (modularity, computational viability, uniformity, formality, and links with related work).

The model of societies of computees that we propose follows a Computational Logic-based approach. In this model, Logic Programming, suitably extended with the concept of IC_S and expectations (interpreted as abducibles), acts as a uniform language for protocols, interaction policies and patterns.

A degree of openness, understood as the freedom of its members to join or leave it, is given by the model of society, presented in Section 2, which caters for new members joining a society and existing members leaving it (see Section 3.5). Moreover, the entry/exit rules can be expressed in terms of IC_S, which makes it possible to implement *semi-open* societies in the sense introduced by [Dav01]. The two "extremes" of semi-open societies are: *open* societies, i.e., those that pose no constraints at all on the entry rules, and *closed* societies, i.e., those that prevent members from joining in or leaving them.

Another degree of openness, understood as the possibility to have a society of heterogeneous computees, is achieved by the fact that the model of the society, including the handling of expectations, the protocol conformance checking and the generation of violations, are only based on the socially observable behaviour of computees: no assumption is made on the internals of computees, but their social behaviour is constrained by the semantics of social actions and protocols. Non-conforming behaviour of computees is still possible, but it will be detected by the society infrastructure and it will have social consequences. Violation handling and recovery is a matter of current and future work (see Section 7.1). This model of society caters for reasoning under incomplete information, in the sense that events that did not happen or that have not been "detected" are treated as unattended expectations, and it is possible to reason over both expectations and happened events.

The formalism for expressing society rules and protocols, together with the semantics of the individual communication utterances, is based on Abductive Logic Programming and constraints over abducible predicates, and its declarative semantics has been given in terms of logical entailment. Therefore, appropriate abductive proof-procedures can provide the operational support for the underlying infrastructure. In Section 4.3 we describe a possible way to "operationalize" the declarative semantics of societies and protocols, but this will be indeed matter of study during the second year of the project, in Workpackage 3, *"Computational models for (societies of) computees"*.

Time is explicit in the model. The "reasoning" at a social level is made over time, and it takes into account issues such as dealing with deadlines, that are important also from a practical viewpoint (see Section 4.2). In this way, we propose a social framework which is suitable for modelling a dynamic setting and able to handle changes in a dynamic environment.

We believe that one of the strong points of our approach is to be found in its formality, not only at a syntactic level (definition of what is the format of the society knowledge, IC_S, protocols, CCL format and constraints), but also, and more importantly, at the semantic level, which allows for describing what are the desirable evolutions of a society and link these formally to the social structure and social behaviour of the computees. This link is facilitated by the uniform way in which the society knowledge is expressed at the different levels (protocols, languages) and by its semantic characterization. Thanks to the society infrastructure, interactions among computees in a society are not only message exchanges, but they are high-level communication mechanisms that come together with a semantics that is at the same level as that of interaction protocols.

Adaptability is another evaluation criteria for the project. Besides a form of adaptability that can be achieved by individual computees, there is another kind of adaptability that we aim to achieve at a social level: the society should adapt itself to a changing environment as

new social events are recorded in the event history. A first degree of adaptability is achieved by the society being able to react to changes in the environment. More sophisticated degrees of adaptability, such as the ability to recover from states of violations, learning how computees interact, and single out "emerging" patterns of interaction, have been investigated throughout the project and some preliminary results have been presented in Section 7.3.

With respect to modularity, in our approach extension and merging of protocols can be accommodated by a suitable composition of constraints representing individual or partial interaction patterns.

As far as computational viability, we are aware that a very expressive formalism such as IC_S can result in computationally costly proof-procedures. For this reason, the evaluation of the SOCS project will be also based on the outcomes of the prototype demonstration and society animation that we aim to achieve at the end of Workpackage 4: *"Prototype demonstrator"*.

We think that the framework here presented is rich enough to accommodate varieties of interactive behaviors of computees. In particular, to evaluate the expressiveness of the designed interaction models, we have considered as case studies a dialogue-based interaction, with a special focus on resource reallocation, a combinatorial auction and an electronic payment network protocol. Other examples may be considered in the future (as work on scenarios).

We would like to conclude with a note about the relationship between D5 and the other activity of the project. In the Technical Annex, page 26, it reads:

> "The models for interaction should be integrated with the logic-based models for computees (WP1) in order to allow, through a computational counterpart (WP3), some interesting and useful global properties of the resulting system to be verified (WP5)."

We already discussed about the computational counterpart (related to WP3) in Section 4.3.

As far as the relationship between D5 and D4/WP1, the formalism developed in D5 has been designed for open and heterogeneous systems, therefore we did not put any constraints on the architecture and design of computees. However, it is a clear advantage of the whole approach of SOCS the fact that both the formalizations proposed in both D4 and D5 build upon a Computational Logic-based common ground. This means that it will be possible to uniformly reason upon both computees and societies, which will be particularly important once we start the third phase of the project (proving properties, in Workpackage WP5). Some hints in this respect, and in particular about on-the-fly verification of properties, are given in Section 6. On the Finally, the social framework of SOCS can be smoothly combined with the model of a computee (see Section 10 of deliverable D4).

Acknowledgments

We would like to thank the project referees for useful comments and suggestions on an earlier version of this document, and the internal reviewers Andrea Bracciali, Paolo Mancarella, and Francesca Toni, for the thorough work undergone in improving its later versions.

References

[ACG+03a] M. Alberti, A. Ciampolini, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Logic Based Semantics for an Agent Communication Language. In Barbara Dunin-Keplicz and Rineke

Verbrugge, editors, *Proceedings of the International Workshop on Formal Approaches to Multi-Agent Systems (FAMAS)*, pages 21–36, Warsaw, Poland, April 12 2003.

[ACG+03b] M. Alberti, A. Ciampolini, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. A social ACL semantics by deontic constraints. In V. Mařík, J. Müller, and M. Pěchouček, editors, *Multi-Agent Systems and Applications III. Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, volume 2691 of *Lecture Notes in Artificial Intelligence*, pages 204–213, Prague, Czech Republic, June 16-18 2003. Springer-Verlag.

[ACL] ACLP: Abductive Constraint Logic Programming. Electronically available. ACLP Home Page URL: `http://www.cs.ucy.ac.cy/aclp/`.

[AD95] H. Adé and M. Denecker. AILP: Abductive Inductive Logic Programming. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[AF97] S. Anthony and A. M. Frisch. Generating numerical literals during refinement. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297, pages 61–76. Springer-Verlag, 1997.

[Age] Agentcities. `http://www.agentcities.org`.

[AGL+03a] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. An Abductive Computational Model for Open Societies. In *Proceedings of the 8th National Congress on Artificial Intelligence (AI*IA), Pisa, Italy*. Springer-Verlag, September 23–26 2003. to appear in the Springer Verlag LNAI series.

[AGL+03b] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Modeling interactions using *social integrity constraints*: a resource sharing case study. In J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, *Declarative Agent Languages and Technologies, First International Workshop, DALT 2003. Melbourne, Victoria, July 15th, 2003. Workshop Notes*, pages 81–96, 2003.

