ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
DEIS – DIPARTIMENTO DI ELETTRONICA, INFORMATICA E SISTEMISTICA

Specification, Execution and Verification of Interaction Protocols

An Approach based on Computational Logic

Settore Disciplinare: ING-INF05
Ciclo XIX

Research Supervisor:
Chiar.ma Prof. Ing. **Paola Mello**

Candidate:
Federico Chesani

Ph.D. School Director
Chiar.mo Prof. Ing. **Paolo Bassi**

Esame Finale 2007

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
DEIS – DIPARTIMENTO DI ELETTRONICA, INFORMATICA E SISTEMISTICA

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**Specification, Execution and Verification of Interaction Protocols**" by **Federico Chesani** in partial fulfillment of the requirements for the degree of **Dottore di Ricerca**.

Dated: March 2007

Research Supervisor: _____
Prof. Ing. Paola Mello

Ph.D. School Director: _____
Prof. Ing. Paolo Bassi

Examing Committee: _____
Prof. Ing. Anna Ciampolini

_____
Prof. Ing. Letizia Leonardi

_____
Prof. Ing. Cesare Stefanelli

Abstract

Interaction protocols establish how different computational entities can interact with each other. The interaction can be finalized to the exchange of data, as in *communication protocols*, or can be oriented to achieve some result, as in *application protocols*. Moreover, with the increasing complexity of modern distributed systems, protocols are used also to control such a complexity, and to ensure that the system as a whole evolves with certain features. However, the extensive use of protocols has raised some issues, from the language for specifying them to the several verification aspects.

Computational Logic provides models, languages and tools that can be effectively adopted to address such issues: its declarative nature can be exploited for a protocol specification language, while its operational counterpart can be used to reason upon such specifications.

In this thesis we propose a proof-theoretic framework, called $\mathcal{S}$CIFF, together with its extensions. $\mathcal{S}$CIFF is based on Abductive Logic Programming, and provides a formal specification language with a clear declarative semantics (based on abduction). The operational counterpart is given by a proof procedure, that allows to reason upon the specifications and to test the conformance of given interactions w.r.t. a defined protocol. Moreover, by suitably adapting the $\mathcal{S}$CIFF Framework, we propose solutions for addressing (1) the *protocol properties verification* (g-$\mathcal{S}$CIFF Framework), and (2) the *a-priori conformance verification* of peers w.r.t. the given protocol (A^{l}LoWS Framework). We introduce also an agent based architecture, the $\mathcal{S}$CIFF Agent Platform, where the same protocol specification can be used to program and to ease the implementation task of the interacting peers.

Dedicated to Elena

Contents

Acknowledgements

I wish to thank all the people who helped and supported me during my Ph.D. studies. First of all I would like to thank my supervisor Paola Mello and my wife Elena.

Prof. Mello helped me in all the areas of my research activity; she guided me during my studies, and she helped me with all the problems I encountered in the last three years.

My wonderful wife Elena supported me all the time in the last years, always encouraging me and trusting in my capabilities. She also gave me Francesco, my fantastic son, who taught me how to work during the night...

All the results I obtained during my research activity would not have been possible without the precious help of all the people in my research group here in Bologna, and of the people in Ferrara. I wish to explicitly thank Anna Ciampolini, Alessio Guerri, Michela Milano, Marco Montali, Paolo Torroni, Zeynep Kiziltan, and Marco Alberti, Evelina Lamma, Marco Gavanelli, Fabrizio Riguzzi, Sergio Storari. A special thank also to Matteo Baldoni, for his precious ideas and suggestions.

A particular thank also to the other Ph.D. students that shared with me this period, and in particular to Luca Foschini and Eugenio Magistretti.

All the work presented in this thesis has been done in strict collaboration with all my colleagues in Bologna and in Ferrara. It has been almost impossible for me to isolate my own contribute: most of the credit for the obtained result should be given to all the people who worked with me.

Links to some resources on the web

- The $\mathcal{S}$CIFF Proof Procedure can be downloaded at
  `http://lia.deis.unibo.it/research/sciff/`
- The *SOCS-SI* tool can be downloaded at
  `http://www.lia.deis.unibo.it/research/socs_si/socs_si.shtml`
- A demo description of the *SOCS-SI* tool can be found at
  `http://lia.deis.unibo.it/research/socs/aamas2004demo/`
- A library with several protocols specifications can be found at
  `http://wikiai.deis.unibo.it/index.php?title=SOCS_Protocol_Repository`
  `http://edu59.deis.unibo.it:8079/SOCSProtocolsRepository/jsp/index.jsp`
- The complete bibliography of the author can be found at
  `http://lia.deis.unibo.it/~fc/`

Chapter 1

Introduction

> *A protocol specifies the "rules of encounter" governing a dialogue between two or more communicating agents.*
>
> Rosenschein and Zlotkin, [126]

Protocols have been used since the beginning of the Computer Science discipline to rule the way different entities interact with each other. Initially, the most common type of protocols where the *communication protocols*: they were strict and mandatory rules that defined how the exchange of data should happen between two peers. The goal of such type of protocols is to allow the exchange of data while guaranteeing certain properties related to the exchange process itself (e.g., the detection of transmission errors, or data losses). The peers involved in the communication could be homogeneous in software and hardware, as well as heterogeneous systems: in the latter case, protocols had also the role of solving incompatibilities due to the heterogeneity.

More recently, protocols have been used also at a higher abstraction level (w.r.t. communication protocols), as a way for achieving tasks more complex than the exchange of data. *Application protocols* have been widely used in almost every computer-related sector, as a mean for ruling the interaction between complex peers, like for example the Post Office Protocol for the email sending/retrieving. Also protocols of this class are still defined in terms of strict rules that the peers must respect in order to complete the interaction.

In the Multi Agent System paradigm, protocols have been object of ulterior interest: if the agent paradigm (and the Multi Agent Systems paradigm, MAS) have been used as good method for modeling systems of increasing complexity, protocols (*interaction protocols*) have been used as a tool for controlling such complexity. Nowadays, interaction protocols are the most used mechanism (and probably the most "immediate" one) to achieve collaboration between distributed entities, and to ensure that complex systems does indeed exhibit certain characteristics.
Note that the word entity is no referred only to the concept of agent, but also to a broader class of software components whose task is to perform or to provide some functionality. E.g., Service Oriented Architectures (SOA) propose a solution addressing the interaction issues, and sketch several different proposals for regimenting the whole set of interactions, thus defining a protocol (orchestration, choreography approaches).

Hence, protocols are used to serve many different purposes; to cite some:

1. to give the possibility to the peers of interacting with each other and, by this way, achieving a certain goal; e.g., the communication protocols used in the Internet;

2. to regulate (in a normative sense) the interactions that can happen between different peers. E.g., in a typical e-commerce scenario, the agents enjoy some freedom, in the sense that they can freely choose to perform several different actions. An agent could decide to agree to a deal or to reject it; however, the act of performing some actions could limit such a freedom. A protocol could state that once a deal has been reached between two agents, then they should fulfill the obligations that are enlisted in the deal;

3. to ensure that all the interactions that are compliant with the protocol, enjoy some properties (peculiar to the protocol domain itself). E.g., a protocol for securely exchanging data between two peers aims to guarantee that anyone that is not the intended recipient cannot access the data. A e-commerce protocol could aim to guarantee the "good atomicity" property.

When speaking of protocols, two major aspects must be considered:

1. How to specify them?

2. How to verify them?

In the following Sections 1.1 and 1.2 we will try to introduce the reader to the problem of specifying the protocols, and to the problem of verifying them.

1.1 Specification Issues

The problem of specifying a protocol can be re-formulated as the problem of finding a language that indeed allows to specify such a protocol. Of course, such a language should be:

- *expressive* enough to "capture" all the protocol peculiarities;
- at the same time, *general* as much as possible to get reused in several application domains;
- *simple* enough to be used by a protocol developer, and to be understood by other developers;
- not *ambiguous*;
- possibly based on a *declarative* approach, with a clear and formal semantics;

A feature that has been at the center of recently research ativities is the *executability*. Protocol specifications should be machine-understandable and support it in some degree. With the *executability* term we mean the possibility of using such a specification to directly implement one or more peers involved in the protocol. Such a possibility mainly depends on the used language, and in some part, it also depends on how the protocol has been defined by the developer. By "directly implement", we mean the possibility of using the protocol specification as a base for developing the peers that uses such a protocol. Of course, implementing such peers would require some efforts. Intuitively, such effort could be intended as a measure of the executability of the specification language.

To make clearer such concepts, let us present some examples. The Transfer Control Protocol (TCP, [124]) specifies the rules that two peers should follow in order to establish a connection and exchange (transfer) data in a controlled way. The specification of TCP describes the rules by means of a Finite State Automata, whose transitions and states are (unfortunately) expressed using natural language. Hence

the software developer is left alone, to read the specification, to interpret it and to code the software in the way he consider to be the more appropriate. Such a development process is prone to the introduction of several bugs, mainly due to the interpretation process applied by the developer to the natural language description.

To cite another example, the Business Process Execution Language (BPEL, [20]) is used to specify business process as the set of interactions between one main process (the *orchestrator*) and other process (the *orchestrated processes*). A BPEL specification is composed of the *Abstract Process* specification and the *Executable Process* specification. While the Executable Process specification defines how the orchestrator should treat the data and other low-level details, the Abstract Process specification enlists all the interactions that should happen, in the specified order, between the orchestrator and the other peers. The conjunction of both can be provided as a input to a BPEL engine, that will execute the process as specified. Hence an Abstract Process specification can serve both the purposes of being a description of the interaction rules, and of being (a part of) an executable prototype.

1.2 Verification Issues

Guerin and Pitt [89] distinguish three possible types of verification, depending on the available information about the protocol players:

Type 1: verify that an agent will always comply. This type of verification can be performed at design time: given a representation of the agent, by means of some proof technique (such as *model checking* [111]) it is possible to prove that the agent will always exhibit the desired behaviour. Unfortunately, this type of verification make the assumption that the peer' internals are accessible.

Type 2: verify compliance by observation. Its goal is to check that the *actual* peer behaviour being observed is compliant to some specification. It does not require any knowledge about the internals, but only the observability of the peer behaviour. Since it is based on the observation, this type of verification can be performed at runtime (or possibly later on some logs).

This type of verification is of the uttermost importance in real systems, where the heterogeneity and the complexity is such that protocols must allow some freedom degrees in order to be effective enough in ruling the interactions: too much strict rules would risk to be useless (see, at this purpose, [44]). In more open scenarios then it becomes of the utmost importance being able to separate "good" interactions from the not compliant ones.

Type 3: verify protocol properties. This type of verification instead can be performed at design time, and aims to prove that some property will hold for all the interactions that correctly follow the protocol (i.e., they respect the protocol rules). This type of verification is of a crucial importance: with the rasing complexity of the protocols, it is harder (if not impossible) to manually verify that a protocol does indeed guarantee a certain property. Protocol Specification Languages should offer (or at least support) tools for expressing such properties, as well as for verifying that such properties are entailed by the protocol. I.e., automatic tools are needed in order to prove that if an interaction is compliant w.r.t. a protocol specification, then the property is true for that interaction.

1.3 Advocating the use of Computational Logic

We consider the interacting peers as autonomous computational entities, autonomous in the sense that their inner activity is not externally controlled. They have their own knowledge, capabilities, resources, objectives and rules of behavior. Each peer typically has only a partial, incomplete and possibly inaccurate view of the environment and of the other peers, and it might have inadequate resources or capabilities to achieve its objectives.

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 particular, at the interaction 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 complex systems composed of many heterogenous interacting peers, based on a formalism which is declarative and easily understandable by the user;

(ii) in the possibility to statically analyze the behavior of the whole system 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 peers' observable behavior (communication exchanges) and to dynamically check the conformance of such behaviour with the constraints posed by the protocols regulating the overall system;

(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.

1.4 Aim of this thesis

This thesis work started within the SOCS project (Societies Of Computees, EU IST-32530), where a specification language and verification tools were defined for protocols in the Multi Agent Systems scenario. The aim of this thesis is on one side to contribute to such language and tools, and on the other to extend the language for addressing the executability, other verification types, and to extend its application to several different scenarios.

In particular, within the SOCS project a declarative language has been defined, with a formal declarative semantics based on Abductive Logic Programming, and tools for the *Type 2* verification

Then, starting from the obtained results, a further research activity has been conducted, in order to address the remaining issues of *Type 1* and *Type 3* verification, as well as the executability property of specification given using the language. Moreover, the proposed approach has been extended to different application domains, in order to fully understand its advantages and limits.

Summarizing, this thesis demonstrate how a single framework, based on Computational Logic, can be used for specifying interactions protocols, and to perform several verifications (*Type 1*, *2* and *3*) on the specified protocols. Moreover, it shows also how the protocol specification (given in terms of the framework language) can be usefully exploited to ease the development process of the interacting peers.

1.5 How this thesis is organized

This thesis is organized in the following way. Chapter 2 introduces the $\mathcal{S}$CIFF Framework, together with the language for specifying interactions protocols, its declarative semantics, the proof procedure for reasoning upon the interactions (given a specification of the protocol), and with some formal properties.

Chapter 3 shows how we exploit the $\mathcal{S}$CIFF Framework, in order to address the *Type 2* verification on conformance of observed behaviours w.r.t. a given protocol specification. Some performances consideration are reported, and some examples of the verification process are documented.

Chapter 4 address the *Type 3* verification, by suitably extending the framework introduce in Chapter 2. The new g-$\mathcal{S}$CIFF Framework is presented, with its specification language, its declarative and operational semantics, and some formal properties. Examples of how the g-$\mathcal{S}$CIFF can be used for verifying protocol properties conclude the chapter.

Chapter 5 instead address the *Type 1* verification, by assuming that the peers make public a description of their behaviour (such description is often known as the *behavioural interface*. The A^{l}LoWS framework, obtained by using both the $\mathcal{S}$CIFF and the g-$\mathcal{S}$CIFF framework, shows how the a-priori conformance of such peers can be proved w.r.t. a given protocol specification.

Chapter 6 instead presents an agent platform where, beside using the $\mathcal{S}$CIFF framework to verify on the fly that agents behave as prescribed, the protocol specification can be directly used to develop the interacting peers.

Finally, Chapter 7 summarize the result presented in this thesis, and some considerations about future research directions are given.

Chapter 2

Specifying Interaction Protocols: The SCIFF Framework

The $\mathcal{S}$ocial Constrained IF-and-only-If framework ($\mathcal{S}$CIFF) has been developed in the context of the SOCS European project (IST-2001-32530, [139]): the focus of that project was about the definition of computational logic models for agents (*Computees*) and Multi-Agent Systems (*Societies of Computees*).

The $\mathcal{S}$CIFF framework was specifically developed to address the issues related to agent interactions, i.e. the protocols regulating these interactions. In particular, the research activity focussed on the issues of specifying such protocols and to verify agents behaviours against such specifications. The problem of specifying a protocol has been tackled by means of computational logic, and in particular by exploiting the Abductive Logic Programming (ALP).

The $\mathcal{S}$CIFF Framework has addressed the *Type 2* verification (Section 1.2) for the MAS settings. However, the framework has been designed to be suitable for more general application domains, and its use is not restricted to the multi agent systems. Generally speaking, $\mathcal{S}$CIFF can be used to specify and reason about any interaction

process. E.g., $\mathcal{S}$CIFF has been successfully used to reason about communication protocols like TCP [124], security communication protocols like the Needham-Schroeder [9, 114], Web Services interactions within a Choreography Specification [7].

The $\mathcal{S}$CIFF framework is made up of several components:

The $\mathcal{S}$CIFF Specification Language A language for specifying the interaction protocols, by means of rules that relate events with other events (in the agent domain, communicative acts with other communicative acts). Moreover, it provides also the possibility of expressing a knowledge base that can be used when reasoning about the interactions. The focus of the $\mathcal{S}$CIFF Specification Language is on the events: when an event happens, the rules specify if other events are expected to happen or are expected *not* to happen (both in the past and in the future w.r.t. to the happened event).

The $\mathcal{S}$CIFF Declarative Semantics A declarative semantics for the $\mathcal{S}$CIFF Language, based on abduction and on Abductive Logic Programs (ALP, [95]).

The $\mathcal{S}$CIFF Proof Procedure An abductive proof procedure, that is used within the framework for reasoning about the interactions logs, and about their compliance w.r.t. a protocol specification.

Moreover, a fourth component, the *SOCS-SI* software tool, is part of the $\mathcal{S}$CIFF Framework: however, this component is presented in Chapter 3, together with some application examples of the framework applied to the verification issues.

Contributions of the author. The author participated to the SOCS project for the final two years over three and half years taken by the project. Although the author

didn't participate directly to the specification of the computational logic model (done in the very beginning of the project), he has actively contributed in the definition, development and implementation of all the parts composing the $\mathcal{S}$CIFF framework.

Chapter organization. This chapter is organized as follows: we begin by introducing some key concepts about the entities (events and expectations, Section 2.1) and by clarifying some terms and assumptions we make (Section 2.2).

In Section 2.3 we formally define the $\mathcal{S}$CIFF Language and its parts, while in Section 2.4 we provide also a declarative semantics.

In Section 2.5 we introduce the $\mathcal{S}$CIFF Proof Procedure and briefly present its implementation; in Section 2.5 instead we enunciate its formal properties (soundness, termination and completeness).

The chapter is concluded by a discussion about related works.

2.1 Events, Happened Events and Expectations about Events

The definition of *Event* greatly varies, depending on the application domain. For example, in the healthcare domain, an event could be the fact that a laboratory has communicated the results of blood analysis to the patient who requested it; in a communication protocol like the TCP, an event could be the fact that a peer has sent a *syn* message; in the Web Service domain, an event could be the fact that a certain web service has been invoked. Moreover, within the same application domain there could be several different notions of events, depending on the assumed perspective, the granularity, etc.

The $\mathcal{S}$CIFF framework abstracts completely from the problem of deciding "what is an event", and rather lets the developers decide which are the important events for modeling the domain, at the desired level of detail and granularity. Each event that can be described by a *Term*, can be used in $\mathcal{S}$CIFF. For example, in a peer-to-peer communication system, an event could be the fact that someone communicates something to someone else (i.e., a *communicative* action has been performed):

$$tell(alice, bob, msgContent)$$

Another event could be the fact that a web service has updated some information stored into an external database, or that a bank clerk, upon the request of a customer, has provided him/her some money. Of course, in order to perform some reasoning about such events, accessibility to such information is a mandatory requirement.

In the $\mathcal{S}$CIFF framework, similarly to what has been done in [39], we distinguish

between the description of the event, and the fact that the event has happened. Typically, an event happens at a certain time instant; moreover the same event could happen many times. In our approach the happening of identical events at the same time instant are considered as if only one event happens; if the same event happens more than once, but at different time instants, then they are indeed considered as different happenings. We will always use the term *Event* as a synonym of its description, while *happened events* (i.e. the fact that the event described by *Event* has happened) will be represented as atoms

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

where $Event$ is a *Term*, and $Time$ is an integer, representing the discrete time point in which the event happened.

One innovative contribution of the $\mathcal{S}$CIFF framework is the introduction of *expectations* about events. Indeed in the framework, beside the explicit representation of "what" happened and "when", it is possible to explicitly represent also "what" is expected, and "when" it is expected. The notion of *expectation* plays a key role when defining global interaction protocols, choreographies, and more in general any dynamically evolving process: it is quite natural, in fact, to think of such processes in terms of rules of the form *"if A happened, then B should be expected to happen"*. Expectations about events have the form

$$\mathbf{E}(Event, Time)$$

where $Event$ and $Time$ can be variables, or they could be grounded to a particular Term/value. Constraints, like $Time > 10$, can be specified over the variables: in the given example, the expectation is about an event (described by $Event$) to happen at

a time greater than 10 (hence the event is expected to happen *after* the time instant 10).

Strictly related to the expectations about the happening of events (*positive expectations*), there are the expectations about events that should not happen (*negative expectations*). The $\mathcal{S}$CIFF framework allows to directly represent such negative expectations, in the form:

$$\mathbf{EN}(Event, Time)$$

where the parameters have the same meaning as for the positive expectations. However, the variables that possibly appears in the negative expectations are ruled by different quantification rules w.r.t. the positive expectations. We provide here an intuition, while the details will be discussed in Section 2.3. Typically, a positive expectation is about a certain event to happen: e.g., writing $\mathbf{E}(tell(alice, bob, hello), T_1)$ in the $\mathcal{S}$CIFF framework means that there is an expectation about the happening of the event, at a non specified time T_1. In this case, T_1 is a variable whose quantification is *existential*. Differently, writing $\mathbf{EN}(tell(alice, bob, gossip), T_2)$ means that there is an expectation that the event will not happen at any time T_2. In this case, T_2 is quantified *universally*.

Given the notions of *happened event* and of *expected/expected not event*, two fundamental issues arise: first, how it is possible to specify the link between these two notions. Second, how it is possible to verify if all the expectations have been effectively satisfied. The first issue is fundamental in order to easy the definition of an interaction protocol, and it will be addressed in the the Section 2.3. The second issue, instead, is inherently related to the problem of establishing if a software component,

given its observable external behaviour, does indeed respect a given protocol specification: the solution proposed by the $\mathcal{S}$CIFF framework is presented in Sections 2.4 and 2.5.

2.2 The terms "Open" and "Closed" in the SCIFF Framework

In the $\mathcal{S}$CIFF framework, the adjectives *open* and *closed* are used in several different contexts, referring sometimes to some different concepts. In order to easy the comprehension of the framework, we try here to provide an intuitive (and very informal) description of the cases where open and close adjectives are used.

2.2.1 Open vs. Closed Histories

We call history a set of happened events, that represents somehow the trace (or the log) of an interaction instance. The same interaction instance (i.e., the grounded instance of a generic interaction defined by a protocol) can be considered as a synonym of the term "history".

The $\mathcal{S}$CIFF framework has been thought in order to be able to perform reasoning at run-time, i.e. when the interaction is taking place. At every instant, the $\mathcal{S}$CIFF Proof Procedure can reason upon the actual history (the log of the events happened until that precise instant): hence, it can reason upon a partial and incomplete version of the whole history.

However, one of the key features of the $\mathcal{S}$CIFF Proof Procedure is the ability to reason also upon dynamically happening events without re-considering the reasoning on the past events: in this way, each time a new event happens, the reasoning process is not performed reconsidering again all the history. Instead, the partial result obtained from the previous reasoning is used as a starting point in order to perform further reasoning.

This very powerful mechanism however has a limit: in order to perform some

types of reasoning, it is necessary to know if more (newer) events can still happen or, instead, if no more events can happen anymore. E.g., suppose an absolute prohibition of performing a certain action has been hypothesized:

$$\mathbf{EN}(perform(Perfomer, aCertainAction), Time)$$

In order to establish if this prohibition has been respected or not by a certain history, it is necessary to know if the interaction represented by the history has terminated (and no more events can happen anymore) or not. If no more events can happen in a certain interaction instance, we say that the representing history is *closed*. Is newer vent can still happen (in the context of the same interaction), we say that the history is still *open*.

The distinction between open and closed histories is formalized in Section 2.4.2 (Definitions 2.4.2 and 2.4.3), and has also some practical consequences on the properties of the proof procedure, as well as on the proof procedure itself (e.g., a specific transition called *Closure*, see Section 2.5.4).

2.2.2 Open vs. Closed Interactions Models

Open and Closed are used also in a different context, with a completely different meaning, if related to the Interaction Models (often abbreviated to Interactions). Typical protocols assume a "closed interaction model": every event that happens in the interaction must be explicitly allowed by the interaction specification (by the protocol).

E.g., the TCP protocol [124] defines, for every interaction stage, which are the messages that can be uttered if the interaction is in that particular stage. All the

messages not explicitly listed, are implicitly prohibited. If a peer utters a message not explicitly envisaged by the protocol specification, the interaction is automatically considered as faulty, and the other peer reset the connection. [1]

Another interesting example is provided instead by the recent Business Processes/SOA scenario: due to the openness degree of the environment where business processes are envisaged to be employed (wide local networks/internet, with hardware and software heterogeneity of the peers), the "reset connection" behaviour might result in a too strong reaction; moreover, the fault tolerance of the whole system could result undermined. Although not specified as a principle, the choice of many service engine vendors is to *discard* unwanted messages, and to keep a high flexibility on the decision of elaborating messages not explicitly envisaged by some choreography/orchestration specification.

With the term *closed interaction model*, we mean an interaction specification (a protocol specification) where only events explicitly envisaged by the protocol can happen, and where the happening of any other event is considered as prohibited and a violation of the protocol itself. With the term *open interaction model* instead we mean those protocols that allows for some freedom degrees in the allowed interaction instances. Peers that perform an action or utter a message not envisaged by the protocol do not automatically violates the protocol specification, unless that particular action or message weren't already explicitly prohibited.

The $\mathcal{S}$CIFF framework, as already stated previously, has been developed to provide a logic-based formalization for interaction in the MAS scenarios. Multi Agent

[1] Although the behaviour in case of wrong messages is not clearly specified by the TCP specification [124], the reset action has been chosen as default behaviour by the majority of the TCP implementation stacks.

systems are implicitly heterogeneous both for the hardware as well for the software aspects. Moreover, as discussed by Singh in [44], close interaction models could not be expressive enough in order to capture the complexity of interactions in the agent models.

For these reasons, in the $\mathcal{S}$CIFF Framework it is possible to explicitly specify which are the expected events, and which are the prohibited ones. Events that are not expected, nor prohibited, can happen. However, please note that this characteristic does not guarantee that the happened event will not generate some violation due, e.g., to some inconsistency with previous happened events.

2.2.3 Open vs. Closed Agent Societies

The agent paradigm has raised several problems: the architecture of the various agents, the interactions amongst the agents, the social organization, the rules, the roles of the agents in the society, to cite some. In particular the Multi Agent Systems paradigm has stressed the society related issues, raising questions about the "openness" degree that such societies should entail.

According to Davidsson [54], there can be four types of societies:

Closed societies are predefined societies, in which no agent can enter. Only the designer of the society can create new agents in the society itself.

Semi-closed are societies in which agents cannot enter, but they can nominate or spawn representatives in the society.

Semi-open are societies in which there exists one agent taking the role of gatekeeper, which receives the requests for entering the society. A potential member applies

at the gate, can provide some credentials, and can possibly be admitted in the society by the gatekeeper.

Open are societies in which any agent can enter without restriction.

The classification by Davidsson is based on rules for entering the society, as this is the most pressing issue. Leaving the society could be done by considering a leaving protocol (in semi-open or semi-closed societies), or, in some cases, it can be a way to punish misbehaving agents: when an agents does not comply to the rules, it is ejected from the society. In open societies, there are no given protocols to exit: agents may leave at any time without restriction.

Clearly, open societies are the most flexible, but can also be very unstable. The set of members is not fixed, nor even computable in general, as new agents may join anytime, and current members could leave without any notification. Also openness à la Davidsson implies heterogeneity: any agent may join, so they are not required to share concepts such as beliefs, intentions, knowledge bases, or architectures. Some agents may exhibit powerful reasoning capabilities, while others may only be able to react to stimuli with predefined patterns. Foreign agents can join the society without restrictions and profit from interacting with the agents in the society. On the other hand, malicious agents could enter and disrupt the harmonious evolution of the society, threatening the usability of the whole MAS. Thus, mastering open societies in order to drive them to a coherent, useful global behaviour is a challenge.

2.3 The SCIFF language

The language is composed of entities for expressing happened events, expectations about events, hypotheses, and relationships between happened events and expectations/hypotheses. The final goal of the $\mathcal{S}$CIFF language is to provide a way for specifying agent societies.

A Social specification, i.e, a specification of an agent society in the $\mathcal{S}$CIFF framework, is composed of the following elements:

- a knowledge base, often named *Social Knowledge Base*, (*SOKB*);
- a set of *Integrity Constraints* (ICs);
- possibly a *society goal*, i.e. a result that the whole society, through the interaction of its member, should manage to achieve.

We provide meaning to a social specification by means of Abduction (see Section 2.4): since the $\mathcal{S}$CIFF framework has been designed to be general enough to be use for generic interaction protocols, we will use the term *Abductive Specification* for general cases, and *Social Specification* to indicate an abductive specification in the MAS context.

2.3.1 Syntax of Happened Events and Expectations

Hapened events are the abstraction used to represent the actual observations.

Definition 2.3.1 *An* Happened Event *is an atom:*

- *with predicate symbol* $\boldsymbol{H}$*;*
- *whose first argument is a ground term; and*

- *whose second argument is an integer.*

Intuitively, the first argument is meant to represent the description of the happened event, according to application-specific conventions, and the second argument is meant to represent the time at which the event has happened:

Example 2.3.1

$$\mathbf{H}(tell(alice, bob, query\_ref(phone\_number), dialog\_id), 10) \tag{2.3.1}$$

could represent the fact that *alice* asked *bob* his *phone_number* with a *query_ref* message, in the context identified by *dialog_id*, at time 10.

A *negated happened event* is an event with the unary prefix operator *not* applied to it. [2] We will call *history* a set of happened events, and denote it with the symbol **HAP**.

While *happened events* represent the observed facts, *Expectations* are the abstraction we use to represent the desired events. In a MAS setting, they would represent the ideal behaviour of the system, i.e., the actions that, once performed, would make the system compliant to its specifications. Our choice of the terminology "expectation" is intended to stress that observations cannot be enforced, but only expected, to be as we would like them to be. Expectations are of two types:

- *positive*: representing some event that is expected to happen;
- *negative*: representing some event that is expected *not* to happen.

Definition 2.3.2 *A* positive expectation *is an atom:*

[2] *not* represents default negation (see declarative semantics of the $\mathcal{S}$CIFF framework, Sect. 2.4).

- *with predicate symbol* ***E****;*
- *whose first argument is a term; and*
- *whose second argument is a variable or an integer.*

Intuitively, the first argument is meant to represent an event description, and the second argument is meant to tell for what time the event is expected.

Example 2.3.2 The atom

$$\mathbf{E}(tell(bob, alice, inform(phone\_number, X), dialog\_id), T_i) \tag{2.3.2}$$

could represent that *bob* is expected to *inform alice* at some time T_i that the value for the piece of information identified by *phone_number* is X, in the context identified by *dialog_id*.

A *negated positive expectation* is a positive expectation with the explicit negation operator $\neg$ applied to it.

As the example shows, expectations can contain variables, as it might be desirable to leave the expected behaviour not completely specified. Variables in positive expectations will be existentially quantified, supporting the intuition, as we have seen in Ex. 2.3.2.

Definition 2.3.3 *A* negative expectation *is an atom:*

- *with predicate symbol* ***EN****;*
- *whose first argument is a term; and*
- *whose second argument is a variable or an integer.*

Again, the first argument is meant to represent an event description, and the second argument is meant to tell for what time the event is expected not to happen.

Example 2.3.3 The atom

$$\mathbf{EN}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{refuse}(\mathit{phone\_number}), \mathit{dialog\_id}), T_r) \tag{2.3.3}$$

could represent that *bob* is expected *not* to *refuse* to *alice* his *phone_number*, in the context identified by *dialog_id*, at *any* time.

A *negated negative expectation* is a negative expectation with the explicit negation operator $\neg$ applied to it. Note that $\neg\mathbf{E}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{refuse}(\mathit{phone\_number}), \mathit{dialog\_id}), T_r)$ is different from $\mathbf{EN}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{refuse}(\mathit{phone\_number}), \mathit{dialog\_id}), T_r)$. The intuitive meaning of the former is: no *refuse* is expected by Bob (if he does, we simply did not expect him to), whereas the latter has a different, stronger meaning: it is expected that Bob does not utter *refuse* (by doing so, he would frustrate our expectations). As the example shows, variables in negative expectations are naturally interpreted as universally quantified (Bob should *never* refuse). However, the same variable may occur in two distinct expectations, one of which positive, the other negative. In that case, the quantification will be existential (i.e., the convention adopted for positive expectations will prevail). This follows the intuitions, as we can see in the following example.

Example 2.3.4 It is expected that (at least one) agent A performs task t_1, and that no other agent B interrupts A:

$$\mathbf{E}(\mathit{perform}(A, t_1)), \mathbf{EN}(\mathit{interrupt}(B, A)).$$

Variable A is existentially quantified, while B is quantified universally.

Table 2.3.1 Syntax of events and expectations

EventLiteral	::=	$[not]Event$
Event	::=	$\mathbf{H}(GroundTerm, Integer)$
ExpLiteral	::=	$PosExpLiteral \mid NegExpLiteral$
PosExpLiteral	::=	$[\neg]PosExp$
NegExpLiteral	::=	$[\neg]NegExp$
PosExp	::=	$\mathbf{E}(Term, Variable \mid Integer)$
NegExp	::=	$\mathbf{EN}(Term, Variable \mid Integer)$
ExistLiteral	::=	$PosExpLiteral \mid AbducibleLiteral \mid Literal$
NbfLiteral	::=	$not\ Atom \mid not\ AbducibleAtom$
Literal	::=	$[not]Atom$
AbducibleLiteral	::=	$[not]AbducibleAtom$

The syntax of events and expectations is summarised in Tab. 2.3.1, and it will be used as such by the subsequent Tab. 2.3.2 and 2.3.3. We also introduce, for ease of presentation, the syntactic element *ExistLiteral*, that lists the literals that are existentially quantified. Again, for simplifying the following presentation, we define *NbfLiteral*, that intuitively indicates negative literals with negation by failure. By *AbducibleAtom* we mean an atom built on an abducible predicate (i.e., a predicate in the set *Ab*; see Sect. 2.4).

2.3.2 Specification of the Social Knowledge Base

The Social Knowledge Base ($SOKB$) is a set of *Clause*s in which the body can contain (besides defined and abducible literals), expectation literals and constraints. Intuitively, the $SOKB$ is used to express declarative knowledge about the specific application domain.

The syntax of the Knowledge Base is given in Tab. 2.3.2, and it will be used as

Table 2.3.2 Syntax of the Knowledge Base

$$
\begin{array}{rcl}
SOKB & ::= & [Clause]^{\star} \\
Clause & ::= & Head \leftarrow Body \\
Head & ::= & Atom \\
Body & ::= & ExtLiteral\ [\ \wedge\ ExtLiteral\ ]^{\star} \mid true \\
ExtLiteral & ::= & Literal \mid AbducibleLiteral \mid ExpLiteral \mid Restriction
\end{array}
$$

such also in Tab. 2.3.3.

Allowedness conditions

The operational semantics (see Section 2.5) will require some syntactic restrictions, which we will now introduce. In the sequel and throughout this thesis, we will assume that such restrictions hold in all cases we consider. As usual in Logic Programming, we need to avoid floundering of variables in negative literals [108]:

Definition 2.3.4 *A clause Head* $\leftarrow$ *Body is* allowed *if and only if every variable that occurs in a NbfLiteral in Body, also occurs in the Head or in at least one ExistLiteral.*

Variable quantification and scope

The quantification and scope of variables is implicit. In each clause, the variables are quantified as follows:

- universally with scope the *Clause* if they occur in the *Head* or in at least one *ExistLiteral*;
- otherwise (if they occur only in negative expectations and possibly restrictions) universally, with scope the *Body*.

