A Proof-system for the Safe Execution of Tasks in Multi-Agent Systems

A. Ciampolini[1], E. Lamma[2], P. Mello[1], and P. Torroni[1]

[1] DEIS, Università di Bologna,
Viale Risorgimento 2, 40136 Bologna, Italy
{aciampolini,pmello,ptorroni}@deis.unibo.it

[2] Dip. di Ingegneria, Università di Ferrara,
Via Saragat 2, 44100 Ferrara, Italy
elamma@ing.unife.it

Abstract. In this work, we propose an operational semantics based on a proof system for the consistent execution of tasks in a constrained multi-agent setting. The tasks represent services, and are associated with abstract specifications that express conditions on such services. The constraints, contained in the body of the agents, may include – but are not limited to – policies on provided services, and limitations about the use and allocation of bounded resources. The contribution of this work is twofold. Firstly, a formalism and an operational semantics is introduced, to express the way agents can coordinate their requests of services, and to verify that they do not collide with each other's conditions. Then, we prove the soundness and completeness of such operational semantics to be used to verify the correct execution of tasks.

1 Introduction

Multi-agent systems are generally conceived as societies of entities that are *sociable* and *autonomous*, and exhibit both a *reactive* and a *pro-active* behaviour, in that they should act in anticipation of future goals, while coping with the (generally unpredictable) changes of a dynamic environment. While on the one hand the agent metaphor is appealing and challenging, because of its ability to reflect current computing reality and complex organizations, on the other hand, if we want to actually implement such complex entities, we must tackle all aspects of this scenario.

According to [1], 'there are two major drawbacks associated with the very essence of an agent-based approach: (1) the patterns and the outcomes of the interaction are inherently unpredictable; and (2) predicting the behaviour of the overall system based on its constituent components is extremely difficult (sometimes impossible) because of the strong possibility of emerging behaviour.'

This could be a drawback in many cases. In fact, the need for making agents 'predictable', and – for most applications – as deterministic as possible, is indeed in contrast with the concept itself of autonomy. Nonetheless, it is reasonable to

believe that societies (of agents) can only exist as long as the individuals' autonomy does not represent a threat for the other individuals, and for the society in general. Therefore, the important counterpart of autonomy is represented by private constraints and public laws, that can make the agents, if not predictable, at least not colliding with each other needs and constraints.

In this paper, we tackle the problem of ensuring that the execution of tasks in a constrained multi-agents setting is consistent with respect to its constraints. We call *safe* this kind of execution. In the paper, the tasks represent services, and are associated with abstract specifications that express conditions on such services. The constraints, contained in the body of the agents, may include – but are not limited to – policies on provided services, and limitations about the use and allocation of bounded resources. The agent model presented in the paper is abstracted from our previous work on abductive logic agent [2, 3], and it is generalized in the paper.

We start by introducing a formalism and an operational semantics, to express the way agents can coordinate the requests of services, and to verify that they do not collide with each other's conditions, in dynamic, possibly adaptive systems. To this purpose, we introduce some operators, that could be mapped, in a concrete implementation, into part of a library of the language(s) used to encode the agents in the system. The operators provided by such library could be used to specify the need to obtain different services form diverse sources, in a *coordinated* fashion. For example, let A_0, A_1, and A_2 be agents in the system, and s_1 and s_2 services, respectively provided by A_1 and A_2. It could be the case that A_0 needs both s_1 from A_1 and s_2 from A_2, but in a coordinated way, that is, if either of the two cannot be provided, the other becomes useless. Moreover, we can imagine that due to system policy regulations, the two services require conflicting authorizations (e.g., there is a policy stating that an agent cannot be provided both service s_1 and s_2). A_0, as a client, could be unaware of any system policies. Therefore, if we want the system to comply with such constraints, we should give the agents an infrastructure to check the requests against any possible integrity threat that could arise from the execution of their tasks.

In this work, we propose a dynamic mechanism to ensure the system consistency, assuming that not all the hypotheses on the external world are necessarily known a priori by the system programmer(s). In particular we propose a protocol that is able to guarantee that a request or set of requests is allowed only if at execution time there is no constraint in the system that is violated.

Once a formalism is defined, we give such formalism an operational semantics, in the form of inference rules, in a sequent-style notation. Inference rules can be used as an abstract specification of an algorithm, that guarantees the correct execution of multiple, possibly conflicting, tasks. In this work we prove that, if the abstract machine underlying the agent code implements such operational semantics, all requests involving integrity constraints are allowed only if no constraint in the system is violated. In doing that, each agent performs a consistency check only within its own (private) set of constraints (as for instance, in an abductive logic programming setting). That is, we ensure the consistency

of the constraints in the whole system, without each agent needing to disclose its constraints to other agents.