[AGL+03c] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interactions using social integrity constraints. In W. van der Hoek, A. Lomuscio, E. de Vink, and M. Wooldridge, editors, *Proceedings of the Workshop on Logic and Communication in Multi-Agent Systems (LCMAS)*, Eindhoven, the Netherlands, June 29 2003. To appear.

[ALF99] ALFEBIITE: A Logical Framework for Ethical Behaviour between Infohabitants in the Information Trading Economy of the universal information ecosystem. IST-1999-10298. Web page, 1999. http://www.iis.ee.ic.ac.uk/ alfebiite/ab-home.htm.

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[AOR+99] K. A. Arisha, F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus. IMPACT: a Platform for Collaborating Agents. *IEEE Intelligent Systems*, (2):64–72, March/April 1999.

[APS99] J. Alferes, L. M. Pereira, and T. Swift. Well-founded abduction via tabled dual programs. In *Proceedings of the International Conference on Logic Programming*, pages 426–440, 1999.

[APS02] A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III, Bologna, Italy*, pages 1053–1061. ACM, 2002.

[Aus62] J. L. Austin. *How to Do Things with Words*. Harvard University Press, 1962.

[Axe97] R. Axelrod. *The Complexity of Cooperation.* Princeton studies in complexity. Princeton University Press, 1997.

[BF95] M. Barbuceanu and M. S. Fox. Cool: A language for describing coordination in multi-agent systems. In V. Lesser, editor, *Proceedings of the First Intl. Conference on Multi-Agent Systems*, pages 17–25. AAAI Press/The MIT Press, June 1995.

[BG94] F. Bergadano and D. Gunetti. Learning Clauses by Tracing Derivations. In *Proceedings 4th Int. Workshop on Inductive Logic Programming*, 1994.

[BG95] F. Bergadano and D. Gunetti. *Inductive Logic Programming.* MIT press, 1995.

[BGNR96] F. Bergadano, D. Gunetti, M. Nicosia, and G. Ruffo. Learning logic programs with negation as failure. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 107–123. 1996.

[BHS93] B. Burmeister, A. Haddadi, and K. Sundermeyer. Generic, configurable, cooperation protocols for multi-agent systems. In Cristiano Castelfranchi and Jean-Pierre Müller, editors, *From Reaction to Cognition, 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'93)*, number 957 in Lecture Notes in Computer Science, pages 157–171, Neuchatel, Switzerland, August 1993. Springer-Verlag.

[BLMM97] A. Brogi, E. Lamma, P. Mancarella, and P. Mello. A unifying view for logic programming with non-monotonic reasoning. *Theoretical Computer Science*, 184:1–59, 1997.

[BM92] M. Bain and S. Muggleton. Non-monotonic learning. In S. Muggleton, editor, *Inductive Logic Programming*, chapter 7, pages 145–161. Academic Press, 1992.

[BM98] K. Bouzouba and B. Moulin. Kqml+: An extension of kqml in order to deal with implicit information and social relationships. In *Proceedings of the Eleventh International Florida Artificial Intelligence Research Society Conference*, pages 289–293. AAAI Press, May 1998.

[BMO01] B. Bauer, J. P. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent interaction. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, pages 91–103. Springer, Berlin, 2001.

[CCF+00] R. S. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling conversations with colored Petri nets. In *[GB99]*, pages 178–192. 2000.

[CDJT99] C. Castelfranchi, F. Dignum, C.M. Jonker, and J. Treur. Deliberative normative agents: Principles and architecture. In Nicholas R. Jennings and Yves Lespérance, editors, *Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL), 6th International Workshop*, number 1757 in Lecture Notes in Computer Science, pages 364–378, Orlando, Florida, July 15-17 1999. Springer-Verlag.

[CF98] C. Castelfranchi and R. Falcone. Principles of trust for MAS: Cognitive anatomy, social importance, and quantification. In *Proceedings of the 3^{rd} International Conference on Multi-Agent Systems*, pages 72–79. IEEE Press, 1998.

[CFS99] R. Conte, R. Falcone, and G. Sartor. Special issue on agents and norms. *Artificial Intelligence and Law*, 1(7), March 1999.

[CFV02] M. Colombetti, N. Fornara, and M. Verdicchio. The role of institutions in multiagent systems. In *Proceedings of the Workshop on Knowledge based and reasoning agents, VIII Convegno AI*IA 2002, Siena, Italy*, 2002.

[CHH00] C. F. Camerer, D. Hsia, and T. Ho. Learning in bilateral call markets. Technical report, Caltech Working Paper, 2000.

[CLM+03] A. Ciampolini, E. Lamma, P. Mello, F. Toni, and P. Torroni. Co-operation and competition in *ALIAS*: a logic framework for agents that negotiate. *Computational Logic in Multi-Agent Systems. Annals of Mathematics and Artificial Intelligence*, 37(1–2):65–91, 2003.

[CLMT02] A. Ciampolini, E. Lamma, P. Mello, and P. Torroni. LAILA: A language for coordinating abductive reasoning among logic agents. *Computer Languages*, 27(4):137–161, February 2002.

[CLZ02] G. Cabri, L. Leonardi, and F. Zambonelli. Modeling role-based interactions for agents. In J. Debenham, B. Henderson-Sellers, N. Jennings, and J. Odell, editors, *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, Seattle, USA, November 4-8 2002.

[CMK96] A. Chavez, P. Maes, and Kasbah. An agent marketplace for buying and selling goods. In *Proceedings of the first International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, 1996.

[Cor02] P. A. Corning. The re-emergence of "emergence": A venerable concept in search of a theory. *Complexity*, 7(6):18–30, 2002.

[CP86] P. T. Cox and T. Pietrzykowski. Causes for events: Their computation and applications. In *Proceedings CADE-86*, page 608, 1986.

[Cru94] J. P. Crutchfield. Is anything ever new? In G. Cowan, D. Pines, and D. Melzner, editors, *Complexity: Metaphors, Models, and Reality*, SFI Series in the Sciences of Complexity XIX, pages 479–497. Addison-Wesley, 1994.

[CTS95] B. Cox, J.C. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *Proceedings of the First USENIX Workshop on Electronic Commerce*, New York, July 1995.

[Dav00] P. Davidsson. Emergent societies of information agents. In *Fourth International Workshop on Cooperative Information Agent – CIA'00*, volume 1860 of *LNCS series*. Springer Verlag, 2000.

[Dav01] P. Davidsson. Categories of artificial societies. In Omicini et al. [OPT01], pages 1–9. 2nd International Workshop (ESAW'01), Prague, Czech Republic, 7 July 2001, Revised Papers.

[DDRFK96] M. Denecker, L. De Raedt, P. A. Flach, and A. C. Kakas, editors. *Proceedings of ECAI96 Workshop on Abductive and Inductive Reasoning*. Catholic University of Leuven, 1996.

[DE02] J. Denzinger and S. Ennis. Being the new guy in an experienced team - enhancing training on the job. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III, Bologna, Italy*, pages 1246–1253. ACM, 2002.