This means that clauses will be quantified as in most other abductive logic programming languages, and in particular, in the language interpreted by the IFF proof-procedure, except for negative expectations. Variables that occur only in a negative expectation will be universally quantified with scope the *Body*. Let us see an example:

Example 2.3.5 In order to have a task completed, it is expected that an agent performs it, and no agent is expected to interrupt the agent performing that task.

$$completed(Task) \leftarrow \mathbf{E}(perform(A, Task)), \mathbf{EN}(interrupt(B, A)).$$

The quantification of the variables is most intuitive:

$(\forall Task, \forall A)$ (
$\quad completed(Task) \leftarrow \mathbf{E}(perform(A, Task)), (\forall B)$ ($\mathbf{EN}(interrupt(B, A))$)
).

Definition 2.3.5 *A Clause is* restriction allowed *if the variables that are universally quantified with scope the body do not occur in quantifier Restrictions, and each variable that occurs in a restriction also occurs in at least one positive expectation PosExp, or in AbducibleLiteral in the body.* [3]

For example, the clause:

$$p \leftarrow \mathbf{EN}(X), X < 10$$

is not restriction allowed, because it contains a variable X that is universally quantified with scope the *Body*, and that is also in a quantifier restriction. Similarly, the

[3] *Def. 2.3.5 is needed for a correct handling of defined predicates literals in the integrity constraints. In fact, it turns out that unfolding a clause which is not restriction allowed could generate an integrity constraint which is not restriction allowed (see Def. 2.3.7). Note that if there is no chance a predicate will appear in the body of an integrity constraint, then the restriction allowedness condition could be safely relaxed.*

clause:

$$p \leftarrow a(Y), Y < 10$$

is not restriction allowed, because it contains an existentially quantified variable Y, with scope the *Body*, which does not appear in any *PosExp* literal (**E**) in the *Body*.

Goal

Thanks to the abductive interpretation, goal-directed societies are possible in the $\mathcal{S}$CIFF framework; non-goal directed societies are also supported, by considering the atom *true* as goal. The syntax of the goal is the same as the *body* of a clause (Tab. 2.3.2). In order to avoid floundering, variables in the goal cannot occur only in *NbfLitera*s. The quantification rules are the following:

- All variables that occur in an *ExistLiteral* are existentially quantified.
- All remaining variables are universally quantified.

Note that these rules are equivalent to those of the variables in the body of a clause (Sect. 2.3.2), considering that $\forall X.(H \leftarrow B)$ is equivalent to $H \leftarrow (\exists X.B)$ when X does not occur in H.

2.3.3 Syntax of the Integrity Constraints

Integrity Constraints (also ICs, for short, in the following) are implications that, operationally, are used as forward rules, as will be explained in Sect. 2.5. Declaratively, they relate the various entities in the $\mathcal{S}$CIFF framework, i.e., expectations, happened events, abducibles, and constraints/restrictions, together with the predicates in the knowledge base.

Table 2.3.3 Syntax of Integrity Constraints (ICs)

$$
\begin{array}{rcl}
\mathcal{IC}_{\mathcal{S}} & ::= & [IC]^{\star} \\
IC & ::= & Body \rightarrow Head \\
Body & ::= & (EventLiteral \mid ExpLiteral \mid AbducibleLiteral)\ [\ \wedge BodyLiteral\ ]^{\star} \\
BodyLiteral & ::= & EventLiteral \mid ExtLiteral \\
Head & ::= & HeadDisjunct\ [\ \vee HeadDisjunct\ ]^{\star} \mid false \\
HeadDisjunct & ::= & ExtLiteral\ [\ \wedge ExtLiteral]^{\star}
\end{array}
$$

The syntax of ICs is given in Tab. 2.3.3: the *Body* of ICs can contain conjunctions of all elements in the language (namely, **H**, **E**, and **EN** literals, definite and abducible literals and restrictions), and their *Head* contains a disjunction of conjunctions of all the literals in the language, except for **H** literals. Let us now consider an interaction protocol taken from the MAS literature:

Specification 2.3.1 Integrity Constraints and Knowledge Base for the *query_ref* specification.

$$
\begin{array}{rl}
 & \mathbf{H}(tell(A, B, query\_ref(Info), D), T)\ \wedge \\
 & qr\_deadline(TD) \\
\rightarrow & \mathbf{E}(tell(B, A, inform(Info, Answer), D), T1)\ \wedge \\
 & T1\ <\ T\ +\ TD \\
\vee & \mathbf{E}(tell(B, A, refuse(Info), D), T1)\ \wedge \\
 & T1\ <\ T\ +\ TD \\
 & \\
 & \mathbf{H}(tell(A, B, inform(Info, Answer), D), Ti) \\
\rightarrow & \mathbf{EN}(tell(A, B, refuse(Info), D), Tr) \\
qr\_deadline(10). &
\end{array}
$$

Example 2.3.6 Tab. 2.3.1 shows the ICs for the *query_ref* [76] specification. Intuitively, the first IC means that if agent A sends to agent B a *query_ref* message, then

B is expected to reply with either an *inform* or a *refuse* message by TD time units later, where TD is defined in the Knowledge Base by the *qr_deadline* predicate. The second IC means that, if an agent A sends an *inform* message, then it is expected not to send a *refuse* message about the same *Info*, to the same agent B and in the context of the same interaction D at any time.

Variable quantification and scope

All variables in an integrity constraint should occur in an *EventLiteral*, *ExpLiteral*, or *AbducibleAtom*. The rules of scope and quantification for the variables in an integrity constraint $Body \rightarrow Head$ are as follows:

1. Each variable that occurs both in *Body* and in *Head* is quantified universally, with scope the integrity constraint.

2. Each variable that occurs only in *Head* cannot occur only in *NbfLiteral*s and

   - if it occurs in at least one $ExistLiteral$ is existentially quantified and has as scope the disjunct where it occurs;
   - otherwise it is quantified universally.

3. Each variable that occurs only in *Body* is quantified with scope *Body* as follows:

   (a) existentially if it occurs in at least one $ExistLiteral$ or $Event$;

   (b) universally, otherwise.

The given quantification rules let the user write integrity constraints without explicitly stating the quantification of the variables, and typically capture the intuitive meaning of the rules in protocols. Let us show it with an example.

Example 2.3.7 Consider the following example:

$$\mathbf{H}(p(X,Y)), \mathit{not}\ \mathbf{H}(q(Z,X)) \rightarrow \mathbf{E}(r(X,K)), \mathbf{EN}(f(Y,J))$$

Variables X and Y occur both in the body and in the head. Coherently with the literature in abduction, they will be universally quantified with scope the whole IC. Variables K and J occur only in the $Head$. The quantification rules for those variables are the same as for the *Goal* (see Sect. 2.3.2), i.e., existential for K and universal for J. Finally, $\neg\mathbf{H}(q(Z,X))$ means that, if no event happens matching $q(Z,X)$, then the IC's head should be true. For instance, if the set of happened events is

$$\mathbf{H}(p(2,1)), \mathbf{H}(q(3,2))$$

it is quite natural to understand the *Body* as false (the second event makes $\mathit{not}\ \mathbf{H}(q(Z,X))$ false). So, the existence of one atom ($\mathbf{H}(q(3,2))$ in the example) is enough for making $\mathit{not}\ \mathbf{H}(q(Z,X))$ false. This means that the IC should be read as *"if* $\boldsymbol{H}(p(X,Y))$ *and for all values* Z, $\boldsymbol{H}(q(Z,X))$ *is false, the Head must hold"*. Variable Z should be quantified as follows:

$$[\forall Z \ldots, \mathit{not}\ \mathbf{H}(q(Z,X))] \rightarrow \ldots$$

thus, the quantification rules give the quantification

$$\forall X, Y \exists Z, K \forall J.\ \mathbf{H}(p(X,Y)), \mathit{not}\ \mathbf{H}(q(Z,X)) \rightarrow \mathbf{E}(r(X,K)), \mathbf{EN}(f(Y,J))$$

Allowedness conditions

As in the case of the Knowledge Base syntax, the following syntactic restrictions are motivated by the operational semantics, and will be supposed to hold throughout the

paper. A variable cannot occur in an IC only in *NbfLiteral*s. If it does occur in a literal with negation by failure, it necessarily has to appear in the same IC also in at least another literal within predicate symbol **H**, **E**, **EN**, or an abducible atom. Since variables in positive expectations are existentially quantified, integrity constraints should not entail universally quantified positive expectations. For example,

$$not\ \mathbf{H}(p(A)) \rightarrow \mathbf{E}(q(A))$$

would entail in an empty history that $\forall A.\mathbf{E}(q(A))$. We avoid such situations with the following allowedness condition.

Definition 2.3.6 *An Integrity Constraint Body* $\rightarrow$ *Head is* quantifier allowed *if*

- *each variable that occurs in an ExistLiteral in Head either does not occur in Body, or it occurs in the Body in at least one Event or in a PosExpLiteral, or in an AbducibleAtom;*
- *each variable that occurs in a NbfLiteral in Body also occurs in at least one Event or PosExpLiteral or in an AbducibleAtom in Body*[4].

Definition 2.3.7 *An integrity constraint is* restriction allowed *if*

- *all the variables that are universally quantified with scope Body do not occur in Restrictions;*
- *the other variables (that occur only in Head, or both in Head and in Body) can occur in Restrictions. Each Restriction occurring in the integrity constraint should:*

[4] *This rule descends from the previous one, considering that* $not(A), B \rightarrow C$ *is equivalent to* $C \rightarrow A \vee B$.

- *either involve only variables that also occur in PosExpLiterals, Events or AbducibleAtom (in the same disjunct, or in the body),*
- *or involve* one *variable that also occurs in at least one NegExpLiteral, and possibly other variables which only occur in Events.*

Abductive Specification

Given a Knowledge Base $SOKB$ and a set $\mathcal{IC}_\mathcal{S}$ of Integrity Constraints, we can define an *Abductive Specification*:

Definition 2.3.8 (Abductive Specification). *An* Abductive Specification *is the pair:*

$$\mathcal{S} = \langle SOKB, \mathcal{IC}_\mathcal{S} \rangle$$

and will be indicated with the symbol $\mathcal{S}$*.*

Definition 2.3.9 *An abductive specification* $\mathcal{S} = \langle SOKB, \mathcal{IC}_\mathcal{S} \rangle$ *is* quantifier allowed *if all the integrity constraints in* $\mathcal{IC}_\mathcal{S}$ *are quantifier allowed.* $\mathcal{S}$ *is* restriction allowed *if all the clauses in* $SOKB$ *and all the integrity constraints in* $\mathcal{IC}_\mathcal{S}$ *are restriction allowed.* $\mathcal{S}$ *is* allowed *if it is quantifier allowed and restriction allowed, and* $SOKB$ *is allowed.*

As a recap on allowedness conditions, we have the following table. For all the three syntactic elements (Clauses, Goal, and ICs), variables cannot occur only in *NbfLiteral*s. Besides, the following conditions must hold in order for Clauses/ICs to be restriction-/quantifier-allowed:

	Clause	Integrity Constraint
is restriction-allowed	if QRs only appear on vars in $\exists$ abducibles	if no QR appears in $\forall$ vars with scope *Body* and QRs do not involve more than one $\forall$ var
is quantifier-allowed	*always*	if *ExistLiterals* in *Head* do not contain vars that occur in the *Body* only in *not* **H**, [¬]**EN** and all vars of *NbfLiterals* in *Body* also occur in other **H**, [¬]**E**, or abducibles in *Body*

2.4 SCIFF Declarative Semantics

2.4.1 Background

We assume the reader has a basic familiarity with logics and logic programming; a good introduction is the book by Lloyd [108]. As will be clear soon, the $\mathcal{S}$CIFF framework is based on Abductive Logic Programming and on Constraint Logic Programming; we introduce the two concepts in an intuitive way, and provide pointers to the formal parts.

Abduction

Abduction is a powerful mechanism for hypothetical reasoning in the presence of incomplete knowledge, that is handled by labelling some pieces of information as "abducibles". Abducibles can be viewed as possible hypotheses which can be assumed, provided that they are consistent with the current knowledge base. The abduction process is typically applied when looking for an explanation for some observation. Starting from some observed facts, possible causes are hypothesised (they are abduced). Then it is possible to confirm the hypotheses by performing some additional observation: for example, the scientist postulates some theory, and then develops new experiments to confirm (or disconfirm) such theory. Another common application of abduction is *diagnosis*: the physician, by observing the symptoms, formulates some alternative hypothesis about the disease. The physician tries to find more facts by prescribing a patient another test, that will possibly support a smaller set of explanations. Some of the previously made hypotheses could be discarded because they are now incompatible with the new facts, or because some pairs of explanations cannot be assumed at the same time. Formally, an *abductive logic program* (ALP) [96] is a

triple $\langle P, Ab, IC \rangle$ where:

- P is a (normal) logic program, i. e., 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 with scope the clause. 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 is 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;

- IC is a set of integrity constraints, that is, a set of formulae in the language of P.

Given an abductive logic program $\langle P, Ab, IC \rangle$ and a formula G, the goal of abduction is to find a (possibly minimal) set of ground atoms Δ (the *abductive explanation*), with $\Delta \subseteq Ab$, and which, together with P, entails G, and satisfies IC:

$$P \cup \Delta \models G \tag{2.4.1}$$

$$P \cup \Delta \models IC \tag{2.4.2}$$

The notion of entailment $\models$ depends on the semantics associated with the logic program P. Several abductive proof procedures can be found in the literature (like the Kakas-Mancarella [97], limited to ground literals, SLDNFA [57], that can abduce literals with existentially quantified variables, ACLP [3] and $\mathcal{A}$-system [99], that integrate constraints, to cite some). The $\mathcal{S}$CIFF proof procedure (Section 2.5) is an extension of the If-and-only-If (IFF) abuctive proof procedure [82]. The integrity constraints,

in the IFF proof procedure, are expressed as a set of implications of the form:

$$B_1 \wedge \cdots \wedge B_n \rightarrow A_1 \vee \cdots \vee A_m$$

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.

Constraint Logic Programming

Constraint Logic Programming [91, 92] (CLP) is a class of programming languages that extend logic programming by giving an interpretation to some of the symbols. In classical Logic Programming, the symbols are not interpreted, so the term 2+3 does not mean 5, but simply a structure whose functor is + and whose terms are 2 and 3. Unification performs a syntactical operation, and does not provide any interpretation, so the term 5 will not unify with the term 3+2, and the goal 5=3+2 simply fails. In Constraint Logic Programming, a subset of the terms and atoms are given a standard interpretation: the symbol 5 stands for the number *five* and the symbol + represent the addition operation. Unification is extended, and treated as a *constraint*. For example, the goal $5 = A + 3$ succeeds in CLP, providing the answer $A = 2$. This behaviour is obtained by identifying syntactically the set of interpreted atoms, called *constraints*, and inserting them into a *constraint store* instead of applying resolution. The constraints in the store are then evaluated by a *constraint solver*, that detects possible failures and infers new constraints. Each language of the CLP class is identified by a *domain*, representing the set of values that a variable subject to constraints can assume, the set of constraints, the set of interpreted symbols. For example, CLP($\mathcal{R}$) [93] is the instance of CLP that works on the reals; this means that a variable in CLP(R) can have a real value, and it can be subject to constraints

on the reals. Current implementations typically employ the simplex algorithm as constraint solver. CLP(FD) is the specialisation of CLP on the Finite Domains [64]. Variables are initially assigned a domain through the predicate $Variable :: Domain$. For instance $X :: [red, green, blue]$ states that X can take only the values *red*, *green* or *blue*. On numeric values, CLP(FD) languages typically interpret the symbols $<$, $\leq$, $=$, $\neq$, etc., plus the usual operations $+$, $-$, $*$, $/$. In CLP(FD), imposing constraints typically deletes inconsistent values from the domains of the variables; for example, if $A :: [0..10]$, $B :: [1..5]$, $A < B$ would remove the values that cannot satisfy the imposed constraint, in this case the values greater than 4 in the domain of A. When a domain becomes empty, there cannot be an assignment for the corresponding variable, so the system fails. Various languages and efficient solvers have been developed [64, 137, 4]. Such languages have been successfully used for hard combinatorial problems, such as scheduling [36], planning [29], bioinformatics [118], and many others. These solvers typically deal only with problems that contain existentially quantified variables.

2.4.2 ALP Interpretation of a Society Specification

Definition 2.4.1 (Abductive Instance). *An* instance $\mathcal{S}_{\boldsymbol{HAP}}$ *of a society/abductive specification* $\mathcal{S}$ *is represented as an ALP, i.e., a triple* $\langle P, \mathcal{E}, \mathcal{IC}_{\mathcal{S}} \rangle$ *where:*

- P *is the SOKB together with the history of happened events* $\boldsymbol{HAP}$*;*
- $\mathcal{E}$ *is the set of* abducible predicates *of* $\mathcal{S}$*;*
- $\mathcal{IC}_{\mathcal{S}}$ *are the social integrity constraints of* $\mathcal{S}$*.*

In this way, our social framework (and its dynamic counterpart, as instance of a society) has been smoothly given an abductive interpretation.

If the society is goal driven, then there exists a goal G at the society level (which is simply *true* if the society is not goal driven).

Definition 2.4.2 *Given two instances, $\mathcal{S}_{\boldsymbol{HAP}}$ and $\mathcal{S}_{\boldsymbol{HAP'}}$, of a society $\mathcal{S}$, $\mathcal{S}_{\boldsymbol{HAP'}}$ is a* proper extension *of $\mathcal{S}_{\boldsymbol{HAP}}$ if and only if $\boldsymbol{HAP} \subset \boldsymbol{HAP'}$.*

Definition 2.4.3 *Given an instance, $\mathcal{S}_{\boldsymbol{HAP}}$, of a specification $\mathcal{S}$, the instance is* closed *iff it has no proper extensions. We denote a closed instance as $\mathcal{S}_{\overline{\boldsymbol{HAP}}}$.*

In the following, we indicate a closed history by means of an overline: $\overline{\mathbf{HAP}}$. Notice that in a closed instance, we assume that no further event might occur (i.e., the instance has no further extensions and the history is closed under CWA).

2.4.3 Declarative Semantics

We describe then the (abductive) declarative semantics of the $\mathcal{S}$CIFF framework, which is inspired by other abductive frameworks, but introduces the concept of fulfilment, used to express a correspondence between the expected and the actual observations. The declarative semantics of an abductive/social specification is given for each specific history.

In this way, $\mathcal{S}_{\mathbf{HAP}^i}$, $\mathcal{S}_{\mathbf{HAP}^f}$ will denote different instances of the same abductive specification $\mathcal{S}$, based on two different histories: $\mathbf{HAP}^i$ and $\mathbf{HAP}^f$. We adopt an abductive semantics for the society instance. The abductive computation produces a set Δ of hypotheses, which is partitioned in a set ΔA of general hypotheses and a set **EXP** of *expectations*. The set of abduced literals should entail the goal and satisfy the integrity constraints.

Definition 2.4.4 (Abductive Explanation).

Given an abductive specification $\mathcal{S} = \langle SOKB, \mathcal{IC}_{\mathcal{S}} \rangle$*, an instance* $\mathcal{S}_{\boldsymbol{HAP}}$ *of* $\mathcal{S}$*, and a goal* $\mathcal{G}$*,* Δ *is an abductive explanation of* $\mathcal{S}_{\boldsymbol{HAP}}$ *if:*

$$Comp(SOKB \cup \boldsymbol{HAP} \cup \Delta) \cup \text{CET} \ \cup T_{\mathcal{X}} \models \mathcal{IC}_{\mathcal{S}} \tag{2.4.3}$$

$$Comp(SOKB \cup \Delta) \cup \text{CET} \ \cup T_{\mathcal{X}} \models \mathcal{G} \tag{2.4.4}$$

where Comp represents the completion *of a theory,* CET *is Clark's Equational Theory [48], and* $T_{\mathcal{X}}$ *is the theory of constraints [92].*

The symbol $\models$ is interpreted in three valued logics, as it is in the IFF Proof Procedure. We also require consistency with respect to explicit negation [21] and between positive and negative expectations.

Definition 2.4.5 *A set* $\boldsymbol{EXP}$ *of expectations is* $\neg$-consistent *if and only if for each (ground) term p:*

$$\{\boldsymbol{E}(p), \neg\boldsymbol{E}(p)\} \not\subseteq \boldsymbol{EXP} \qquad \textit{and} \quad \{\boldsymbol{EN}(p), \neg\boldsymbol{EN}(p)\} \not\subseteq \boldsymbol{EXP}. \tag{2.4.5}$$

Definition 2.4.6 *A set* $\boldsymbol{EXP}$ *of expectations is* **E**-consistent *if and only if for each (ground) term p:*

$$\{\boldsymbol{E}(p), \boldsymbol{EN}(p)\} \not\subseteq \boldsymbol{EXP} \tag{2.4.6}$$

The following definition establishes a link between happened events and expectations, by requiring positive expectations to be matched by events, and negative expectations not to be matched by events.

Definition 2.4.7 *Given a history* $\boldsymbol{HAP}$*, a set* $\boldsymbol{EXP}$ *of expectations is* **HAP**-fulfilled *if and only if*

$$\forall \boldsymbol{E}(p) \in \boldsymbol{EXP} \Rightarrow \boldsymbol{H}(p) \in \boldsymbol{HAP} \qquad \forall \boldsymbol{EN}(p) \in \boldsymbol{EXP} \Rightarrow \boldsymbol{H}(p) \notin \boldsymbol{HAP} \tag{2.4.7}$$

Otherwise, ***EXP*** *is* **HAP**-violated.

When all the given conditions (2.4.3-2.4.7) are met, we say that the goal is *achieved* and **HAP** is compliant to $\mathcal{S}_{\mathbf{HAP}}$ with respect to $\mathcal{G}$, and we write $\mathcal{S}_{\mathbf{HAP}} \models_{\Delta} \mathcal{G}$. In the remainder of this thesis, when we simply say that a history **HAP** is compliant to an abductive specification $\mathcal{S}$, we will mean that **HAP** is compliant to $\mathcal{S}$ with respect to the goal *true*. We will often say that a history **HAP** *violates* a specification $\mathcal{S}$ to mean that **HAP** is not compliant to $\mathcal{S}$. When **HAP** is apparent from the context, we will often omit mentioning it.

Example 2.4.1 Consider the *query_ref* abductive specification $\mathcal{S} = \langle SOKB, \mathcal{IC}_{\mathcal{S}} \rangle$, where $SOKB$ and $\mathcal{IC}_{\mathcal{S}}$ are defined in Tab. 2.3.1. The history

$$\begin{aligned} \{&\mathbf{H}(tell(alice, bob, query\_ref(phone\_number), dialog\_id), 10), \\ &\mathbf{H}(tell(bob, alice, inform(phone\_number, 5551234), dialog\_id), 12)\} \end{aligned} \tag{2.4.8}$$

is compliant to $\mathcal{S}$.

2.5 The SCIFF Proof Procedure

The operational semantics of $\mathcal{S}$CIFF is given by an abductive proof procedure. Since the language and declarative semantics of the $\mathcal{S}$CIFF framework are closely related with the IFF abductive framework [82], the $\mathcal{S}$CIFF proof procedure has also been inspired by the IFF proof procedure. However, some modifications were necessary. As a result, $\mathcal{S}$CIFF is a substantial extension of IFF, and the main differences between the frameworks are, in a nutshell:

- $\mathcal{S}$CIFF supports the dynamical happening of events, i.e., the insertion of new facts in the knowledge base during the computation;
- $\mathcal{S}$CIFF supports universally quantified variables in abducibles;
- $\mathcal{S}$CIFF supports quantifier restrictions;
- $\mathcal{S}$CIFF supports the concepts of fulfilment and violation (see Def. 2.4.7).

2.5.1 Data Structures

The $\mathcal{S}$CIFF proof procedure is based on a rewriting system transforming one node to another (or to others). In this way, starting from an initial node, it defines a proof tree.

A node can be either the special node *false*, or defined by the following tuple

$$T \equiv \langle R, CS, PSIC, \Delta A, \Delta P, \mathbf{HAP}, \Delta F, \Delta V \rangle. \tag{2.5.1}$$

We partition the set of expectations **EXP** into the confirmed (ΔF), disconfirmed (ΔV), and pending (ΔP) expectations. The other elements are:

- R is the resolvent: a conjunction, whose conjuncts can be literals or disjunctions of conjunctions of literals
- CS is the constraint store: it contains CLP constraints and quantifier restrictions
- $PSIC$ is a set of implications, called partially solved integrity constraints
- ΔA is the set of general abduced hypotheses (the set of abduced literals, except those representing expectations)
- **HAP** is the history of happened events, represented by a set of events, plus a *open/closed* attribute (see transition *closure* in the following)

If one of the elements of the tuple is *false*, then the whole tuple is the special node *false*, which cannot have successors. In the following, we indicate with Δ the set $\Delta A \cup \Delta P \cup \Delta F \cup \Delta V$.

2.5.2 Initial Node and Success

A derivation D is a sequence of nodes

$$T_0 \rightarrow T_1 \rightarrow \cdots \rightarrow T_{n-1} \rightarrow T_n.$$

Given a goal $\mathcal{G}$, a set of social integrity constraints $\mathcal{IC}_\mathcal{S}$, and an initial history $\mathbf{HAP}^i$, we build the first node in the following way:

$$T_0 \equiv \langle \{\mathcal{G}\}, \emptyset, \mathcal{IC}_\mathcal{S}, \emptyset, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$$

i.e., the resolvent R is initially the query ($R_0 = \{G\}$) and the set of partially solved integrity constraints $PSIC$ is the set of integrity constraints ($PSIC_0 = \mathcal{IC}_\mathcal{S}$).

The other nodes $T_j, j > 0$, are obtained by applying the transitions that we will define in the next section, until no further transition can be applied (we call this last condition *quiescence*).

Definition 2.5.1 *Given an instance* $\mathcal{S}_{\boldsymbol{HAP}^i}$ *of an abductive specification* $\mathcal{S} = \langle SOKB, \mathcal{IC}_\mathcal{S} \rangle$ *and a set* $\boldsymbol{HAP}^f \supseteq \boldsymbol{HAP}^i$ *there exists a* successful derivation *for a goal* G *iff the proof tree with root node* $\langle \{G\}, \emptyset, \mathcal{IC}_\mathcal{S}, \emptyset, \emptyset, \boldsymbol{HAP}^i, \emptyset, \emptyset \rangle$ *has at least one leaf node*

$$\langle \emptyset, CS, PSIC, \Delta A, \Delta P, \boldsymbol{HAP}^f, \Delta F, \emptyset \rangle$$

where CS *is consistent, and* ΔP *contains only negations of expectations* $\neg \boldsymbol{E}$ *and* $\neg \boldsymbol{EN}$*. In such a case, we write:*

$$\mathcal{S}_{\boldsymbol{HAP}^i} \vdash_\Delta^{\boldsymbol{HAP}^f} \mathcal{G}.$$

From a non-failure leaf node N, answers can be extracted in a very similar way to the IFF proof procedure. Answers of the $\mathcal{S}$CIFF proof procedure are called *expectation answers*. To compute an abductive answer, a substitution σ' is computed such that

- σ' replaces all variables in N that are not universally quantified by a ground term
- σ' satisfies all the constraints in the store CS_N.

If the constraint solver is (theory) complete [92] (i.e., for each set of constraints c, the solver always returns *true* or *false*, and never *unknown*), then there will always exist a substitution σ' for each non-failure leaf node N. Otherwise, if the solver is incomplete, σ' may not exist. The non-existence of σ' is discovered during the answer

extraction phase. In such a case, the node N will be marked as a failure node, and another success node can be selected (if there is one).

Definition 2.5.2 (Abductive Answer). *Let $\sigma = \sigma'|_{vars(G)}$ be the restriction of σ' to the variables occurring in the initial goal $\mathcal{G}$. Let $\Delta_N = (\Delta F_N \cup \Delta P_N \cup \Delta A_N)\sigma'$. The pair (Δ_N, σ) is the* abductive answer *obtained from the node N.*

2.5.3 Variables Quantification and Scope

Concerning variable quantification, $\mathcal{S}$CIFF differs from IFF in the following aspects:

- in IFF, all the variables that occur in the resolvent or in abduced literals are existentially quantified, while the others (that occur only in implications) are universally quantified; in $\mathcal{S}$CIFF, variables that occur in the resolvent or in abducibles can be universally quantified (as **EN** expectations can contain universally quantified variables);
- in IFF, variables in an implication are existentially quantified if they also occur in an abducible or in the resolvent, while in $\mathcal{S}$CIFF variables in implications can be existentially quantified even if they do not occur elsewhere.

For these reasons, in the $\mathcal{S}$CIFF proof procedure the quantification of variables is explicit.

The scope of the variables differs depending on where they occur:

- if they occur in the resolvent or in abducibles, their scope is the whole tuple representing the node (see Sect. 2.5.1);
- otherwise they occur in an implication; their scope, in such a case, is the implication in which they occur.

In the first case, we say that the variable is *flagged*. In the following, when we want to make explicit the fact that a variable X is flagged (when it is not clear from the context), it will be indicated with $\hat{X}$, while if we want to highlight that it is not flagged, it will be indicated with $\check{X}$.

Copy of a formula Since the $\mathcal{S}$CIFF syntax allows for abducibles with both existentially and universally quantified variables, the classical concept of renaming of a formula should be extended. Intuitively, universally quantified variables are renamed, in a sense, doubling the original formula, while existentially quantified variables are not. Let us call this operation *copy* of the formula.

When making a copy of a formula, we keep into account the scope of the variables it contains by means of their flagging status, as follows.

Definition 2.5.3 *Given a formula F, we call* copy *of F a formula F' where the universally quantified variables and the non flagged variables are renamed. We write*

$$F' = copy(F).$$

For example,

$$\exists_{\hat{Y}} \forall_{\hat{X}'>50} \forall_{\check{Z}'} \mathbf{E}(p(\hat{Y})) \wedge \mathbf{EN}(q(\hat{X}', \hat{Y})) \wedge [\mathbf{EN}(r(\hat{Y}, \check{Z}')) \rightarrow \exists_{\check{K}'} \mathbf{E}(p(\check{K}'))]$$

is a copy of the formula:

$$\exists_{\hat{Y}} \forall_{\hat{X}>50} \forall_{\check{Z}} \mathbf{E}(p(\hat{Y})) \wedge \mathbf{EN}(q(\hat{X}, \hat{Y})) \wedge [\mathbf{EN}(r(\hat{Y}, \check{Z})) \rightarrow \exists_{\check{K}} \mathbf{E}(p(\check{K}))]$$

Notice that, by Definition 2.5.3, if F contains only flagged existentially quantified variables, then $copy(F) \equiv F$ (so, for instance, the selected literal of SLD resolution

would not be renamed, as in SLD resolution), while a universally quantified formula would be renamed (for instance, a clause would be renamed, as in SLD resolution).

Intuitively, by copying a formula we obtain a new fresh copy (unrelated to previous ones) of universally quantified variables and non flagged variables.

2.5.4 Transitions

The transitions are based on those of the IFF proof procedure, enlarged with those of CLP [91], and with specific transitions accommodating the concepts of fulfilment, dynamically growing history and consistency of the set of expectations with respect to the given definitions (Defs. 2.4.5, 2.4.6 and 2.4.4).

Here, for sake of completeness of the presentation of the $\mathcal{S}$CIFF Framework, we will briefly cite and define the transitions of the Proof Procedure. The interested reader can refer to [15] for the complete and detailed presentation of the transitions.

IFF-like transitions

The IFF proof-procedure. The IFF 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 its *inference rules*. By such 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 solved) integrity constraints. Given a formula, its variables' quantification is defined 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 negated atom *not* 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 *not* A, the variables in A necessarily occur also in a positive atom, and must be considered existentially quantified. The inference rules which IFF is based on are: *Unfolding*, *Propagation*, *Splitting*, *Case analysis*, *Factoring*, *Rewrite rules for equality*, *Logical simplifications*. In the following, we will show how these IFF transitions are adapted for the purposes of $\mathcal{S}$CIFF.

Unfolding. Is adapted from the IFF proof-procedure. Let L_i be the selected literal in the resolvent $R_k = L_1, \ldots, L_r$. Suppose that L_i is a predicate defined in the $SOKB$ of the social specification. Unfolding generates a child node for each of the definitions of L_i; in each node, L_i is replaced with its definition.

Moreover, as in the IFF proof procedure, unfolding is also applied to a defined atom in the body of an implication. In this case, only one child node is generated, which contains a new implication for each definition of the atom.

Abduction. Since the $\mathcal{S}$CIFF proof procedure (differently from the IFF) keeps the set of abducibles separate from the resolvent, a transition has been introduced for

abduction which, intuitively, moves an abducible from the resolvent to the set of abduced atoms ($\Delta A \cup \Delta P$).

Propagation. Given a partially solved PSIC and a literal A (an happened event or an abducible) that unifies with a literal L_i in the body of the PSIC, a new node is generated where an equality constraint is imposed between A' and L_i, and a new $PSIC'$ is added (where L_i has been removed). A' is the copy of A, ($A' = copy(A)$), and the equality will be handled by transition *Case Analysis*.

Splitting. Given a node where the resolvent R_k contains a disjunction, two new child nodes are generated, each one containing only one of the disjunct atoms of the parent node.

In the $\mathcal{S}$CIFF proof procedure, disjunctions may appear also in the constraint store. Depending on the type of underlying Constraint Solver, clever reasoning can be possible. For instance, when using a CLP(FD) solver, constructive disjunction [148] or the cardinality operator [147] can be used to handle disjunctions of constraints. If the adopted constraint solver does not provide such facilities, *Splitting* can be applied also to disjunctions in the store.

Case Analysis. Given a node with an implication

$$PSIC_k = PSIC' \cup \{A = B, L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j\}$$

the node is replaced by two identical nodes, except for the following: in Node 1 we hypothesize that the equality $A = B$ holds, while in Node 2, we hypothesize the opposite. Since our proof procedure also needs to deal with constraints in the body, we also extend case analysis to such situation.

Factoring. In the IFF proof procedure, transition factoring separates answers in which abducible atoms are merged from answers in which they are distinct. It is important for keeping the set of assumptions small (ideally, minimal). It generates two nodes: in one node two hypotheses unify, in the other one a constraint is imposed in order to avoid the unification of the hypotheses.

In the $\mathcal{S}$CIFF proof procedure, abducibles can contain universally quantified variables; it is not reasonable to unify atoms with universally quantified variables, because we would lose some of the information given by the abduced atoms.

For this reason, we apply *factoring* only if the two atoms only contain existentially quantified variables. Notice that this coincides with the factoring transition of the IFF proof procedure.

Equivalence Rewriting. The equivalence rewriting operations are delegated to the constraint solver. Note that a constraint solver works on a constraint domain which has an associated interpretation. In addition, the constraint solver should handle the constraints among terms derived from unification. Therefore, beside the specific constraint propagation on the constraint domain, we assume that the constraint solver is equipped with further inference rules for coping with unification. Moreover, we also have to consider that our language is more expressive than that of the IFF proof-procedure, as we can abduce atoms with universally quantified variables. For this reason, we introduced *flagged* variables, and we deal with them in the theory of unification.

Logical Equivalence. Intuitively, when the body of a $PSIC$ becomes *true*, then a new child node is generated, where $PSIC$ is removed and its head is added to the

resolvent R. Moreover, also all the logic equivalence rules of the IFF are considered in this transition.