The paper is structured as follows. In Section 2 we introduce the formalism used to express the way agents can coordinate the requests of services. In Section 3 we give such formalism an operational semantics, and define a proof system to check the consistency of service requests in a constrained multi-agent system. In Section 4 we present the soundness and completeness results of the system with respect with the safe execution of tasks. A brief discussion follows.

2 Formalism

In this section, we introduce the formalism expressing the way agents can coordinate the requests of services, while verifying that they do not collide with each other's conditions. In doing this, we draw inspiration from a logic language for constrained cooperative problem solving, LAILA [2], which is tailored to the particular setting of logic agents provided with hypothetical (abductive) reasoning capabilities [3]. Here, we abstract away from the specific reasoning capabilities and internal knowledge representation of the agents. We generalize the language to cope with service requirements and constraints in general. The only assumption that we make is that it is possible to describe the problem in question, in terms of constraints and conditions on the services, possibly expressed in the form of logic predicates.

Before we proceed with the syntax, we define what we intend by agent and by agent system.

Definition 1 (*Agent*)
An agent is a triple $< P, S, IC >$, representing a program P written in a language that allows the definition of *functions*, a set S of such functions called *services*, and a set IC of *integrity constraints*.
The services $s \in S$ are annotated with pairs $< s, \delta >$, that represent the agent's denotation, as defined below. We do not make assumptions on the syntax of the agent programs, services, and ICs, although we assume that the agents are provided with a mechanism, that we call 'local consistency check', that is able to determine if the constraints are violated. In the examples of this paper we use a syntax for the ICs and a semantics for the (local) consistency check that is derived from the abductive logic programming literature [4] (see *footnote* 1 in Section 3).

Definition 2 (*Agent denotation*)
Let A be an agent, s a service and δ a set of conditions. We call *denotation of agent* A ($Den(A)$) the following set:
$Den(A) \stackrel{def}{=} \{< s, \delta >: s$ is locally provided by A under the conditions δ and $\delta \cup IC_{\{A\}} \nvDash \bot\}$ (where $\bot$ stands for false). We assume that $Den(A)$ is sound and complete with respect to the notion of local computation.

Definition 3 (*Agent system*)
An *agent system* is represented as a finite set of atoms, each one standing of an agent of the system.

2.1 Syntax

We give the syntax of the operators in BNF. Let $\mathcal{V}$ be the vocabulary of the language used to write the agents' program P. We add to its vocabulary the set $\{ \& , \ ; \ , \ > \ , \downarrow\}$ of operators, where:

- $\downarrow$ is the *local execution* operator;
- $>$ is the *communication* operator;
- $\&$ is the *collaborative* coordination operator;
- ; is the *competitive* coordination operator.

Such operators are part of *expressions*, possibly enclosed in the agent programs, where the *local execution* operator has a maximum priority, followed by the *communication* operator; the *competitive* coordination operator has the minimum priority. As far as associativity, they are all left-associative. An expression is defined as follows:

$$
\begin{array}{lll}
\mathit{Expression} & ::= & \mathit{Formula} \ ; \ \mathit{Expression} \mid \mathit{Formula} \\
\mathit{Formula} & ::= & \mathit{SingleFormula} \ \& \ \mathit{Formula} \mid \\
 & & \mathit{SingleFormula} \\
\mathit{SingleFormula} & ::= & \mathit{Agent} \ > \ \mathit{SingleFormula} \mid \\
 & & \downarrow \mathit{Service} \\
\mathit{Agent} & ::= & \mathbf{A_0} \mid \mathbf{A_1} \mid \ \ldots \ \mid \mathbf{A_n}
\end{array}
$$

We label the agents by $\mathbf{A_0} \ \ldots \ \mathbf{A_n}$. *Service*, as introduces in Definition 1, is a function call in the language that embeds the communication, competitive and collaborative coordination operators; we will often label such services s_0, s_1, etc. Services are associated with *conditions*. Conditions are grouped into sets that in the operational semantics are usually labelled δ, δ', δ_1, etc. The communication operator can also be used to issue requests to 'local' services involving some local constraints, as it is shown by the following example.