[DEG01] DEGAS: Design Environment for Global Applications IST–2001–32072. Web page, 2001. http://www.omnys.it/degas.

[Del02] C. Dellarocas. The design of reliable trust management systems for online trading communities, 2002. Working paper. Available electronically.

[Dem95] Y. Demazeau. From interactions to collective behaviour in agent-based systems. In *Yves Demazeau. From interactions to collective behaviour in agent-based systems. In European Conference on Cognitive Sciences, 1995.*, 1995.

[DFCF91] A. Drogoul, J. Ferber, B. Corbara, and D. Fresneau. A behavioral simulation model for the study of emergent social structures. In *Proceedings of First European Conference on Articial Life*, pages 161–170. MIT Press, 1991.

[DHvdT00] M. Dastani, J. Hulstijn, and L. van der Torre. Negotiation protocols and dialogue games. In *Proceedings of the Belgium/Dutch Artificial Intelligence Conference (BNAIC00)*, pages 13–20, Kaatsheuvel, 2000.

[DK95] Y. Dimopoulos and A. C. Kakas. Learning non-monotonic logic programs: Learning exceptions. In *Proceedings of the 8th European Conference on Machine Learning*, 1995.

[DK96] Y. Dimopoulos and A. C. Kakas. Abduction and inductive learning. In *Advances in Inductive Logic Programming*. IOS Press, 1996.

[dKL98] M. d'Inverno, D. Kinny, and M. Luck. Interaction protocols in agentis. In *Mark d'Inverno, David Kinny, and Michael Luck. Interaction protocols in agentis. In Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*, pages 261–268, 1998.

[DM87] T. L. Dean and D. W. McDermott. Temporal data base management. *Artificial Intelligence*, 32:1–55, 1987.

[DMDW02] V. Dignum, J. J. Meyer, F. Dignum, and H. Weigand. Formal specification of interaction in agent societies. In *Proceedings of the Second Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS), Maryland*, October 2002.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[DMW02] V. Dignum, J. J. Meyer, and H. Weigand. Towards an organizational model for agent societies using contracts. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II, Bologna, Italy*, pages 694–695. ACM, 2002.

[DMWD02] V. Dignum, J. J. Meyer, H. Weigand, and F. Dignum. An organizational-oriented model for agent societies. In *Proceedings of International Workshop on Regulated Agent-Based Social Systems: Theories and Applications. AAMAS'02, Bologna*, 2002.

[DP98] C. V. Damásio and L. M. Pereira. A survey on paraconsistent semantics for extended logic programs. In D.M. Gabbay and Ph. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 2, pages 241–320. Kluwer Academic Publishers, 1998.

[DRL95] L. De Raedt and W. Van Lear. Inductive constraint logic. In *Proceedings of the 5th International Workshop on Algorithmic Learning Theory*, 1995.

[DS92] M. Denecker and D. De Schreye. SLDNFA: An abductive procedure for normal abductive programs. In Krzysztof Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSLP-92)*, pages 686–702, Cambridge, November 9–13 1992. MIT Press.

[DS93] M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. In *Proc. International Logic Programming Symposium ILPS93*, pages 147–163. The MIT Press, 1993.

[DS98] M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1998.

[EdlCS02] M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III, Bologna, Italy*, pages 1045–1052. ACM, 2002.

[EK89] K. Eshgi and R. A. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *Proceedings of the 6th International Conference on Logic Programming*, pages 234–255. MIT Press, 1989.

[ELM+96] F. Esposito, E. Lamma, D. Malerba, P. Mello, M. Milano, F. Riguzzi, and G. Semeraro. Learning abductive logic programs. In P. A. Flach and A. C. Kakas, editors, *Proceedings of the ECAI'96 Workshop on Abductive and Inductive Reasoning*, 1996. Available on-line at `http://www.cs.bris.ac.uk/~flach/ECAI96/`.

[EM02] T. Eiter and K. Makino. On Computing all Abductive Explanations. In *Proceedings Eighteenth National Conference on Artificial Intelligence (AAAI '02)*, pages 62–67, 2002.

[EMS02] Thomas Eiter, Viviana Mascardi, and V. S. Subrahmanian. Error-tolerant agents. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic. Logic Programming and Beyond*, volume 2407 of *Lecture Notes in Computer Science*, pages 586–625. Springer, 2002.

[EMS+03] U. Endriss, N. Maudet, F. Sadri, F. Toni, and P. Torroni. Preliminary list of verifiable properties of societies of computees. Technical report, SOCS Consortium, 2003. Internal document.

[EMST03a] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Aspects of Protocol Conformance in Inter-Agent Dialogue. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003), Poster*, 2003.

[EMST03b] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003)*. Morgan Kaufmann Publishers, August 2003.

[ERA+00] M. Esteva, J. Rodriguez, J. Arcos, C. Sierra, and P. Garcia. Formalising agent mediated electronic institutions. In *Esteva M., Rodriguez J.A., Arcos J.L., Sierra C., Garcia P. (2000); Formalising Agent Mediated Electronic Institutions, Catalan Congres on AI (CCIA 00)*, pages 29–38, 2000.

[ESP99] T. Eiter, V.S. Subrahmanian, and G. Pick. Heterogeneous active agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255, March 1999.

[FC02] N. Fornara and M. Colombetti. Operational specification of a commitment-based agent communication language. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002). Part II, Bologna, Italy*, pages 535–542. ACM, 2002.

[FIP01a] FIPA Communicative Act Library Specification. Experimental specification XC00037H, Foundation for Intelligent Physical Agents, August 2001. Published on August 10th, 2001, available for download from the FIPA website: `http://www.fipa.org`.

[FIP01b] FIPA Interaction Protocol Library Specification (XC00025E). Experimental specification, Foundation for Intelligent Physical Agents, 2001. Available for download from the FIPA website: `http://www.fipa.org`.

[FIP03] FIPA Modeling Area: Temporal Constraints. Technical report, 2003. Available for download from the AUML website: `http://auml.org/auml/documents/Temporal-Constraints.pdf`.

[FK97] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, November 1997.

[FLM97] T. Finin, Y. Labrou, and J. Mayfield. *Software Agents*, chapter KQML as an agent communication language. MIT Press, Cambridge, 1997.

[Frü98] T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1-3):95–138, October 1998.

[GB99] M. Greaves and J. Bradshaw, editors. *Proceedings of the International Workshop on Specifying and Implementing Conversation Policies, Seattle, WA*, May 1999.

[GC] Global Computing: Co-operation of Autonomous and Mobile Entities in Dynamic Environments. Global Computing Website: `http://www.cordis.lu/ist/fetgc.htm`.

[GHB00] M. Greaves, H. Holmback, and J. Bradshaw. What is a conversation policy? In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, number 1916 in Lecture Notes in Computer Science, pages 118–131. Springer-Verlag: Heidelberg, Germany, 2000.

[GMM98] R. Guttman, A. Moukas, and P. Maes. Agent-mediated electronic commerce: A survey. *Knowledge Engineering Review*, 13(2):143–147, 1998.