Dynamically growing history

This set of transitions deals with a dynamically growing history **HAP**. The transitions are used to reason upon the happening (or non-happening) of events.

Closure. In order to reason about non-happening of events, we adopt Closed World Assumption (CWA, [123]) on the set of currently happened events. Of course, this assumption is not acceptable if other events will happen in the future. For this reason, we non-deterministically assume that no other event will happen, i.e., we generate two child nodes. In the first we assume that no other events will happen, in the second that there will be other events. The *open/closed* attribute of the history (see Sect. 2.5.1) records if closed world is assumed on the happening of events.

Transition *Closure* is only applicable when no other transition is applicable. In other words, it is only applicable at the quiescence of the set of the other transitions.

Happening of Events. The happening of events is handled by a transition *Happening*. This transition takes an event $\mathbf{H}(Event)$ from an external queue and puts it in the history **HAP**; the transition *Happening* is applicable only if an *Event* such that $\mathbf{H}(Event) \notin \mathbf{HAP}$ is in the external queue.

Non-Happening. The *Non-Happening* transition can be considered an application of *constructive negation*. Constructive negation is a powerful inference that is particularly well suited in CLP [141].

Rule *Non-Happening* applies when the history is closed and a literal *not* **H** is in the body of a PSIC. Given a node where:

- $PSIC_k = \{not\ \mathbf{H}(E_1), L_2, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_m\} \cup PSIC'$
- closed($\mathbf{HAP}_k$) = $true$

Non-Happening produces a new node. Intuitively, we hypothesise that all the events matching with E_1 that are not in the history, do not happen at all. Intuitively, we hypothesise that every event that would be able to match with E_1, and is not in the current history, will not happen. This can be seen as abducing an atom $non\mathbf{H}(E_1')$ where all the variables are substituted with universally quantified variables. We impose that the hypothesis holds in all cases except those already in the $\mathbf{HAP}_k$; we can state this by means of the quantifier restrictions, i.e., we impose that the hypothesis $non\mathbf{H}(E_1')$ does not unify with any of the happened events. This is equivalent to imposing a conjunction (for all the events in the history that match with E_1') of a disjunction (for all the variables appearing in E_1') of non unification restrictions (written $\neq$).

Fulfilment and Violation

These transitions nondeterministically try and match expectations with events. In general, these transitions generate two child nodes: in one we assume that one expectation and one event match, while in the other we assume they will not match.

Violation EN. Given a node N with the following situation:

- $\Delta P_k = \Delta P' \cup \{\mathbf{EN}(E_1)\}$
- $\mathbf{HAP}_k = \mathbf{HAP'} \cup \{\mathbf{H}(E_2)\}$

Violation ***EN*** produces two nodes N^1 and N^2, where N^1 is as follows:

- $\Delta V^1_{k+1} = \Delta V_k \cup \{\mathbf{EN}(E_1)\}$
- $CS^1_{k+1} = CS_k \cup \{E_1 = E_2\}$

and N^2 is as follows:

- $\Delta V^2_{k+1} = \Delta V_k$
- $CS^2_{k+1} = CS_k \cup \{E_1 \neq E_2\}$

Fulfilment E. Starting from a node N as follows:

- $\Delta P_k = \Delta P' \cup \{\mathbf{E}(Event_1)\}$
- $\mathbf{HAP}_k = \mathbf{HAP'} \cup \{\mathbf{H}(Event_2)\}$

Fulfilment ***E*** builds two nodes, N^1 and N^2, that are identical to their father except for the following: in node N^1 we hypothesise that the expectation and the happened event unify; in node N^2 instead we hypothesise that the two will not unify.

Violation E. Violation of an **E** expectation can be proven only if there will not be an event matching the expectation. It is possible when we assume that no other event will happen; i.e., either when the transition *Closure* has been applied, or when a deadline has expired. The **E** atom that is violated is then added to the ΔV set in the new child node.

Fulfilment EN. Symmetrically to violation **E**, we can prove fulfilment of **EN** expectations, either when the history **HAP** is closed, or when a deadline has expired and the correspondent event didn't happen at all.

Consistency

E-Consistency. In order to ensure **E**-consistency (see Def. 2.4.6) of the set of expectations, we impose the following integrity constraint:

$$\mathbf{E}(T) \wedge \mathbf{EN}(T) \rightarrow \mathit{false} \tag{2.5.2}$$

¬-Consistency. In order to ensure ¬-consistency (see Def. 2.4.5) of the set of expectations, we impose the following integrity constraints:

$$\begin{array}{lcll} \mathbf{E}(T) & \wedge & \neg\mathbf{E}(T) & \rightarrow \mathit{false} \\ \mathbf{EN}(T) & \wedge & \neg\mathbf{EN}(T) & \rightarrow \mathit{false} \end{array} \tag{2.5.3}$$

CLP

The $\mathcal{S}$CIFF proof-procedure inherits the same transitions of CLP [91]. We suppose that the symbols $=$ and $\neq$ are in the constraint language and the theory behind them is, for equality, the same used by the *Equivalence Rewriting* transition. Concerning $\neq$, we will again suppose that it is possible to syntactically distinguish the CLP-interpreted terms and atoms; the solver will perform some inference on the interpreted terms (typically, depending on the CLP sort, e.g., by deleting inconsistent values from domains in CLP(FD)), and will moreover contain the rules for uninterpreted terms.

The constraint solver deals also with quantifier restrictions. If a quantifier restriction (due to unification) gets all the variables existentially quantified, then we replace it with the corresponding constraint.

Constrain. Given a node with

- $R_k = L_1, \ldots, L_r$

and the selected literal, L_i is a quantifier restriction, *Constrain* produces a node with

- $R_{k+1} = L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_r$
- $CS_{k+1} = CS_k \cup \{L_i\}$

Infer. Given a node, the transition *Infer* modifies the constraint store by means of a function *infer*(CS). This function is typical of the adopted constraint sort. E.g., the function *infer* in a FD (Finite Domain) sort will typically compute (generalised) arc-consistency.

- $CS_{k+1} = \mathit{infer}(CS_k)$

Consistent. Given a node, the transition *Consistent* will check the consistency of the constraint store (by means of a solver of the domain) and will generate a new node. The new node can either be a special node *fail* or a node identical to its father. Again, this transition is typical of the chosen constraint solver: in CLP(FD), for example, failures are discovered when a domain is empty.

2.5.5 Implementation of the SCIFF Proof Procedure

As its ancestor, the IFF proof procedure [82], the $\mathcal{S}$CIFF proof-procedure is a transition system that rewrites logic formulae into equivalent logic formulae. Each formula is a *Node* of the proof-procedure, and can be rewritten into one or more nodes, logically in OR (so building an OR-tree). The $\mathcal{S}$CIFF proof-procedure has more features: it accepts dynamically incoming events (**H**), uses a constraint solver, generates expectations (**E**, **EN**). For these reasons, elements in a formula (node) are arranged in a tuple which is more structured than the node of the IFF, and that carries the

following information:

$$T \equiv \langle R, CS, PSIC, \Delta P, \mathbf{HAP}, \Delta F, \Delta V \rangle \tag{2.5.4}$$

where R is the resolvent, CS is the constraint store (as in CLP), PSIC is a set of implications (initially set as the set of all integrity constraints), **HAP** is the current history, ΔP, ΔF, and ΔV are, respectively, the set of pending, fulfilled, and violated expectations. For the implementation of the $\mathcal{S}$CIFF proof-procedure, SICStus PROLOG [137] has been chosen.

As the IFF proof-procedure, the $\mathcal{S}$CIFF proof-procedure specifies a mechanism for building proof trees, leaving the search strategy to be defined at implementation level. The implementation is based on a depth-first strategy. This choice, enabling us to tailor the implementation for the built-in computational features of PROLOG, allows for a simple and efficient implementation of the proof-procedure. Experiments in a practical application (namely, combinatorial auctions) show that the proof-procedure is scalable enough to address real-life size situations. The PROLOG-Constraint Handling Rules (CHR, [81]) module implements the transitions of the proof-procedure. The data structures of the proof-procedure (e.g., PSIC, ΔP) are implemented as CHR constraints, so the transitions can be straightforwardly implemented as CHR rules. For example, each happened event is represented by means of a `h/2` CHR constraint, whose (ground) arguments are the content and the time of the event. An example of event is:

```
h(request(seller,buyer,give(10€),1),10am)
```

Expectations are represented by means of CHR constraints `e` for **E** expectations and `en` for **EN** expectations. CHR interfaces easily with other constraint solvers, so we

can impose constraints on the variables, such as:

```
e(do(buyer,seller,give(10€),1),T), T<5pm
```

and this expectation will not match with (and be fulfilled by), for example, a happened event

```
h(do(buyer,seller,give(10€),1),8pm).
```

Given this representation, the $\mathcal{S}$CIFF transitions can be mapped into CHR rules, in a sense defining a new constraint solver for the resolution of expectations. For example, we have a transition of **E**-consistency, that ensures that the final derivation node does not contain both the expectations of an event to happen and to not happen:

```
e_consistency @
        e(E1,T1), en(E2,T2)
        ==>
        reif_unify((E1,T1),(E2,T2),0).
```

Such a rule, for each pair $(\mathbf{E}(E_1, T_1), \mathbf{EN}(E_2, T_2))$, imposes the dis-unification constraint $(E_1, T_1) \neq (E_2, T_2)$ (`reif_unify`(T_1, T_2, B) is a constraint that imposes unification between two terms T_1 and T_2 according to a boolean variable B: the logical reading is $(T_1 = T_2) \Leftrightarrow (B = 1)$). The fulfillment rule is also rather straightforward:

```
fulfilment @
    h(HEvent,HTime), e(EEvent,ETime)
    ==>
    may_unify(HEvent,EEvent)
    |
    renaming((EEvent,ETime),(EEvent1,ETime1)),
    case_analysis_fulfilment((HEvent,HTime),(EEvent1,ETime1)).
```

The rule is applied when an event and a pending expectation whose content have the same functor and arity (checked by the `may_unify/2` predicate in the guard of the

rule) are in the CHR store. In this case, a renaming is made of the expectation[5] and the `case_analysis_fulfillment/2` predicate is called. Two nodes are created by `case_analysis_fulfillment/2`:

- a first node where unification is imposed between the expectation and the event, the `e(EEvent,ETime)` constraint for the expectation is removed from the constraint store and the `fulf(e(EEvent,ETime))` CHR constraint is imposed (implementing the fact that the expectation is moved from the set ΔP of pending expectations to the ΔF one of fulfilled expectations);
- and a second node where dis-unification between the expectation and the event is imposed.

[5]This step is necessary because some expectations may contain universally quantified variables. The issue is discussed in detail in a technical report [14].

2.6 Properties of the SCIFF proof procedure

2.6.1 Soundness of the SCIFF Proof Procedure

Here we will report only the main results about the soundness of the $\mathcal{S}$CIFF Proof Procedure. The interested reader can refer to [85] for the complete and detailed proofs of soundness.

Premises and definitions

First of all, we need to distinguish between closed histories and open histories (see Def. 2.4.2 and Def. 2.4.3). This distinction reflects upon the concepts of a goal $\mathcal{G}$ *achievable* or *achieved*. In particular:

Definition 2.6.1 ***Goal achievability*** *Given an open instance of an abductive specification, $\mathcal{S}_{\boldsymbol{HAP}}$, and a ground goal G, we say that G is* achievable *(and we write $\mathcal{S}_{\boldsymbol{HAP}} \not\approx_{\Delta} G$) iff there exists an (open) admissible set of abducibles Δ (and whose $\boldsymbol{EXP} \subseteq \Delta$ is also* fulfilled*) such that:*

$$SOKB \cup \boldsymbol{HAP} \cup \Delta \models G \tag{2.6.1}$$

(which is a shorthand for $Comp(SOKB \cup \Delta) \cup \boldsymbol{HAP} \cup CET \models G$).

Definition 2.6.2 ***Goal achievement*** *Given a closed instance of an abductive specification, $\mathcal{S}_{\overline{\boldsymbol{HAP}}}$, and a ground goal G, we say that G is* achieved *(and we write $\mathcal{S}_{\overline{\boldsymbol{HAP}}} \vdash_{\Delta} G$) iff there exists a (closed) admissible set of abducibles Δ (and whose $\boldsymbol{EXP} \subseteq \Delta$ is also* fulfilled*) such that:*

$$SOKB \cup \overline{\boldsymbol{HAP}} \cup \Delta \models G \tag{2.6.2}$$

(i.e., $Comp(SOKB \cup \overline{\boldsymbol{HAP}} \cup \Delta) \cup CET \models G$*).*

Recalling the deifinitions of open/closed successfull derivation of the proof, we are now ready to enunciate the main soundness results.

Soundness Properties of the $\mathcal{S}$CIFF Proof Procedure

The following theorem relates the operational notion of open successful derivation with the corresponding declarative notion of goal achievability.

Theorem 2.6.1 Open Soundness. *Given an open instance* $\mathcal{S}_{\boldsymbol{HAP}^i}$*, if*

$$\mathcal{S}_{\boldsymbol{HAP}^i} \mid\sim_{\Delta}^{\boldsymbol{HAP}^f} G$$

with abductive answer (Δ, σ) *then*

$$\mathcal{S}_{\boldsymbol{HAP}^f} \mid\approx_{\Delta\sigma} G\sigma$$

The theorem above states that if there exists an open successful derivation for a goal G starting from an initial history $\mathbf{HAP}^i$ and leading to the (open) instance $\mathcal{S}_{\mathbf{HAP}^f}$ with abduced set Δ, and with expectation answer (Δ, σ), then $G\sigma$ is achievable in $\mathcal{S}_{\mathbf{HAP}^f}$ (with the abduced set $\Delta\sigma$).

In the closed case, the soundness property is stated as follows, relating the operational notion of closed successful derivation with the corresponding declarative notion of goal achievement.

Theorem 2.6.2 Closed Soundness. *Given a closed instance* $\mathcal{S}_{\overline{\boldsymbol{HAP}^f}}$*, if*

$$\mathcal{S}_{\boldsymbol{HAP}^i} \vdash_{\Delta}^{\overline{\boldsymbol{HAP}^f}} G$$

with abductive answer (Δ, σ) *then*

$$\mathcal{S}_{\overline{\boldsymbol{HAP}^f}} \models_{\Delta\sigma} G\sigma$$

Soundness in the closed case states that if there exists a closed successful derivation for a goal G starting from an initial history $\mathbf{HAP}^i$ and leading to the closed instance $\mathcal{S}_{\overline{\mathbf{HAP}^f}}$ with abduced set Δ, and with abductive answer (Δ, σ), then $G\sigma$ is achieved in $\mathcal{S}_{\overline{\mathbf{HAP}^f}}$ (with the abductive set $\Delta\sigma$).

2.6.2 Termination of the SCIFF Proof Procedure

As already states previously, the $\mathcal{S}$CIFF Proof Procedure is an extension of the IFF Proof Procedure [82]. The termination property for the IFF was proven by Xanthakos [154], by establishing sufficient conditions on the IFF program for guaranteeing the termination of the proof procedure. Here we will present a set of restrictions that, if applied to a $\mathcal{S}$CIFF program, are indeed a sufficient condition for the termination. Then, we will enunciate the termination property that holds under such restrictions. The interested reader can refer to [85] for the detailed proof.

New restrictions on the $\mathcal{S}$CIFF Proof Procedure

We give here the equivalent of the restrictions proposed by Xanthakos.

Splitting Citing Xanthakos:

> The first new restriction we enforce is the (exhaustive) application of splitting on any disjunctions in a node (i.e. whenever possible, splitting should be applied after an unfolding, propagation, case analysis, or previous splitting step). Then, any execution tree is an or-tree where any node is a conjunction of literals, implications and at most one disjunction $D_1 \vee \cdots \vee D_n$, where each D_i is a conjunction of literals and implications.

Equality rewriting and logical simplification Citing Xanthakos[154]:

> The second restriction that we pose is that we give logical simplification and equality rewrite rules the highest priority, i.e. they should be applied whenever possible.

Equality rewriting is substituted in the $\mathcal{S}$CIFF proof-procedure by more general transition rules, called *Constraint Solving*. We impose that *Constraint Solving* transitions are applied (together with logical simplification) before the other transitions (i.e., they have highest priority).

Case Analysis Citing Xanthakos [154]:

> Some equalities in the body of implications are not dealt with by equality rewrite rules, but by case analysis. Our third restriction is that case analysis is given the highest priority (after equality rewriting and logical simplification have been performed) when an implication is selected. Similarly to equality rewriting, we enforce that the left-most equality is selected first. This restriction simplifies the implications in a node and may also reduce the computational cost.

We take the same restriction proposed by Xanthakos. Notice that in the $\mathcal{S}$CIFF proof-procedure, Case Analysis can also be applied to a constraint in the body of an implication.

Assumptions on the Constraint Solver

As in Constraint Logic Programming [91], the Constraint Solving is not completely specified in the $\mathcal{S}$CIFF proof procedure. In order to prove termination, we need to make some assumption on the Constraint Solver.

Definition 2.6.3 Assumptions on the Constraint Solver

- *The constraint solving process always terminates*
- *The constraint solving process cannot generate an infinite constraint store*
- *If the constraint solving process generates a disjunction of constraints* $CS = (c_1 \vee c_j) \wedge CS'$ *then splitting can be applied. We require that the alternation of Constraint Solving and splitting always terminates.*
- *The constraint solving process will not change the quantification of a variable (a variable universally quantified will not become quantified existentially and vice-versa).*
- *The constraint solving process can change a literal* L *into* L'*, but the new version,* L' *must be an instance of the previous version,* L*.*

Thanks to these assumptions, we can now state the following lemma:

Lemma 2.6.1 *Constraint Solving steps cannot cause other transitions, except*

- *Case Analysis*
- *failing transitions.*

Moreover, an infinite sequence of case analysis and constraint solving steps is impossible.

Acyclicity for $\mathcal{S}$CIFF programs

Definition 2.6.4 *Given a SOKB, an atom L* depends *on a literal M w.r.t. SOKB if*

- *an instance of a clause in SOKB is $L\theta \leftarrow K \wedge M$, or*
- *an instance of a clause in P is $L\theta \leftarrow K \wedge N$ and N depends on M*

where K is a conjunction of literals, possibly true, and $L\theta$ is an instance of L.

Given a logic program P, an atom L weakly depends *on a literal M wrt SOKB if*

- *L is M, or*
- *L depends on M wrt SOKB.*

Note that, since we interpret a specification $\mathcal{S}$ by means of an abductive logic program (see Sect. 2.4.2, Def. 2.4.1), the logic program P is the union of the $SOKB$ and of the set **HAP** of happened events.

We report here some definitions given by Xanthakos, adapted to our terminology.

Definition 2.6.5 *Given a $SOKB$, two literals L, M are* related *w.r.t. an atom N if an instance of a clause in P is $N\theta \leftarrow K, L', M'$ (where K is a conjunction of literals, possibly true, and $N\theta$ is an instance of N) and L' weakly depends on L and M' weakly depends on M.*

Intuitively, two literals are related w.r.t. a goal, if a sequence of unfolding steps for the goal can lead to the introduction of a node with both literals.

Definition 2.6.6 *Given a SOKB, a* level mapping $||$ *is a function that maps all ground atoms in B_{SOKB} (where B_{SOKB} is the Herbrand base of the logic program SOKB) to $\mathbb{N} \setminus \{0\}$ and false to 0. Also, $||$ is extended to map a ground negative literal $\neg A$ to $|A|$, where $A \in B_{SOKB}$.*

Given the definitions above, we can introduce the definition of acyclic implication, properly restated in our terminology:

Definition 2.6.7 (Acyclic implication) *Given a society with SOKB acyclic w.r.t. a level mapping $||$, a ground implication, say $L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_m$, is called* acyclic w.r.t. *SOKB* and $||$*, if for every non-constraint atom L_i, $i = 1, \ldots, n$, for every ground atom K which L_i weakly depends upon w.r.t. SOKB,*

- $|K| > |H_r|$, $r = 1, \ldots, m$ *and*
- $|K| > |N|$, *for every non constraint atom N such that some L_j, $j = 1, \ldots, i-1, i+1, \ldots, n$ depends upon the negative literal $\neg N$ and*
- $|K| > |N|$, *for every non equality atom N such that K is related to $\neg N$ wrt L_i.*

An implication is called acyclic w.r.t. *SOKB* and $||$ *if every ground instance of it is acyclic w.r.t. $||$. An implication is called* acyclic w.r.t. *SOKB if it is acyclic w.r.t. some level mapping.*

The definition of Acyclic Implication considers CLP constraints as an extension of the concept of unification (as is usual in CLP [91]). In other words, constraints are

not assigned a level; this is reasonable, because they do not depend upon definitions, nor upon integrity constraints, but their semantics is defined only by the underlying constraint theory.

We now extend the notion of acyclicity to the society knowledge (which is the definition of acyclic ALP [154, Def 4.2.5 pag 65] rewritten in our terminology)

Definition 2.6.8 Acyclic Society Specification

- *Given a logic program P that is acyclic w.r.t. a level mapping $||$, a negative defined literal $\neg N$ is called* acyclic w.r.t. P and $||$ *if the implication $N \rightarrow false$ is acyclic w.r.t. P and $||$. A negative defined literal is called* acyclic w.r.t. P *if it is acyclic w.r.t. some level mapping.*

- *An* Abductive Specification $\mathcal{S}$ is acyclic w.r.t. a level mapping $||$ *if*

  1. *$SOKB$ is acyclic w.r.t. $||$*
  2. *all negative defined literals in $SOKB$ are acyclic w.r.t $SOKB$ and $||$*
  3. *every implication in $\mathcal{IC}_{\mathcal{S}}$ is acyclic wrt $SOKB$ and $||$.*

  $\mathcal{S}$ is called acyclic *if it is acyclic w.r.t. some level mapping.*

- *A query G to an abductive specification $\mathcal{S}$ where the $\mathcal{S}$ is acyclic w.r.t. some level mapping $||$, is called* acyclic w.r.t. $\mathcal{S}$ and $||$ *if every negative defined literal in G is acyclic w.r.t. $SOKB$ and $||$. $\mathcal{S}$ and G are then called acyclic w.r.t. $||$. An abductive specification $\mathcal{S}$ and a query G are called acyclic if they are acyclic w.r.t. some level mapping.*

Notice that the definition of *acyclic negative literal* is slightly different from the IFF, because the $\mathcal{S}$CIFF proof procedure does not rewrite *all* negative literals $\neg N$

to $N \rightarrow false$, but only the negative defined literals, while abducibles have explicit negation [86], and constraints depend on the solver (for example, $\neg(A < B)$ is typically rewritten as $A \geq B$). Thus, literals $\neg\mathbf{E}$, $\neg\mathbf{EN}$ and $\neg c$ (where c is a constraint) are always acyclic.

Termination properties

We state the theorem of termination for a *"static version* of $\mathcal{S}$CIFF proof-procedure, i.e., for a version of $\mathcal{S}$CIFF that does not have Happening, non-Happening, and closure transitions. In other words, we proved termination for a version of $\mathcal{S}$CIFF provided with a static history.

Theorem 2.6.3 (Termination of static $\mathcal{S}$CIFF).
Let G be a query to a society $\mathcal{S}$, where $SOKB$, $\mathcal{IC}_\mathcal{S}$ and G are acyclic w.r.t. some level mapping, and G and all implications in $\mathcal{IC}_\mathcal{S}$ bounded w.r.t. the level-mapping. Then, every $\mathcal{S}$CIFF derivation for G, where transitions Happening, non-happening, and closure are not applied, for each instance of $\mathcal{S}$ is finite.

The termination for the dynamic case (i.e., where happening events can happen dynamically at run-time, and hence happening/non-happening transitions can be applied in any order), we need to assert two further assumptions. The first states that the new events will arrive only when the $\mathcal{S}$CIFF is in a stable state (i.e., new events are considered only if no other transition is applicable).

Definition 2.6.9 *A $\mathcal{S}$CIFF derivation has a* slow happening rate *if happening transitions apply only if no other transition is applicable.*

Non happening transitions are applicable only after closure of the history. We will assume that after closure of the history, non happening is applied as soon as possible (this can be seen as a *preprocessing*):

Definition 2.6.10 *A $\mathcal{S}$CIFF derivation has* non happening high priority *if, whenever non happening is applicable, it is indeed applied.*

We can now state our termination theorem for $\mathcal{S}$CIFF:

Theorem 2.6.4 (Termination of $\mathcal{S}$CIFF). *Let G be a query to a society $\mathcal{S}$, where $SOKB$, $\mathcal{IC}_\mathcal{S}$ and G are acyclic w.r.t. some level mapping, and G and all implications in $\mathcal{IC}_\mathcal{S}$ bounded w.r.t. the level-mapping.*

Then, every $\mathcal{S}$CIFF derivation with high priority for non happening and with slow happening rate for G, starting from an initial history $\boldsymbol{HAP}^i$ ending in a (possibly closed) finite final history $\boldsymbol{HAP}^f$ is finite.

2.6.3 Completeness of the SCIFF Proof Procedure

In the following we state the completeness results that have been achieved, respectively for the open and closed case. We do not present the proofs: the interested reader can refer to [83].

Theorem 2.6.5 (Open Completeness).

Given an open society instance $\mathcal{S}_{\boldsymbol{HAP}}$, and a (ground) goal G, for any set of ground abducibles, $\Delta = \boldsymbol{EXP} \cup \Delta A$, such that $\mathcal{S}_{\boldsymbol{HAP}} \not\approx_\Delta G$ then $\exists \Delta'$ such that $\mathcal{S}_\emptyset \not\sim^{\boldsymbol{HAP}}_{\Delta'} G$ with an expectation answer (Δ', σ) such that $\Delta'\sigma \subseteq \Delta$.

Completeness in the open case states that if goal G is achievable in an open society instance under the abducible set Δ, then an open successful derivation can be obtained for G, possibly computing a set Δ' of the abducibles whose grounding (according to the expectation answer) is a subset of Δ.

Theorem 2.6.6 (Closed Completeness).

Given a closed society instance $\mathcal{S}_{\overline{\boldsymbol{HAP}}}$*, a (ground) goal* G*, for any set of ground abducibles,* $\Delta = \boldsymbol{EXP} \cup \Delta A$ *such that* $\mathcal{S}_{\overline{\boldsymbol{HAP}}} \models_{\Delta} G$ *then* $\exists \Delta'$ *such that* $\mathcal{S}_{\emptyset} \vdash_{\Delta'}^{\boldsymbol{HAP}} G$ *with an expectation answer* (Δ', σ) *such that* $\Delta'\sigma \subseteq \Delta$*.*

Completeness in the closed case states that if goal G is achieved in a closed society instance under the abducible set Δ, then a closed successful derivation can be obtained for G, possibly computing a set Δ' of the abducibles whose grounding (according to the expectation answer) is a subset of Δ.

2.7 Related Works

In this section we relate the $\mathcal{S}$CIFF framework with other relevant work of literature. We will focus on other ALP frameworks and on other applications of computational logic to multi-agent systems. We do not intend to give an exhaustive account of the work done, but we will only touch the most closely related proposals and focus on the differences with respect with our own work.

2.7.1 ALP frameworks

By reading Kakas and colleagues' survey on ALP [96], one will be impressed by the amount of work done on this topic. We will try to relate our work with some of the most influential proposal of literature, although we are aware that many others will have to be left out.

Kakas and Mancarella [97] define a proof procedure (herein and below referred to as *KM*) for ALP, building on previous work by Eshghi and Kowalski [72]. *KM* assumes that the integrity constraints are in the form of *denials*, with at least one abducible literal in the conditions.[6] The semantics given by *KM* to the integrity constraints is that at least one of the literals in the integrity constraint must be false (otherwise, procedurally, *false* is derived). The procedure starts from a query 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 query. The proof procedure uses the notion of *abductive* and *consistency derivations*. Intuitively, an abductive derivation is a standard SLD-derivation suitably extended

[6] The syntax of integrity constraints varies from framework to framework; while some frameworks require integrity constraints to be denials of literals, this is not true of other frameworks, such as $\mathcal{S}$CIFF, and IFF, as we will see.

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 unifies with an abducible in it is satisfied. For this purpose, a consistency derivation for δ is started. Since the integrity constraints are denials only (i.e., queries), this corresponds to proving that every such query fails to hold. Therefore, δ is removed from all the denials with which it unifies, and it is proved that all the resulting queries fail. In this consistency derivation, 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 integrity constraint is satisfied.

Operationally, in *KM* abducibles must be ground when they are considered by the proof, and the procedure *flounders* if a selected abducible is 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. Therefore, extensions to *KM* have been proposed to cope with such limitations. Notably, *ACLP* [98] extends *KM* to deal with non-ground abduction and with constraints. ACLP programs can contain constraints on finite domains. ACLP interleaves consistency checking of abducible assumptions and constraint satisfaction.

Denecker and De Schreye [57, 59] introduce a proof procedure for normal abductive logic programs by extending SLDNF resolution to the case of abduction. The procedure is called *SLDNFA* and it is correct with respect to the completion semantics, and interestingly, it presents a crucial property: the treatment of *non-ground* abductive queries. [57] does not consider general integrity constraints, but only constraints of the kind $a, not\ a \Rightarrow false$. In later work [58], they propose adding integrity

constraints by extending the program with rules $false \leftarrow \neg F$, for each integrity constraint F; the literal $\neg false$ is then added as an extra literal to the query. SLDNFA has been extended towards CLP constraints handling, giving rise to SLDNFA(C) [149].

The $\mathcal{A}$-System [99] is a merger of ACLP and SLDNFA(C), but it differs from them by its explicit treatment of non-determinism, which permits to perform heuristic search with different types of heuristics. Also $\mathcal{A}$-System, like $\mathcal{S}$CIFF, copes with non-ground abduction.

The *Active-KM* proof procedure by Terreni *et al.* [110] integrates in the original abductive computational scheme a limited but powerful type of implicative-form integrity constraints. It supports forward reasoning via integrity constraints (implications) which fire when their conditions (body) are satisfied. However, this procedure does not deal with non-ground abducibles.

The *KM* proof-procedure has been also used and extended in the context of MAS. In particular, Ciampolini *et al.*'s ALIAS framework [46] and the LAILA language [47] define mechanisms for the coordination of agent reasoning based on it.

Surely the most related abductive framework to $\mathcal{S}$CIFF is Fung and Kowalski's IFF proof-procedure [82], on which $\mathcal{S}$CIFF is based. The IFF proof procedure uses backward reasoning with the selective Clark completion [48] of the logic program[7] to compute abductive explanations for given queries. Forward reasoning is applied based on the conjunction of queries plus integrity constraints, which is done at the beginning of the abductive process. The integrity constraints can be any (closed) implications. The authors describe IFF as a sort of "hybrid of the proof procedure

[7]The term "selective" refers to the fact that IFF does completion, but only of non-abducible predicates.

of Console *et al.* [51] and the SLDNFA procedure of Denecker and De Schreye (see [57]),” mainly due to its use of the Clark completion semantics and because neither of them requires a safe selection rule for abducibles and negation.

IFF has been used to model the rational part of logic-based agents, since Kowalski and Sadri’s seminal paper [102], and in further developments and refinements [103, 131, 106]. $\mathcal{S}$CIFF also applies ALP to the context of MAS, but differently from other work it does it at the social level, its initial purpose being to perform the compliance check of externally observable agent behaviour.

Recently, IFF has been refined to deal with negation as failure in integrity constraints [130], and extended with the definition of frameworks that treat abducibles and constraints uniformly [105, 69]. This last work also presents an implementation of IFF (the only one published, to the best of our knowledge), based on a meta-interpreter. Although these extensions improve IFF in several aspects, none of them handles universally quantified variables in abducible predicates, and of course do not deal with expectations. Finally, $\mathcal{S}$CIFF is implemented in CHR with attributed variables, which is a considerably efficient technology.

Given the CHR-based implementation of $\mathcal{S}$CIFF, we will also mention Abdennadher and Christiansen’s work [2], which further developed into the HYPROLOG system [45]. HYPROLOG is not limited to abduction, but also encloses assumptive logic programming features. The abductive part of HYPROLOG, however, is much more restrictive in scope than $\mathcal{S}$CIFF: it has a limited use of negation, and integrity constraints cannot involve defined predicates (but only abducibles and built-ins). Thanks to these simplifications, the necessary machinery is much simpler than the one used by $\mathcal{S}$CIFF. A subset of the $\mathcal{S}$CIFF language based on ideas similar to

HYPROLOG has been implemented at the beginning of the SOCS project: this is documented in [84, 11].

Finally, related to our work on ALP are the abductive query evaluation method proposed by Satoh and Iwayama [134], and *Abdual* [16]: a system to perform abduction from extended logic programs grounded on the well-founded semantics. Abdual, which relies on tabled evaluation inspired to SLG resolution [43], handles only ground programs.

A little bit outside of ALP, but related to our work, Sergot [135] proposed a framework, *query-the-user*, in which some of the predicates are labelled as "askable"; the truth of askable atoms can be asked to the user. Our **E** predicates may be understood as information asking, while **H** atoms may be considered as new information provided during search. However, differently from Sergot's query-the-user, $\mathcal{S}$CIFF is not intended to be used interactively, but rather to provide a means to generate and to reason upon generated expectations, be them positive or negative. Moreover, $\mathcal{S}$CIFF presents expectations in the context of an abductive framework (integrating them with other abducibles). Hypotheses confirmation was studied also by Kakas and Evans [74], where hypotheses can be corroborated or refuted by matching them with observable atoms: an explanation fails to be corroborated if some of its logical consequences are not observed. The authors suggest that their framework could be extended to take into account dynamic events, possibly, queried to the user: *"this form of reasoning might benefit from the use of a query-the-user facility"*. In a sense, our work can be considered as a merger and extension of these works: it has confirmation of hypotheses, as in corroboration, and it provides an operational semantics for dynamically incoming events, as in query-the-user.

Also related to reasoning with dynamic incoming events are two additional works, which we briefly mention before we conclude this roundup. Speculative Computation [133] is a propositional framework for a multi-agent setting with unreliable communication. When an agent asks a query, it also abduces a default answer; if the real answer arrives within a deadline, the hypothesis is (dis-)confirmed; otherwise the computation continues with the default. In our work, expectations can be confirmed by events, with a wider scope: they are not only questions, and they can have variables, possibly constrained. The dynamics of incoming events can be seen as an instance of an Evolving Logic Program [17]. In EvoLP, the knowledge base can change both because of external events or because of internal results. $\mathcal{S}$CIFF does not generate new events, but only expectations about external events. Our focus is more on the expressivity of the expectations than on the evolution of the knowledge base.

2.7.2 Computational Logic and societies of agents

To the best of our knowledge, the SOCS approach to agent societies, upon which $\mathcal{S}$CIFF found its main motivations, is the first attempt to use ALP to reason about agent interaction at a social level. Many other logics have been proposed to represent richer social and institutional entities, such as normative systems and electronic institutions. Here also the literature is broad, and slightly aside of the focus of this thesis. However, our work shares some concepts with normative systems, being **E** related with the $\mathcal{O}$ (obligation) operator of deontic logic [132], and **EN** with the $\mathcal{F}$ (forbidden) operator.[8] We enucleate similarities and differences in [12], and comment

[8]The reduction of deontic concepts such as obligations and prohibitions has been the subject of several past works: notably, by [19] (according to which, informally, A is obligatory iff its absence produces a state of violation) and by [112] (where, informally, an action A is prohibited iff its being performed produces a state of violation).

on the main differences between our approach and others based on social semantics in a number of published papers [10, 13, 8]. Below we will only give a very synthetic and by no means exhaustive account of work based on computational logic, applied to agent interaction and social agent systems in the broader sense.