Example 1. Let the following expression be enclosed in an agent program, say A_0:

$$\texttt{A0} \ > \ \downarrow \texttt{s1} \ ; \ \texttt{A1} \ > \ \downarrow \texttt{s2}$$

It means that A_0 must either perform a local service, s_1, *or* ask agent A_1 for service s_2. Should both service be available, possibly under different conditions, the system will select non-deterministically only one of them.

Let us consider, now, the following expression, also embodied in agent A_0's program, representing a collaborative request composed of two different subrequests, whose conditions must be coherent with one another:

$$\texttt{A1} \ > \ \downarrow \texttt{s3} \ \& \ \texttt{A2} \ > \ \downarrow \texttt{s4}$$

Agent A_0 asks agent A_1 for the service s_3 and A_2 to for the service s_4; after both A_1 and A_2 reply to A_0 by giving each a set of conditions for the requested service, the result is obtained by merging such sets of conditions in a unique consistent set, with respect to the bunch of agents (A_0, A_1, and A_2) dynamically considered along the computation. Such set could be bigger than the union of the parts, due to additional constraints that are fired in the cross checking phase. □

3 Operational Semantics

We have so far described the operators' syntax. In this section, we give them an operational semantics, in the form of inference rules of a proof system. Such a system can be used as an abstract specification of an algorithm that guarantees the correct execution of multiple, possibly conflicting, tasks. The inference rules could be implemented to extend the operating system or virtual machine that supports the execution of agents.

For the sake of simplicity, the rules we define refer to the case of propositional expressions, not including variables, but they can be easily generalized. In the following, $\mathcal{U}$ is the universe of agents, and $A_0, \ldots, A_n$ denote agents in the system, i.e., $\{A_0, \ldots, A_n\} \subset \mathcal{U}$. The entailment symbol adopted is $\vdash_\delta^B$, whose superscript denotes the bunch B of agents involved in the computation, and whose subscript the set of conditions δ, both of which are output parameters of the derivation process. Given $A \vdash_\delta^B F$, F is the *formula* to prove (a *formula* is in general an expression), and A is the agent asking for its derivation. The agent's code embodies both the integrity constraints IC and the program P of Definition 1: for the sake of simplicity, and with abuse of notation, we will write the name of the agent instead of its code. Therefore, $A \vdash_\delta^B F$ means that the formula F is proven within A's code, producing as an output a set of hypotheses δ and a bunch B. Finally, we will adopt the following notation for the consistency derivation:

$$B \overset{cons}{\vdash}_\delta \Delta$$

where Δ is a set of conditions on a (set of) service(s), $\overset{cons}{\vdash}$ denotes the "consistency check" of the conjunction of all atoms in Δ, with respect to the integrity constraints of all agents in bunch B; in particular, let $S \subseteq \{0, \ldots, n\}$. Given a bunch of agents $B \subseteq \{A_i | i \in S\}$, let IC_i denote the set of integrity constraints of agent A_i. The declarative semantics of $B \overset{cons}{\vdash}_\delta \Delta$ is defined as follows:

$$\begin{cases} \delta \cup IC_B \not\models \bot \\ \Delta \subseteq \delta \end{cases}$$

where $IC_B = \bigcup_{A_j \in B} IC_j$. That is, δ satisfies[1] all the integrity constraints of agents in B. In Section 4, Lemma 1 formally proves the equivalence between the operational notion of consistency in a bunch, and this declarative counterpart. We achieve in that a separation among the agents knowledge bases, ensuring the absence of conflicts between δ and each of their integrity constraints. Moreover, we would like to notice that, while in describing the declarative semantics we refer – for the sake of simplicity – to the union of the agents' integrity constraints, the actual implementation of the system does not require at all that the agents share

[1] There are several notions of constraint satisfaction. The one adopted in this work refers to a formalization of integrity constraints used in abductive logic programming of the kind: $\bot \leftarrow body$, where *body* is a set of conditions. $\delta \cup IC \not\models \bot$ means that $\forall ic \in IC, body(ic) \not\subseteq \delta$, i.e., δ does not make $body(ic)$ true. We also rely on the hypothesis, from now on, that $\forall x \in ic, \forall ic \in IC$, x is a *condition*.

any knowledge in that respect (see the consistency check operational semantics further on for details). Knowledge bases and therefore integrity constraints are kept separate, as we would expect in a multi-agent setting.