[GP01] F. Guerin and J. Pitt. A denotational semantics for agent communication languages. In J.P. Andre, E. Muller, S. Sen, and C. Frasson, editors, *Fifth International Conference on Autonomous Agents*, pages 497–504. ACM Press, New York, USA, 2001.

[GP02] F. Guerin and J. Pitt. Proving properties of open agent systems. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II, Bologna, Italy*, pages 557–558. ACM, 2002.

[Ham70] C.L. Hamblin. *Fallacies*. Methuen, London, 1970.

[Hew91] C. Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47(1-3):79–106, 1991.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8), 1978.

[Hug02] M. Huget. Agent uml class diagrams revisited, 2002.

[Hum48] D. Hume. *An Enquiry Concerning Human Understanding*. 1748.

[HW98] J. Hu and M.P. Wellman. Online learning about other agents in a dynamic multiagent system. In *Proceedings of the Second International Conference on Autonomous Agents (Agents-98)*, 1998.

[IS00] M. Ito and J.S. Sichman. Dependence based coalitions and contract net: A comparative analysis. In *Pacific Rim International Conference on Artificial Intelligence*, page 812, 2000.

[JM94] J. Jaffar and M.J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20:503–582, 1994.

[Kak00] A. C. Kakas. ACLP: integrating abduction and constraint solving. In *Proceedings NMR'00, Breckenridge, CO*, 2000.

[KKT98] 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.

[KL91] H. A. Kautz and P. B. Ladkin. Integrating metric and qualitative temporal reasoning. In *Proceedings AAAI91*, pages 241–246, 1991.

[KM90] A. C. Kakas and P. Mancarella. On the relation between Truth Maintenance and Abduction. In T. Fukumura, editor, *Proceedings of the first Pacific Rim International Conference on Artificial Intelligence, PRICAI-90, Nagoya, Japan*, pages 438–443. Ohmsha Ltd., 1990.

[KR96] A. C. Kakas and F. Riguzzi. Abductive concept learning. Technical Report TR-96-15, University of Cyprus, Computer Science Department, 1996.

[KR97] A. C. Kakas and F. Riguzzi. Learning with abduction. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, 1997.

[KR00] A. C. Kakas and F. Riguzzi. Learning with abduction. *New Generation Computing*, 18(3), 2000.

[Kro95] C. Krogh. Obligations in multiagent systems. In A. Aamodt and J. Komorowski, editors, *Proceedings of the 5th Scandinavian Conference on Artificial Intelligence*, pages 19–30, Trondheim, Norway, May29–31 1995. ISO Press, Amsterdam.

[KS86] R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, (4):67–95, 1986.

[KS99] R. A. Kowalski and F. Sadri. From logic programming to multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 1999.

[KSST03] A. Kakas, F. Sadri, K. Stathis, and F. Toni. A logic-based approach to model computees. Technical report, SOCS Consortium, 2003. Deliverable D4.

[KTW98] R.A. Kowalski, F. Toni, and G. Wetzel. Executing suspended logic programs. *Fundamenta Informaticae*, (34):203–224, 1998. http://www-lp.doc.ic.ac.uk/UserPages/staff/ft/PAPERS/slp.ps.Z.

[KvND01] A.C. Kakas, B. van Nuffelen, and M. Denecker. A-System: Problem solving through abduction. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 591–596, Seattle, Washington, USA, August 2001. Morgan Kaufmann.

[LD94] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[LFP99] Y. Labrou, Tim Finin, and Yun Peng. Agent communication languages: The current landscape. *IEEE Intelligent Systems*, 14(2):45–52, March/April 1999.

[LL02] F. López y López and M. Luck. Towards a model of the dynamics of normative multi-agent systems. In *Proceedings of the International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02)*, pages 175–193, 2002.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd extended edition, 1987.

[LMM+03] E. Lamma, P. Mello, P. Mancarella, A. Kakas, K. Stathis, and F. Toni. Self-assessment: parameters and criteria. Technical report, SOCS Consortium, 2003. Deliverable D3. Distribution restricted to the GC programme.

[LMMR97a] E. Lamma, P. Mello, M. Milano, and F. Riguzzi. Integrating induction and abduction in logic programming. In P. P. Wang, editor, *Proceedings of the Third Joint Conference on Information Sciences*, volume 2, pages 203–206, 1997.

[LMMR97b] E. Lamma, P. Mello, M. Milano, and F. Riguzzi. Introducing Abduction into (Extensional) Inductive Logic Programming Systems. In M. Lenzerini, editor, *AI*IA97, Advances in Artificial Intelligence, Proceedings of the 5th Congress of the Italian Association for Artificial Intelligence*, number 1321 in LNAI. Springer-Verlag, 1997.

[LR02] S. Lynden and O. F. Rana. LEAF: A FIPA compliant software toolkit for learning based MAS. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III, Bologna, Italy*, pages 1260–1261. ACM, 2002.

[Mar94] S. Marsh. Trust in distributed artificial intelligence. In C. Castelfranchi and E. Werner, editors, *Artificial Social Societies, LNAI 830*, pages 94–112. Springer, Heidelberg, 1994.

[Mat94] M. Mataric. Learning to behave socially. In *Third International Conference on Simulation of Adaptive Behavior*, 1994.

[Mat98] M. Mataric. Coordination and learning in multirobot systems. *IEEE Intelligent Systems*, pages 6–8, March/April 1998.

[MC02] N. Maudet and B. Chaib-draa. Commitment-based and dialogue-game based protocols – new trends in agent communication languages. *The Knowledge Engineering Review*, 17(2):157–179, 2002.

[MCM83] R. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. *Machine Learning - An Artificial Intelligence Approach*. Springer-Verlag, 1983.

[MEH02] H. Mazouzi, A. El Fallah Seghrouchni, and S. Haddad. Open protocol design fo complex interactions in multi-agent systems. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part I, Bologna, Italy*, pages 402–409. ACM, 2002.

[Mei91] I. Meiri. Combining qualitative and quantitative constraints in temporal reasoning. In *Proceedings of AAAI91*, pages 260–267, 1991.

[MM02] M. Mui and M. Mohtashemi. Rational decision making using social information, 2002. Submitted to Rationality and Society. Available electronically.

[MOB03] A. Mounier and F. Jacquenet O. Boissier. Conversation mining in multi-agent systems. Submitted, 2003.

[MP02] P. McBurney and S. Parsons. Games that agents play: a formal framework for dialogue between autonomous agents. *Journal of Logic, Language, and Information —Special issue on logic and games*, 11(3), 2002.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Part I and II.*, 100(1):1–77, 1992.

[MPW02] P. McBurney, S. Parsons, and M. Wooldridge. Desiderata for agent argumentation protocols. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part I, Bologna, Italy*, pages 402–409. ACM, 2002.

[MV96] L. Martin and C. Vrain. A three-valued framework for the induction of general logic programs. In *Advances in Inductive Logic Programming*. IOS Press, 1996.

[MYT01] MYTHS: Models and Types for Security in Mobile Distributed Systems IST–2001–32617. Web page, 2001. http://www.cogs.susx.ac.uk/users/vs/myths/myths.htm.