The social approach to the semantic characterization of agent interaction is adopted by many researchers to allow for flexible, architecture-independent and verifiable protocol specification. Prominent schools, including Castelfranchi's [41], Singh *et al.*'s [138, 155], and Colombetti *et al.*'s [79, 49, 50] indicate commitments as first class entities in social agents, to represent the state of affairs in the course of social agent interaction. The resulting framework is more flexible than traditional approaches to protocol specification, as it does not necessarily define action sequences, nor it prescribes initial/final states or necessary transitions.

In [155], a variant of the Event Calculus [71] 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*. Similarly, [79] defines an operational specification of an ACL 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*, *canceled*, *conditional*, *active*, *fulfilled* and *violated*) can change by application of methods of the commitment class, or of rules triggered by external conditions. The semantics of communicative acts is specified in terms of methods to be applied to a commitment

when a communicative act is issued. The use of the SOCS framework for the social semantic specification of agent interaction protocols has been discussed in [10].

Artikis *et al.* [24] 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 [153, 145]. 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).

Differently from [24] (and from other work on normative systems), we do not explicitly represent concepts such as institutional power of the society members and validity of action. Instead, permitted are all social events that do not determine a violation, i.e., all events that are not explicitly forbidden are allowed. Permission instead, if explicitly needed, is mapped the negation of a negative expectation ($\neg\mathbf{EN}$).

[129] provides a first-order framework of deontic reasoning that can model and compute social regulations and norms, and among the organizational models, [60, 62, 61] exploit deontic logic to specify the society norms and rules. Several papers discuss

"sub-ideal" situations, i.e., how to manage situations in which some of the norms are not respected. For instance, [146] show the relation between diagnostic reasoning and deontic logic, importing the "principle of parsimony" from diagnostic reasoning into their deontic system, in the form of a requirement to minimize the number of violations. [121] proposes a solution to the problem and paradoxes stemming from earlier logical representations of *contrary-to-duty* obligations, i.e., obligations that become active when other obligations are violated. The Interactive Maryland Platform for Agents Collaborating Together (IMPACT) [22, 67] also uses deontic operators: not to describe social stances, but to program intelligent agents.

Chapter 3

Verifying Compliance by Observation: the SOCS-SI tool

In this Chapter we present how the components of the $\mathcal{S}$CIFF Framework have been exploited to practically apply the *Type 2* verification (Section 1.2) to Multi Agent Systems (MAS).

In particular, we present the *SOCS-SI* software tool (SI stands for Society Infrastructure): developed within the SOCS project, it has been extended to cope also with other scenarios than the initial MAS setting.

The key aspect of *Type 2* Verification is that in heterogenous systems (like the ones we are considering), it is not reasonable to assume that agents/peers internals are accessible. Therefore any verification process should be made on the *observable behaviour*, from an external viewpoint. The $\mathcal{S}$CIFF Proof Procedure (Section 2.5) already considered this viewpoint by taking in account the dynamic happening of events. The *SOCS-SI* tool uses the proof procedure to perform the reasoning (for the verification purposes), and provide monitoring facilities for accessing the external, observable behaviour.

Contribution of the author. The author contributed in a substantial way to the content of this chapter, and in particular to the development of the *SOCS-SI* tool. all the performances test on the $\mathcal{S}$CIFF Proof Procedure and on the *SOCS-SI* have been done in collaboration with Marco Alberti. Finally, the examples presented at the end of the chapter are the result of the collaboration with all the research group.

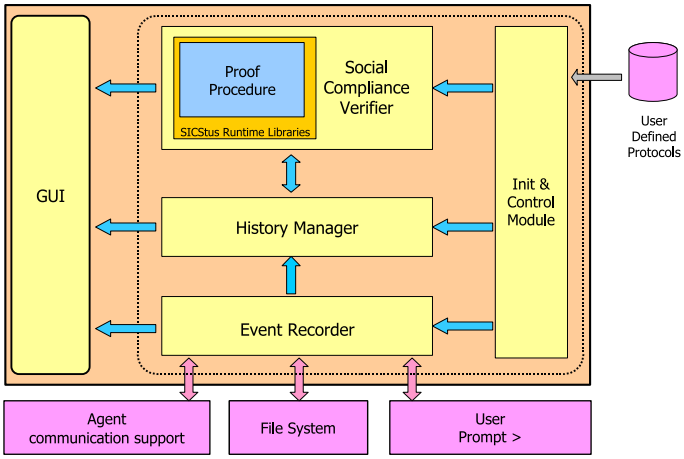
Chapter organization. The chapter is organized as follows. In Section 3.1 we describe the software *SOCS-SI*, while in Section 3.2 we discuss some performances results obtained through experimentation, on the proof procedure alone and in conjunction with the *SOCS-SI* tool.

Finally, in Section 3.3 we present few application examples of conformance verification on the observable behaviour of peers.

3.1 The *SOCS-SI* tool

The implementation of the *SOCS-SI* tool for compliance verification of agent interaction is composed of an implementation of the proof-procedure described in Section 2.5, interfaced to a Graphical User Interface and to a component for the observation of agent interaction. The *SOCS-SI* software application is composed of a set of modules. All the components except the proof-procedure have been implemented in the JAVA language. The core of *SOCS-SI* is composed of three main modules (see Fig. 3.1), namely:

- *Event Recorder*: fetches events from different sources and stores them inside the *History Manager*.
- *History Manager*: receives events from the *Event Recorder* and composes them into an "event history".
- *Social Compliance Verifier*: fetches events from the *History Manager* and passes them on to the proof-procedure in order to check the compliance of the history to the specification. It receives the expectations from the proof-procedure and visualizes them in the GUI.



In our model, agents communicate by exchanging messages, which are then translated into **H** events. The *Event Recorder* fetches events and records them into the *History Manager*, where they become available to the proof-procedure (see Section 2.5). As soon as the proof-procedure is ready to process a new event, it fetches one from the *History Manager*. The event is processed and the results of the computation are returned to the GUI. The proof-procedure then continues its computation by fetching another event if there is any available, otherwise it suspends, waiting for new events.

A fourth module, named *Init&Control Module* provides for initialisation of all the components in the proper order. It receives as initial input a set of protocols defined by the user, which will be used by the proof-procedure in order to check the compliance of agents to the specification.

The JAVA-PROLOG Interface

The main task of the JAVA portion of the *Social Compliance Verifier* is to interact with the proof-procedure. The SICStus Runtime libraries are accessed from JAVA using the Jasper package and native interfaces. All data exchanged between the JAVA side and the PROLOG program is translated into String objects. In order to process and filter the String objects, JAVA regular expressions are extensively used. These expressions are defined in a configuration file, loaded at initialisation time. Our software application can deal with different proof-procedure implementations and with different ACL performatives, without any a priori assumption about the format of the exchanged parameters. It is sufficient to properly re-define the regular expressions in the config file, and a new proof-procedure can be easily integrated into the software application.

The Recorder Interface

The *Event Recorder* fetches events from the external world using modules, each module being specialised for a specific source. We developed modules for interfacing with various agent platforms, starting with PROSOCS [37]. We are currently experimenting with other platforms: we had some successful experiments with JADE [32] and TuCSoN [125], and with checking compliance of e-mail messages. For testing and debugging purposes, we also developed modules to interact with the user prompt, as well as with the file system; it is possible to add as many specialised modules as desired, provided that they implement the interface `RecorderInterface`. In order to integrate our application with an already existing platform the user should:

1. create a JAVA class that implements the `RecorderInterface`
2. select it as message source during the application configuration (either through the configuration GUI, or by modifying the config file).

The `RecorderInterface` that we propose defines three methods, where the class `SOCSEvent` is our internal representation of events:

- `public SOCSEvent listen()`. Returns an instance of the `SOCSEvent` class if a message is available, or it waits (suspends) until a message arrives.
- `public long speak(SOCSEvent aMsg)`. Gives our application the capability to communicate with agents, by sending a message. It returns the time the message is sent.
- `public long getTime()`. Returns the current time. It is used to check temporal deadlines.

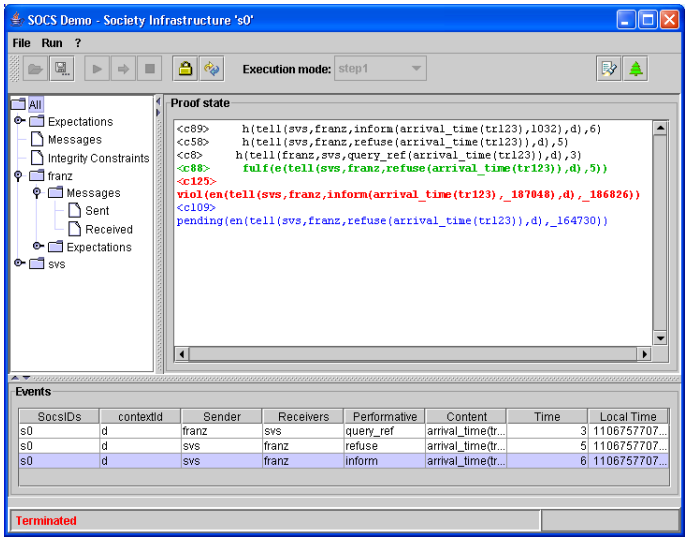


Figure 3.1: A screenshot of the application

The `RecorderInterface` has originally been defined as a subset of the low level communication API defined in the PROSOCS platform, which is used to perform controlled experiments in the context of global computing applications, within the SOCS project. However, one of the design specifications we strove to obtain was to have an interface general enough to allow integration with most agents platforms currently available.

The Graphical User Interface

The Graphical User Interface is implemented by using the Swing graphic library, and implements the Model-View-Control programming pattern. The main window is composed of three areas (or sub-windows), and of a button bar containing the controls (Figure 3.1). The bottom area contains the list of all the messages received by the *SOCS-SI*: the next message to be processed by the proof-procedure is emphasised (in Fig. 3.1 it is the third row, which is darker). The area on the left contains

the list of agents known by the society, i.e., agents that have performed at least one communicative action (coherently with the notion of openness by Davidsson [54]). The larger frame on the right contains the proof state, i.e., the results of the computation, returned by the proof-procedure. These results are expressed in terms of society expectations about the future behaviour of agents, and also in terms of fulfilled expectations and violations of social rules. By selecting an agent from the left pane, it is possible to restrict the information shown on the larger pane to be only that which is relevant to that particular agent. Among other features, it is possible to execute the application step-by-step, so that it elaborates one message at a time and then waits for a user acknowledge (similarly to debugger interfaces). Protocols are loaded into the tool by means of a button; they are simply provided as text files with a syntax strictly adherent to the formal one presented earlier. Finally, a tree-view of the whole computation is provided (Figure 3.2); interestingly, the shown tree bears both an operational and a logical interpretation. The operational interpretation is an intuitive graphical form of a log-file, showing the most significant computational steps, useful for debugging purposes. The logical meaning is an or-tree of the possible derivations timed by the incoming events. For each incoming event that enriches the knowledge base, the frontier of the explored proof-tree (which is a logical disjunction, as in various proof-procedures [82]) is shown. The user can inspect each of the nodes, and see in the main window the state of the computation, i.e., the tuple given in Eq. (2.5.4). The tuple is logically a conjunction of logical formulae of the types in the $\mathcal{S}$CIFF language: abducibles, constraints, literals, implications. Presentation of the frontier of the derivation tree is important for explanation reasons. Typically, logic languages can provide two types of answers: a *success/failure* answer and an

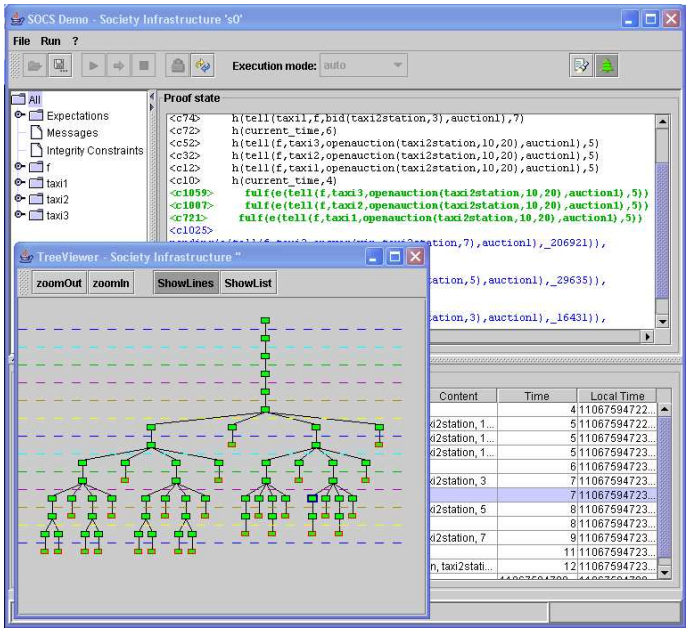


Figure 3.2: The Logic OR-Tree

explanation answer. In case of success, logic languages explain *why*: PROLOG returns simply a binding for the variables in the goal, CLP can return also constraints, and ALP (Abductive Logic Programming) returns a set of abducibles, just to name a few. But in case of failure, there is typically no explanation. The tree-view provides information also in case of failure: the set of failing nodes. In each node, the GUI shows underlined the cause of failure (e.g., a violated expectation or an unsatisfiable CLP constraint).

3.2 Performances of SCIFF and SOCS-SI

In this section we present some tests e have conducted, in order to better understanding the computational time required by $\mathcal{S}$CIFF/*SOCS-SI* to process some simple protocols and histories.The aim of this group of tests is not to establish absolute values about performances, but rather to understand how the computation time required to provide an answer is affected by changes in the length of the history processed (Section 3.2.1), and by changes in the alternatives dialogues allowed by the protocol (Section 3.2.2). The last experiment in particular roughly corresponds to increasing the breadth of the search tree explored by $\mathcal{S}$CIFF, whether the former corresponds to increasing the depth of the tree.

The output considered is only the computational time required to elaborate an answer. For test instances of certain dimension (see Tables 3.1 and 3.2), it was not possible to achieve the completion of the test, mainly for limitations of the hardware. This condition is expressed by placing a question mark in the results tables, instead of a value.

Qualitatively, the computational complexity of $\mathcal{S}$CIFF can be evaluated as follows. Each $\mathcal{S}$CIFF computation produces a search tree whose *depth* and *breadth* determine the total number of nodes, and thus the time needed to explore the (whole) tree. As the proof tree is explored by $\mathcal{S}$CIFF in a depth-first fashion, the depth of the tree, together with the *size* of a single node, also impacts on space requirements. For both time and space, the worst case is when each branch leads to failure, because in this case the whole tree is explored in search of a success node.

Intuitively, the *depth* of the search tree depends on the total number of *events* (the facts added dynamically to the knowledge base).

The *breadth* of the search tree, instead, is influenced by both the number of disjuncts in the head of the $\mathcal{S}$CIFF integrity constraints, and the alternative branches arising in several of the $\mathcal{S}$CIFF transitions. For example, one of the branches generated by the transition *Fulfillment* can be safely pruned, provided that the set $\mathcal{IC}_{\mathcal{S}}$ respects some syntactic conditions, whose discussion is beyond the scope of this paper. In such cases, it is possible to optimize the performance of $\mathcal{S}$CIFF by reducing the number of generated branches. In this paper, we call this optimized $\mathcal{S}$CIFF behaviour *f-deterministic*, as opposed to the *f-non-deterministic*, which does not perform the pruning.

In the following, we present three tests: the first and the second test (Sections 3.2.1 and 3.2.2) uses meaningless protocols specification, with the only purposes of understanding how the depth and the breadth of the search tree affect the required computational time. In Section 3.2.3 instead a third test is presented, where a meaningful protocol specification (a combinatorial auction) is taken as the playground for stressing the $\mathcal{S}$CIFF Proof Procedure. All the tests were designed to provide a positive answer by the $\mathcal{S}$CIFFProof Procedure, and were executed on a PC with a 2 GHz Pentium IV CPU, 512 MB of RAM, Windows XP Professional Edition and SICStus Prolog 3.10.1.

3.2.1 Increasing the depth of the explored derivation tree

In order to evaluate the impact of histories of various length on the $\mathcal{S}$CIFF Proof Procedure and on *SOCS-SI*, we have considered a very simple protocol, presented in the Specification 3.2.1, along with the history used to test it. The considered results, presented in Table 3.1, are the time required to the proof procedure and to *SOCS-SI* respectively to elaborate the histories of various length, and to provide the expected

positive answer. The used protocol does not contain any alternative (disjunction) in the head of the rule: each time an appropriate event is processed, a new expectation is generated and, if possible, fulfilled. The parameter varied was the number of messages (events) composing each history, and results are shown in Table 3.1.

Specification 3.2.1 The protocol and the histories used for testing when increasing the *depth* of the search tree.

$$
\begin{aligned}
&IC_1: \\
&\quad \mathbf{H}(tell(A, B, aQuestion(aParameter), D), T) \\
&\rightarrow \mathbf{E}(tell(B, A, anAnswer(aParameter), D), T_1) \ \wedge \\
&\quad T_1 \ \geq \ T
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{HAP}: \\
&\quad tell(a, b, aQuestion(aParameter), d_1, 1) \\
&\quad tell(b, a, anAnswer(aParameter), d_1, 1) \\
&\quad tell(a, b, aQuestion(aParameter), d_2, 2) \\
&\quad tell(b, a, anAnswer(aParameter), d_2, 2) \\
&\quad \ldots \\
&\quad tell(a, b, aQuestion(aParameter), d_x, x) \\
&\quad tell(b, a, anAnswer(aParameter), d_x, x)
\end{aligned}
$$

Due to the simplicity of the protocol used it is not correct to assume the results as meaningful in their absolute values. However, the test shows two significant aspects about the behavior of the proof and of *SOCS-SI*. First of all, the time required to elaborate longer histories increases in an almost quadratic way, as it is possible to observe in Figure 3.3. Secondly, the *SOCS-SI* has a big impact on performances, with respect to $\mathcal{S}$CIFF running without a GUI. Not only does *SOCS-SI* lower the maximum number of processable events before an "Out–of–Memory" error, but also the performances are worsened, with a factor that is not constant, but that tends to increase.

Table 3.1: Testing performances of $\mathcal{S}$CIFF Proof Procedure and *SOCS-SI* while increasing the depth of the search tree.

No. of Messages	*SOCS-SI* Time(sec.)	$\mathcal{S}$CIFF Proof Time(sec.)
2	0,13	0,42
4	0,18	0,43
20	0,68	0,51
30	0,98	0,51
50	1,82	0,57
100	6,13	0,76
150	13,13	0,97
200	21,68	1,35
250	33	1,62
300	47,05	1,99
400	82,47	3,00
1000	?	12,16
2000	?	41,21
3000	?	92,56
4000	?	165,89
5000	?	259,02

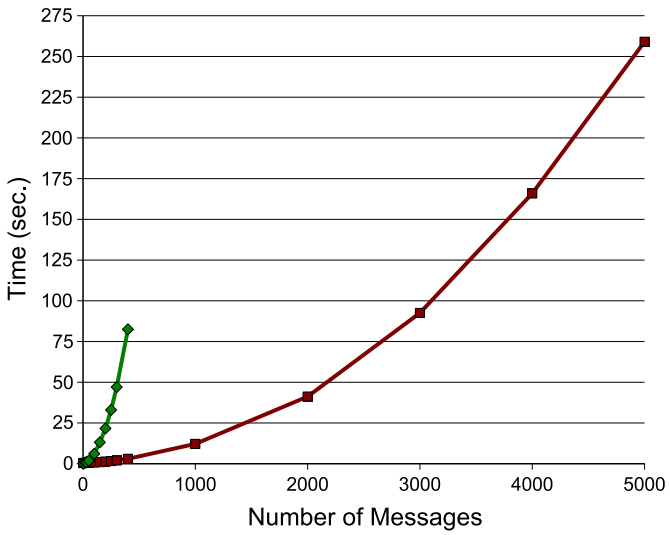


Figure 3.3: Performances with histories of increasing length.

3.2.2 Increasing the breadth of the exploration tree

The purpose of this test is to understand how the performance of the $\mathcal{S}$CIFF Proof and *SOCS-SI* changes if the breadth of the search tree increases. Here we vary the

Specification 3.2.2 The protocol and the history used for testing when increasing the *breadth* of the derivation tree.

$$
\begin{aligned}
&IC_1 : \\
&\quad \mathbf{H}(tell(A, B, aQuestion(aParameter), D), T) \\
&\rightarrow \mathbf{E}(tell(B, A, answer_1(aParameter), D), T_1) \ \wedge \\
&\quad T_1 \ \geq \ T \\
&\vee \mathbf{E}(tell(B, A, answer_2(aParameter), D), T_2) \ \wedge \\
&\quad T_2 \ \geq \ T \\
&\vee \ \ldots \\
&\vee \mathbf{E}(tell(B, A, end(aParameter), D), T_x) \ \wedge \\
&\quad T_x \ \geq \ T
\end{aligned}
$$

HAP :

$tell(a, b, aQuestion(aParameter), d_1, 1)$
$tell(b, a, end(aParameter), d_1, 2)$
$close\_history.$

protocol definition by increasing the number of disjuncts in the head of an Integrity Constraint; the history used, instead, is of a fixed length. The protocol, presented in Specification 3.2.2, is again a simple one: a subscript x indicates the total number of disjuncts. The history has been thought in order to fulfill the protocol only w.r.t. the expectation presented in the last disjunction. Since the $\mathcal{S}$CIFF Proof explores the search tree by expanding the possible branches following the order of occurrence of the disjuncts in the integrity constraint, this history forces the $\mathcal{S}$CIFF Proof Procedure to explore all the tree. The results obtained are presented in the Table 3.2.

Again, the absolute values are not really meaningful, due to the to simplicity of the protocol used. However, the results show that the time requirements increase with a almost quadratic coefficient w.r.t. the increasing of the disjunctions in the protocol. In Figure 3.4 it is possible to appreciate the overhead introduced by the *SOCS-SI*,

suggesting how much the GUI impacts on the overall performances.

Table 3.2: Testing performances of the $\mathcal{S}$CIFF Proof Procedure and *SOCS-SI* while increasing the depth of the derivation tree.

No. of Messages	*SOCS-SI* Time(sec.)	$\mathcal{S}$CIFF Proof Time(sec.)
2	0,032	0,015
5	0,047	0,016
10	0,093	0,031
20	0,219	0,063
30	0,375	0,109
40	0,578	0,188
50	0,859	0,282
100	2,813	0,921
200	10,968	3,360
300	25,423	7,516
400	46,390	12,937
500	71,496	19,875
1000	?	111,750

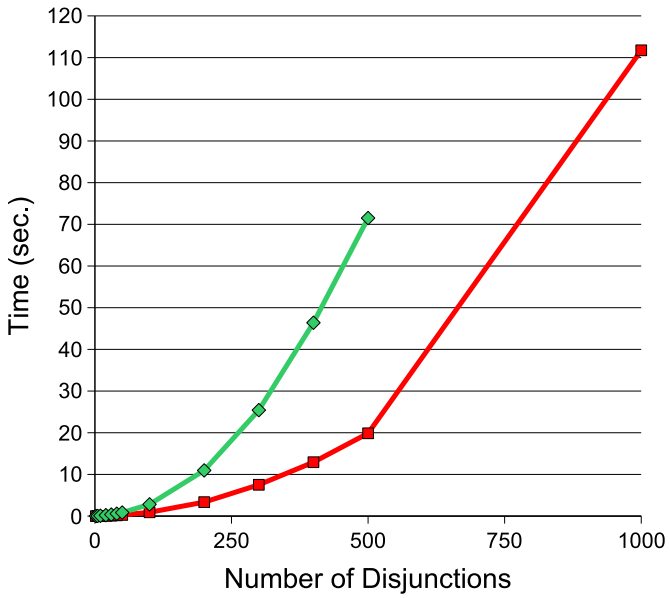


Figure 3.4: Performances with protocols with increasing number of alternatives.

We did not test how performances could have changed using the *f-deterministic* version of $\mathcal{S}$CIFF. In fact, in this dummy scenario, it has no sense to introduce

knowledge about the domain, and thus it is not possible to take advantage of the *f-deterministic* $\mathcal{S}$CIFF.

3.2.3 Tests in a real scenario

In this section, we show some experimental results obtained applying the $\mathcal{S}$CIFF Proof Procedure to the verification of compliance to the combinatorial auction protocols (described in [6]).

There exist different kinds of auctions. For this test, we consider *single unit reverse auctions*. In a *single unit auction*, the auctioneer wants to sell a set M of goods/tasks maximizing the profit. Goods are distinguishable. Each bidder j posts a bid B_j where a set S_j of goods/tasks $S \subseteq M$ is proposed to be bought at the price p_j, i.e., $B_j = (S_j, p_j)$. The *single unit reverse auction* is a single unit auction where the auctioneer wants to buy and bidders are suppliers. The protocol definition for the auction is given in Specification 3.2.3.

While not being an exhaustive experimentation, the results show the effect on the time costs of $\mathcal{S}$CIFF of the breadth and depth of the search tree. The tests have been performed varying the following parameters:

1. $\mathcal{S}$CIFF version (*f-non-deterministic* vs. *f-deterministic*);

2. the abductive program used. Specification 3.2.3 shows two versions of the IC 4, which are semantically equivalent (i.e., an agent behaviour that respects one will respect the other, and vice-versa), but are verified by $\mathcal{S}$CIFF in a computationally different way. IC_{4b} expresses with a disjunction in the head that the auctioneer can either declare a bid winning (first disjunct) or declare it losing (second disjunct). Instead in IC_{4a} this alternative is expressed by

Specification 3.2.3 Combinatorial Auction

IC_1 :
$\mathbf{H}(tell(B, A, bid(ItemList, P), D), T_{bid})$
$\rightarrow \mathbf{E}(tell(A, B, openauction(Items, T_{end}, T_{deadline}), D), T_{open})$
$\wedge\ T_{open} < T_{bid} \wedge T_{bid} \leq T_{end}.$

IC_2 :
$\mathbf{H}(tell(A, B, openauction(Items, T_{end}, T_{deadline}), D), T_1)$
$\wedge\ \mathbf{H}(tell(B, A, bid(ItemBids, P), D), \_)$
$\wedge\ not\ included(ItemBid, Items)$
$\rightarrow \mathbf{E}(tell(A, B, answer(lose, B, ItemBids, P), D), \_).$

IC_3 :
$\mathbf{H}(tell(A, B, openauction(Items, T_{end}, T_{deadline}), D), T_{open})$
$\rightarrow \mathbf{E}(tell(A, B, closeauction, D), T_{end}).$

IC_{4a} :
$\mathbf{H}(tell(B, A, bid(ItemList, P), D), T_{bid})$
$\wedge\ \mathbf{H}(tell(A, B, openauction(Items, T_{end}, T_{deadline}), D), T_{open})$
$\rightarrow \mathbf{E}(tell(A, B, answer(X, S, ItemList, P), D), T_{answer})$
$\wedge\ T_{answer} > T_{end} \wedge T_{answer} < T_{deadline} \wedge X :: [win, lose].$

IC_{4b} :
$\mathbf{H}(tell(B, A, bid(ItemList, P), D), T_{bid})$
$\wedge\ \mathbf{H}(tell(A, B, openauction(Items, T_{end}, T_{deadline}), D), T_{open})$
$\rightarrow \mathbf{E}(tell(A, B, answer(win, S, ItemList, P), D), T_{answer})$
$\wedge\ T_{answer} > T_{end} \wedge T_{answer} < T_{deadline}$
$\vee \mathbf{E}(tell(A, B, answer(lose, S, ItemList, P), D), T_{answer})$
$\wedge\ T_{answer} > T_{end} \wedge T_{answer} < T_{deadline}.$

IC_5 :
$\mathbf{H}(tell(A, B, answer(lose, S, ItemList, P), D), \_)$
$\rightarrow \mathbf{EN}(tell(A, B, answer(win, S, ItemList, P), D), \_).$

IC_6 :
$\mathbf{H}(tell(A, B_1, answer(win, B_1, ItemList, P), D), \_)$
$\wedge\ \mathbf{H}(tell(B_2, A, bid(ItemList', P), D), T_{bid})$
$\wedge\ B_1 \neq B_2 \wedge intersect(ItemList, ItemList')$
$\rightarrow \mathbf{EN}(tell(A, B_2, answer(win, B_2, ItemList', P'), D), \_).$

$SOKB$:
$included([], \_).$
$included([H|T], L) : -member(H, L), included(T, L).$
$intersect([X|\_], L) : -member(X, L).$
$intersect([\_|Tx], L) : -intersect(Tx, L).$

means of a domain variable: intuitively, the auctioneer must declare each bid *Answer*, where *Answer* can be either *win* or *lose*. Operationally, in the first case,

two branches are generated by $\mathcal{S}$CIFF; in the second case, only one branch is generated and the binding of the domain variable is delayed.

In particular, we measure the computation time for sequences of auctions with different numbers of bidders in the two following implementations of the protocol:

1. *f-non-deterministic* $\mathcal{S}$CIFF, protocol with disjunction (which we call the *first setup* of $\mathcal{S}$CIFF and protocol);

2. *f-deterministic* $\mathcal{S}$CIFF, protocol with no disjunction (which we call the *second setup* of $\mathcal{S}$CIFF and protocol).

The protocols have been run by varying the number N of bidders, in two different cases.

- In each run of the first case:

    1. the auctioneer sends an *openauction* message to each of the N bidders;
    2. each of the N bidders places a *bid*;
    3. the auctioneer issues a *closeauction* message to each of the N bidders;
    4. the auctioneer notifies each of the N bidders with either a *win* or a *lose* message,

    thus resulting in $4N$ total messages exchanged.

- In each run of the second case, the last notification to one of the bidders is missing, thus resulting in a violation of the protocol and $4N-1$ total messages.

Table 3.3: Combinatorial Auction case 1: Fulfillment

f-non-deterministic,disjunction		*f-deterministic,domain*	
Bidders	Time(sec.)	Bidders	Time(sec.)
5	1	5	1
10	1	10	1
15	2	15	2
20	3	20	6
25	4	25	8
30	6	30	10
35	9	35	15
40	10	40	18
45	12	45	23
50	21	50	30

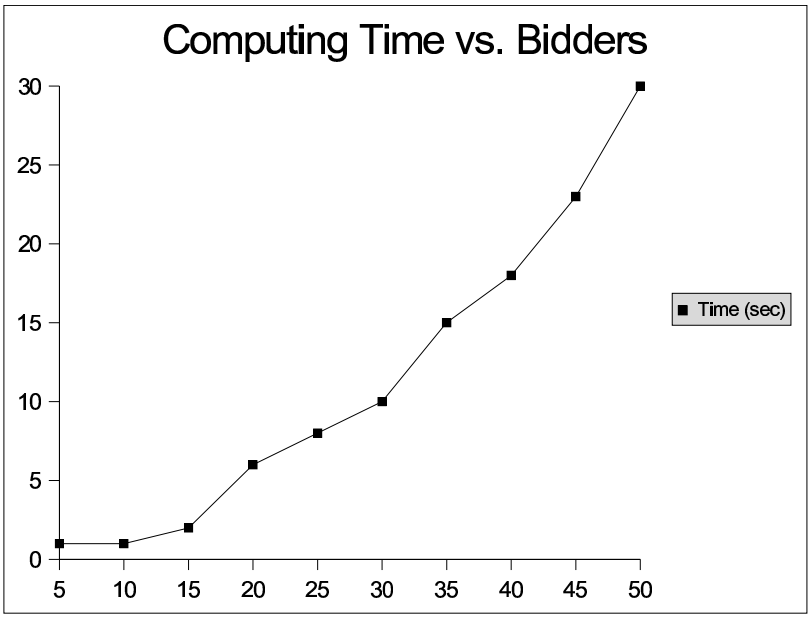


Figure 3.5: Proof performance on a basic auction (compliant)

In case of fulfillment (see Table 3.3), the first setup of $\mathcal{S}$CIFF and protocol seems to scale well with the number of bidders and, in fact, it achieves better execution timing than the second (also shown in Fig. 3.5).

This is basically due to the fact that the chosen setup of interactions directly leads to a successful $\mathcal{S}$CIFF derivation, and only one branch of the tree is explored.

Table 3.4: Combinatorial Auction case 2: Violation

f-non-deterministic,disjunction		*f-deterministic,domain*	
Bidders	Time(sec.)	Bidders	Time(sec.)
3	7	3	0
4	55	4	0
5	?	5	0
10	?	10	1
15	?	15	3
20	?	20	4
25	?	25	7
30	?	30	10
35	?	35	14
40	?	40	17
45	?	45	22
50	?	50	26

In the case of violation (see Table 3.4), however, the first setup of $\mathcal{S}$CIFF and protocol explodes for a very small number of bidders. The experiment with 5 bidders was suspended since this did not reach the answer of violation after several minutes of computing time; no experiments were performed with a higher number of bidders, which would have made things even worse. The second setup (also shown in Fig. 3.6), instead, scales very well also in case of violation. In this case, a CLP(FD) solver, written in CHR, directly manages the two alternative values for variable `Answer`.

The difference between the two setups of $\mathcal{S}$CIFF and protocol becomes apparent in the worst case (i.e., the case of violation) when the whole tree is explored. With the first setup, the choice points left open in case of fulfillment and the disjunctions in the head of the integrity constraint make the number of nodes in the proof tree explode even for small number of bidders. With the second setup, instead, the tree has only one branch, and is thus explored in a reasonable time when the number of bidders increases.

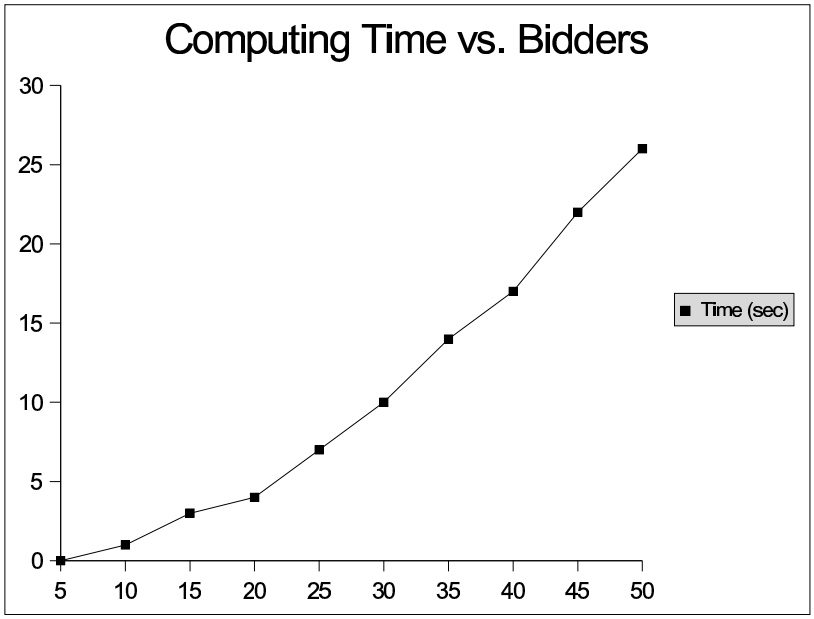


Figure 3.6: Proof performance on a basic auction (non compliant)

3.3 Applications of the Run-time Conformance Verification

In the following we present some real cases where we have successfully applied the $\mathcal{S}$CIFF Framework, and in particular the *SOCS-SI* tool. Several other scenarios have been studied and correspondent protocols have been specified using our formalism. The interested reader can refer to `http://wikiai.deis.unibo.it/index.php?title=SOCS_Protocol_Repository` for an online protocol repository.