We also define a concept of 'local consistency', and adopt the following notation: $A \overset{l-cons}{\rightsquigarrow}_{\delta} \Delta$, where Δ is a set of conditions. By 'local consistency' we mean that Δ is consistent with agent A's integrity constraints, IC, returning a (possibly enlarged) set of conditions δ:

$$\begin{cases} \delta \cup IC \not\models \bot \\ \Delta \subseteq \delta \end{cases}$$

Beside understanding the role that integrity constraints play to ensure the correct behaviour of a system, as briefly exemplified above, it is also important to understand the consequences that it brings to extend the concept of consistency to the multi-agent case. The main point is that while the union of two given sets of hypotheses (conditions) could be enough to ensure the consistency of two separate sets of integrity constraints, it may become insufficient if we want to ensure the consistency of the union of the integrity constraints, as it is shown by the following example.

Example 2. Let us consider two different sets of constraints, $IC_1 = \{\leftarrow a, c\}$ and $IC_2 = \{\leftarrow b, not\ c\}$, a, b and c being possible conditions on a service. If we check the consistency of the set of hypotheses $\{a, b\}$ against either of IC_1 and IC_2 separately, we obtain no inconsistency. In fact, as a result of such a check we could obtain (assuming for instance that we are adopting a proof-procedure such as that defined in [5]) two separate sets of conditions: $\{a, b, not\ c\}$ and $\{a, b, c\}$. But, if we join the constraints in a unique set $IC_3 = IC_1 \cup IC_2$, it is understood that $\{a, b\}$ cannot be the case, since we would have both c and its negation in the (enlarged) set of hypotheses that includes a and b. Therefore, it is not enough to check $\{a, b\}$ against, say, IC_1, and then $\{a, b\}$ again against the other constraint IC_2, but we need instead a different mechanism to verify the consistency of an enlarged set of hypotheses $\{a, b, not\ c\}$ resulting from the first check, against the other constraint IC_2, thus detecting the inconsistency. □

For this reason, in a distributed setting such as that of multi-agents, the individuals must communicate to each other the possible sources of inconsistency, like it was c in Example 2. The sets of conditions may therefore grow bigger while the consistency check is made, and the enlarged set of conditions checked for consistency again by all the involved agents, until fixpoint is reached.

Let us describe now the operational behavior of the system, by giving, in a sequent-style notation, the inference rules of the operators. The *communication* operator is used to request a service to an agent, which could possibly be the requester itself.

Definition 4 (*Communication formula*)

$$\frac{Y \vdash^{B}_{\delta_1} F \quad \wedge \quad B \cup X \overset{cons}{\vdash}_{\delta} \delta_1}{X \vdash^{B \cup X}_{\delta} Y > F}$$

where X and Y may either be two distinct agents, or the same one, $X, Y \in \{A_0, ..., A_n\}$. F is a *single formula* that contains properly combined service re-

quests. If F is simply a local execution such as $\downarrow s$, X asks Y to locally perform the service s. In general, however, F may include several nested communication formulas, involving several agents possibly enclosing some private integrity constraints. Therefore, the communication formula $Y > F$ used by X to ask Y to perform the services F requires a consistency check within the bunch composed by X and by those agents that participated in F. Clearly, this bunch will at least include X and Y. This makes the communication operator more than just a simple call to another agent's knowledge base and reasoning module: the answer possibly returned back from Y to X is put in the form of a set of conditions, δ_1, and needs to be agreed upon and possibly modified (enlarged) by the whole bunch of agents, through a consistency step.

There could be the case that an agent requires a service that can be performed in several ways, but it does not really matter which one is selected for execution. This is why we introduce the *competitive* operator, that introduces a degree of non-determinism, since in the expression $f \ ; \ F$ there is no precedence relationship between the two operands, f and F.

Definition 5 (*Competitive formula*)

$$\frac{A \vdash^{B}_{\delta} f}{A \vdash^{B}_{\delta} f;F}$$

$$\frac{A \vdash^{B}_{\delta} F}{A \vdash^{B}_{\delta} f;F}$$

where A is an agent, $A \in \{A_0, ..., A_n\}$, f is a *formula*, F is an *expression* and B is a bunch of agents, $B \subseteq \{A_0, ..., A_n\}$. The competitive formula results in a non-deterministic choice of one inference rule between the two listed above.

The third operator that we define is the *collaborative* coordination operator. It is used to indicate that two services are both required, and must be consistent with each other. From the inference rule below, we see that the collaborative coordination requires a consistency step involving all agents that contribute to provide the service.