[Nis00] N. Nisan. Bidding and allocation in combinatorial auction. In *Proceedings of the International Conference on Electronic Commerce (EC-00)*, 2000.

[NS02] P. Noriega and C. Sierra. Institutions in perspective: An extended abstract. In *Sixth International Workshop CIA-2002 on Cooperative Information Agents*, volume 2446 of *Lecture Notes in Artificial Intelligence*. Springer, 2002.

[OPT01] A. Omicini, P. Petta, and R. Tolksdorf, editors. *Engineering Societies in the Agents World II*, volume 2203 of *LNAI*. Springer-Verlag, December 2001. 2nd International Workshop (ESAW'01), Prague, Czech Republic, 7 July 2001, Revised Papers.

[Ost98] E. Ostrom. A behavioral approach to the rational-choice theory of collective action. *American Political Science Review*, 92:1–22, 1998.

[PG02] J. Pitt and F. Guerin. Guaranteeing properties for e-commerce systems. Technical Report TRS020015, Department of Electrical and Electronic Engineering, Imperial College, London, UK, 2002.

[PNJ99] P. Panzarasa, T. Norman, and N. Jennings. Modeling sociality in the BDI framework. In *Proceedings of the First Asia-Pacific Conference on Intelligent Agent Technology*. World Scientific Publishing, 1999.

[Poo88] D. L. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.

[Pop68] K. Popper. *The Logic of Scientific Discovery*. Hutchinson, London, 1968.

[PV90] R. Cohen P. VanBeek. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.

[RG92a] A. Rao and M. Georgeff. An abstract architecture for rational agents. In *Proceedings of the International Workshop on Knowledge Representation (KR'92)*, 1992.

[RG92b] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KRR92), Boston, MA*, 1992.

[RZ94] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, Cambridge, Massachusetts, 1994.

[San96] T. Sandholm. *Negotiation among Self-Interested Computationally Limited Agents*. Computer science, University of Massachusetts at Amherst, September 1996.

[San00] T. Sandholm. eMediator: a next generation electronic commerce server. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents-2000)*, 2000.

[San02] T. Sandholm. Algorithm for optimal winner determination in combinatorial auction. *Artificial Intelligence*, 135, 2002.

[Sat02] K. Satoh. Speculative computation with multi-agent belief revision. In *Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems*, pages 897 – 904, Bologna, Italy, 2002.

[SCDC98] J.S. Sichman, R. Conte, Y. Demazeau, and C. Castelfranchi. *Readings in Agents*, chapter A social reasoning mechanism based on dependence networks, pages 416–420. Morgan Kaufmann Publishers, 1998.

[SD95] J.S. Sichman and Y. Demazeau. Exploiting social reasoning to deal with agency level inconsistency. In V. Lesser, editor, *Proceedings First International Conference on Multi-agent Systems, San Francisco, California*, pages 352–359. AAAI Press/The MIT Press, Menlo Park, June 1995.

[SD01] J.S. Sichman and Y. Demazeau. On social reasoning in multi-agent systems. *Revista Iberoamericana de Inteligencia Artificial*, 13:68–84, 2001.

[SEC01] SECURE: Secure Environments for Collaboration among Ubiquitous Roaming Entities IST–2001–32486. Web page, 2001. http://www.dsg.cs.tcd.ie/index.php?category_id=206.

[SI92] K. Satoh and N. Iwayama. A query evaluation method for abductive logic programming. In *In proceedings of the 1992 Joint International Conference and Symposium on Logic Programming*, pages 671–685, 1992.

[Sic01] J.S. Sichman. A dependence-based model for social reasoning in multi-agent systems. Technical Report BT/PCS/0108, Escola Politécnica da Universidade de São Paulo, 2001.

[Sin98] M. Singh. Agent communication language: rethinking the principles. *IEEE Computer*, pages 40–47, December 1998.

[Sin00] M. P. Singh. A social semantics for agent communication languages. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, pages 31–45. Springer-Verlag, Heidelberg, Germany, 2000.

[Smi80] R.G. Smith. The contract net protocol: High level communication and control in a distributed porblem solver. *IEEE Transaction on Computers*, C-29(12):1104–1113, 1980.

[SOC] SOCS: Societies Of ComputeeS (SOCS): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees. SOCS Home Page URL: `http://lia.deis.unibo.it/Research/SOCS/`.

[ST97] Y. Shoham and M. Tennenholtz. On the emergence of social conventions: modeling, analysis and simulations. *Artificial Intelligence*, 94:139–166, 1997.

[ST00] F. Sadri and F. Toni. Abduction with negation as failure for active and reactive rules. In E. Lamma and P. Mello, editors, *Proceedings AI*IA 99, 6th Congress of the Italian Association for Artificial Intelligence*, number 1792 in LNAI, pages 49–60. Springer Verlag, 2000.

[STT02a] F. Sadri, F. Toni, and P. Torroni. An abductive logic programming architecture for negotiating agents. In S. Greco and N. Leone, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 2424 of *LNCS*, pages 419–431. Springer Verlag, September 2002.

[STT02b] F. Sadri, F. Toni, and P. Torroni. Dialogues for negotiation: agent varieties and dialogue sequences. In *Intelligent Agents VIII, 8th International Workshop, ATAL 2001 Seattle, WA, USA, August 1-3, 2001 Revised Papers*, volume 2333 of *LNAI*, pages 405–421. Springer Verlag, 2002.

[STT02c] F. Sadri, F. Toni, and P. Torroni. A multi-stage negotiation architecture for sharing resources amongst logic-based agents (preliminary report). short paper. In *UK Multi-Agent Systems (UKMAS) Annual Conference, Liverpool, UK*, December 2002.

[STT03] F. Sadri, F. Toni, and P. Torroni. Minimally intrusive negotiating agents for resource sharing. In G. Gottlob, editor, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. AAAI Press, August 2003.

[SV00] P. Stone and M. M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[TMM+02] P. Torroni, P. Mello, N. Maudet, M. Alberti, A. Ciampolini, E. Lamma, F. Sadri, and F. Toni. A logic-based approach to modeling interaction among computees (preliminary report). In *UK Multi-Agent Systems (UKMAS) Annual Conference, Liverpool, UK*, December 2002.

[vdT03] L. van der Torre. Contextual deontic logic: Normative agents, violations and independence. *Annals of Mathematics and Artificial Intelligence*, 37(1):33–63, 2003.

[vH89] P. van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.

[VK89] M. B. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of AAAI89*, pages 377–382, 1989.

[vND00] B. van Nuffelen and M. Denecker. Problem solving in ID-logic with aggregates. In *Proceedings of NMR'2000, special track on Abductive Reasoning*, pages 1–9, 2000.

[vP96] H. van Parunak. Visualizing agent conversations: using enhanced dooley graphs for agent design and analysis. In *Proceedings of the Second International Conference on Multi-agent Systems (ICMAS96)*, pages 275–282. AAAI Press, 1996.

[Wel93] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[Wil85] R. Wilson. Reputations in games and markets. In A. Roth, editor, *Game-Theoretic Models of Bargaining*, pages 27–62. Cambridge University Press, 1985.