3.3.1 The Opening phase of the Transmission Control Protocol

The Transmission Control Protocol [124] is one of the most known and used protocols for the transmission of data over an Internet Connection (over the IP protocol). It

has been published in the 1981, and since then several different implementations of the protocol stack have been proposed, developed and deeply tested.

Recently, with the advent of the "third generation" of mobile phones, the use of the TCP protocol has been adopted for supporting application protocols over radio connections, from the core network of the telecommunication providers to the user terminals. Each phone maker has equipped its products with its own TCP implementation.

Since some details of the TCP protocol have not been completely specified, it can happen that different phones exhibit slightly different behaviours when connecting to the core networks of telecommunications providers. In collaboration with one provider, we have formalized the opening phase of the TCP protocol, and we have studied the logs of the connections with *SOCS-SI*. Main objective of the analysis was to (possibly) identify non-compliances between the behaviour of the peers (traced in the form of event logs) and the formalization (based on $\mathcal{S}$CIFF) of the protocol.

We present here the ICs regarding the "three-way handshaking" open modality, that can be summarized as follows:

1. a peer A sends to another peer B a *syn* segment;
2. B replies by acknowledging (with a *ack* segment) A's *syn* segment, and by sending a *syn* segment;
3. A acknowledges B's *syn* segment with a *ack* segment, and starts sending data.

Specification 3.3.1 shows how the opening phase has been represented by means of the $\mathcal{S}$CIFF Language. In particular, IC_1 says that if A sends to B a *syn* segment, whose sequence number is $NSynA$, then B is expected to send to A an *ack* segment,

Specification 3.3.1 The Three-way Handshake opening phase of the TCP Protocol

$$
\begin{aligned}
&IC_1: \\
&\quad \mathbf{H}(tell(A, B, tcp(syn, null, NSynA, AckNumber), D), T1) \\
&\quad \rightarrow \mathbf{E}(tell(B, A, tcp(syn, ack, NSynB, NSynAAck), D), T2) \\
&\quad\quad \wedge\ NSynAAck = NSynA + 1\ \wedge\ T2 > T1. \\
&IC_2: \\
&\quad \mathbf{H}(tell(A, B, tcp(syn, null, NSynA, AckNumber), D), T1) \\
&\quad\quad \wedge\ \mathbf{H}(tell(B, A, tcp(syn, ack, NSynB, NSynAAck), D), T2) \\
&\quad\quad \wedge\ T2 > T1\ \wedge\ NSynAAck = NSynA + 1 \\
&\quad \rightarrow \mathbf{E}(tell(A, B, tcp(null, ack, NSynAAck, NSynBAck), D), T3) \\
&\quad\quad \wedge\ T3 > T2\ \wedge\ NSynBAck = NSynB + 1. \\
&IC_3: \\
&\quad \mathbf{H}(tell(A, B, tcp(syn, null, NSynA, ANY), D), T1) \\
&\quad\quad \wedge ta(TA) \\
&\quad \rightarrow \mathbf{EN}(tell(A, B, tcp(syn, null, NSynA, ANY), D), T2) \\
&\quad\quad \wedge\ T2 < T1\ \wedge\ T2 > T1 - TA. \\
\\
&SOKB: \\
&\quad ta(1000msec).
\end{aligned}
$$

whose acknowledgment number is $NSynA + 1$, at a later time. Moreover (three way handshake), B is expected to send (within the same message) a *syn* with another sequence number $NSynB$.

IC_2 says that, if the previous two messages have been exchanged, then A is expected to send to B an *ack* segment acknowledging B's *syn* segment, and with acknowledgement number is $NSynB + 1$, where $NSynB$ is the sequence number of B's *syn*.

The opening phase (restricted to the three way handshake) would be completely specified by the integrity constraints IC_1 and IC_2. However, within the collaboration with a telecom provider, some domain experts explicitly required to focus our attention on a problem they had previously spotted. The TCP protocol definition [124] explicitly states that if a first *syn* message has been sent and no *ack*message

has been received, it is allowed to repeat the initial *syn* message. Unfortunately, the specification does not specify the minimum time interval between each transmission of a *syn* message.

As a consequence, the follow situation can happen: a fast peer A send a *syn* message to a slower peer B. B's answer is delayed because its computational load is very high. As a consequence, A starts to re-transmit the *syn* message, causing problems to B (typically, a denial of service). In order to verify this hypothesis, the integrity constraint IC_3 has been added.

Specification 3.3.1 has been used to check the correctness of the interaction between mobile phones and a central server, taking the history from a log file. Evidence has been found that, after an initial *syn* message and no *ack* received, different mobile phones retransmit a *syn* with different timings (depending on different implementations). If the server does not answer rapidly enough, certain mobile phones repeats the *syn* message causing a denial of service on the server side. The use of the $\mathcal{S}$CIFF tools to this scenario has provided two results:

1. it was proved on the logs that a behaviour of certain mobile phones was responsible for the server problems (indeed human experts had already hypothesized the problem, but a punctual proof was appreciated);

2. it allowed to identify which phones exhibited that particular behaviour, paving the way for elaborating different solutions;

3. it allowed to establish a minimum time interval between each *syn* transmission; then this minimum time interval was used to define the Quality of Service (QoS) for the core network.

3.3.2 Run-Time Verification of Web Services Choreographies

Service Oriented Architectures (SOA) have recently emerged as a new paradigm for structuring inter-/intra- business information processes. While SOA is indeed a set of principles, methodologies and architectural patterns, a more practical instance of SOA can be identified in the Web Services technology, where the business functionalities are encapsulated in software components, and can be invoked through a stack of Internet Standards.

The standardization process of the Web Service technology is at a good maturation point: in particular, the W3C Consortium has proposed standards for developing basic services and for interconnecting them on a point-to-point basis. These standards have been widely accepted; vendors like Microsoft and IBM are supporting the technology within their development tools; private firms are already developing solutions for their business customer, based on the web services paradigm. However, the needs for more sophisticated standards for service composition have not yet fully satisfied. Several attempts have been made (WSFL, XLang, BPML, WSCL, WSCI), leading to two dominant initiatives: BPEL [20] and WS-CDL [151].

Both these initiatives however have missed to tackle some important issues. We agree with the view [28, 144] that both BPEL and WS-CDL languages lack of declarativeness, and more dangerous, they both lack an underlying formal model and semantics. Hence, issues like *run-time conformance testing*, *composition verification*, *verification of properties* are not fully addressed by the current proposals. Also semantics issues, needed in order to verify more complex properties (besides properties like livelock, deadlock, leak freedom, etc.), have been left behind.

Some of these issues have been already subject of research: generally, a mapping

between choreographed/orchestrated models to specific formalisms is proposed, and then single issues are solved in the transformed model. E.g., the *composition verification* is addressed in [27, 100]; *process mining* and *a-posteriori conformance testing* are addressed in [143]; livelock, deadlock, etc. properties are tackled in [116, 127].

Taking inspiration by the many analogies between the Web Services research field and the Multi Agent System (MAS) field [27], we have used the $\mathcal{S}$CIFF Framework for verifying at run-time (or a-posteriori using an *event log*) if the peers behave in a conformant manner w.r.t. a given choreography. Global choreographies have been defined by means of abductive specifications, and the conformance verification (also called *run-time behaviour conformance* in [28]) of the interactions have been performed by means of *SOCS-SI*.

To our purposes, let us consider a revised version of the choreography proposed in [28]. The choreography (shown in Figure 3.7) models a 3-party interaction, in which a supplier coordinates with its warehouse in order to sell and ship electronic devices. Due to some laws, the supplier should trade only with customers who do not belong to a publicly known list of banned countries.

The choreography starts when a $Customer$ communicates a purchase order to the $Supplier$. $Supplier$ reacts to this request asking the $Warehouse$ about the availability of the ordered item. Once $Supplier$ has received the response, it decides to cancel or confirm the order, basing this choice upon $Item$'s availability and $Customer$'s country. In the former case, the choreography terminates, whereas in the latter one a concurrent phase is performed: $Customer$ sends an order payment, while $Warehouse$ handles the item's shipment. When both the payment and the shipment confirmation

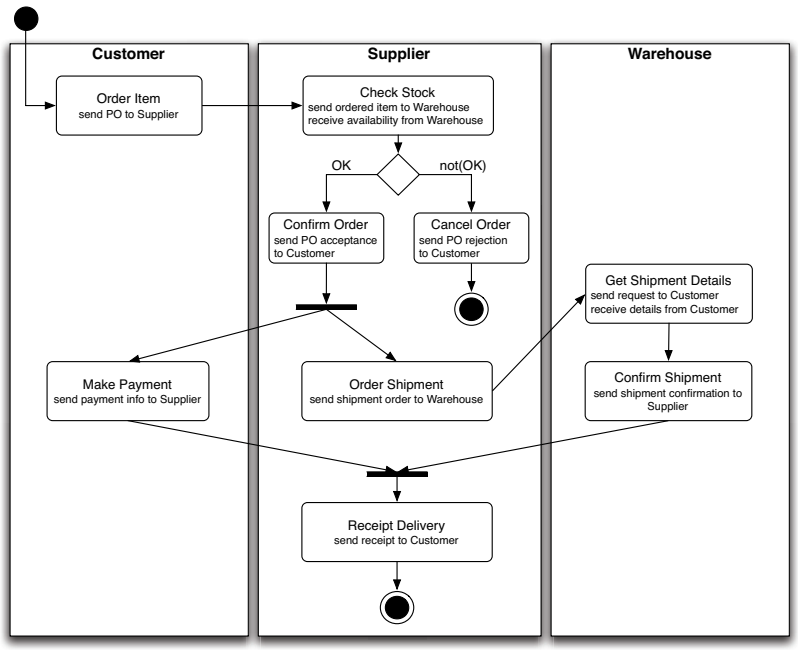


Figure 3.7: A Choreography example taken from [28]

are received by *Supplier*, it delivers a final receipt to the *Customer*. The specification of this choreography is given in Specification 3.3.2. The events are represented in the form $msgType(sender, receiver, content_1, \ldots, content_n)$, where the $msgType$, $sender$, $receiver$ and $content_i$ retain their intuitive meaning.

(IC_1) specifies that, when *Customer* sends to *Supplier* the purchase order, including the requested *Item* and his/her *Country*, *Supplier* should request *Item*'s availability to *Warehouse*. *Warehouse* should respond within 10 minutes to *Supplier*'s request giving the corresponding quantity *Qty* (IC_2). The deadline is imposed as a CLP constraint over the variable T_{qty}, that represents the time in which the response is sent.

After having received the requested quantity, *Supplier* decides whether to accept or reject *Customer*'s order (IC_3). As we have pointed out, the decision depends

Specification 3.3.2 Definition of the choreography shown in figure 3.7

$$\begin{aligned}&\mathbf{H}(purchase\_order(Customer, Supplier, Item, Country), T_{po})\\ \rightarrow&\mathbf{E}(check\_availability(Supplier, Warehouse, Item), T_{ca}) \wedge T_{ca} > T_{po}\end{aligned} \tag{IC_1}$$

$$\begin{aligned}&\mathbf{H}(check\_availability(Supplier, Warehouse, Item), T_{ca})\\ \rightarrow&\mathbf{E}(inform(Warehouse, Supplier, Item, Qty), T_{qty})\\ &\wedge T_{qty} > T_{ca} \wedge T_{qty} < T_{ca} + 10\end{aligned} \tag{IC_2}$$

$$\begin{aligned}&\mathbf{H}(purchase\_order(Customer, Supplier, Item, Country), T_{po})\\ &\wedge \mathbf{H}(inform(Warehouse, Supplier, Item, Qty), T_{qty})\\ \rightarrow&\mathbf{E}(accept\_order(Supplier, Customer, Item), T_{ao})\\ &\wedge ok(Qty, Country) \wedge T_{ao} > T_{po} \wedge T_{ao} > T_{qty}\\ \vee&\mathbf{E}(reject\_order(Supplier, Customer, Item), T_{ro})\\ &\wedge \neg ok(Qty, Country) \wedge T_{ro} > T_{po} \wedge T_{ro} > T_{qty}\end{aligned} \tag{IC_3}$$

$$\begin{aligned}&\mathbf{H}(accept\_order(Supplier, Customer, Item), T_{ao})\\ \rightarrow&\mathbf{E}(shipment\_order(Supplier, Warehouse, Item, Customer), T_{so})\\ &\wedge \mathbf{E}(payment(Customer, Supplier, Item), T_p) \wedge T_{so} > T_{ao} \wedge T_p > T_{ao}\end{aligned} \tag{IC_4}$$

$$\begin{aligned}&\mathbf{H}(shipment\_order(Supplier, Warehouse, Item, Customer), T_{so})\\ \rightarrow&\mathbf{E}(request\_details(Warehouse, Customer), T_{rd}) \wedge T_{rd} > T_{so}\end{aligned} \tag{IC_5}$$

$$\begin{aligned}&\mathbf{H}(request\_details(Warehouse, Customer), T_{rd})\\ \rightarrow&\mathbf{E}(inform(Customer, Warehouse, Details), T_{det}) \wedge T_{det} > T_{rd}\end{aligned} \tag{IC_6}$$

$$\begin{aligned}&\mathbf{H}(shipment\_order(Supplier, Warehouse, Item, Customer), T_{so})\\ &\wedge \mathbf{H}(inform(Customer, Warehouse, Details), T_{det})\\ \rightarrow&\mathbf{E}(confirm\_shipment(Warehouse, Supplier, Item), T_{cs}) \wedge T_{cs} > T_{so} \wedge T_{cs} > T_{det}\end{aligned} \tag{IC_7}$$

$$\begin{aligned}&\mathbf{H}(payment(Customer, Supplier, Item), T_p)\\ &\wedge \mathbf{H}(confirm\_shipment(Warehouse, Supplier, Item), T_{cs})\\ \rightarrow&\mathbf{E}(delivery(Supplier, Customer, Item, Receipt), T_{del}) \wedge T_{del} > T_{cs} \wedge T_{del} > T_p\end{aligned} \tag{IC_8}$$

$SOKB$:

```
ok( Qty, Country):-
    Qty>0,
    not banned_country( Country).

banned_country( shackLand).
banned_country( badLand).
```

upon the quantity and the *Country* the *Customer* belongs to; *Supplier* may accept the order only when *Qty* is positive and customer's *Country* is not in the list of banned countries. This last condition has been expressed using a predicate defined in the KB_{chor}, shown in Specification 3.3.2. If *Supplier* has accepted the purchase order, then *Customer* is expected to pay for the requested *Item* and, at the same time, *Supplier* will send a shipment order to *Warehouse*, communicating the involved *Item* and *Customer*'s identity (IC_4). *Warehouse* will use *Customer*'s identity in order to communicate with him/her and asking for shipment details (IC_5).

When *Customer* receives the request for details, then he/she is expected to respond giving his/her own *Details* (IC_6). After having received them, *Warehouse* should sends to *Supplier* a shipment confirmation (IC_7). Finally, (IC_8) states that when both the payment and the shipment confirmation actually happen *Supplier* is expected to deliver a *Receipt* to *Customer*.

Example of Run-Time Conformance Verification

In our scenario, the criminal *bankJob* beagle wants to buy a device from the online shop *devOnline*, whose warehouse is *devWare*. *devOnline* is quite greedy, and therefore trades with everyone, without checking if the customer comes from one of the banned countries. As a consequence, even if *bankJob* comes from *shackLand*, one of the banned countries, *devOnline* sells him the requested device, thus violating the choreography. Table 3.5 contains the log of the scenario from the viewpoint of *devOnline*; note that messages are expressed in high level way, abstracting from the SOAP exchange format.

When the first event (labeled m_1 in Table 3.5) happens, (IC_1) is triggered, and

Table 3.5: Log of messages exchanged by *devOnline* in our scenario

Id	message	sender	receiver	content	time
m_1	purchase_order	bankJob	devOnline	[device,shackLand]	2
m_2	check_availability	devOnline	devWare	[device]	3
m_3	inform	devWare	devOnline	[device,3]	10
m_4	accept_order	devOnline	bankJob	[device]	12
m_5	shipment_order	devOnline	devWare	[device,bankJob]	13
m_6	confirm_shipment	devWare	devOnline	[device]	16
m_7	payment	bankJob	devOnline	[device]	19
m_8	delivery	devOnline	bankJob	[device,receipt]	21

an expectation about *devOnline*'s behaviour is consequently generated:

$$\Delta P = \{ \quad \mathbf{E}(check\_availability(devOnline, Warehouse, device), T_{ca}) \wedge T_{ca} > 2\}$$

The happening of m_2 fullfills the pending expectation and matches with the body of (IC_2), generating a new one:

$$\Delta F = \{ \quad \mathbf{E}(check\_availability(devOnline, devWare, device), 3)\}$$

$$\Delta P = \{ \quad \mathbf{E}(inform(devWare, devOnline, device, Qty), T_{Qty})$$
$$\wedge T_{qty} > 3 \wedge T_{qty} < 13\}$$

The happening of m_3 fulfills the current pending expectation respecting the deadline. Moreover, it triggers (IC_3), and two different hypotheses are considered (acceptance and rejection of the order). However, since the predicate `ok(3,shackLand)` is evaluated by $\mathcal{S}$CIFF to false, only the expectation about the order rejection is considered:

$$\Delta F = \{ \quad \mathbf{E}(check\_availability(devOnline, devWare, device), 3),$$
$$\mathbf{E}(inform(devWare, devOnline, device, 3), 10)\}$$

$$\Delta P = \{ \quad \mathbf{E}(reject\_order(devOnline, bankJob, device), T_{ro})$$
$$\wedge T_{ro} > 3 \wedge T_{ro} > 10\}$$

As a consequence, when *devOnline* accepts the purchase order of *bankJob* sending the message m_4, the $\mathcal{S}$CIFF proof procedure detects a violation, since m_4 is not explicitly expected.

3.3.3 Medical guidelines

Medical guidelines [88] are clinical behaviour's recommendations that are used to support physicians in the definition of the most appropriate diagnosis and/or therapy within determinate clinical circumstances.

Unfortunately, guidelines are today described by using several formats, such as flow charts and tables, so that physicians are not properly supported in the detection of possible errors and incompleteness: it is difficult to evaluate who made an error within the protocol's flow and when. As a consequence, guideline's application often loses its benefits.

In the following we show that the logic-based formalism provided by the $\mathcal{S}$CIFF framework is general enough to allow us to formally describe medical protocols. The main advantage of using ICs in the context of medical guidelines is the capability to discover some forms of inconsistency and to perform an on-the-fly verification of the protocol's application on a specific patient.

In order to effectively test the potentialities of this approach, we formalized a microbiological guideline [38] which describes how to manage an infectious patient from his arrival at a hospital's emergency room to his recovery and tested this guideline on a set of clinical trials.

The guideline may be structured in seven phases: patient's arrival at the hospital's emergency room; patient examination at the emergency room; possible admission in a

specific hospital ward and first therapy prescription made by the ward physician; request of a microbiological test (consisting of many sub-phases, involving both human and artificial actors); return of the microbiological test report to the ward physician, who must decide the definitive therapy; management of drugs by nurses; evaluation of patient's health and, in case of symptoms persistence, new prescription of microbiological test. In order to formalize the guideline described before, we detected, first of all, all the actors involved (e.g. the patient, wards physicians, the microbiological laboratory, etc.) and secondly pointed out all the actions which should be executed (or not, i.e. expected or not expected) for an appropriate patient's disease treatment. Each actor has been then mapped into an agent with a specific role, and actors actions (e.g, examinations, analysis, etc) has been modeled as SOCS events. For example, the following IC:

$$\begin{aligned} &\mathbf{H}\left(enter\left(Patient, emergency\_ward\right), T_{ent}\right) \\ &\rightarrow \mathbf{E}\left(examinate\left(Physician, Patient\right), T_{exam}\right) \\ &\wedge T_{exam} < T_{ent} + 6 * 60 \end{aligned} \tag{3.3.1}$$

expresses that when a patient arrives at the emergency room (at time T_{ent}), we expect that at least one physician would visit him (at time T_{exam}) within the deadline of 6 hours. This deadline is expressed as a CLP constraint, which says that T_{exam} should be lower than T_{ent} plus 6 hours. The complete specification of this protocol consists of about 20 social ICs. It has been tested via the *SOCS-SI* software, using different set of events, compliant and not. For instance, a non compliant set is the following: a patient (*patientA*) arrives at the hospital's emergency room at time 10, but no physician visits him within 6 hours. The event

$$enter\left(patientA, emergency\_ward\right), 10$$

matches with the antecedent of (1), generating the expectation in the consequent that a physician should visit *patientA* at time T_{exam}, such that $T_{exam} < 10+6*60$. No event is afterward registered until this deadline, therefore a violation is raised by the proof procedure.

In this way a simple medical guideline may be mapped into a set of integrity constraints in the context of $\mathcal{S}$CIFF infrastructure, thus enabling an on-the-fly verification about the compliance of the hospital staff to it. We have successfully tested this specification using the *SOCS-SI* tool with some set of events, compliant and not.

In literature, several formalisms have been proposed for representing medical protocols, like for example GLARE [142] and PROforma [80]. These are complete tool capable to manage both guidelines acquisition and execution, but, to the best of our knowledge, their are not able to verify compliance of actions and interactions of the kind here presented.

3.3.4 E-learning by doing

E-learning is a new paradigm for the learning process, based on the growing availability of technology resources such as personal computers and the Internet. The main idea of e-learning consist of distributing the knowledge onto new media support like cd, dvd, or directly through the internet. Around this idea a set of support technologies have been developed, such as content management systems and applications for real-time streaming and interactions. Many advantages are offered by this paradigm: just to mention the more evident, teacher and student are not constrained anymore to be in the same place. Moreover, teacher and student can be decoupled also in the time dimension: it is no longer needed that teacher and student attend the lesson at the same time instant. The learning process can be adapted to each student's needs,

taking into account previous knowledge, time availability, and learning capabilities of the student himself.

Several e-learning paradigms have been developed, and amongst them, e-learning "by doing" is one of the most promising in terms of the learning quality. The "by doing" paradigm consists of teaching a topic by letting the student directly practice the argument onto a real system, or a model that simulates the real system. This approach can be applied also to the e-learning processes, and in particular to software applications learning. Of course, the degree of interaction between the student and the teacher, and the possibility to receive help when needed, are of the utmost importance in such process. The student in fact must not be left alone during the learning process, but rather he should be followed interactively, and he should receive help, hints and feedback whenever it is opportune.

To support the e-learning by doing process, it is necessary to tackle several issues: firstly, a mechanism for evaluating the acquired skills is needed, in order to be able to proceed to advanced topics. The evaluation mechanism must provide support for a-posteriori evaluation, as well as run-time evaluation to hint the student. Secondly, it is quite common that the same learning goal can be achieved in more than one way: the tutoring system must be able to evaluate all the options, and should adapt in response to the student choices.

The $\mathcal{S}$CIFF framework, and in particular the *SOCS-SI* application, are general enough to be used also in the context of e-learning by doing. We have successfully used our protocol definition language for representing the action expected by the user of a e-learning by doing system (a sort of a protocol where only one peer participate). We have focussed our experiments on the learning process of a writing application

within the offices program suites. We developed our prototype on two applications, the MS Word program (part of the Microsoft Office Suite), and the Writer application of the OpenOffice suite. For both applications, a specific filter has been developed, with the purpose of capturing the actions performed by the student. Those actions, after a transformation process, are communicated to the *SOCS-SI* application, that provide to check the conformance to a special protocol definition. Such definition can be seen in the Specification 3.3.3, where it is defined how the student can achieve the goal of closing the application after printing a file.

Specification 3.3.3 An e-learning goal represented through the $\mathcal{S}$CIFF Language.

$$
\begin{aligned}
& \mathbf{H}(tell(U,S,keyboard\_event(print),DialogId),T_{Print}) \\
\rightarrow\ & \mathbf{E}(tell(U,S,mouse\_event(menu\_File\_Close),DialogId),T_{Close}) \\
& \quad \wedge T_{Close} > T_{Print} \\
\vee\ & \mathbf{E}(tell(U,S,mouse\_event(menu\_File\_Exit),DialogId),T_{Exit}) \\
& \quad \wedge T_{Exit} > T_{Print} \\
\vee\ & \mathbf{E}(tell(U,S,keyboard\_event(quit),DialogId),T_{Exit}) \\
& \quad \wedge T_{Exit} > T_{Print} \\
\vee\ & \mathbf{E}(tell(U,S,keyboard\_event(alt+f),DialogId),T_{File}) \\
& \quad \wedge \mathbf{E}(tell(U,S,mouse\_event(menu\_File\_Close),DialogId),T_{Close}) \\
& \quad \wedge T_{Print} < T_{File} \wedge T_{File} < T_{Close} \\
\vee\ & \mathbf{E}(tell(U,S,keyboard\_event(alt+f),DialogId),T_{File}) \\
& \quad \wedge \mathbf{E}(tell(U,S,mouse\_event(menu\_File\_Exit),DialogId),T_{Exit}) \\
& \quad \wedge T_{Print} < T_{File} \wedge T_{File} < T_{Exit} \\
\vee\ & \mathbf{E}(tell(U,S,close\_document,DialogId),T_{Close}) \\
& \quad \wedge T_{Close} > T_{Print} \\
\vee\ & \mathbf{E}(tell(U,S,close\_office,DialogId),T_{Close}) \\
& \quad \wedge T_{Close} > T_{Print}
\end{aligned}
\tag{3.3.2}
$$

The IC 3.3.2 shows how it is possible to represent multiple solutions for solving the learning goal. Seven different alternatives are considered, from using the “File” menu and the corresponding voice, to closing directly all the application.

Once the learning goal has been defined through IC, the *SOCS-SI* application can use it in three different ways:

1. the tool can be used as evaluator of the actions of the student: if at the end of the practicing session, at least one expectation is not satisfied, then the goal has not been achieved;

2. *SOCS-SI* can be used also as an on-the-fly checker: if the student perform an action that will block him for reaching the goal, then it is possible to advice him immediately, rather than waiting for the end of the exercise;

3. the tool can be finally used as a suggesting system: if the student does not know how to achieve the goal, it is possible to hint him the next action by communicating the expectations about his future behavior.

Of course it is up to the teacher (or the e-learning content manager) to decide which modality is more opportune.

3.4 Related Works

The social approach to the definition of interaction protocols has been documented in several noteworthy contributions of the past years. Among them, Artikis et al. [24] present a formal framework for specifying systems where the behaviour of the members and their interactions cannot be predicted in advance, and for reasoning about and verifying the properties of such systems. The framework relies upon a deontic logic formalism, and on the concepts of permission, prohibition, and empowerment. The paper also describes a *Society Visualiser* to demonstrate animations of protocol runs in such systems. A noteworthy difference with [24] is that we do not explicitly represent the institutional power of the members and the concept of valid action. "Permitted" are all the events that do not determine a violation, i.e., all events that are not explicitly "forbidden" are "allowed". Being detached from any deontic infrastructure, our framework can be used for a broader spectrum of application domains, from intelligent agents to reactive systems.

Caire et al. [40] propose an agent-oriented CASE tool for implementing and testing Multi-Agent Systems. The testing framework is divided into two steps: the agent test and the society test. The agent test verifies the behaviour of the agent with regard to the system requirements under the responsibility of that agent; the agents are checked both in their black-box behaviour, and in a white-box checking of the behaviour of their internal modules. The "agent society testing is a kind of integration testing": the successful integration of the different agents is verified. The testing is performed automatically, without the need for intervention of the user.

Our work is devoted to testing on-the-fly the compliance of peers to protocol rules, without having any knowledge on the internals of the entities. We provided a

language, based on logics, to define the interaction protocols, and a proof-procedure, based on abduction, to check the compliance. Our *SOCS-SI* tool can be used to check the behaviour of Multi-Agent Systems that are *open*: members of the society are *not* only the ones defined by the MAS designer, but new agents, possibly malicious, may unpredictably join the society, and interact with the other agents. As far as their behaviour follows the society's prescriptions, such interactions may enrich the society, but they must be checked for conformance in order to avoid abuses.

Yolum and Singh [155] apply a variant of the Event Calculus [104] 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*. Such a way of specifying protocols is more flexible than traditional approaches based on action sequences in that it prescribes no initial and final states or transitions explicitly. It only restricts the agent interaction in 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, by means of an Abductive Event Calculus planner. Our notion of expectation is more general than that of commitment adopted by Yolum and Singh [155] or by other work, such as [79]: it represents the expectation about a (past or future) event, without any reference to specific roles of agents (such as a commitment's debtor and creditor), and it does not necessarily need to be brought about by a specific agent.

Several other frameworks in the literature aim at verifying properties about the behaviour of social agents at design time. Often, such frameworks define structured

hierarchies, roles, and deontic concepts such as norms and obligations as first class entities. Notably, ISLANDER [73] is a tool for the specification and verification of interaction in complex social infrastructures, such as electronic institutions. ISLANDER allows for the analysis of situations, called scenes, and visualise liveness or safety properties in some specific settings. The kind of verification involved is static and is used to help designing institutions.

Chapter 4

Proving Protocol-specific Properties: the g-SCIFF Framework

The recent and fast growth of network infrastructures, such as the Internet, is allowing for a new range of scenarios and styles of business-making, secure data communication, and in general of interactions between different peers. Protocols have become one of the key design point through which such interactions can be somehow managed. Protocols are used as a mean for assuring that the overall system exhibits the desired behaviour.

Important key aspects of this behaviour are called *properties*, that are guaranteed by the protocol. For example, a property of a protocol for a typical english auction is that the winner is the bidder with the highest bid, provided that he submitted its bid within a certain deadline (from the previous submitted bid).

Another example can be taken from the security protocol field: the use of such protocols has become common practice in a community of users who often operate in the hope (and sometimes in the trust) that they can rely on a technology which

protects their private information and makes their communications secure and reliable. Such hypothesis of security and privacy are properties of the used protocol: a large number of tools and formal methods has been developed in the research literature to explicitly address the security issues. To cite some, the model checking based techniques [94] and the state of the art On-the-Fly Model Checker (OFMC, [30]).

In this chapter we focus on proving protocol properties, without restricting to any specific application domain. We propose the g-$\mathcal{S}$CIFF Framework (where *g* stands for *generative*), an extension of the $\mathcal{S}$CIFF Framework presented in Chapter 2. Our aim is to adopt a unified approach to both verification types *2* and *3* (as defined in Section 1.2).

The g-$\mathcal{S}$CIFF Framework has been built as an extension of the $\mathcal{S}$CIFF Framework, and like the latter one, the former framework offers several components: a specification language, a declarative semantics, a proof procedure. All the components have been defined as extensions/modifications of the respective original components, pursuing (as much as possible) a unified approach for both the frameworks.

Contributions of the author. The author contributed in a substantial way to the results presented in this chapter. The "proving properties" topic has been partially addressed in the Ph.D. thesis of Marco Alberti. However, here it is possible to find a more complete and comprehensive presentation of the topic and the results achieved. Moreover, the results previously presented have been properly extended and reviewed, and formal properties have been demonstrated.

Chapter organization. The chapter is organized as follow. In Section 4.1 we introduce and formalize the concepts of property, and of proving them.

In Section 4.2 and 4.3 we define the language used in the g-$\mathcal{S}$CIFF Framework, and its declarative semantics. Then we provide the definition of the g-$\mathcal{S}$CIFF Proof Procedure, and compare it with the $\mathcal{S}$CIFF one in Section 4.4.

Section 4.5 to the formal proof of the properties, while some example applications are discussed in Section 4.6. The chapter is concluded with some remarks on related works, in Section 4.7.

4.1 Proving Properties

We first define what is a property in the g-$\mathcal{S}$CIFF Framework:

Definition 4.1.1 (Protocol Property). *A* Protocol Property $\mathcal{P}$ *in the g-$\mathcal{S}$CIFF Framework is a formula, expressed as a goal (in logic programming), with the same syntactics restrictions and allowedness rules as for the Goal $\mathcal{G}$ as stated in Sect. 2.3.2. For sake of completeness and to ease the comprehension:*

- *the syntax of $\mathcal{P}$ is the same as the body of a clause (Tab. 2.3.2);*
- *variables in $\mathcal{P}$ cannot occur only in NbfLiteras;*
- *All variables that occur in an ExistLiteral are existentially quantified.*
- *All remaining variables are universally quantified.*

Example 4.1.1 Let be $\mathcal{S} = \langle \emptyset, \{IC_1, IC_2\} \rangle$ an abductive specification as defined below:

$$
\begin{aligned}
[IC_1]\ & \mathbf{H}(event_1, T_1) \\
& \rightarrow \mathbf{E}(event_2, T_2)\ \wedge\ \mathbf{E}(event_x, T_x) \\
& \vee \mathbf{E}(event_3, T_3)\ \wedge\ \mathbf{E}(event_x, T_x). \\
[IC_2]\ & \mathbf{H}(event_x, T_x) \\
& \rightarrow \mathbf{E}(event_4, T_4). \\
\mathcal{G}\ &= \{\ \mathbf{E}(Event_1, T1)\ \} \\
\mathcal{P}_1\ &= \mathbf{E}(event_4, T_4) \\
\mathcal{P}_2\ &= \mathbf{E}(event_3, T_3)
\end{aligned}
$$

the histories compliant with the protocol (w.r.t. $\mathcal{G}$) are:

$$\begin{aligned}
\mathbf{HAP}_1 &= \{\ \mathbf{H}(event_1, T_1),\\
&\qquad \mathbf{H}(event_2, T_2),\\
&\qquad \mathbf{H}(event_x, T_x),\\
&\qquad \mathbf{H}(event_4, T_4)\}\\
\mathbf{HAP}_2 &= \{\ \mathbf{H}(event_1, T_1),\\
&\qquad \mathbf{H}(event_3, T_3),\\
&\qquad \mathbf{H}(event_x, T_x),\\
&\qquad \mathbf{H}(event_4, T_4)\}
\end{aligned}$$

$\mathcal{P}_1$ holds in both the histories, while $\mathcal{P}_2$ holds only for history $\mathbf{HAP}_2$.