Definition 6 (*Collaborative formula*)

$$\frac{A \vdash^{B'}_{\delta'} f \ \ \wedge \ \ A \vdash^{B''}_{\delta''} F \ \ \wedge \ \ B' \cup B'' \overset{cons}{\vdash}_{\delta} \ \ \delta' \cup \delta''}{A \vdash^{B' \cup B''}_{\delta} f \& F}$$

where A is an agent, $A \in \{A_0, ..., A_n\}$, f is a *formula*, F is an *expression* and B, B' and B'' are three possibly distinct bunches of agents, $B, B', B'' \subseteq \{A_0, ..., A_n\}$.

Finally, we describe the semantics of the consistency check through the following inference rules:

Definition 7 (*Consistency check*)

$$\frac{\forall A_i \in B \ A_i \overset{l-cons}{\rightsquigarrow}_{\delta_i} \delta \ \wedge \ \delta \subset \bigcup_{A_j \in B} \delta_j \ \wedge \ B \overset{cons}{\vdash}_{\delta'} \bigcup_{A_i \in B} \delta_i}{B \overset{cons}{\vdash}_{\delta'} \delta}$$

$$\frac{\forall A_i \in B \quad A_i \overset{l-cons}{\rightsquigarrow}_{\delta} \delta}{B \overset{cons}{\vdash}_{\delta} \delta}$$

where A_i, A_j are agents, $A_i, A_j \in \{A_0, ..., A_n\}$, h is a literal, G is an *expression* and B is a bunch of agents, $B \subseteq \{A_0, ..., A_n\}$.

Therefore, the consistency check of δ is first performed individually by every single agent via an abductive derivation, which could result in an enlargement of δ (in particular, this happens if exists a δ_i such that $\delta \subset \delta_i$). In case no agent raises new hypotheses, i.e., for all $A_i \in B$, $\delta \equiv \delta_i$, then $\delta \subset \bigcup_{A_j \in B} \delta_j$ is not true any more, fixpoint is reached, and the consistency check terminates; otherwise the union of the δ_i has to be checked again within the bunch.

Definition 8 (*Local computation formula*)

$$\frac{local\_computation(A, s, \delta') \quad \wedge \quad A \overset{l-cons}{\rightsquigarrow}_{\delta} \quad \delta'}{A \vdash_{\delta}^{\{A\}} \quad \downarrow \; s}$$

where A is an agent, $A \in \{A_0, ..., A_n\}$, s is a service, and $\{A\}$ is the bunch composed by the only agent A. *local_computation* is a relation evaluated by agent A, that returns the conditions δ' associated with a certain service s, that A is asked to execute. δ' must be checked for consistency with the other constraints of A, which could result in an enlargement of δ' leading to a $\delta \supseteq \delta'$. The local computation formula, characterized by the $\downarrow$ operator, is intended not as an actual execution of a service, but rather as a preliminary step that anticipates the execution of a service.

Finally, we can introduce the definition of a successful top-down derivation.

Definition 9 (*Successful top-down derivation*)
Let A be an agent and F an expression that possibly describes the request for a service. A *successful top-down derivation* for F in A, which returns a set of conditions δ and the bunch of agents B dynamically involved in the service, can be traced in terms of a (finite) tree such that:

- The root node is labeled by $A \vdash_{\delta}^{B} F$;
- The internal nodes are derived by using, backwards, the inference rules defined above;
- All the leaves are labeled by the empty formula, or represent a successful local computation.

For instance, we have seen how a collaborative formula $A \vdash_{\delta}^{B} f \;\&\; F$ develops into three branches (sub-trees), one for proving f, another one for F and a last one for the consistency check. Similarly, the communication formula produces two sub-trees, while the down reflection and the competitive formulas produce only one branch, and so on.

4 Soundness and Completeness

In this section, we prove the soundness and completeness properties of the proof system, which guarantees the safe execution of tasks. To this purpose, we start by introducing two basic properties we rely upon, Properties 1 and 2, and concerning

the soundness and completeness of local consistency derivations. We assume that they hold for all the local consistency computations.[2]

Property 1 (*Soundness of the local consistency derivation*)
Given an agent A_i and a set of conditions δ,

$$A_i \overset{l-cons}{\rightsquigarrow}_{\delta_i} \delta \Rightarrow \delta_i \cup IC_i \not\models \bot$$

where IC_i represents the integrity A_i's constraints. This means that, if there exists a local consistency derivation for δ in A_i that returns a set of conditions δ_i, then δ_i is consistent with the integrity constraints IC_i of A_i itself.