[WJK00] M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, September 2000.

[Wri51] G.H. Wright. Deontic logic. *Mind*, 60:1–15, 1951.

[WW95] A. Walker and M. Wooldridge. Understanding the emergence of conventions in multi-agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems*, 1995.

[WW98] M. P. Wellman and P. R. Wurman. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24:115–125, 1998.

[WWT99] D.H. Wolpert, K.R. Wheeler, and K. Tumer. General principles of learning-based multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)*, 1999.

[WWW98] P.R. Wurman, M.P. Wellman, and W.E Walsh. The michigan internet auctionbot: A configurable auction server for human and software agents. In *Proceedings of the Second International Conference on Autonomous Agents (Agents-98)*, 1998.

[YS02] P. Yolum and M.P. Singh. Flexible protocol specification and execution: applying event calculus planning using commitments. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II, Bologna, Italy*, pages 527–534. ACM, 2002.

[ZM99] G. Zacharia and P. Maes. Collaborative reputation mechanisms in electronic marketplaces. In *Proceedings of the 32^{nd} Hawaii International Conference on System Sciences*, 1999.

[ZS96] D. Zeng and K. Sycara. Bayesian learning in negotiation. In Sandip Sen, editor, *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, pages 99–104, Stanford University, CA, USA, 1996.

[ZS97] D. Zeng and K. Sycara. Benefits of learning in negotiation. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 36–42, Menlo Park, 27–31 1997. AAAI Press.

A Abductive Logic Programming

In this section we give some background to those who are not very familiar with Logic Programming. In particular, we will review a technology that we use to give an interpretation of our model of societies of computees: Abductive Logic Programming (ALP). Some background on ALP is also given in deliverable D4 [KSST03], in more general terms. Here, we want to focus on the computational aspects of the abductive process, by briefly introducing the IFF proof-procedure [FK97], which we refer to very often in this report.

Abduction has been widely recognized as a powerful mechanism for hypothetical reasoning in the presence of incomplete knowledge [CP86, EK89, KM90]. Incomplete knowledge is handled by labelling some pieces of information as abducibles, i.e., possible hypotheses which can be assumed, provided that they are consistent with the current knowledge base. More formally, given a theory T and a formula G, the goal of abduction is to find a (possibly minimal) set of atoms Δ which together with T "entails" G, with respect to some notion of "entailment" that the language of T is equipped with.

Some results about the computational complexity of the abductive process are presented by Eiter & Makino in [EM02].

In the following, we briefly recall the framework of ALP [KKT98], where T is a logic program. Within ALP, abductive proof-procedures (well-assessed in the Computational Logic setting, e.g. see [EK89, KM90, FK97]) can be exploited to compute any Δ as above, given G and T.

A.1 Abductive Logic Programming

An *abductive logic program* is a triple $\langle P, Ab, IC\rangle$ where:

- P is a (normal) logic program, that is, a set of clauses of the form $A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_{m+n}$, where $m, n \geq 0$, each A_i $(i = 1, \ldots, m+n)$ is an atom, and all variables are implicitly universally quantified from the outside. A_0 is called the *head* and $A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_{m+n}$ is called the *body* of any such clause.
- Ab a set of *abducible predicates*, p, such that p is a predicate in the language of P which does not occur in the head of any clause of P (without loss of generality, see [KKT98]).
- IC is a set of integrity constraints, that is, a set of sentences in the language of P.

Abducible predicates (or simply abducibles) are the predicates about which assumptions (or abductions) can be made. These predicates carry all the incompleteness of the domain, they can have a partial definition or no definition at all, while all other predicates have a complete definition in the logic program.

Given an abductive logic program $T = \langle P, Ab, IC\rangle$ and a formula G, the goal of abduction is to find a (possibly minimal) set of ground atoms Δ (*abductive explanation*) in predicates in Ab which, together with P, "entails" G, i.e., $P \cup \Delta \vdash G$, and such that $P \cup \Delta$ "satisfies" IC, e.g. $P \cup \Delta \vdash IC$ (see [KKT98] for other possible notions of integrity constraint "satisfaction"). Here, the notion of "entailment" $\vdash$ depends on the semantics associated with the logic program P (there are many different choices for such semantics, as it is well-documented in the Computational Logic literature).

Operationally, we write $T \vdash_{Abd} G$ when there exists an abductive explanation computed by an abductive proof-procedure for G in T.

In [KM90] Kakas and Mancarella define a proof-procedure (here referred to as KM) for ALP, as an extension of that in [EK89]. This procedure assumes that the integrity constraints are in the form of *denials*, i.e.:

$$\leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_{m+n}$$

where each A_i, $i = 1, \ldots, m+n$ is an atom, $m > 0$, at least one of the A_i, $i = 1, \ldots m$ is abducible, and all variables are implicitly universally quantified from the outside. The semantics of such integrity constraints within the KM proof-procedure requires that at least one of the literals in the constraint does not hold. The procedure starts from a goal and a set of initial assumptions Δ_i and results in a set of consistent hypotheses (abduced literals) Δ_o such that $\Delta_o \supseteq \Delta_i$ and Δ_o together with the program P entails the goal. The proof-procedure uses the notion of *abductive* and *consistency derivations*. Intuitively, an abductive derivation is the standard Logic Programming SLD-derivation suitably extended in order to consider abducibles. As soon as an abducible atom δ is encountered which does not already occur in the current set of hypotheses, it is added to the current set of hypotheses, and it must be proved that any integrity constraint such that δ is an instance of any of the A_i, $i = 1, \ldots, m$, is satisfied. For this purpose, a consistency derivation for δ is started. Since the constraints are denials only (i.e., goals), this corresponds to proving that every such goal fails to hold. Therefore, δ is removed from all the constraints containing it, and we prove that all the resulting goals fail. In this consistency derivations, when an abducible is encountered, an abductive derivation for its complement is started in order to prove the abducible's failure, so that the initial constraint is satisfied. When the procedure succeeds for the goal G and the initial set of assumptions Δ_i, producing as output the set of assumptions Δ_o, we say that T *abductively derives* G or that G is *abductively derivable* from T and we write $T \vdash_{\Delta_o} G$.

The original KM proof-procedure can only deal with abducibles which are ground at selection time, and *flounders* if the selected abducibles are not ground. Moreover, it treats *constraint predicates*, such as $<, \leq, \neq, \ldots$, as ordinary predicates, thus being unable to use specialized constraint solvers for such predicates. Some work have proposed abductive proof-procedures dealing with non-ground abducibles and/or constraints which are relevant for our purposes. In the following, we survey some of them.

A.2 Abductive proof-procedures dealing with variables in hypotheses and with constraints

In [DS92], Denecker and De Schreye introduce a proof procedure for normal abductive logic programs by extending the SLDNF resolution to the case of abduction. The resulting proof-procedure (SLDNFA) is correct with respect to the completion semantics. A crucial property of this abductive procedure is the treatment of *non-ground* abductive goals. In [DS92], the authors do not consider general integrity constraints, but only constraints of the kind $\leftarrow a, not\ a$. To overcome this limitation, in a later work [DS93], they consider the treatment of general integrity constraints but in a quite inefficient way. In practice, they check all the integrity constraints at the end of the proof for a query, i.e., only when the overall set of abductive hypotheses supporting the query has been computed. More recent work is represented by the SLDNFA(C) system [vND00] which extends SLDNFA with constraints.