We then provide a definition that a protocol specification (an abductive specification, as defined in 2.3.8) $\mathcal{S}$ does indeed enjoy (or *guarantee*) a property $\mathcal{P}$. In the g-$\mathcal{S}$CIFF framework, this is stated as follows:

Definition 4.1.2 ($\mathcal{P}$ holds for $\mathcal{S}$ w.r.t. $\mathcal{G}$) *Given an abductive specification $\mathcal{S} = \langle SOKB, ICs\rangle$, and a goal $\mathcal{G}$,* a property $\mathcal{P}$ holds for $\mathcal{S}$ *if and only if:*

$$\forall \boldsymbol{HAP}\ \forall \Delta\ :\\ \left\{ \left[\begin{array}{l} SOKB \cup \boldsymbol{HAP} \cup \Delta \models \mathcal{IC}_{\mathcal{S}} \\ SOKB \cup \boldsymbol{HAP} \cup \Delta \models \mathcal{G} \\ \boldsymbol{EXP} \textit{ is fulfilled, } \neg, \textit{ } \boldsymbol{E}\textit{-consistent} \end{array} \right] \Rightarrow \left[\ SOKB \cup \boldsymbol{HAP} \cup \Delta \models \mathcal{P}\ \right] \right\} \tag{4.1.1}$$

The definition simply states that a property $\mathcal{P}$ holds for a protocol if for each history **HAP** that is compliant with the protocol, then such history (together with every abductive explanation Δ_i for the specification $\mathcal{S}$, see 2.4.4) does indeed entail the property $\mathcal{P}$. Note that Equation 4.1.1 states that $\mathcal{P}$ must hold for every abductive explanation Δ_i. In fact, a history **HAP** might be compliant to a protocol

specification, w.r.t. to only some abductive answers

Moreover, a property $\mathcal{P}$ is always said to hold for a specification w.r.t. a specific goal. In fact, a property could hold for all the histories compliant with the protocol specification, but different goals could prevent this (see Example 4.1.2).

Example 4.1.2 Let be $\mathcal{S} = \langle \emptyset, IC_1 \rangle$ an abductive specification as defined below, where p is an abducible predicate:

$$
\begin{aligned}
[IC_1]\ \ & \mathbf{H}(Event_1, T_1) \\
& \rightarrow \mathbf{E}(Event_2, T_2)\ \wedge\ p \\
& \vee\ \mathbf{E}(Event_2, T_2)\ \wedge\ \neg p. \\
\mathcal{G}\ &=\ \{\ \mathbf{E}(Event_1, T1)\ \} \\
\Delta_1\ &=\ \{\ \mathbf{H}(Event_1, T_1),\ \mathbf{H}(Event_2, T_2),\ p\ \} \\
\Delta_2\ &=\ \{\ \mathbf{H}(Event_1, T_1),\ \mathbf{H}(Event_2, T_2),\ \neg p\ \} \\
\mathcal{P}\ &=\ \{\ p\ \}
\end{aligned}
$$

Δ_1 and Δ_2 are two abductive explanation for $\mathcal{S}$ w.r.t. goal $\mathcal{G}$. The property $\mathcal{P}$ holds if we consider Δ_1, but does not hold for Δ_2.

Properties as in Definition 4.1.2 are always referred as *safety* properties. Safety properties state that something bad will not happen. Of course, peers are also free to behave badly in an open environment, so trying to prove that violations will not happen is indeed unrealistic. As stated earlier, however, there is a reaction to bad behaviour of this type: the detection of violation. In our setting, we want to answer formally to an even more subtle question: is there some *undesirable* property that could happen even if there is no violation detected? In order to rely on a system, we want, in all possible histories, either to find a violation (i.e., something bad happens, but we can detect it on-line), or the system to be safe. Stated otherwise, in all histories compliant to the protocol, the desired property must hold. Safety properties are often stated with an empty initial history ($\mathbf{HAP}^i = \emptyset$).

Disproving that a property $\mathcal{P}$ holds for a protocol amounts to disprove Eq. 4.1.1. This can be done by looking for a set **HAP** of events such that an abductive answer Δ exists and $\neg\mathcal{P}$ is entailed:

$$\exists \mathbf{HAP}\ \exists\Delta\ : \begin{cases} SOKB \cup \mathbf{HAP} \cup \Delta \models \mathcal{IC}_{\mathcal{S}} \\ SOKB \cup \mathbf{HAP} \cup \Delta \models (\mathcal{G}\ \wedge\ \neg\mathcal{P}) \\ \mathbf{EXP} \text{ is fulfilled, } \neg, \mathbf{E}\text{-consistent} \end{cases} \tag{4.1.2}$$

A different class of properties is that of *liveness* properties. Liveness means that something good will happen, eventually in the future; it could also mean that given an unpleasant situation, there is an escape: given an initial history (usually, not very promising), there exists, nevertheless, an extension to such a history that entails the desired property. Proving a liveness property $\mathcal{P}$ amounts to prove that, given an initial history $\mathbf{HAP}^i$, there exist a history $\mathbf{HAP}^f$ ($\mathbf{HAP}^i \subseteq \mathbf{HAP}^f$) and an abductive answer Δ_i s.t. $\mathcal{P}$ holds. Such condition is the one expressed in Eq. 4.1.2.

Proving safety and liveness properties can be done by means of Equation 4.1.2: this leads to the idea of proving automatically both types of properties in a uniform way, with same proof methods and same language for formally defining the requested properties.

4.2 The g-SCIFF Language

The language used in the g-$\mathcal{S}$CIFF Framework is a subset of the language defined in the $\mathcal{S}$CIFF Framework (see Section 2.3): here we will point out only the differences and briefly recall the common parts.

Entities of the language. As in the $\mathcal{S}$CIFF Language (Sections 2.1 and 2.3.1, the main entities of the language are:

- atoms representing the concept of *Happened Events* (by means of the functor **H**);
- atoms representing *Positive* and *Negative Expectations* (functors **E** and **EN**).

However, we restrict the language in the following way: in the g-$\mathcal{S}$CIFF Language it is not possible to use the negation $\neg$ in conjunction with an happened event **H**. This restriction is motivated by the fact that, as it will be clearer later in Section 4.3, we are going to abduce **H** atoms. As a consequence, the *constructive negation* applied to **H** atoms is meaningless.

As a consequence, the syntax of events and expectations in the g-$\mathcal{S}$CIFF Framework is the same presented in Table 2.3.1, with the exceptions that the non- terminal symbol *EventLiteral* is not allowed.

The Social Knowledge Base. The SOKB is defined by the grammar shown in Table 2.3.2 (same syntax as in the $\mathcal{S}$CIFF Framework).

Integrity Constraints. The syntax of the Integrity Constraints in the g-$\mathcal{S}$CIFF Framework is pretty much the same of the one shown in Table 2.3.3. However, due to

the restriction on $\neg\mathbf{H}$ atoms, the *Body* of an IC can not contain such negated atoms. Hence, the re-writing rule of the non-terminal symbol *Body* of the grammar shown in Table 2.3.3 is slightly different. We report in Table 4.2.1 the modified syntax of the Integrity Constraints in the g-$\mathcal{S}$CIFF Framework.

Table 4.2.1 Integrity Constraints (ICs) in the g-$\mathcal{S}$CIFF

$$
\begin{array}{rcl}
\mathcal{IC}_{\mathcal{S}} & ::= & [IC]^{\star} \\
IC & ::= & Body \rightarrow Head \\
Body & ::= & (Event \mid ExpLiteral \mid AbducibleLiteral)\ [\ \wedge BodyLiteral\ ]^{\star} \\
BodyLiteral & ::= & Event \mid ExtLiteral \\
Head & ::= & HeadDisjunct\ [\ \vee HeadDisjunct\ ]^{\star} \mid false \\
HeadDisjunct & ::= & ExtLiteral\ [\ \wedge ExtLiteral]^{\star}
\end{array}
$$

4.3 Declarative Semantics of g-SCIFF

In order to be able to hypothesize new happened events, in the g-$\mathcal{S}$CIFF framework **H** atoms are not considered anymore as facts (and then being part of the program P in the abductive interpretation given in Section 2.4.2), but rather as abducibles (and then belonging to the set $\mathcal{E}$ of predicates that can be abduced).

Although considering **H** atoms as abducibles is indeed an important extension w.r.t. to the $\mathcal{S}$CIFF framework, from a formal viewpoint the declarative semantics is the same. Formally, the only difference is that **H** atoms now belongs to the abducibles set, and that the abductive explanation Δ (Definition 2.4.4) is defined as:

$$\Delta \equiv \langle \mathbf{EXP}, \Delta A, \mathbf{HAP} \rangle$$

where the set **EXP** is the set of all the expectations (positive and negative) that have been hypothesized; ΔA is the set of abducibles predicates that have been hypothesized; and **HAP** is the set of happened events that has been hypothesized for disproving the property $\mathcal{P}$ (as explained in Section 4.1).

Note that, as it is in the $\mathcal{S}$CIFF framework, also here $SOKB$, **HAP**, ΔA and **EXP** are subject to the completion of the program (compare Equations 2.4.3 and 2.4.4, referred to $\mathcal{S}$CIFF, with Equations 4.3.1 and 4.3.2, referred instead to g-$\mathcal{S}$CIFF).

For the sake of comprehension, we report here the most important definitions for the declarative semantics of the g-$\mathcal{S}$CIFF Framework.

Definition 4.3.1 *Given an abductive specification $\mathcal{S} = \langle SOKB, \mathcal{IC}_{\mathcal{S}} \rangle$ and a history $\boldsymbol{HAP}$, $\mathcal{S}_{\boldsymbol{HAP}}$ represents the pair $\langle \mathcal{S}, \boldsymbol{HAP} \rangle$, called the* **HAP**-instance of $\mathcal{S}$ *(or simply an* instance of $\mathcal{S}$*).*

Definition 4.3.2 *An abductive specification $\mathcal{S}$ is represented as an ALP, i.e., a triple $\langle P, \mathcal{E}, \mathcal{IC}_{\mathcal{S}} \rangle$ where:*

- *P is the $SOKB$;*
- *$\mathcal{E}$ is the set of* abducible predicates *of $\mathcal{S}$ (**E**, **EN**, **H** predicates and normal abducibles predicates);*
- *$\mathcal{IC}_{\mathcal{S}}$ are the social integrity constraints of $\mathcal{S}$.*

Given the definition of an abductive specification in the g-$\mathcal{S}$CIFF framework, we are now able to re-define the *abductive explanation* for the g-$\mathcal{S}$CIFF:

Definition 4.3.3 *Given an abductive specification $\mathcal{S} = \langle SOKB, \mathcal{IC}_{\mathcal{S}} \rangle$, and a goal $\mathcal{G}$, $\Delta \equiv \langle \boldsymbol{EXP}, \Delta A, \boldsymbol{HAP} \rangle$ is an abductive explanation of $\mathcal{S}$ if:*

$$Comp(SOKB \cup \Delta) \cup \text{CET} \cup T_{\mathcal{X}} \models \mathcal{IC}_{\mathcal{S}} \tag{4.3.1}$$

$$Comp(SOKB \cup \Delta) \cup \text{CET} \cup T_{\mathcal{X}} \models \mathcal{G} \tag{4.3.2}$$

where Comp represents the completion *of a theory,* CET *is Clark's Equational Theory [48], and $T_{\mathcal{X}}$ is the theory of constraints [92].*

The symbol $\models$ is interpreted in three valued logics, as it is in the IFF Proof Procedure.

As for the $\mathcal{S}$CIFF framework, we require consistency with respect to explicit negation [21] and between positive and negative expectations.

Definition 4.3.4 *A set Δ of abducibles is* ¬-consistent *if and only if for each (ground)*

term p and for each abducible predicate q:

$$\begin{aligned} &\{\boldsymbol{E}(p), \neg\boldsymbol{E}(p)\} \not\subseteq \boldsymbol{EXP} \\ &\{\boldsymbol{EN}(p), \neg\boldsymbol{EN}(p)\} \not\subseteq \boldsymbol{EXP} \\ &\{q, \neg q\} \not\subseteq \Delta A \end{aligned} \tag{4.3.3}$$

Definition 4.3.5 *A set $\boldsymbol{EXP}$ of expectations is* **E**-consistent *if and only if for each (ground) term p:*

$$\{\boldsymbol{E}(p), \boldsymbol{EN}(p)\} \not\subseteq \boldsymbol{EXP} \tag{4.3.4}$$

Moreover, we require also that the specification $\mathcal{S}$ is fulfilled (as for $\mathcal{S}$CIFF in Def. 2.4.7), i.e.:

Definition 4.3.6 *Given an abductive explanation Δ ($\Delta = \langle \Delta A, \boldsymbol{EXP}, \boldsymbol{HAP}\rangle$), $\mathcal{S}$ is* fulfilled *if and only if*

$$\begin{aligned} &\forall \boldsymbol{E}(p) \in \boldsymbol{EXP} \Rightarrow \boldsymbol{H}(p) \in \boldsymbol{HAP} \\ &\forall \boldsymbol{EN}(p) \in \boldsymbol{EXP} \Rightarrow \boldsymbol{H}(p) \notin \boldsymbol{HAP} \end{aligned} \tag{4.3.5}$$

When all the given conditions (4.3.1-4.3.5) are met, we say that the goal is *achieved*, and we write

$$\mathcal{S} \models_{\Delta A, \mathbf{EXP}, \mathbf{HAP}} \mathcal{G}$$

.

4.4 g-SCIFF Proof Procedure

The g-$\mathcal{S}$CIFF Proof Procedure is obtained by modifying the $\mathcal{S}$CIFF one: in particular, the modifications affect mainly the set of transitions. Some transitions have been removed, as a consequence of the syntactic restriction introduced in Section 4.2, while a new transition has been added, in order to be able to generate new hypotheses about happened events.

4.4.1 Data Structures

As in the $\mathcal{S}$CIFF Proof Procedure, a node can be either the special node *false*, or defined by the following tuple

$$T \equiv \langle R, CS, PSIC, \Delta A, \Delta P, \mathbf{HAP}, \Delta F, \Delta V \rangle. \tag{4.4.1}$$

We partition the set of expectations **EXP** into the confirmed (ΔF), disconfirmed (ΔV), and pending (ΔP) expectations. The other elements are:

- R is the resolvent: a conjunction, whose conjuncts can be literals or disjunctions of conjunctions of literals
- CS is the constraint store: it contains CLP constraints and quantifier restrictions
- $PSIC$ is a set of implications, called partially solved integrity constraints
- ΔA is the set of general abduced hypotheses (the set of abduced literals, except those representing expectations)
- **HAP** is the history of hypothesized happened events, represented by a set of abducted atoms with functor **H**.

If one of the elements of the tuple is *false*, then the whole tuple is the special node *false*, which cannot have successors. In the following, we indicate with Δ the set $\Delta A \cup \Delta P \cup \Delta F \cup \Delta V \cup \mathbf{HAP}$.

4.4.2 Initial Node and Success

The definitions of initial node, derivation and success/failure of such derivation are exactly the same as shown in Section 2.5.2. Here we will briefly recall that:

- let $\mathcal{S}_{\mathbf{HAP}^i}$ be an abductive specification as in Definition 2.3.8, where the set of abduced happened events is $\mathbf{HAP}^i$ (possibly the empty set);
- a derivation D is a sequence of nodes

$$T_0 \rightarrow T_1 \rightarrow \cdots \rightarrow T_{n-1} \rightarrow T_n.$$

- the first node of a derivation is defined as

$$T_0 \equiv \langle \{\mathcal{G}\}, \emptyset, \mathcal{IC}_{\mathcal{S}}, \emptyset, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$$

  where the inital set of hypothesized happened events is $\mathbf{HAP}^i$;

- the other nodes $T_j, j > 0$, are obtained by applying the transitions;
- if a *successful derivation* exists and a success node is reached (see Definition 2.5.1), s.t. $\mathcal{S}_{\mathbf{HAP}^f}$ is a proper extension of $\mathcal{S}_{\mathbf{HAP}^i}$ (Definition 2.4.2), then we write:

$$\mathcal{S}_{\mathbf{HAP}^i} \overset{g}{\vdash}_{\mathbf{EXP},\Delta A,\overline{\mathbf{HAP}^f}} G$$

- abductive answers can be extracted as for $\mathcal{S}$CIFF from an abductive explanation, by means of a substitution σ s.t. each existentially quantified variable of any term in $\Delta\sigma$ is ground (see 2.5.2).

4.4.3 Removed Transitions

In Section 4.2 we have introduced some syntactic restriction (about $\neg\mathbf{H}$ literals). Moreover, in Section 4.3 we have defined the **H** atoms as abducible predicates. As a consequence, some transitions defined in the $\mathcal{S}$CIFF Proof Procedure are not needed anymore, hence simplifying the g-$\mathcal{S}$CIFF Proof Procedure. In particular:

- *Happening*: since the happened events now are hypothesized and represented by means of abducibles, transition *Happenining* is not needed anymore;
- *Non-Happening*: since the $\neg\mathbf{H}$ literals are not allowed anymore in the syntax, the transition non-Happening can be safely removed;

It is worth to notice that the removed transitions were part of the transition group introduced for coping with dynamically happening events. This is quite reasonable, since in the g-$\mathcal{S}$CIFF Framework happened events are only hypotheses made in order to prove/disprove some property $\mathcal{P}$.

4.4.4 Added Transition

In order to be able to produce new hypotheses, we have decided to define the **H** atoms as abducibles. However, it is necessary to "guide" the process of generating new hypotheses about the happened events, in order to fulfill all the positive/negative expectations (possibly generated by the process of abducing happened events), as given by Definition 4.3.6. For this reason, we have introduced in the g-$\mathcal{S}$CIFF Proof Procedure the following transition:

Definition 4.4.1 (Fulfiller Transition). *The g-$\mathcal{S}$CIFF Proof Procedure extends the $\mathcal{S}$CIFF Proof Procedure by adding the following transition:*

Fulfiller. *Given a node N_k in which*

- $\Delta P_k = \Delta P' \cup \{\boldsymbol{E}(E,T)\}$
- *closed(*$\boldsymbol{HAP}_k$*)=* $false$

and Fulfillment **E** *transition is not applicable, transition* Fulfiller *is applicable and generates a node N_{k+1} identical to N_k except:*

- $\Delta P_{k+1} = \Delta P'$
- $\Delta F_{k+1} = \Delta F_k \cup \{\boldsymbol{E}(E,T)\}$
- $\boldsymbol{HAP}_{k+1} = \boldsymbol{HAP}_k \cup \{\boldsymbol{H}(E,T)\}$

i.e., a new happened event is inserted in the history, fulfilling the expectation.

Otherwise, given a state where

- *closed(*$\boldsymbol{HAP}_k$*)=* $true$

the transition Fulfiller *produces a single successor*

$$false.$$

Note that the *Fulfiller* transition must be applied only when *Fulfillment* transition is not applicable. This condition is not mandatory: we have introduce it in order to preserve the minimality of the abductive explanation.

Example 4.4.1 Let a node N_k of a g-$\mathcal{S}$CIFF derivation be as follow (see Definition 4.4.1):

- $\Delta P = \mathbf{E}(p(X),T)$

- $\mathbf{HAP} = \mathbf{H}(p(1), 3)$
- all the other sets empty.

If the transition *Fulfiller* is applied firstly, the following derivation is computed:

$$\langle \Delta P_k = \{\mathbf{E}(p(X), T)\}\ \mathbf{HAP}_k = \{\mathbf{H}(p(1), 3)\}\rangle$$

$$\begin{gathered}\Delta P_{k+1} = \emptyset \\ \Delta F_{k+1} = \{\mathbf{E}(p(X), T)\} \\ \mathbf{HAP}_{k+1} = \{\mathbf{H}(p(1), 3), \mathbf{H}(p(X), T)\}\end{gathered}$$

Although in $\mathbf{HAP}_k$ there is an happened event that can fulfill the pending expectation, the transition *Fulfiller* generate (hypothesize) a new happened event. $\mathbf{HAP}_{k+1}$ is not minimal.

4.5 g-SCIFF properties

4.5.1 Soundness

We prove the soundness property of the g-$\mathcal{S}$CIFF Proof Procedure in a similar way as it was proved for the $\mathcal{S}$CIFF Proof Procedure [85]. As $\mathcal{S}$CIFF was proved sound relying on the soundness result of IFF, we prove soundness of g-$\mathcal{S}$CIFF relying on soundness result of $\mathcal{S}$CIFF. Intuitively, we proceed in the following way: we first show in Lemma 4.5.1 that an abductive answer Δ_g extracted by a successful g-$\mathcal{S}$CIFF derivation is also a computed answer for a $\mathcal{S}$CIFF program; then, based on this lemma, we prove the soundness result.

Lemma 4.5.1 *Let:*

- $\mathcal{S}_{\boldsymbol{HAP}^i}$ *be* $\langle \mathcal{S}, \boldsymbol{HAP}^i \rangle$ *an abductive instance, where* $\mathcal{S} = \langle SOKB, \mathcal{IC}_\mathcal{S} \rangle$*;*
- (Δ_g, σ) *be the abductive answer extracted from a successful derivation* $(\mathcal{S}_{\boldsymbol{HAP}^i} \overset{g}{\vdash}_{\boldsymbol{EXP}, \Delta A, \overline{\boldsymbol{HAP}^f}} G)$ *for an initial goal* G *and an initial abductive instance* $\mathcal{S}_{\boldsymbol{HAP}^i}$ *evolving to a proper extension* $\mathcal{S}_{\overline{\boldsymbol{HAP}^f}}$*, such that*
  - $\Delta_g = \langle \boldsymbol{EXP}, \Delta A, \boldsymbol{HAP}^f \rangle$ *;*
  - $\Delta_{SCIFF} = \langle \boldsymbol{EXP}, \Delta A \rangle$

Then
(Δ_{SCIFF}, σ) *is a* $\mathcal{S}$*CIFF computed answer for* G *for the program* $\langle SOKB \cup \overline{\boldsymbol{HAP}^f}\delta, \mathcal{E}, \mathcal{IC}_\mathcal{S} \rangle$.

Proof. We construct a successful closed $\mathcal{S}$CIFF derivation from the given successful g-$\mathcal{S}$CIFF derivation, by mapping every step except *Fulfiller* onto itself.

Let us consider then the new transition *Fulfiller*. As described in Definition 4.4.1, given a node $\Delta P_k = \Delta P' \cup \{\mathbf{E}(E,T)\}$, it performs the following actions:

(i) $\Delta P_{k+1} = \Delta P'$

(ii) $\Delta F_{k+1} = \Delta F_k \cup \{\mathbf{E}(E,T)\}$

(iii) $\mathbf{HAP}_{k+1} = \mathbf{HAP}_k \cup \{\mathbf{H}(E,T)\}$

The action (iii) constructs the $\overline{\mathbf{HAP}^f}$ set of happened events that is provided as definition of the $\mathcal{S}$CIFF equivalent program. It is directly mapped on the $\mathcal{S}$CIFF transition *Happening*.

Actions (i) and (ii) are almost equivalent to the *Fulfillment* $\boldsymbol{E}$ transition (see Section 2.5.4). In this particular case the selected happened event for fulfillment has exactly the same variables of the matching expectation. The only difference is about the fact that *Fulfillment* $\boldsymbol{E}$ generates two children nodes from the parent:

- in the first node N^1_{k+1} the actions (i) and (ii) are performed as in g-$\mathcal{S}$CIFF. Moreover, an equality constraint is added to CS^1_{k+1} between the variables of the happened event and the variables of the expected event; in the *Fulfiller* transition this can be safely avoided since the variables are exactly the same (and then the equality constraint is entailed).

- in the second node N^2_{k+1} the parent node is copied identically, and an inequality constraint between the variables of the happened event and the variables of the expected event is added to CS^2_{k+1}. This second node is not generated in the g-$\mathcal{S}$CIFF proof.

Since g-$\mathcal{S}$CIFF generates only the first node, we can draw the conclusion that the derivation tree of the g-$\mathcal{S}$CIFF is a subset of the derivation tree of $\mathcal{S}$CIFF. Note that the choice of not generating the second node N^2_{k+1} in g-$\mathcal{S}$CIFF has been made for performances issues: in fact such a node would lead immediately to a failure, due to the fact that the inequality constraint can not be satisfied (the variables of the expected event and of the happened event are exactly the same by construction).

Note that any success node in the g-$\mathcal{S}$CIFF derivation has a correspondent success node in the $\mathcal{S}$CIFF derivation, and that no other $\mathcal{S}$CIFF transition can be applied anymore to that success node. If a $\mathcal{S}$CIFF transition (except *Non-Happening* and *Happening*) could be applied to the success node, the correspondent g-$\mathcal{S}$CIFF transition would be applicable to the success node too, and that node could not be a success node (contradicting the initial hypothesis). *Non-Happening* transition could not be applied because of the syntax limitations introduced in Section 4.2, while *Happening* transition would not applicable because we are considering a close derivation for $\mathcal{S}$CIFF.

Summarizing, since the derivation tree of g-$\mathcal{S}$CIFF proof procedure is a subset of the derivation tree of $\mathcal{S}$CIFF proof procedure for the specified program, the answer $\Delta_{SCIFF}\sigma$ computed by g-$\mathcal{S}$CIFF is an abductive answer for $\mathcal{S}$CIFF. □

We are now ready to state the soundness result for the case without universally quantified variables:

Theorem 4.5.1 (Soundness). *Given an instance $\mathcal{S}_{\boldsymbol{HAP}^i}$ (i.e., the set of happened events is set initially to $\boldsymbol{HAP}^i$),if*

$$\mathcal{S}_{\boldsymbol{HAP}^i} \overset{g}{\vdash}_{\boldsymbol{EXP},\Delta A,\overline{\boldsymbol{HAP}^f}} G$$

with abductive answer (Δ, σ) *(*$\Delta = \langle \boldsymbol{EXP}, \Delta A, \overline{\boldsymbol{HAP}^f} \rangle$*),*

then

$$\mathcal{S} \models_{\Delta\sigma} G\sigma$$

Proof. As already done in the proofs for the $\mathcal{S}$CIFF framework [85], we rely upon the 3-value completion [107] of $SOKB$ and Δ. Let us consider the proof for an atomic goal (the extension to other structures of the formula G is trivial).

Proving the latter condition stated in the theorem corresponds to prove the following ones, separately, w.r.t. the extensions introduced in Sect. 4.4:

(i) $SOKB \cup [\overline{\mathbf{HAP}^f} \cup \Delta F \cup \Delta P \cup \Delta A]\sigma \models G\sigma$;

(ii) $SOKB \cup [\overline{\mathbf{HAP}^f} \cup \Delta F \cup \Delta P \cup \Delta A]\sigma \models \mathcal{IC}_{\mathcal{S}}$;

(iii) $\{\mathbf{E}(p), \neg\mathbf{E}(p)\} \not\subseteq [\Delta F \cup \Delta P]\sigma$ ($\neg$-consistency for $\mathbf{E}$ atoms);

(iv) $\{\mathbf{EN}(p), \neg\mathbf{EN}(p)\} \not\subseteq [\Delta F \cup \Delta P]\sigma$ ($\neg$-consistency for $\mathbf{EN}$ atoms);

(v) $\{q, \neg q\} \not\subseteq (\Delta A)\sigma$ ($\neg$-consistency for generic abducible atoms);

(vi) $\{\mathbf{E}(p), \mathbf{EN}(p)\} \not\subseteq [\Delta F \cup \Delta P]\sigma$ ($\mathbf{E}$-consistency);

(vii) $\overline{\mathbf{HAP}^f} \cup [\Delta F \cup \Delta P]\sigma \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{EN}(p) \rightarrow \neg\mathbf{H}(p)\} \not\models \bot$ (fulfillment).

Conditions $(i)-(vi)$ are guaranteed by Lemma 4.5.1. In particular these conditions are exactly the same that are stated in the soundness of the $\mathcal{S}$CIFF proof procedure (Proposition 6.2, [85]). This is a consequence of the fact that the g-$\mathcal{S}$CIFF and $\mathcal{S}$CIFF share the same declarative semantics. From Lemma 4.5.1, we have that a computed answer for a g-$\mathcal{S}$CIFF derivation is also a computed answer for $\mathcal{S}$CIFF derivation, and as a consequence of the soundness result of the $\mathcal{S}$CIFF, conditions $(i)-(vi)$ hold.

Condition (*vii*) holds because: (1) the fulfillment of the negative expectations (**EN** atoms) is still guaranteed by the *Fulfillment* ***EN*** transition; and (2) the fulfillment of the positive expectations instead is enforced by the *Fulfiller* transition itself, that is applied whenever a positive expectation (**E** atoms) is pending and still not fulfilled (i.e., it still belongs to the set ΔP).

□

4.6 Application Examples

4.6.1 Needham-Schroeder Public Key Security Protocol

The Needham-Schroeder Public Key Security Protocol (NSPK,[114]) aims to allow two peers, A and B, to exchange two secret numbers (*nonces*), while mutually authenticating each other. The protocol consists of seven steps, but – as other authors have previously done – we focus on a simplified version consisting of only three steps, which are the kernel of the protocol. In support of the authentication procedure, peers rely on the well-known public key encryption technology. The three messages of the protocol that we consider in this example are those listed in Figure 4.1. Basically, with the simplified version, we assume that all the agents know the public key of the other agents, and that no previous stages for discovering the public keys is needed. Thanks to the public/private key technology, a peer is able to generate

(1) $A \rightarrow B : \{N_A, A\}_{pub\_key(B)}$

(2) $B \rightarrow A : \{N_A, N_B\}_{pub\_key(A)}$

(3) $A \rightarrow B : \{N_B\}_{pub\_key(B)}$

Figure 4.1: The Needham-Schroeder protocol (simplified version)

two keys, a public key which is made available to the others, and a private key which must remain undisclosed. A sequence of bytes encrypted using the public key can be decrypted only by using the corresponding private key. The idea of the protocol is to challenge the fellow peer in a communication session (conversation), to make sure that he is actually the holder of the private key associated with his public key.

During the authentication phase, peers can generate some special items of data

called *nonces*. Therefore, during the conversation, if peer A sends a nonce N_A generated by himself to B, and if A sends N_A encrypted with the public key of B, only B will be able to decrypt N_A and send it back to A. A will then know that the peer to whom he sent N_A is actually the holder of B's private key.

As shown in Fig. 4.1, by message (1) A challenges B to decrypt his nonce N_A encrypted using B's public key. By message (2) B responds to A's challenge, by attaching to N_A a new nonce N_B, which he generated himself, and encrypting the whole set of two nonces using A's public key, thus challenging A to decrypt N_B and prove to be the holder of A's private key. By message (3) A responds to B's challenge.

The security property of the NSPK protocol has been stated under the assumptions of *perfect cryptography*, *insecure communication channels* and the intruder I able to intercept/generate messages. Practically, we assume that:

1. when a peer sends a message to another peer, the sender has no way to know if the message has been received or not;

2. when a peer receives a message, there is no way to be sure about the sender, unless this information is somehow coded into the payload;

3. the content of a message could be compromised someway;

4. there is no way a peer can guess the content of a message encrypted with the public key of another agent;

5. there is no way a peer can guess the *nonce* that another agent has generated (unless it was explicitly communicated).

Lowe's attack on the protocol

It turns out that (at least) one situation may occur in which B trusts that he is proving his own identity to another agent A, by following this protocol, but in fact a third agent I (standing for *intruder*) manages to successfully pretend that he is A and authenticate himself as A with B. This attack was suggested by Lowe [109], and it consists of the messages listed in Figure 4.2.

(1) $A \rightarrow I : \{N_A, A\}_{pub\_key(I)}$

(2) $I(A) \rightarrow B : \{N_A, A\}_{pub\_key(B)}$

(3) $B \rightarrow I(A) : \{N_A, N_B\}_{pub\_key(A)}$

(4) $I \rightarrow A : \{N_A, N_B\}_{pub\_key(A)}$

(5) $A \rightarrow I : \{N_B\}_{pub\_key(I)}$

(6) $I(A) \rightarrow B : \{N_B\}_{pub\_key(B)}$

Figure 4.2: Lowe's attack on the Needham-Schroeder protocol

It is a nesting of the Needham-Schroeder protocol, in which A happens to start a conversation with I, thus transmitting him his nonce N_A. Instead of answering to the challenge with a new nonce N_I, I exploits the information contained in A's request for authentication (namely, its nonce) to handcraft a message to send to B. Such a message (2) will be encrypted using B's public key, and will contain A's name along with A's nonce N_A. I therefore sends this message pretending that he is A (we use the notation $I(A)$ for this purpose). B will reply to the challenge contained in message (2) by generating a nonce N_B and encrypting N_A and N_B together using A's public key. Since I is unable to decrypt a message encrypt with another agent's public key, I simply forwards B's message to A. This is understood by A as the continuation of

the protocol initiated with message (1). In this way, I manages to receive back from A the nonce N_B encrypted using his own public key (message 5), and to respond to B with message (6).

Messages (1), (4) and (5) represent a conversation between A and I, while (2), (3), and (6) represent a conversation between $I(A)$ and B; both conversations happen to be compliant to the protocol. But, as we have seen, a combination of two compliant conversations generates a situation in which an agent (I) authenticates himself with an identity ($I(A)$) which is not his own.

It is important to stress that in the attack proposed by Lowe it is never the case that an intruder manages to guess a nonce or a private key. In particular, initially only agent A knows the content of its own nonce N_A and only B knows the content of its own nonce N_B, and an agent knows the content of a nonce if either he initially knows it or if it is sent to him encrypted in his own public key.

Formalizing peers's authentication

In the idea of the NSPK protocol, an agent trusts the identity of the agent with whom he is communicating by associating his name with his public key and receiving back a nonce that he forged, encrypted in his own public key. If we had to define the idea of an agent B 'trusting' that he is communicating with A, we could do it by using a combination of messages in which an agents responds to a challenge posed by another agent and successfully decrypts a nonce.

Definition 4.6.1 *B trusts that agent X, he is communicating with, is indeed A,*[1] *and we write $trust_B(X, A)$ once two messages have been exchanged at times T_1 and T_2,*

[1] *We restrict ourselves to only one communication session, all the definitions will therefore have as a scope the session.*

$T_1 < T_2$, having the following sender, recipient, and content:

$$(T_1)\ B \rightarrow X : \{N_B, \dots\}_{pub\_key(A)}$$

$$(T_2)\ X \rightarrow B : \{N_B, \dots\}_{pub\_key(B)}$$

where N_B is a nonce generated by B.

Note that B is unable to judge whether N_A is a nonce actually generated by X or not, therefore no condition is posed on the origin of such nonce.

Symmetrically, we can consider, from A's viewpoint, messages (1) and (2) as those that prove the identity of B. We therefore implement Def. 4.6.1 in Def. 4.6.2, where messages are expressed using the notation of the $\mathcal{S}$CIFF language, namely as events which are part of some "history" **HAP**. The content of messages will be composed of three parts, the first showing the public key used to encrypt it, the second and third containing agent names or nonces or nothing (in particular, the last part may be empty).

Definition 4.6.2 *Let A, B and X be agents, K_A and K_B respectively A's and B's public key, N_B a nonce produced by B, and let $\boldsymbol{HAP}_1$ and $\boldsymbol{HAP}_2$ be two sets of events each composed of two elements, namely:*

$\boldsymbol{HAP}_1 = \{$

$\boldsymbol{H}(send(B, X, content(key(K_A), agent(B), nonce(N_B))), T_1),$

$\boldsymbol{H}(send(X, B, content(key(K_B), nonce(N_B), nonce(...))), T_2)$

$\}$, *and*

$\boldsymbol{HAP}_2 = \{$

$\boldsymbol{H}(send(B, X, content(key(K_A), nonce(...), nonce(N_B))), T_1),$

$\boldsymbol{H}(send(X, B, content(key(K_B), nonce(N_B), empty(0))), T_2)$

}. *Then,* $trust_B(X, A)$ *holds if and only if* $\boldsymbol{HAP}_1 \subseteq \boldsymbol{HAP}$ *or* $\boldsymbol{HAP}_2 \subseteq \boldsymbol{HAP}$.