Property 2 (*Completeness of the local consistency derivation*)
Given an agent A_i and a set of conditions δ,

$$\forall \delta : \delta \cup IC_i \not\models \bot \Rightarrow \exists \delta_i : A_i \overset{l-cons}{\rightsquigarrow}_{\delta_i} \delta \wedge \delta_i \supseteq \delta$$

where IC_i represents the integrity A_i's constraints. This means that, if a set of conditions δ is consistent with the integrity constraints of an agent A_i, then there exists a local consistency computation for δ in A_i itself, which possibly adds some new conditions to δ, returning $\delta_i \supseteq \delta$. For instance, the proof-procedure of [5] is such that an initial set δ of conditions (abducible predicates) gets enlarged to falsify all the bodies of constraints that contain some predicates that are also contained in δ (see Example 2).

In the following, we prove two lemmas about the soundness and completeness of the consistency check in a bunch of agents of Definition 7, that we will use to prove Theorem 1 and 2 below.

Lemma 1 (*Soundness of the consistency check*) Given a bunch of agents B and a set of conditions δ,

$$B \overset{cons}{\vdash}_{\delta'} \delta \Rightarrow \delta' \cup IC_B \not\models \bot \wedge \delta' \supseteq \delta$$

where IC_B represents the union of the ICs of all the agents in B.

Proof The proof is given by separately proving the two conjuncts in the right hand side of the implication. In particular, we prove inductively that $\delta' \supseteq \delta$. If $B \overset{cons}{\vdash}_{\delta'} \delta$ is 1-length computation, then by the second inference rule in Def. 7, $\delta' = \delta$. If $B \overset{cons}{\vdash}_{\delta'} \delta$ is n-length, we assume the lemma holds for n-1. Then, by the first inference rule in Def. 7, $\delta' \supseteq \bigcup_{A_i \in B} \delta_i$. Since $\delta \subset \bigcup_{A_i \in B} \delta_i$, then follows: $\delta' \supset \delta$.

We prove inductively that $\delta' \cup IC_B \not\models \bot$. For one-length computation, $B \overset{cons}{\vdash}_{\delta} \delta$ is equivalent to $\forall A_i \in B\ A_i \overset{l-cons}{\rightsquigarrow}_{\delta} \delta$ by the second inference rule in Def. 7. By the soundness of the local consistency derivation (Prop. 1) follows that $\forall A_i \in B, \delta \cup IC_i \not\models \bot$. For an n-length computation, $B \overset{cons}{\vdash}_{\delta'} \delta$ by the first inference rule of Def. 7 is equivalent to $\forall A_i \in B, A_i \overset{l-cons}{\rightsquigarrow}_{\delta_i} \delta \wedge \delta \subset \bigcup_{A_j \in B} \delta_j \wedge B \overset{cons}{\vdash}_{\delta'} \bigcup_{A_i \in B} \delta_i$.

[2] Although our system does not rely upon a logic system, there exist in literature some proof procedures that implement a local consistency derivation, such as [5, 6], some of which are proved correct and complete.

By the inductive hypothesis, $B \overset{cons}{\vdash}_{\delta'} \bigcup_{A_i \in B} \delta_i$ implies $\delta' \cup IC_B \not\models \bot$. ■

It is worth stressing that, when proving $B \overset{cons}{\vdash}_{\delta} \delta \equiv \forall A_i \in B\ A_i \overset{l-cons}{\rightsquigarrow}_{\delta} \delta$, we refer to the case in which the local consistency derivation does not affect δ. In particular, with respect to Example 2, if we use [5], which indeed affects δ, neither $\{a, b, c\}$ is consistent with IC_1 nor $\{a, b, not\ c\}$ is consistent with IC_2, and of course both are inconsistent with $\{IC_1, IC_2\}$. If we use a weaker notion of consistency, instead, $\{a, b\}$ could be consistent with both IC_1, IC_2 and $\{IC_1, IC_2\}$.

Lemma 2 (*Completeness of the consistency check*) Given a bunch of agents B,

$$\forall \delta' : \delta' \cup IC_B \not\models \bot \Rightarrow \exists \delta \supseteq \delta' : B \overset{cons}{\vdash}_{\delta} \delta'$$

Proof $\delta' \cup IC_B \not\models \bot$ implies $\forall ic \in IC_B : \delta' \cup ic \not\models \bot$. This implies, by Prop. 2, that $\forall A_i \in B\ \forall ic_i \in IC_i \exists \delta_i : A_i \overset{l-cons}{\rightsquigarrow}_{\delta_i} \delta' \wedge \delta_i \supseteq \delta'$. By induction on the length of the computation, for a 1-length computation, by the second inference rule of Def. 7, the statement is trivially proven. For an n-length computation, by the first inference rule of Def. 7, and by the inductive hypothesis the statement is proven as well. ■

In the following, we state the soundness theorem for our proof system. The theorem states that, for any successful top-down derivation, under the assumption of soundness of the local consistency derivations (Prop. 1), the computed conditions satisfy all the integrity constraints of the bunch of agents dynamically involved in the service in question. This implements in a sense the introduced notion of *safe execution*.