A recent abductive proof-procedure dealing with constraints is also contained in [ACL], where the ACLP system is presented. ACLP programs can contain constraints on finite domains. ACLP interleaves consistency checking of abducible assumptions and constraint satis-

faction. Finally, $\mathcal{A}$-system [KvND01] is a followup of ACLP and SLDNFA(C). $\mathcal{A}$-system differs from the previous two for the explicit treatment of non-determinism that allows the use of heuristic search with different types of heuristics.

Among other abductive proofs existing in literature, we cite here the abductive query evaluation method proposed by Satoh and Iwayama in 1992 [SI92] and Abdual, by Alferes, Pereira & Swift, [APS99].

A.3 The IFF proof-procedure

Another recent proof-procedure dealing with non-ground abducibles is presented in [FK97], referred to as IFF. This uses backward reasoning with the selective completion of the given logic program (namely its completion, but only with respect to the non-abducible predicates) to compute abductive explanations for given goals. The goals are conjoined with the integrity constraints at the beginning of the abductive process, and forward reasoning with them is applied. The integrity constraints do not need to be denials, but can be any (closed) implications. This procedure is proposed as the engine underlying the reasoning mechanism of intelligent agents in [KS99]. An extension of this procedure to deal with constraint predicates is given in [KTW98]. An extension of this procedure to deal with negation as failure in integrity constraints is proposed in [ST00].

The IFF proof-procedure [FK97] uses a syntax very related to the one in our framework; in the following we discuss it in detail.

A.3.1 Syntax

An abductive program is a triple $\langle T, Ab, IC \rangle$, where:

- T is a theory given by a set of definitions

  $$P(X_1, \ldots, X_k) \leftrightarrow D_1 \vee \cdots \vee D_n$$

  where D_i is a conjunction of literals (positive or negative atoms). The semantics is similar to that of the completion of a logic program, except for the fact that there can be more definitions for a unique predicate. This is used to distinguish *suspended* atoms: in fact, when using the *unfolding* rule (that replaces *resolution*), only one-way unification (a.k.a. pattern-matching) is used. For example, the user can either define a predicate `p/1` as

  $$p(1) \leftrightarrow true.$$
  $$p(2) \leftrightarrow true.$$

  or as

  $$p(X) \leftrightarrow X = 1 \vee X = 2.$$

  the two are declaratively equivalent, but have a different operational semantics: a query $? - p(Y)$ is suspended in the first case and opens a choice point in the second.

  The syntax forbids to write two definitions with unifying heads; this avoids backtracking in the selection of the definition.

  Apart from this, the meaning is the same as for the completion: all variables in the head are universally quantified and the remaining (in the body) are existentially quantified.

  A definition is *allowed* if every variable in a negated atom also occurs in a positive atom in the same D_i (intuitively, in the same branch);

- Ab is a set of abducible predicates;
- IC is a set of Integrity constraints. It is a set of implications

$$A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$$

where all variables are universally quantified, A_i and B_i are atoms (can be abducibles or defined predicates), but they cannot be the negation of an atom. An IC is *allowed* if every variable in the conclusion occurs in the condition.

A.3.2 Proof-procedure

The IFF proof-procedure is based on rewriting. It starts with a formula (that replaces the concept of *resolvent* in logic programming) built as a conjunction of the initial query and the ICs. Then it repeatedly applies one of the following *inference rules*:

unfolding replaces resolution;

propagation propagates ICs;

splitting distributes conjunctions and disjunctions, making the final formula in a sum-of-products form;

case analysis if the body of an IC contains $X = t$, case analysis nondeterministically tries $X = t$ or $X \neq t$,

factoring tries to reuse an previously made hypothesis;

rewrite rules for equality use the inferences in the Clark Equality Theory;

logical simplifications try to simplify a formula through equivalences like $A \wedge false \leftrightarrow false$, $[A \leftarrow true] \leftrightarrow true$, etc.

Thanks to these inference rules, each node is always translated into a (disjunction of) conjunctions of atoms and implications; e.g., it can look like:

$$\begin{array}{ll} & (A_1 \wedge A_2 \wedge [A_3 \leftarrow B_1 \wedge B_2] \wedge [A_4 \leftarrow B_3 \wedge B_4]) \\ \vee & (A_i \wedge A_j \wedge A_k \wedge [A_z \leftarrow B_y] \wedge [false \leftarrow B_5]) \end{array}$$

the atoms have a similar meaning to those in the resolvent in LP, while the implications are (partially-propagated) integrity constraints.

Given a formula, it is always clear the quantification of the variables by the following rules:

- *if* a variable is in the initial query, then it is *free*;
- *else if* it occurs in an atom, it is existentially quantified;
- *else* (it occurs only in implications) it is universally quantified.

A.3.3 Negation

A negated atom $\neg A$ is rewritten as $false \leftarrow A$. Notice that this does not change the existential quantification of the atom because of the *allowedness condition*. A variable can occur in a negated atom only if it also occurs in a positive atom. A variable is universally quantified only if it occurs only in implications. Thus, if an implication $false \leftarrow A$ was generated by the transformation of a negated atom $\neg A$, the variables in A necessarily occur also in a positive atom, and must be considered existentially quantified.

B Comparing IC_S with the integrity constraints of the IFF proof-procedure

The syntax of IC_S is strictly related with that of integrity constraints in the IFF proof-procedure [FK97]. This leads to the idea of using an extension of the IFF proof-procedure as a proof-procedure for our language. We can map **E** and **NE** as abducible predicates, while **H** atoms are considered as facts in the history of the society.

A first, obvious difference stands in the fact that our framework requires more dynamics. New facts (**H** events) continuously occur in the KB of the society, and must be taken into account by the proof.

Other differences in the syntax are given in the following.

Syntactic restrictions - In the IFF definitions, integrity constraints and queries are required to be *allowed*. A definition and a query are allowed if every variable occurring in a negative atom also occurs in a positive atom (or in the head of the definition). This ensures that all variables in a negative atom will be existentially quantified (or free). An Integrity Constraint is allowed if every variable in the conclusion also occurs in the condition. All these restrictions are present in order to avoid explicit quantification of the variables, as explained in the next paragraph.

Implicit/explicit quantification of variables - The IFF proof-procedure is a rewriting systems that produces, at each node, a *Formula*. A formula is a conjunction of atoms and implications. Thanks to the syntactic restrictions explained earlier, for every variable it is always clear if the variable is quantified existentially or universally; thus, there is no need to add an explicit quantification.

Quantification of variables in abduced atoms - In the IFF proof-procedure, abduced atoms are always existentially quantified. In our framework, we need to express concepts like *"An event is expected not to happen in a time interval"* or *"All computees are expected not to perform a given action"* or *"A computee is expected not to perform all the actions in a given class"*. To express an expected behavior one needs to forbid actions for all the possible values of the variables. For this reason, new variables in **NE** atoms are universally quantified.