Specification by means of ICs of the scenario assumptions

The assumptions about perfect cryptography , etc. stated previously, can be stated as ICs that rules which messages can/can not be sent between peers. Taking this perspective, we can sat for example that an agent X can send to another agent Y a message containing a nonce N_X which he does not initially know only if one of the following two cases hold: either (i) X received N_X from another agent, encrypted in X's own public key, or (ii) X received a message containing N_X and encrypted with a public key K_Y, in which case X can forward exactly the same message, without operating any modification on it.

Such integrity constraints about the impossibility to guess a nonce are shown in Spec. 4.6.1. In order to maintain relevant information about the ownership of public keys and nonces, we define a number of predicates in the Social Organization Knowledge Base, ash shown in Spec. 4.6.1.

Needham-Schroeder protocol specification

The relevant ICs are shown in Spec. 4.6.2. The IC1 of Spec. 4.6.2 expresses that if a message is sent from X to B, containing the name of some agent A and some nonce N_A, encrypted together with some public key K_B:

$$\mathbf{H}(send(X, B, content(key(K_B), agent(A), nonce(N_A))), T1) \in \mathbf{HAP},$$

then a message is expected to be sent at a later time (and by some deadline T_{Max}) from B to X , containing the original nonce N_A and a new nonce N_B, encrypted with

Specification 4.6.1 ICs and SOKB expressing that an agent cannot guess the content of a *nonce*

ICs:

$$
\begin{aligned}
[IC_{A1}]\ & \mathbf{H}(send(X,Y,content(key(KY),agent(W),nonce(NX))),T1) \wedge \\
& X\ !=\ Y\ \wedge\ notIsNonce(X,NX) \\
\rightarrow\ & \mathbf{E}(send(V,X,content(key(KX),agent(V),nonce(NX))),T0) \wedge \\
& isAgent(V)\ \wedge\ X\ !=\ V\ \wedge\ isPublicKey(X,KX)\ \wedge\ isNonce(V,NX) \wedge \\
& T0\ <\ T1\ \wedge\ T0\ >\ 0 \\
\vee\ & \\
& \ldots
\end{aligned}
$$

$$
\begin{aligned}
[IC_{A2}]\ & \mathbf{H}(send(X,Y,content(key(KY),nonce(NX),nonce(NY))),T1) \wedge \\
& X\ !=\ Y\ \wedge\ notIsNonce(X,NX) \\
\rightarrow\ & \mathbf{E}(send(Z,X,content(key(KX),agent(V),nonce(NX))),T0) \wedge \\
& isAgent(V)\ \wedge\ isAgent(Z)\ \wedge\ X\ !=\ V\ \wedge\ Z\ !=\ X\ \wedge\ isPublicKey(X,KX) \wedge \\
& T0\ <\ T1\ \wedge\ T0\ >\ 0 \\
\vee\ & \\
& \ldots
\end{aligned}
$$

$$
\begin{aligned}
[IC_{A3}]\ & \mathbf{H}(send(X,Y,content(key(KY),nonce(NX),empty(0))),T1) \wedge \\
& X\ !=\ Y\ \wedge\ notIsNonce(X,NX) \\
\rightarrow\ & \mathbf{E}(send(Y,X,content(key(KX),nonce(NW),nonce(NX))),T0) \wedge \\
& isPublicKey(X,KX)\ \wedge\ isNonce(NW)\ \wedge\ NW\ !=\ NX\ \wedge \\
& T0\ <\ T1\ \wedge\ T0\ >\ 0 \\
\vee\ & \\
& \mathbf{E}(send(Z,X,content(key(KX),nonce(NX),empty(0))),T0) \wedge \\
& isPublicKey(X,KX)\ \wedge\ isAgent(Z)\ \wedge\ X\ !=\ Z\ \wedge\ Y\ !=\ Z\ \wedge \\
& T0\ <\ T1\ \wedge\ T0\ >\ 0 \\
\vee\ & \\
& \ldots
\end{aligned}
$$

SOKB:

```
isPublicKey( PK) :-
  isPublicKey( _, PK).
isPublicKey( i, ki).
isPublicKey( b, kb).
isPublicKey( a, ka).

isMaxTime( 7).

isAgent( i).
isAgent( a).
isAgent( b).
```

```
isNonce( N) :-
   isNonce( _, N).

isNonce( A, N) :-
  checkIfNonce( A, N).

notIsNonce( A, NB) :-
  \+( checkIfNonce( A, NB)).

checkIfNonce( b, nb).
checkIfNonce( a, na).
```

the public key of A:

$$\mathbf{E}(send(B, X, content(key(K_A), nonce(N_A), nonce(NB))), T2)$$

is therefore generated and put into ΔP.

The IC2 of Fig. 4.6.2 expresses that if a message of the protocol is sent from X to B, containing the name of some agent A and some nonce N_A, encrypted together with some public key K_B:

$$\mathbf{H}(send(X, B, content(key(K_B), agent(A), nonce(N_A))), T1) \in \mathbf{HAP},$$

and a message is sent at a later time from B to X, containing the original nonce N_A and a new nonce N_B, encrypted with the public key of A:

$$\mathbf{H}(send(B, X, content(key(K_A), nonce(N_A), nonce(N_B))), T2) \in \mathbf{HAP},$$

then a third message is expected to be sent from X to B, containing N_B, and encrypted with the public key of B:

$$\mathbf{E}(send(X, B, content(key(KB), nonce(NB), empty(0))), T3)$$

is therefore generated and put into ΔP.

Generation of compliant histories

A first result that we obtain by running the g-$\mathcal{S}$CIFF is that, given as a social goal the expectation about some event, the proof-procedure is able to generate a compliant (and complete) history which includes such event. For instance, given the goal *g1* representing the start of a protocol run between a and i:

$$g1 \leftarrow \mathbf{E}(send(a, i, content(key(ki), agent(a), nonce(na))), 1),$$

Specification 4.6.2 Social Integrity Constraints defining the Needham-Schroeder protocol

$$
\begin{aligned}
[IC1]\ \ & \mathbf{H}(send(X,B,content(key(KB),agent(A),nonce(NA))),T1) \wedge \\
& \rightarrow \mathbf{E}(send(B,X,content(key(KA),nonce(NA),nonce(NB))),T2) \wedge \\
& \quad isPublicKey(A,KA) \ \wedge \ isNonce(NB) \ \wedge \ NA\,!=\,NB \wedge \\
& \quad isMaxTime(TMax) \ \wedge \ T2 \ > \ T1 \ \wedge \ T2 \ < \ TMax.
\end{aligned}
$$

$$
\begin{aligned}
[IC2]\ \ & \mathbf{H}(send(X,B,content(key(KB),agent(A),nonce(NA))),T1) \wedge \\
& \quad \mathbf{H}(send(B,X,content(key(KA),nonce(NA),nonce(NB))),T2) \wedge \\
& \quad T2 \ > \ T1 \\
& \rightarrow \mathbf{E}(send(X,B,content(key(KB),nonce(NB),empty(0))),T3) \wedge \\
& \quad isMaxTime(TMax) \ \wedge \ T3 \ > \ T2 \ \wedge \ T3 \ < \ TMax.
\end{aligned}
$$

and given a deadline of 6 "time units" to the completion of the protocol, the execution of the proof returns the following compliant history:

$$
\begin{aligned}
&\mathbf{HAP}_{g1} = \{ \\
&\mathbf{H}(send(a,i,content(key(ki),agent(a),nonce(na))),1), \\
&\mathbf{H}(send(i,a,content(key(ka),nonce(na),nonce(nb))),T_A), T_A \in [2..5], \\
&\mathbf{H}(send(a,i,content(key(ki),nonce(nb),empty(0))),T_B), T_B \in [3..6], T_B > T_A \\
&\},
\end{aligned}
$$

while given the goal *g2*, representing the last step of a protocol run between i and b:

$$g2 \leftarrow \mathbf{E}(send(i,b,content(key(kb),nonce(nb),empty(0))),6),$$

and again a range of 6 "time units" to the completion of the protocol (from time 1 to time 6), it is possible to obtain a compliant history such as the following:

$$
\begin{aligned}
&\mathbf{HAP}_{g2} = \{ \\
&\mathbf{H}(send(a,i,content(key(ki),agent(a),nonce(na))),T_C), T_C \in [1..3], \\
&\mathbf{H}(send(i,a,content(key(ka),nonce(na),nonce(nb))),T_D), T_D \in [2..5], T_D > T_C \\
&\mathbf{H}(send(a,i,content(key(ki),nonce(nb),empty(0))),T_E), T_E \in [3..6], T_E > T_D
\end{aligned}
$$

$\mathbf{H}(send(i, b, content(key(kb), agent(i), nonce(na))), T_B), T_B \in [2..4], T_B > T_C$

$\mathbf{H}(send(b, i, content(key(ki), nonce(na), nonce(nb))), T_A), T_A \in [3..5], T_A > T_B$

$\mathbf{H}(send(i, b, content(key(kb), nonce(nb), empty(0))), 6)$

}.

It is worthwhile noticing that $\mathbf{HAP}_{g2}$ contains two possibly interleaved communication sessions (one between a and i and another between i and a) which do not represent an attack to the protocol. In fact, it is not the case that $trust_X(Y, W)$ and $\neg trust_W(Y, X)$, for all X, Y and W. What happens is, i uses to communicate with b the content of the nonce na obtained from a (but does not pretend to be himself a). This is in fact perfectly allowed by the protocol and does not contradict the assumptions on the generation of nonces specified by the constraints of Spec. 4.6.1.

Generation of Lowe's attack

A second important result, which shows how the g-$\mathcal{S}$CIFF can effectively be used for protocol verification, is the generation of Lowe's attack. The property that we want to disprove is $\mathcal{P}_{trust}$ defined as $trust_B(X, A) \rightarrow X = A$, i.e., if B trusts that he is communicating with A, then he is indeed communicating with A. We obtain a problem which is symmetric in the variables A, B, and X. In order to check if we have a solution we can ground $\mathcal{P}_{trust}$ and define its negation $\neg\mathcal{P}_{trust}$ as a goal, $g3$, where we choose to assign to A, B, and X the values a, b and i:

$g3 \leftarrow isNonce(NA), NA \neq nb,$

$\mathbf{E}(send(b, i, content(key(ka), nonce(NA), nonce(nb))), 3),$

$\mathbf{E}(send(i, b, content(key(kb), nonce(nb), empty(0))), 6).$

Besides defining $g3$ for three specific agents, we also assign definite time points (3 and 6) in order to improve the efficiency of the proof.

Running the g-$\mathcal{S}$CIFF on $g3$ results in a compliant history:

$\mathbf{HAP}_{g3} = \{$

$h(send(a, i, content(key(ki), agent(a), nonce(na))), 1),$

$h(send(i, b, content(key(kb), agent(a), nonce(na))), 2),$

$h(send(b, i, content(key(ka), nonce(na), nonce(nb))), 3),$

$h(send(i, a, content(key(ka), nonce(na), nonce(nb))), 4),$

$h(send(a, i, content(key(ki), nonce(nb), empty(0))), 5),$

$h(send(i, b, content(key(kb), nonce(nb), empty(0))), 6)$

$\},$

which is indeed Lowe's attack on the protocol. $\mathbf{HAP}_{gL}$ represents a counterexample of the property $\mathcal{P}_{trust}$.

4.6.2 NetBill Transaction Protocol

NetBill [53] is a security and transaction protocol optimized for the selling and delivery of low-priced information goods, like software, journal articles or songs/videos. The protocol rules transactions between two peers: the seller of the good, namely the *Merchant*, and the client, namely the *Customer*.

A NetBill server is used to deal with financial issues such as those related to credit card accounts of customer and merchant. In this example, we focus on the NetBill protocol version designed for non zero-priced goods, and do not consider the variants that deal with zero-priced goods. A typical protocol run is composed of three phases:

1. **Price Negotiation**. The customer requests a quote for a good identified by Product Id ($PrId$) $priceRequest(PrId)$ and the merchant replies with the requested price $priceQuote(PrId, Quote)$

2. **Good Delivery**. The customer requests the good $goodRequest(PrId, Quote)$ and the merchant delivers it in an encrypted format $deliver(crypt(PrId, Key), Quote)$

3. **Payment**. The customer issues an Electronic Payment Order (EPO) to the merchant, for the amount agreed for the good $payment(epo(C, crypt(PrId, K), Quote)))$; the merchant appends the decryption key for the good to the EPO, signs the pair and forwards it to the NetBill server $endorsedEPO(epo(C, crypt(PrId, K), Quote), M)$; the NetBill server deals with the actual money transfer and returns the result to the merchant $signedResult(C, PrID, Price, K)$, who will, in her turn, send a receipt for the good and the decryption key to the customer $receipt(PrId, Price, K)$.

The *Customer* can withdraw from the transaction until she has issued the *EPO* message; the *Merchant* can withdraw from the transaction until she has issued the *endorsedEPO* message.

NetBill protocol specification based on IC.

In Table 4.6.1 the specification of the Netbill protocol is presented: ICs $[1-6]$ are *backward* ICs (i.e.,integrity constraints that state that if some set of event happens, then some other set of event is expected to have happened before), while ICs $[7-8]$ are *forward* ICs.

IC1, for example, imposes that if M has sent a *priceQuote* message to C, stating that M's quote for the good identified by $PrId$ is $Quote$, in the interaction identified by Id, then C is expected to have sent to M a *priceRequest* message for the same good, in the same interaction, at an earlier time; IC7 instead, imposes that an *endorsedEPO* message from M to the *netbill* server be followed by a *signedResult* message, with the corresponding parameters.

Note that we only impose forward constraints from the *endorsedEPO* message onwards, because both parties (merchant and customer) can withdraw from the transaction at the previous steps: hence the uttering of messages in the first part of the protocol does not lead to any expectation to utter further messages.

Verification of a NetBill property

In this example, we show the verification of the following property:

Table 4.6.1 NetBill protocol specification

$[IC1]$ $\mathbf{H}(tell(M, C, priceQuote(PrId, Quote), Id), T)$
$\rightarrow \mathbf{E}(tell(C, M, priceRequest(PrId), Id), T2) \wedge$
$T2 < T.$

$[IC2]$ $\mathbf{H}(tell(C, M, goodRequest(PrId, Quote), Id), T)$
$\rightarrow \mathbf{E}(tell(M, C, priceQuote(PrId, Quote), Id), Tpri) \wedge$
$Tpri < T.$

$[IC3]$ $\mathbf{H}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T)$
$\rightarrow \mathbf{E}(tell(C, M, goodRequest(PrId, Quote), Id), Treq) \wedge$
$Treq < T.$

$[IC4]$ $\mathbf{H}(tell(C, M, payment(C, crypt(PrId, K), Quote), Id), T)$
$\rightarrow \mathbf{E}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), Tdel) \wedge$
$Tdel < T.$

$[IC5]$ $\mathbf{H}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \wedge$
$M != netbill$
$\rightarrow \mathbf{E}(tell(M, netbill, endorsedEPO(epo(C, PrId, Quote), K, M), Id), T) \wedge$
$T < Tsign.$

$[IC6]$ $\mathbf{H}(tell(M, C, receipt(PrId, Quote, K), Id), Ts)$
$\rightarrow \mathbf{E}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \wedge$
$Tsign < Ts.$

$[IC7]$ $\mathbf{H}(tell(M, netbill, endorsedEPO(epo(C, PrId, Quote), K, M), Id), T)$
$\rightarrow \mathbf{E}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \wedge$
$T < Tsign.$

$[IC8]$ $\mathbf{H}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign)$
$\rightarrow \mathbf{E}(tell(M, C, receipt(PrId, Quote, K), Id), Ts) \wedge$
$Tsign < Ts.$

> *"As long as the protocol is respected, the merchant receives the payment for a good G if and only if the customer receives the good G."*
>
> (**Good Atomicity Property**)

Since the $\mathcal{S}$CIFF deals with (communicative) events and not with the states of the peers, we need to express the properties in terms of happened events. To this purpose, we can assume that merchant has received the payment once the NetBill server has issued the *signedResult* message, and that the the customer has received the good if she has received the encrypted good (with a *deliver* message) and the encryption key (with a *receipt* message).

Thus, the property we want to verify can be expressed as

$$\begin{aligned} & \mathbf{H}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ \Longleftrightarrow\ & \mathbf{H}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T) \\ & \wedge \mathbf{H}(tell(M, C, receipt(PrId, Quote, K), Id), Ts) \end{aligned} \tag{4.6.1}$$

whose negation is

$$\begin{aligned} & (\neg \mathbf{H}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ & \wedge \mathbf{H}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T) \\ & \wedge \mathbf{H}(tell(M, C, receipt(PrId, Quote, K), Id), Ts)) \\ & \vee \\ & (\mathbf{H}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ & \wedge \neg \mathbf{H}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T) \\ & \vee \\ & (\mathbf{H}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ & \wedge \neg \mathbf{H}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T)) \end{aligned} \tag{4.6.2}$$

In other words, an history that entails Eq. (4.6.2) is a counterexample of the property that we want to prove. In order to search for such a history, we define the $\mathcal{S}$CIFF goal as follows:

$$
\begin{aligned}
g \leftarrow & EN(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign), \\
& E(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T), \\
& E(tell(M, C, receipt(PrId, Quote, K), Id), Ts)). \\
g \leftarrow & E(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign), \\
& EN(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T). \\
g \leftarrow & E(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign), \\
& EN(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T))
\end{aligned}
\tag{4.6.3}
$$

and run g-$\mathcal{S}$CIFFwith the integrity constraints showed in Spec. 4.6.1.

The result of the call is a failure. This suggests that there is no history that entails the negation of the property while respecting the protocol, i.e., the property is likely to hold if the protocol is respected.

If we remove IC8 (which imposes that a *signedResult* message be followed by a *receipt* message), then the following history is generated:

```
h(tell(_E,_F,priceRequest(_D),_C),_M),
h(tell(_F,_E,priceQuote(_D,_B),_C),_L),
h(tell(_E,_F,goodRequest(_D,_B),_C),_K),
h(tell(_F,_E,goodDelivery(crypt(_D,_A),_B),_C),_J),
h(tell(_E,_F,payment(_E,crypt(_D,_A),_B),_C),_I),
h(tell(_F,netbill,endorsedEPO(epo(_E,_D,_B),_A,_F),_C),_H),
h(tell(netbill,_F,signedResult(_E,_D,_B,_A),_C),_G),
_I<_H, _H<_G,
_L>_M, _K>_L, _I>_J, _J>_K,
```

The *receipt* event is missing, hence proving (by means of the counter-example generated) that the protocol step envisaged by IC8 was necessary in order to guarantee the *good atomicity* property.

4.7 Related Works

To the best of our knowledge, this is the first comprehensive and fully operational approach addressing both types of verification (property verification, presented in this chapter, and compliance verification, Section 3.1), and using the same protocol definition language in both cases.

Although the property verification is about properties of any kind, in security research field this issues has acquired an enormous importance, and a huge literature is available on the topic. In the following, we discuss some related logic-based approaches to automatic verification of security properties. Note however that security protocols and their proof of flawedness are, in g-$\mathcal{S}$CIFF viewpoint, instances of the general concepts of interaction protocols and their properties.

Russo *et al.* [128] discuss the application of abductive reasoning for analyzing safety properties of declarative specifications expressed in the Event Calculus. In their abductive approach, the problem of proving that, for some invariant I, a domain description D entails I ($D \models I$), is translated into an equivalent problem of showing that it is not possible to consistently extend D with assertions that particular events have actually occurred (i.e., with a set of abductive hypotheses Δ), in such a way that the extended description entails $\neg I$. In other words, there is no set Δ such that $D \cup \Delta \models \neg I$. They solve this latter problem by a complete abductive decision procedure, thus exploiting abduction in a refutation mode. Whenever the procedure finds such a set Δ, the assertions in Δ act as a counterexample for the invariant. Our work is closely related: in fact, in both cases, goals represent negation of properties, and the proof-procedure attempts to generate counterexamples by means of abduction. However, we rely on a different language (in particular, ours can also be used for

checking compliance on the fly without changing the specification of the protocol, which is a demanding task) and we deal with time by means of CLP constraints, whereas Russo *et al.* employ a temporal formalism based on Event Calculus.

In [30] the authors present a new approach, On-the-Fly Model Checker, to model check security protocols, using two concepts quite related to our approach: the concept of lazy data types for representing a (possibly) infinite transition system, and the use of variables in the messages that an intruder can generate. In particular, the use of unbound variables reduces the state space generated by every possible message that an intruder can utter. Protocols are represented in the form of transition rules, triggered by the arrival of a message: proving properties consists of exploring the tree generated by the transition rules, and verifying that the property holds for each reachable state. They prove results of soundness and completeness, provided that the number of messages is bounded. Our approach is very similar, from the operational viewpoint. The main difference is that the purpose of our language is not limited to the analysis of security protocols: their approach instead is deeply focused on the security issues (e.g., the presence of an intruder is mandatory for each protocol specification, and it is not possible to avoid it). Moreover, we have introduced variables in all the messages, and not only in the messages uttered by the intruder; we can pose CLP constraints on these variables, whereas OFMC can only generate equality/inequality constraints. On the downside, OFMC provides state-of-the-art performance for security protocol analysis; our approach instead suffers for its generality, and its performance is definitely worse than the OFMC.

A relevant work in computer science on verification of security protocols was done

by Abadi and Blanchet [35, 1]. They adopt a verification technique based on logic programming in order to verify security properties of protocols, such as secrecy and authenticity in a fully automatic way, without bounding the number of sessions. In their approach, a protocol is represented in extensions of pi calculus with cryptographic primitives. The protocol represented in this extended calculus is then automatically translated into a set of Horn clauses [1]. To prove secrecy, in [35, 1] attacks are modeled by relations and secrecy can be inferred by non-derivability: if $attacker(M)$ is not derivable, then secrecy of M is guaranteed. More importantly, the derivability of $attacker(M)$ can be used, instead, to reconstruct an attack. This approach was later extended in [34] in order to prove authenticity. By first order logic, having variables in the representation, they overcome the limitation of bounding the number of sessions. We achieve the same generality of [35, 1], since in their approach Horn clause verification technique is not specific to any formalism for representing the protocol, but a proper translator from the protocol language to Horn clause has to be defined. In our approach, we preferred to directly define a rewriting proof-procedure ($\mathcal{S}$CIFF) for the protocol representation language. Furthermore, by exploiting abduction and CLP constraints, also in the implementation of g-$\mathcal{S}$CIFF transitions themselves, in our approach we are able to generate proper traces where terms are constrained when needed along the derivation avoiding to impose further parameters to names as done in [1]. CLP constraints can do this more easily.

Armando *et al.* [23] compile a security program into a logic program with choice lp-rules with answer set semantics. They search for attacks of length k, for increasing values of k, and they are able to derive the flaws of various flawed security protocols. They model explicitly the capabilities of the intruder, while we take the opposite

viewpoint: we explicitly state what the intruder cannot do (like decrypting a message without having the key, or guessing the key or the nonces of an agent), without implicitly limiting the abilities of the intruder.

Our protocol specifications can be seen as intensional formulations of the possible (i.e., compliant) traces of communication interactions. In this respect, our way of modeling protocols is very similar to the one of Paulson's inductive approach [120]. In particular, our representation of the events is almost the same, but we explicitly mention time in order to express temporal constraints. In the inductive approach, the protocol steps are modeled as possible extensions of a trace with new events and represented by (forward) rules, similar to our ICs. However, in our system we have expectations, which allow us to cope with both compliance on the fly and verification of properties without changing the protocol specification. Moreover, ICs can be considered more expressive than inductive rules, since they deal with constraints (and constraint satisfaction in the proof), and disjunctions in the head. As far as verification, the inductive approach requires more human interaction and expertise, since it exploits a general purpose theorem prover, and has the disadvantage that it cannot generate counterexamples directly (as most theorem prover-based approaches). Instead, we use a specialized proof-procedure based on abduction that can perform the proof without any human intervention, and can generate counterexamples.

Millen and Shmatikov [113] define a sound and complete proof-procedure, later improved by Corin and Etalle [52], based on constraint solving for cryptographic protocol analysis. g-$\mathcal{S}$CIFF is based on constraint solving as well, but with a different flavour of constraint: while the approaches by Millen and Shmatikov and by Corin and Etalle are based on abstract algebra, our constraint solver comprises a CLP(FD)

solver, and embeds constraint propagation techniques to speed-up the solving process.

In [140], Song presents Athena, an approach to automatic security protocol analysis. Athena is a very efficient technique for proving protocol properties: unlike other techniques, Athena copes well with state space explosion and is applicable with an unbounded number of peers participating in a protocol, thanks to the use of theorem proving and to a compact way to represent states. Athena is correct and complete (but termination is not guaranteed). Like Athena, the representation of states and protocols in g-$\mathcal{S}$CIFF is non ground, and therefore general and compact. Unlike Athena's, the g-$\mathcal{S}$CIFF's implementation is not optimized, and suffers from the presence of symmetrical states. On the other hand, a clear advantage of our approach is that protocols are written and analyzed in a formalism which is the same used for run-time verification of compliance.

Özkohen and Yolum [117] propose an approach for the prediction of exceptions in supply chains which builds upon the well-known commitment-based approach for protocol specification (see, for instance, Yolum and Singh [155]); their approach is related in many aspects to our on-the-fly verification. They represent the expected agent behaviour by means of commitments between agents; commitments have timeouts, i.e., they must be fulfilled by a deadline, and can be composed by means of conjunction and disjunction. In this perspective, commitments are similar to our expectations, which can have deadlines represented by CLP constraints, and which are composed in disjunctions of conjunctions in the head of the social integrity constraints. However, our expectations can regard any kind of events expected to happen, not only those that can be represented as a commitment of a debtor towards a creditor; and we can also represent negative expectations. Operationally, in [117] the reasoning about

commitments is centralized in a monitoring agents; in our framework, a similar task in performed by the social infrastructure.

One way to prove/disprove the security of the protocols is the cryptographic approach, whose security definitions are based on complexity theory. Such an approach have been used for proofs by hand [87] or, more recently, automatically [25]. Theorem provers, such as Isabelle/HOL [115] have also been applied to such a task, also together with tools for graphically representing and defining the protocols [150]. Another viewpoint is to embody a possible intruder and plan for an attack [5]: if a planner succeeds in developing such a plan, then the protocol is, clearly, flawed.

Dixon *et al.* [65], specify security protocols in $KL_{(n)}$, a language for representing the *Temporal Logic of Knowledge*. $KL_{(n)}$ contains both a linear-time temporal logic and a modal connective for representing what the various agents know. They use clausal resolution to automatically prove properties of security protocols. As an example, they give the specifications of the Needham-Schroeder protocol in $KL_{(n)}$, then they show how to apply clausal resolution for proving formal properties. In a technical report [66], they define a proof system based on resolution rules in a sequent style notation, and a temporal resolution algorithm, then applied to the mentioned protocol. They then show the derivation of some properties of the protocol, for example, that an intruder does not know the sensitive information exchanged by other agents following the protocol. Finally, they show Lowe's attack on the protocol, and state that, if they do not assume that an agent A can send a message to an agent B only encrypted with B's public key, the previous properties (on the intruder's ignorance) cannot be proven. [122] also use a temporal logic enriched with epistemic connectives for representing the agents' knowledge, but exploit efficient data

structures (namely, Ordered Binary Decision Diagrams) to improve the efficiency.

[90] propose a framework for the synthesis of security protocols. Their framework employs SVO logic to express the initial conditions od a protocol run, the goals that the protocol is wanted to achieve and the effect of message exchanges (in terms of the principals' knowledge and beliefs). An efficient (according to a fitness function) protocol is synthesized by simulated annealing in the space of the protocols that achieve the goal starting from the initial conditions. By using g-$\mathcal{S}$CIFF, we can synthesize a history that satisfies a given goal; if we view the synthesized history as a protocol run, the result of a computation can be seen as a synthesized protocol that achieves a given goal. However, we do not use a logic for expressing secrecy and trust properties in terms of exchanged messages; and we have not yet researched efficient search strategies for generating one of the possible histories.

Among other approaches to security protocols verification we cite those developed using hereditary Harrop formulas [55], process-algebraic languages [119], model-checking with pre-configuration [101], proof-theory [56].

We terminate the discussion on related works by citing [136], where Shanahan also introduces a concept of expectation: a robot moves in an office, and has expectations about where it is standing, based on the values obtained by sensors. Both this work and ours share the idea of abducing expectations. The difference is that Shanahan uses the expectations to elaborate plans for the future moves of the robot, while we use them to elaborate all the possible histories and prove properties.

Chapter 5

A-priori Conformance: the ALLOWS Framework

The A^{l}LoWS (Abductive Logic Web-service Specification) Framework aims to verify the a-priori compliance conformance of peers w.r.t. global protocols, i.e. the *Type 1* verification presented in Section 1.2. This verification type is of the utmost importance in modern systems.

The a-priori conformance is a required step to achieve the "off-the-shelf components" business model. Although heterogeneities between different hardware/software components has been solved by introducing standards (like, to cite one, the Web Service approach), at the application level a solution is still missing. The desired goal is to take off-the-shelf components, test a-priori their compliance with existing systems, and in case of positive answer safely introduce the new entities in the old system (hence substituting an older component or extending the system).

The approach we propose here is to assume that each component is described by its behavioural interface: in particular, we propose to describe both the component external behaviour and the global protocol using the same formalism, based on the $\mathcal{S}$CIFF Language discussed in Section 2.3.

Then we provide definitions of conformance (*feeble* and *strong*), and discuss how it is possible to exploit the $\mathcal{S}$CIFF and the g-$\mathcal{S}$CIFF proof procedure to verify the conformance. Hence, using the single framework $\mathcal{S}$CIFF and its extension g-$\mathcal{S}$CIFF, our solution addresses the a-priori conformance issue within a unified $\mathcal{S}$CIFF based framework.

Note that the framework name derives from the fact that we have applied it for the first time to the Web Services/Choreography scenario, where the interoperability issues is heavily studied. However, our approach can be seamlessly used in other application scenarios.

Contribution of the Author. The author contributed in a substantial way to the presented results. In particular, the first architecture was proposed by the author, together with its colleague; Marco Gavanelli spotted some limits of that solution and proposed an extended version. The actual framework is the result of a further extensions applied by the author (of course by following the supervisor hints).

Chapter Organization. The chapter is organized as follow: in Section 5.1 we introduce the language for specifying both the behavioural interface and the global protocol (actually, a subset of the $\mathcal{S}$CIFF language).

In Section 5.2 we provide e declarative semantics to the A^{l}LoWS language, together with definitions of *Feeble Conformance* and *Strong Conformance*. In Section 5.3 then we define the operational semantics of A^{l}LoWS.

In Sections 5.4 and 5.5 we present some simple examples, and then a more complex (and real) example of how our approach can be used, and what are the outcomes. Finally, in Section 5.6 we discuss some related issues.

5.1 The ALLOWS Specification Language

The specification language used in the A^{l}LoWS framework is a subset of the $\mathcal{S}$CIFF language presented in Section 2.3. In particular, negative expectations and explicit negation have been removed from the entities of the language. This simplification has been possible because, as many other do in the literature, we assume a closed model interaction, as discussed in Section 2.2.2. As consequence, **EN** atoms are not necessary.

In the A^{l}LoWS Framework the same notation used in [27] is adopted: a message is described by the term $m_x(Sender, Receiver, Content)$, where m_x is the type of message, and the arguments retain their intuitive meaning. We sometimes simplify the notation, and omit some of the parameters when the meaning is clear from the context. Note that this notation is equivalent to the one adopted in the $\mathcal{S}$CIFF Framework (Section 2.1).

As in $\mathcal{S}$CIFF, happened events are represented as $\mathbf{H}(Message, Time)$, where $Message$ has the syntax previously defined, and $Time$ is an integer, representing the time point in which the event happened. As we will see in the following, the **H** predicate can be abduced, when making hypotheses on the possible interactions (as we do in Chapter 4. In other phases, they are considered as given a priori, thus considered as a defined predicate (as in Chapter 2.

We represent expectations with the predicate

$$\mathbf{E}_X(Message, Time)$$

expressing the fact that the corresponding event is expected to happen, in order to fulfil the coherent evolution, from the viewpoint of X (where X might be either the

global protocol or a peer public policy).

Note that, w.r.t. the $\mathcal{S}$CIFF, here we explicitly state the expectation viewpoint: in fact, since we are using the same language for defining both the global protocol and the peer policy, we need to distinguish between protocol expectations $\mathbf{E}_{prot}$ and the peer expectations $\mathbf{E}_{peer}$.

5.1.1 Specification of a Protocol

A protocol describes, from a global viewpoint, what are the patterns of communication, or interactions, allowed in a system that adopts such protocol [28]. The protocol specification defines the messages that are allowed: it is not possible to exchange other messages except the ones explicitly specified. The protocol usually also enlists the participants, the roles the participants can play, and other knowledge about the peer interaction. Note that we are adopting a closed interaction model, as described in Section 2.2.2.

As in $\mathcal{S}$CIFF, we specify a protocol by means of an abductive logic program [95]. A protocol specification $\mathcal{P}_{chor}$ is defined by the triple:

$$\mathcal{P}_{prot} \equiv \langle KB_{prot}, \mathcal{E}_{prot}, \mathcal{IC}_{prot} \rangle$$

where:

- KB_{prot} is the *Knowledge Base*,
- $\mathcal{E}_{prot}$ is the set of $\mathbf{E}$ atoms and *abducible predicates*, and
- $\mathcal{IC}_{prot}$ is the set of *Integrity Constraints*.

The syntax of the $SOKB$ is reported in Equation (5.1.1).

$$
\begin{aligned}
SOKB &::= [Clause]^{\star} \\
Clause &::= Atom \leftarrow Cond \\
Cond &::= ExtLiteral\ [\ \wedge ExtLiteral\ ]^{\star} \\
ExtLiteral &::= Literal|Expectation|Abducible|Constraint \\
Expectation &::= \mathbf{E}_{prot}(Term\ [,T]) \\
Abducible ::= AtomLiteral &::= Atom \mid \neg Atom \mid true
\end{aligned}
\tag{5.1.1}
$$

The *abducible predicates* are those that can be hypothesized (abduced) in our framework, namely happened events (denoted by the functor $\mathbf{H}$), expectations (denoted by the functor $\mathbf{E}_{prot}$), and generic abducibles predicates.

Integrity Constraints $\mathcal{IC}_{prot}$ are the usual forward rules, of the form $Body \rightarrow Head$, whose *Body* can contain literals and (happened and expected) events, and whose *Head* can contain (disjunctions of) conjunctions of expectations. In Eq. (5.1.2) we report the formal definition of the grammar.

$$
\begin{aligned}
\mathcal{IC}_{prot} &::= [IC]^{\star} \\
IC &::= Body \rightarrow Head \\
Body &::= (Event|Expect)\ [\wedge BodyLit]^{\star} \\
BodyLit &::= Event|Expect|Literal|Constraint \\
Head &::= Disjunct\ [\ \vee Disjunct\ ]^{\star}|false \\
Disjunct &::= Expect|Abducible\ [\ \wedge (Expect|Abducible|Constraint)]^{\star} \\
Expect &::= \mathbf{E}_{prot}(Term\ [,T]) \\
Event &::= \mathbf{H}(Term\ [,T]) \\
Literal &::= Atom \mid \neg Atom
\end{aligned}
\tag{5.1.2}
$$

The syntax of $\mathcal{IC}_{prot}$ is a simplified version of the introduced in Section 2.3. In particular in A^{l}LoWS we do not need negative expectations and explicit negation.