Theorem 1 (*Soundness*) Let A be an agent and F an expression that possibly describes the request for a service. If there exists a *successful top-down derivation* for F in A, which returns a set of conditions δ and a bunch of agents B, then the computed set δ is consistent with the integrity constraints of bunch B. Formally:

$$A \vdash_{\delta}^{B} F \Rightarrow \delta \cup IC_B \not\models \bot$$

Proof The proof is given by induction on the length of the derivation. For a 1-length derivation, F has the form $\downarrow s$, where s is a service. By Def. 8, $A \vdash_{\delta}^{\{A\}} \downarrow s$, which implies that there exists in A a local computation for s returning δ' and $A \overset{l-cons}{\rightsquigarrow}_{\delta} \delta'$. By Prop. 1, it follows that $\delta \cup IC_{\{A\}} \not\models \bot$.

For any n-length computation, we consider several cases:

(i) F is in the form $Y > f$. Then, $A \vdash_{\delta}^{B \cup \{A\}} f$ by Def. 4 implies $Y \vdash_{\delta'}^{B} f$ and $B \cup \{A\} \overset{cons}{\vdash}_{\delta} \delta'$. By the inductive hypothesis, $\delta' \cup IC_B \not\models \bot$, and by Lemma 1, $\delta \cup IC_{B \cup \{A\}} \not\models \bot$ and $\delta \supseteq \delta'$. Then, it follows that $\delta' \cup IC_{B \cup \{A\}} \not\models \bot$.

(ii) When F is in the form $f; F'$, the statement is trivially proven.

(iii) F is in the form $f \& F'$. Then, $A \vdash_{\delta}^{B' \cup B''}$ by Def. 6 implies $A \vdash_{\delta'}^{B'} f \ \wedge\ A \vdash_{\delta''}^{B''} F' \ \wedge\ B' \cup B'' \overset{cons}{\vdash}_{\delta} \delta' \cup \delta''$. The first two conjuncts imply, by the inductive hypothesis, $\delta' \cup IC_{B'} \not\models \bot \wedge \delta'' \cup IC_{B''} \not\models \bot$. The third conjunct implies, by

Lemma 1, $\delta \cup IC_{B' \cup B''} \not\models \bot \wedge \delta \supseteq \delta' \cup \delta''$ ■

In the rest of the section, we also state a completeness theorem for a subset of the possible expressions occurring in a top-level request. The completeness theorem ensures that, if a set of conditions δ can be found, satisfying the integrity constraints of a bunch of agents, and it belongs to the meaning of some agent, as defined below, then there exists an expression F and a successful top-down derivation for F in A leading to the bunch B of agents, and to conditions δ.

Theorem 2 (*Completeness*) Let B be a bunch of n agents, s a service and δ a set of conditions.

$\forall B = \{A_1, \ldots, A_n\}, \forall < s, \delta > \in Den(A_i), A_i \in B : \delta \cup IC_B \not\models \bot \Rightarrow \exists F, A \in B : A \vdash^B_{\hat{\delta}} F \wedge \hat{\delta} \supseteq \delta$ where F is an expression occurring in a top-level request.

Proof The proof is given in the case of a communication formula F expressed in the form: $F = A_{1i} > A_{2i} > \ldots > A_{ki} > A_i >\downarrow s$, where $k \geq n-2$, and $\{A_{ji} : j = i..k\} \cup \{A\} \cup \{A_i\} = B$.

By Def. 2, being $< s, \delta > \in Den(A_i)$, it follows that *(i)* $local\_computation(A_i, s, \delta) \wedge \delta \cup IC_{A_i} \not\models \bot$. By Prop. 2, it follows that *(ii)* $\exists \delta \supseteq \delta'$ such that $A \overset{l-cons}{\rightsquigarrow}_{\delta'} \delta$. Therefore, by Def. 8, *(i)*, and *(ii)*, it follows that *(iii)* $A_i \vdash^{\{A_i\}}_{\delta'} \downarrow s, \delta' \supseteq \delta$.