In the IFF proof-procedure, universal quantification can be put in the integrity constraints; for example, instead of writing

$$\mathbf{H}(tell(X, Y, yes, D), T) \rightarrow \mathbf{NE}(tell(X, Y, no, D), T')$$

(where T' is universally quantified in our framework, and would be existentially quantified in the IFF), one may write (in the IFF clauses use the symbol $\leftrightarrow$, as they represent a completion of a set of clauses):

$$\mathbf{H}(tell(X, Y, yes, D), T), \mathbf{H}(tell(X, Y, no, D), T') \rightarrow false$$

Note that this provides a different set of expectations, thus the result is different. The set of expectations may be visible to the computees, so they can reason about their expected behavior.

Quantification of variables in Integrity Constraints - Variables in the integrity constraints of the IFF proof-procedure are all universally quantified, while in our case some of the variables are universally quantified, and some are existentially quantified. This derives from the fact that positive expectations (**E**) mean that some computee should perform a given action choosing some parameters (e.g., the time at which the action is performed, or the addressee of the action, etc.). In the previous example

$$\begin{array}{rcl} \mathbf{H}(tell(X, Y, ask, D), T) & \rightarrow & \mathbf{E}(tell(Y, X, yes, D), T') \\ & \vee & \mathbf{E}(tell(Y, X, no, D), T') \end{array}$$

Variable T' is existentially quantified, while in the IFF it is not accepted, as the integrity constraint is not *allowed*.

In the IFF one can put existentially quantified variables in the body of a definition, thus one would write:

$$\mathbf{H}(tell(X, Y, ask, D), T) \rightarrow reply(X, Y, D)$$

$$\begin{array}{rcl} reply(X, Y, D) & \leftrightarrow & \mathbf{E}(tell(Y, X, yes, D), T') \\ & \vee & \mathbf{E}(tell(Y, X, no, D), T') \end{array}$$

C Learning

In the following, we briefly recall the main definition of ILP, and some extensions provided for learning abductive logic programs. This part is aimed at providing some background which can help the reader in better understanding the exploitation of ILP to learn social behavior in societies of computees.

C.1 Inductive Logic Programming

ILP is the research area that studies the problem of inductive concept learning from examples when the representation language employed is Logic Programming. In particular, ILP is the research area covering the intersection of Machine Learning [MCM83] and Logic Programming. Its aim is to devise systems that are able to learn logic programs from examples and from a background knowledge.

The ILP problem can be defined as [BG94]:
Given:

- a set $\mathcal{P}$ of logic programs (*hypotheses space*)
- a set E^+ of positive examples

- a set E^- of negative examples
- a logic program B (*background knowledge*)

Find:

- a logic program $P \in \mathcal{P}$ (*target program*) such that
  - $\forall e^+ \in E^+$, $B \cup P \vdash e^+$ (*completeness*)
  - $\forall e^- \in E^-$, $B \cup P \nvdash e^-$ (*consistency*).

The predicates which are defined in P are called *target predicates*. The sets E^+ and E^- are called *training sets* and contain ground atoms for the target predicates. The program B contains the definitions of the predicates that are already known. We say that the program P *covers* an example e if[15] $P \cup B \vdash e$, i.e., if the theory "explains" the example. Therefore the conditions that the program P must satisfy in order to be a solution to the ILP problem can be expressed as "P must cover all positive examples and must not cover any negative example". A theory that covers all positive examples is said to be *complete* while a theory that does not cover any negative example is said to be *consistent*. The importance of the hypotheses space lies in the fact that it defines the search space of the ILP system. In order to be able to effectively learn a program, this space must be restricted as much as possible. If the space is too big, the search could be unaffordable.

The *language bias* is a description of the hypothesis space (i.e., the forms of program clauses to be learned).

There are two broad categories of ILP learning methods: *bottom-up* methods and *top-down* methods. In bottom-up methods clauses in P are generated by starting with a clause that covers one or more positive examples and no negative example, and by generalizing it as much as possible without covering any negative example. In top-down methods clauses in P are constructed starting with a general clause that covers all positive and negative examples and by specializing it until it does no longer cover any negative example while still covering at least one positive. A basic top-down inductive algorithm [BG95, LD94] learns programs by generating clauses one after the other. A clause is generated by starting with an empty body and iteratively adding literals to the body.

ILP has been initially applied to learn definite logic program. However, in the last years, in this research area, a number of works appeared on the problem of learning non-monotonic logic programs [BM92, DK95, BGNR96, MV96]. Particular attention has also been given to the problem of learning abductive logic programs [ELM$^+$96, KR96, LMMR97a, LMMR97b, KR97] and, more generally, to the relation existing between abduction and induction and how they can integrate and complement each other [DDRFK96, DK96, AD95].

Learning abductive logic programs allows one to deal with knowledge incompleteness, and also learn default theories and exceptions to general rules, since default negation can be mapped into abduction. These are very frequent cases in practice, because very often the available data is incomplete and/or noisy. Abduction helps induction by allowing to make assumptions about unknown facts.

[15] In the ILP literature, the derivability relation is often used instead of entailment because real systems adopt the Prolog interpreter for testing the coverage of examples, although this is not sound nor complete.

C.2 Learning with abduction

In [LMMR97a, KR00] members of the SOCS consortium defined a new learning problem called Abductive Concept Learning. In this new framework an abductive logic program is generated from an abductive background knowledge and from a set of positive and negative examples of the concepts to be learned. Moreover, abductive derivability $\vdash_{Abd}$ is used as the example coverage relation instead of deductive derivability as in conventional ILP.

Definition 18 (Abductive Concept Learning).
Given

- *a hypothesis space* $\mathcal{T} = \langle \mathcal{P}, \mathcal{I} \rangle$ *consisting of a space of possible logic programs* $\mathcal{P}$ *and a space of possible integrity constraints* $\mathcal{I}$*,*
- *a set of positive examples* E^+ *(ground facts),*
- *a set of negative examples* E^- *(ground facts),*
- *an abductive logic program* $T = \langle P, A, I \rangle$ *as background theory,*

Find

A set of rules $P' \in \mathcal{P}$ *and a set of integrity constraints* $I' \in \mathcal{I}$ *such that the new abductive logic program* $T' = \langle P \cup P', A, I \cup I' \rangle$ *satisfies the following conditions*

- $T' \vdash_{Abd} E^+$*,*
- $\forall e^- \in E^-$*,* $T' \nvdash_{Abd} e^-$*.*

We say that an individual example e *is* covered *by a theory* T' *if and only if* $T' \vdash_{Abd} e$*.*

The full abductive concept learning problem can be split into two subproblems: (1) learning the rules together with appropriate explanations and (2) learning integrity constraints. The solutions of the two subproblems can be combined to obtain a solution for the original problem. [LMMR97a] gives a top-down algorithm for the first subproblem. The second subproblem can be solved by means of a system that learns from interpretations, like ICL [DRL95].