Now, being $\delta \cup IC_B \not\models \bot$, by Lemma 2 it follows that $B \overset{cons}{\vdash}_{\delta''} \delta, \delta'' \supseteq \delta$. Now, by Lemma 1, $\delta'' \cup IC_B \not\models \bot$, and, again by Lemma 2, $B \overset{cons}{\vdash}_{\tilde{\delta}} \delta''$. From Def. 7 it is possible to prove that $\delta' \subseteq \delta''$. Therefore, $\{A_i\} \cup \{A_{ki}\} \overset{cons}{\vdash}_{\delta'''} \delta'$. From this, Def. 4 and *(iii)*, it follows that $\{A_i, A_{ki}\} \overset{cons}{\vdash}_{\delta'''} A_i >\downarrow s$, and $\delta''' \supseteq \delta'$.

Now, by iteratively applying Def. 4, we can prove that $\exists \hat{\delta} : A \vdash^B_{\hat{\delta}} A_{1i} > A_{2i} > \ldots > A_{ki} > A_i >\downarrow s \wedge \hat{\delta} \supseteq \delta$ ■

In a similar way it is possible to give the proofs referring to other kinds of formula.

5 Conclusion

In this work, we tackled the problem of 'safe' task execution in multi-agent systems. The tasks are services that agents can request / provide, to other agents. The feasibility of such services may be constrained by "policies", i.e., integrity constraints embodied in the agents themselves. We defined a formalism to express the combination of service requests, and an operational semantics as a proof system that can be used to extend an operating system or virtual machine supporting the agents' execution, to ensure a safe task execution. We presented two results, with respect to the semantics. The first one is a soundness result, which implements the notion of safe execution, ensuring that no constraint is violated by a service request which is possibly composed of several services, asked to several agents. In other words, if the operational semantics allows the execution of a certain service, then it is safe to execute it. The second one is a completeness result, which states that if such service could be safely executed, then its execution is allowed by the operational semantics. This result has been

proven for a particular service composition pattern, but it is possible to extend the proof for other more general patterns.

In related work, a considerable effort has been dedicated towards characterizing agents as logical systems, providing several operational semantics to the logical specifications of agent programming languages [7–9]. Nevertheless, to the best of our knowledge, our work is among the few ones that give a sound and complete semantics to a formalism that allows to express the agents' execution, in a way that is independent of the notion of consistency adopted and that at the same time ensures that the system's constraints are not violated. In that, this paper represents a progress from our previous work, which presumes a system of logic based agents and is tailored to the case of abductively reasoning agents.

In the future, we intend to study and formalise the relationship between composition operators of knowledge bases and programs of different agents, and their corresponding operational conunterpart, given as proof systems for the multi-agent case. We also intend to provide a concrete implementation of the operators here introduced, and write a library that extends a virtual machine.

Acknowledgements

We would like to thank the anonymous referees for their precious comments. This work was supported by the SOCS project, funded by the CEC, contract IST-2001-32530.

References

1. Jennings, N.R.: On agent-based software engineering. Artificial Intelligence **117** (2000) 277–296
2. Ciampolini, A., Lamma, E., Mello, P., Torroni, P.: LAILA: A language for coordinating abductive reasoning among logic agents. Computer Languages **27** (2002)
3. Ciampolini, A., Lamma, E., Mello, P., Toni, F., Torroni, P.: Co-operation and competition in *ALIAS*: a logic framework for agents that negotiate. In: Computational Logic and Multi-Agency. Special Issue of the Annals of Mathematics and Artificial Intelligence, Baltzer Science Pub. (to appear)
4. Kakas, A.C., Kowalski, R.A., Toni, F.: The role of abduction in logic programming. Handbook of Logic in AI and Logic Programming **5** (1998) 235–324
5. Kakas, A.C., Mancarella, P.: Generalized stable models: a semantics for abduction. In: Proc. 9th ECAI, Pitman Pub. (1990)
6. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming (1997)
7. Lesprance, Y., Levesque, H.J., Lin, F., Marcu, D., Reiter, R., Scherl, R.B.: Foundations of a logical approach to agent programming. In: Intelligent Agents II – Proc. ATAL'95. LNCS, Springer-Verlag (1996)
8. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Agents Breaking Away, Proc. MAAMAW'97. LNCS 1038, Springer (1996) 42–55
9. Hindriks, K., Boer, F.D., van der Hoek, W., Meyer, J.J.: Formal semantics for an abstract agent programming language. In: Intelligent Agents IV, Proc. ATAL'97. LNCS 1365, Springer-Verlag (1998) 215–229