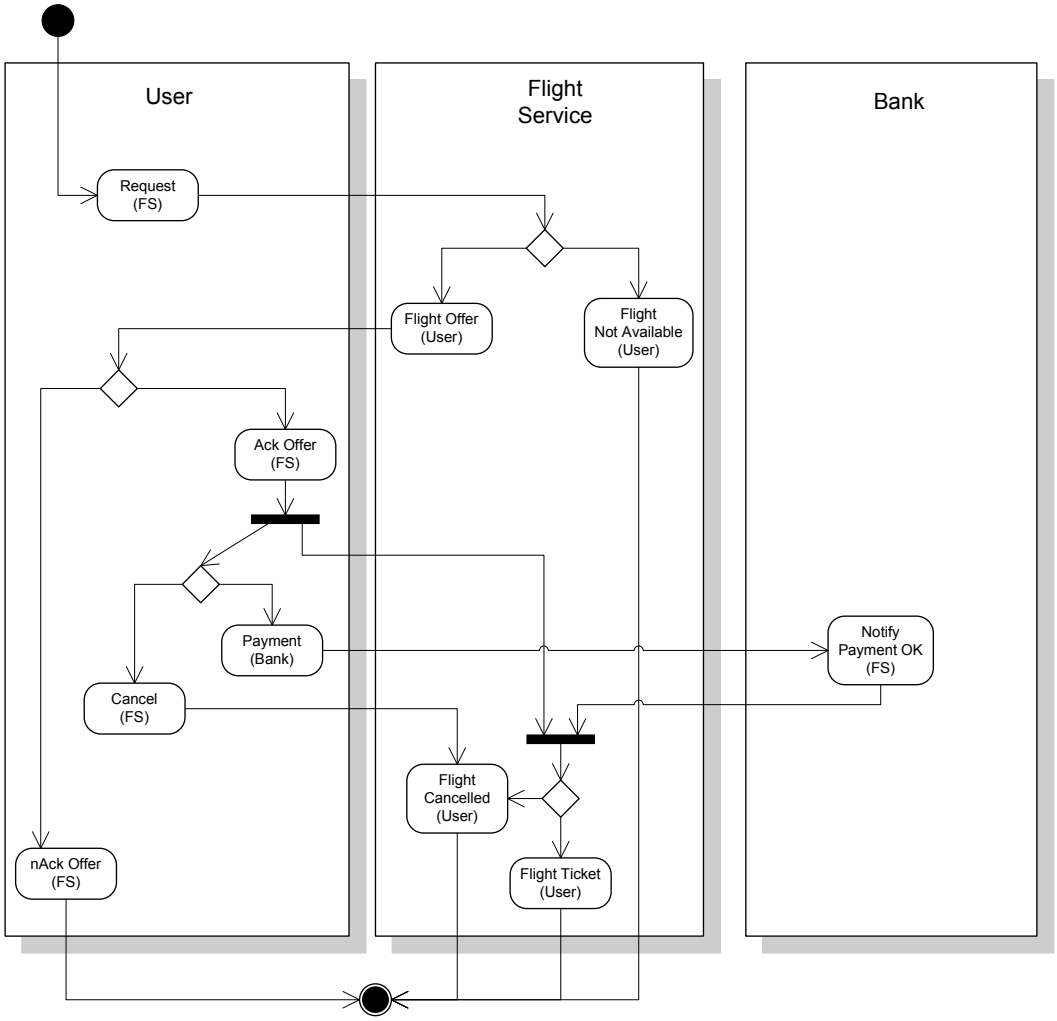


Figure 5.1: Graphical representation of a simple protocol

In Fig. 5.1 a multi-party interaction is shown, expressed by the set of Integrity Constraints in Specification 5.1.1: the depicted scenario is about a User that wants to buy a flight ticket from a Flight Service, and pay by sending a payment order to a Bank.

Specification 5.1.1 The specification of the protocol shown in Fig. 5.1.

$$\begin{aligned}&\mathbf{H}(\mathit{request}(User, FS, Flight), T_r)\\ \rightarrow&\mathbf{E}_{prot}(\mathit{offer}(FS, User, Flight, Price), T_o)\\ \vee&\mathbf{E}_{prot}(\mathit{notAvailable}(FS, User, Flight), T_{na})\end{aligned} \tag{5.1.3}$$

$$\begin{aligned}&\mathbf{H}(\mathit{offer}(FS, User, Flight, Price), T_o)\\ \rightarrow&\mathbf{E}_{prot}(\mathit{ackOffer}(User, FS, Flight, Price), T_a)\\ \vee&\mathbf{E}_{prot}(\mathit{nAckOffer}(User, FS, Flight, Price), T_a)\end{aligned} \tag{5.1.4}$$

$$\begin{aligned}&\mathbf{H}(\mathit{ackOffer}(User, FS, Flight, Price), T_a)\\ \rightarrow&\mathbf{E}_{prot}(\mathit{payment}(User, Bank, Price, FS), T_f)\\ \vee&\mathbf{E}_{prot}(\mathit{cancel}(User, FS, Flight), T_f)\end{aligned} \tag{5.1.5}$$

$$\begin{aligned}&\mathbf{H}(\mathit{ackOffer}(User, FS, Flight, Price), T_a)\\ &\wedge\mathbf{H}(\mathit{notifyPayment}(Bank, FS, Price), T_p)\\ \rightarrow&\mathbf{E}_{prot}(\mathit{flightTicket}(FS, User, Flight), T_f)\\ \vee&\mathbf{E}_{prot}(\mathit{flightCancelled}(FS, User, Flight), T_f)\end{aligned} \tag{5.1.6}$$

$$\begin{aligned}&\mathbf{H}(\mathit{cancel}(User, FS, Flight), T_a)\\ \rightarrow&\mathbf{E}_{prot}(\mathit{flightCancelled}(FS, User, Flight), T_f)\end{aligned} \tag{5.1.7}$$

$$\begin{aligned}&\mathbf{H}(\mathit{payment}(User, Bank, Price, Creditor), T_p)\\ \rightarrow&\mathbf{E}_{prot}(\mathit{notifyPayment}(Bank, Creditor, Price), T_n)\end{aligned} \tag{5.1.8}$$

5.1.2 Representing the peers

In an analogous way as we define the specification of a peers, we describe the interface behaviour of a peer by means of an Abductive Logic Program. In particular, we restrict our analysis to the communicative aspects of the interface behaviour. A Web Service Interface Behaviour Specification $\mathcal{P}_{ws}$ is an Abductive Logic Program [95], represented with the triple

$$\mathcal{P}_{peer} \equiv \langle KB_{peer}, \mathcal{E}_{peer}, \mathcal{IC}_{peer} \rangle$$

where:

- KB_{ws} is the *Knowledge Base* of the peer,
- $\mathcal{E}_{ws}$ is the set of abducible predicates, and
- $\mathcal{IC}_{ws}$ is the set of *Integrity Constraints*.

The *Knowledge Base* (KB_{per}) specifies the knowledge of a peer. In KB_{peer}, clauses may contain in their body literals defined in KB_{peer}, expectations about the behaviour of the web service *peer*, or messages that *peer* expects to receive from other participants. It has the same syntax as the protocol's knowledge base, except for the expectations, that are indicated with the functor $\mathbf{E}_{peer}$ instead of $\mathbf{E}_{prot}$.

$\mathcal{E}_{peer}$ is the set of abducible predicates. Similarly to the choreography specification, this set consists of both expectations (denoted by $\mathbf{E}_{peer}$), happened events ($\mathbf{H}$), and normal abducibles. In the protocol specification the expectations are used for representing the global viewpoint of how things should go, hence all the expectations have the same meaning. In the peer specification instead we are expressing how it

"perceives" the interaction: the viewpoint is local, and the expectations assume a slightly different meaning depending on who is expected to do what. More in detail:

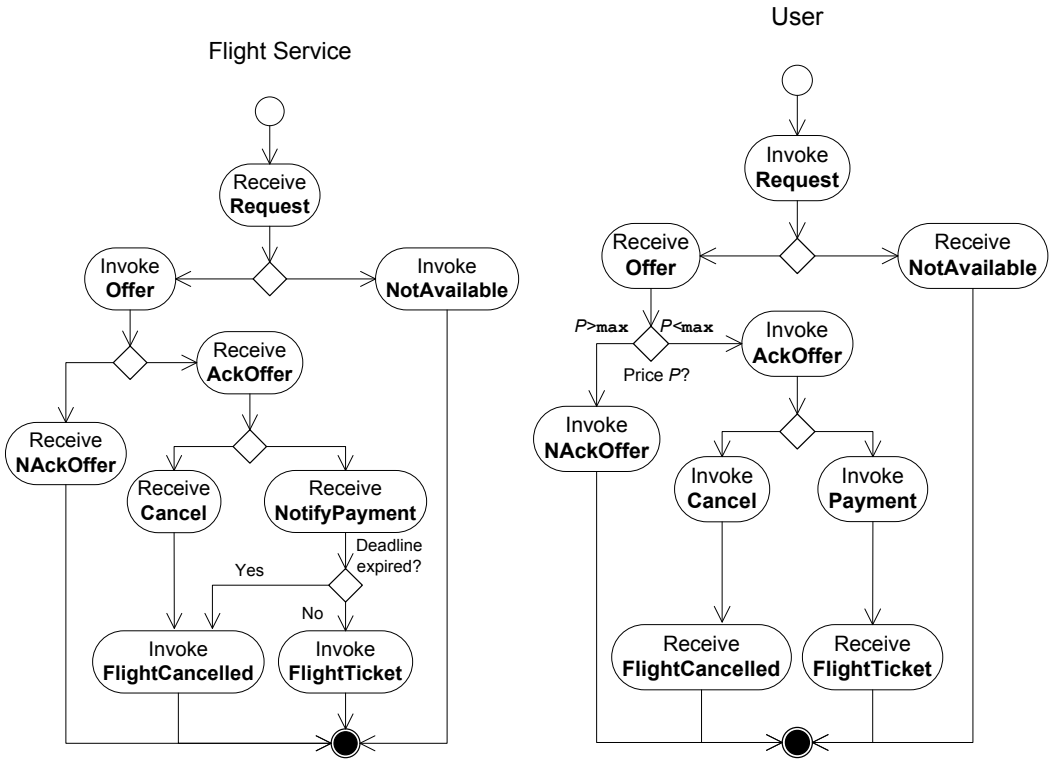
- Expectations about messages where *peer* is the sender are intended as the possible messages that *peer* can indeed utter. Intuitively, expectations of the form $\mathbf{E}_{peer}(m_x(ws, Any, Content))$ represent the "active" behaviour of *peer*, i.e. the actions that it could perform. Hence they represent the "outgoing" communicative behaviour of *peer*. The conformance test should ensure that every possible message that *peer* could utter, is indeed envisaged by the protocol.

- Expectations about messages where other participants are the senders and *peer* is the receiver, can be intended instead as the messages that *peer* is able to understand. They are of the form $\mathbf{E}_{peer}(m_x(Any, ws, Content))$, with $Any \neq peer$.

- Abducibles predicates, that represents extra hypotheses about the interaction (see, for example, Section 5.4.5).

Integrity Constraints $\mathcal{IC}_{peer}$ are forward rules, and they are identical to the $\mathcal{IC}_{prot}$ (except for the fact that expectations are from the peer's viewpoint: $\mathbf{E}_{peer}$ instead of $\mathbf{E}_{prot}$). While in the protocol specification we use them to specify the desired behaviour of the participants, $\mathcal{IC}_{peer}$ are used instead to describe the communication aspects of the interface behaviour of a peer.

In Fig. 5.2(b) the communicative part of the interface behaviour of a peer is represented. The corresponding translation in terms of $\mathcal{IC}_{ws}$ is given in Specification 5.1.2.



(a) (b)

Figure 5.2: Example of behavioural interfaces

Specification 5.1.2 The interface behaviour specification of the web service shown in Fig. 5.2.

$$\begin{aligned}&\mathbf{H}(request(User, fs, Flight), T_r)\\ \rightarrow&\mathbf{E}_{fs}(\mathit{offer}(fs, User, Flight, Price), T_o)\\ \vee&\mathbf{E}_{fs}(notAvailable(fs, User, Flight), T_{na})\end{aligned} \tag{5.1.9}$$

$$\begin{aligned}&\mathbf{H}(\mathit{offer}(fs, User, Flight, Price), T_o)\\ \rightarrow&\mathbf{E}_{fs}(User, fs, ackOffer(User, fs, Flight, Price), T_a)\\ \vee&\mathbf{E}_{fs}(User, fs, nAckOffer(User, fs, Flight, Price), T_a)\end{aligned} \tag{5.1.10}$$

$$\begin{aligned}&\mathbf{H}(\mathit{notifyPayment}(Bank, fs, Price), T_p)\\ \rightarrow&\mathbf{E}_{fs}(\mathit{flightCancelled}(fs, User, Flight), T_c)\\ &\wedge T_p > T_a + \delta \wedge T_c > T_p\\ \vee&\mathbf{E}_{fs}(\mathit{flightTicket}(fs, User, Flight), T_t)\\ &\wedge T_t > T_p\end{aligned} \tag{5.1.11}$$

$$\begin{aligned}&\mathbf{H}(\mathit{ackOffer}(User, fs, Flight, Price), T_a)\\ \rightarrow&\mathbf{E}_{fs}(\mathit{notifyPayment}(Bank, fs, Price), T_p)\\ \vee&\mathbf{E}_{fs}(User, fs, cancel(User, fs, Flight), T_c)\end{aligned} \tag{5.1.12}$$

$$\begin{aligned}&\mathbf{H}(cancel(User, fs, Flight), T_a)\\ \rightarrow&\mathbf{E}_{fs}(\mathit{flightCancelled}(fs, User, Flight), T_f)\end{aligned} \tag{5.1.13}$$

As for the choreographies, also peer specifications can be *goal directed*, by specifying a goal $\mathcal{G}_{peer}$, with the same syntax ($Cond$ in Eq. 5.1.1), in which the expectations are $\mathbf{E}_{peer}$ instead of $\mathbf{E}_{prot}$.

5.2 Declarative semantics

Intuitively, conformance is the characteristics of a peer to comply to a protocol, provided that the other peers will behave according to the protocol. From the declarative semantics viewpoint, the test of conformance requires to assume further hypotheses about events *peer* expects to utter, and events that the protocol specification expects other peers to utter. Both can be mapped into constraints, provided that we consider the predicate **H** as abducible. We use the peer's interface behaviour $\mathcal{P}_{peer}$ to foresee the messages the peer will send in every possible situation, provided that the other peers behave as specified by the protocol. Formally, all the messages the *peer* expects to send will be executed, i.e.:

$$\mathbf{E}_{peer}(m_x(peer, R, C), T) \rightarrow \mathbf{H}(m_x(peer, R, C), T) \tag{5.2.1}$$

Symmetrically for the messages exchanged by other peers as prescribed by the protocol specification $\mathcal{P}_{prot}$:

$$\mathbf{E}_{prot}(m_x(S, R, C), T), S \neq peer \rightarrow \mathbf{H}(m_x(S, R, C), T) \tag{5.2.2}$$

The possible interactions amongst the *peer* and the other peers will be the sets $\mathbf{HAP}^*$ satisfying equations 5.2.1 and 5.2.2. Note also that some extra hypotheses could be made by the peer or by the choreography specification: such hypothesis set (ΔA) must be consistent.

Definition 5.2.1 *Given the abductive program* $\langle KB_U, \mathcal{E}_U, \mathcal{IC}_U \rangle$*, where:*

- $KB_U \triangleq KB_{prot} \cup KB_{peer}$

- $\mathcal{E}_U \triangleq \mathcal{E}_{prot} \cup \mathcal{E}_{peer}$
- $\mathcal{IC}_U \triangleq \mathcal{IC}_{prot} \cup \mathcal{IC}_{peer}$

a possible interaction *amongst a peer in a protocol prot is a triple (****HAP*****,****EXP*****,*ΔA*) such that:*

$$KB_U \cup \boldsymbol{HAP}^* \cup \boldsymbol{EXP}^* \cup \Delta A \models G_U \tag{5.2.3}$$

$$KB_U \cup \boldsymbol{HAP}^* \cup \boldsymbol{EXP}^* \cup \Delta A \models \mathcal{I}C_U \tag{5.2.4}$$

$$KB_U \cup \boldsymbol{HAP}^* \cup \boldsymbol{EXP}^* \models (5.2.1) \cup (5.2.2) \tag{5.2.5}$$

(where by Eq. 5.2.5 we mean that equations 5.2.1 and 5.2.2 must hold). The set $\boldsymbol{HAP}^*$ *is also called* possible history.

When the goal G_U is *true*, the empty set is typically one of the possible histories. The empty history is often of little (or no) interest for proving conformance. When the interesting histories are only those containing at least one event, the expectation of such event can be inserted as the goal G_U. Typically, we use as goal the expectation (both from the peer's viewpoint, $\mathbf{E}_{peer}$ and from the protocol's viewpoint, $\mathbf{E}_{prot}$) of the first event of an interaction. This poses no serious restriction on the types of protocols that can be tested, as if there is not a unique starting event, a dummy event can be inserted as initiation of the protocol.

Example 5.2.1 Suppose a choreography prescribes the following protocol:

$$\mathbf{H}(ask(peer, R, X)) \rightarrow \mathbf{E}_{prot}(answer(R, peer, X)) \tag{5.2.6}$$

$$\mathbf{H}(answer(R, peer, X)) \rightarrow \mathbf{E}_{prot}(ack(peer, R, X))$$

while the peer's integrity constraints contain only the first rule

$$\mathbf{H}(ask(peer, R, X)) \quad \rightarrow \quad \mathbf{E}_{peer}(answer(R, peer, X)).$$

Let $G_U = \mathbf{E}_{peer}(ask(peer, other, X))$, $\mathbf{E}_{prot}(ask(peer, other, X))$. Given the goal G_U, the peer has the intention to send an *ask* message to the *other*, so all the possible histories for G_U will contain the event $\mathbf{H}(ask(peer, other, X))$. The *other*'s behaviour is simulated through the rules in the protocol specification. Since the protocol has an expectation (generated by rule 5.2.6) $\mathbf{E}_{prot}(answer(other, peer, X))$, this will become a happened event in all the possible histories: $\mathbf{H}(answer(other, peer, X))$. Now, the second rule provides a protocol's expectation about the third message: the peer is supposed to send an *ack* message. But, as we can see from the peer's specification, *ws* does not have an expectation to send such message, so the simulation will not suppose it will comply to the protocol's expectation. So, the (only) possible history for the goal G_U is

$$\mathbf{HAP}^* = \{\mathbf{H}(ask(peer, other, X)), \mathbf{H}(answer(other, peer, X))\}. \tag{5.2.7}$$

In a possible history, the messages uttered by the peers comply by definition to the protocol. However, the messages uttered by the peer under test might be non conformant. The peer is conformant if all the possible histories (together with the hypotheses made) are conformant. Also, *peer* should be able to understand all the messages in a possible history, otherwise there might be requests of other peers in the given choreography which *peer* is unable to serve. We require that all the possible histories satisfy both the protocol and the peer expectations.

Definition 5.2.2 (Feeble Conformance). *A possible history* $\boldsymbol{HAP}^*$ *is* Feeble Conformant *if there exists a pair* $(\boldsymbol{EXP}, \Delta A)$ *such that*[1]

$$KB_U \cup \boldsymbol{HAP}^* \cup \boldsymbol{EXP} \cup \Delta A \models G \tag{5.2.8}$$

$$KB_U \cup \boldsymbol{HAP}^* \cup \boldsymbol{EXP} \cup \Delta A \models \mathcal{IC}_U \tag{5.2.9}$$

$$\boldsymbol{HAP}^* \cup \boldsymbol{EXP} \models \boldsymbol{E}_{peer}(X) \rightarrow \boldsymbol{H}(X) \tag{5.2.10}$$

$$\boldsymbol{HAP}^* \cup \boldsymbol{EXP} \models \boldsymbol{E}_{prot}(X) \rightarrow \boldsymbol{H}(X) \tag{5.2.11}$$

A peer is feeble conformant if all the possible histories are feeble conformant. A triple $(\boldsymbol{HAP}^*, \boldsymbol{EXP}, \Delta A)$ *is a* Feeble Conformant Interaction *if* $\boldsymbol{HAP}^*$ *is a feeble conformant history, and* $\boldsymbol{EXP}$ *and* ΔA *satisfy equations (5.2.8-5.2.11) (and* $\boldsymbol{EXP}$ *is minimal with respect to set inclusion).*

Example 5.2.2 Consider again the situation in Example 5.2.1. Given the possible history of Equation 5.2.7, the expectation of the protocol for the third message (ask) remains not fulfilled, so the peer *peer* is clearly non conformant.

Feeble conformance ensures that the peer *peer* will utter all the messages requested by the protocol, but it still does not require *peer* to avoid the messages forbidden by the protocol. We extend feeble conformance to a stronger version in the following.

A possible history is strong conformant if (it is feeble conformant and) all the happened events were expected both by the protocol and the interacting peer, and the hypothesis set ΔA is consistent. We include in this concept only the communications that involve the peer under observation (the other events, e.g., messages exchanged

[1] *Note the difference between Equations (5.2.10-5.2.11) and Equations (5.2.1-5.2.2): Equation (5.2.10) is used as a test, and requires* all *the expectations of the peer to be fulfilled, while Equation (5.2.1) is used to* generate *the behaviour of the peer and imposes only the fulfilment of the expectations the peer has about* itself. *Analogously for Equations (5.2.11) and (5.2.2).*

by other peers in a multi-party interaction, are always considered conformant). Also, by definition the messages sent by the peer comply to its own specifications, and symmetrically the messages sent by the other peers comply to the protocol.

Definition 5.2.3 (Strong Conformance). *A feeble conformant interaction* $(\boldsymbol{HAP}^*, \boldsymbol{EXP}, \Delta A)$ *is also a* Strong Conformant Interaction *if the following conditions hold:*

$$\boldsymbol{H}(m_x(peer, R, C)) \leftrightarrow \boldsymbol{E}_{prot}(m_x(peer, R, C)) \tag{5.2.12}$$

$$\boldsymbol{H}(m_x(S, peer, C)) \leftrightarrow \boldsymbol{E}_{peer}(m_x(S, peer, C)). \tag{5.2.13}$$

A Strongly Conformant History *is a history for which there exists a strongly conformant interaction. A peer is* Strongly Conformant *if all the possible histories are strongly conformant.*

Example 5.2.3 Let us change in the previous example the specifications of the protocol and of the peer, i.e., the peer specification is

$$\begin{array}{rcl} \mathbf{H}(ask(peer, R, X)) & \rightarrow & \mathbf{E}_{peer}(answer(R, peer, X)) \\ \mathbf{H}(answer(R, peer, X)) & \rightarrow & \mathbf{E}_{peer}(ack(peer, R, X)) \end{array}$$

and the protocol is

$$\mathbf{H}(ask(peer, R, X)) \rightarrow \mathbf{E}_{prot}(answer(R, peer, X)).$$

In this case, the peer has the intention to send the *ack*, so it will indeed send it in all the possible histories. The protocol does not prescribe this third message. The possible history becomes

$$\begin{array}{rcl} \mathbf{HAP^*}_2 & = & \{\mathbf{H}(ask(peer, other, X)), \\ & & \mathbf{H}(answer(other, peer, X)), \\ & & \mathbf{H}(ack(peer, other, X))\}. \end{array}$$

All the expectations of the protocol are fulfilled by one message of *peer*, so it is feeble conformant. However, *peer* will also send an unrequested message *ack*, that might confound *other*, undermining the interoperability. There exists no expectation from the protocol for the *ack* message, therefore *peer* is non strong conformant.

5.3 Operational semantics

The operational semantics is based on the $\mathcal{S}$CIFF proof procedure, presented in Section 2.5, and on the g-$\mathcal{S}$CIFFproof procedure, discussed in 4.4.

In order to prove conformance, we apply the two proof procedures to the two phases implicitly defined in the previous section. We decompose the proof of feeble conformance into a *generative* phase and a *test* phase. In the generative phase, we generate, by means of g-$\mathcal{S}$CIFF, all the possible histories. Of course, those histories need not be generated as ground histories (the set of ground histories can be infinite), but intensionally: the **H** events can contain variables, possibly with constraints à la Constraint Logic Programming [91].

In the test phase, we check with $\mathcal{S}$CIFF the compliance of the generated histories both with respect to the peer and the protocol specifications. If all the histories are conformant, the peer is feeble conformant to the protocol. Otherwise, if there exists at least one history that is not conformant, the peer is not (feeble) conformant.

Finally, we can prove strong conformance by checking that all the happened events were indeed expected both by the protocol and by the peer. This can in principle be done by using a version of $\mathcal{S}$CIFF that does not have abducibles, but it can also be performed during the second phase ($\mathcal{S}$CIFF) by adopting the same technique used in the fulfilment transition: if a **H** event matches both with an $\mathbf{E}_{peer}$ and a $\mathbf{E}_{prot}$ expectation, it is labelled *expected*; after the application of the *closure* transition, all events that were not expected are considered *unexpected*, showing that the peer was not strongly conformant.

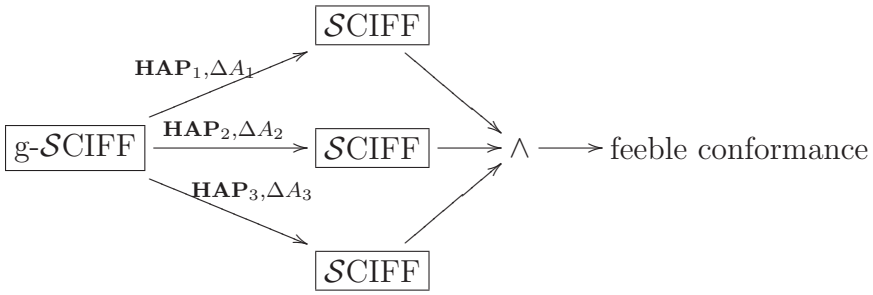


Figure 5.3: A^{l}LoWS architecture

5.4 Examples

Baldoni et al. [27, 26] show various examples of conformance and non-conformance of a web service to a choreography, and propose a framework based on Finite State Automata, to prove conformance. We show how their examples are addressed in A^{l}LoWS, based on Computational Logics. In particular, we will consider a choreography specification as a global protocol, and as interacting peers we will consider different Web Services.

5.4.1 Web service with more capabilities

The first example taken by [27] is the following. The choreography specification defines only one allowed interaction: ws sends a message m_1 and the other peer will reply with m_2:

$$\mathbf{H}(m_1(ws, X)) \rightarrow \mathbf{E}_{chor}(m_2(X, ws))$$

The web service specification is wider: after the first message the web service accepts as reply either m_2 or m_3:

$$\mathbf{H}(m_1(ws, X)) \rightarrow \mathbf{E}_{ws}(m_2(X, ws)) \vee \mathbf{E}_{ws}(m_3(X, ws))$$

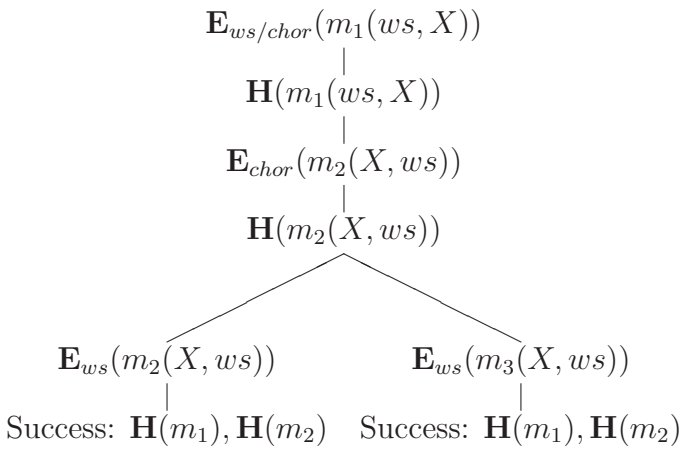


Figure 5.4: g-$\mathcal{S}$CIFF derivation for Example 5.4.1

In this case, Baldoni et al. state that the web service is conformant. In fact, in a legal conversation the message m_3 will never be received by ws, so the interoperability is ensured.

The g-$\mathcal{S}$CIFF proof procedure is started with the goal containing the expectation, both from the web service's and from the choreography's viewpoint, of the first event: $G_U = \mathbf{E}_{ws}(m_1(ws, X)) \wedge \mathbf{E}_{chor}(m_1(ws, X))$. g-$\mathcal{S}$CIFF in this case derives that there exists one possible history: $\{m_1, m_2\}$. A simplified representation of the derivation[2] is reported in Fig. 5.4. There are two alternative sets of expectations from the web service viewpoint, $\{\mathbf{E}_{ws}(m_1), \mathbf{E}_{ws}(m_2)\}$ and $\{\mathbf{E}_{ws}(m_1), \mathbf{E}_{ws}(m_3)\}$, but in the first phase the correspondence between expectations and happened events is not required (open derivation). In the second phase, the (only) generated history is checked; since there exists one set of expectations that is fulfilled by the generated history, the web service is considered feeble conformant. Since in the generated history there are no unexpected events, the web service is also strong conformant.

[2]The derivation does not report all the events and expectations, but for each node it gives only those that are added. We use $\mathbf{E}_{ws/chor}$ to indicate that the event is expected both by the choreography and by the web service.

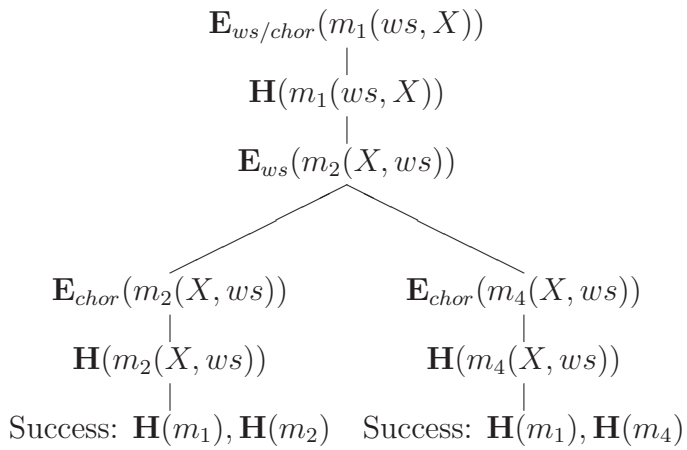


Figure 5.5: g-$\mathcal{S}$CIFF derivation for Example 5.4.2

5.4.2 Missing capability

The second example by Baldoni et al. is dual to the first: the web service accepts as reply only m_2

$$\mathbf{H}(m_1(ws, X)) \rightarrow \mathbf{E}_{ws}(m_2(X, ws))$$

while the choreography defines as valid two interactions

$$\mathbf{H}(m_1(ws, X)) \rightarrow \mathbf{E}_{chor}(m_2(X, ws)) \vee \mathbf{E}_{chor}(m_4(X, ws))$$

In this case, g-$\mathcal{S}$CIFF provides two possible histories: $\{\mathbf{H}(m_1), \mathbf{H}(m_2)\}$ and $\{\mathbf{H}(m_1), \mathbf{H}(m_4)\}$. In the second phase, $\mathcal{S}$CIFF detects non conformance of the history $\{\mathbf{H}(m_1), \mathbf{H}(m_4)\}$, because the web service's expectation $\mathbf{E}_{ws}(m_2(S, ws, C))$ remains unfulfilled in all possible derivation paths. This means that the web service is blocked waiting for message m_2, and will not process other messages, so it is non (feeble) conformant.

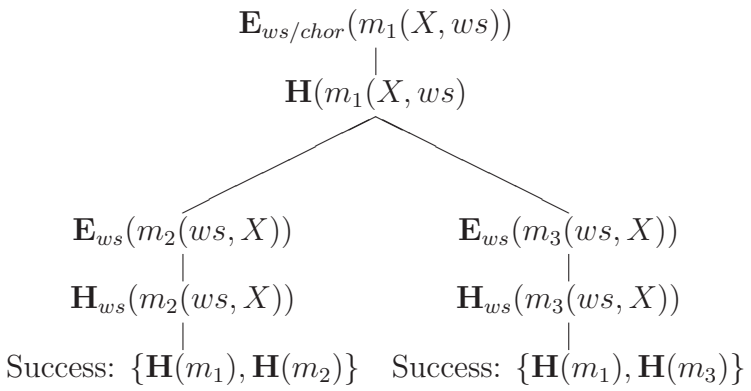


Figure 5.6: g-$\mathcal{S}$CIFF derivation for Example 5.4.3

5.4.3 Wrong reply

In the third example the web service assumes to have the freedom to reply either m_2 or m_3 to a question m_1

$$\mathbf{H}(m_1(X, ws)) \rightarrow \mathbf{E}_{ws}(m_2(ws, X)) \vee \mathbf{E}_{ws}(m_3(ws, X))$$

while the choreography does not grant such a freedom: only m_2 is legal

$$\mathbf{H}(m_1(X, ws)) \rightarrow \mathbf{E}_{chor}(m_2(ws, X))$$

This case is judged non conformant by Baldoni et al, as there might be paths in which the web service utters the forbidden message m_3. g-$\mathcal{S}$CIFF computes two possible histories ($\{\mathbf{H}(m_1), \mathbf{H}(m_2)\}$ and $\{\mathbf{H}(m_1), \mathbf{H}(m_3)\}$). The first is compliant, according to $\mathcal{S}$CIFF, while in the second the choreography's expectation $\mathbf{E}_{chor}(m_2)$ remains pendent.

5.4.4 Predefined answer

The dual of example 5.4.3 is when the choreography lets the web service choose to reply m_2 or m_3 to a question m_1,

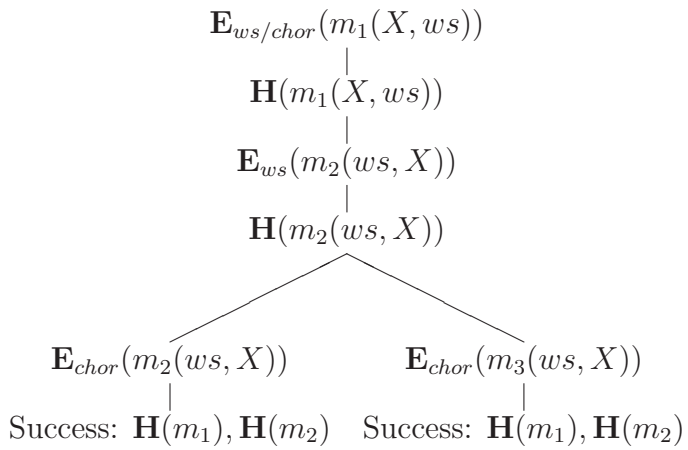


Figure 5.7: g-$\mathcal{S}$CIFF derivation for Example 5.4.4

$$\mathbf{H}(m_1(X, ws)) \rightarrow \mathbf{E}_{chor}(m_2(ws, X)) \vee \mathbf{E}_{chor}(m_3(ws, X))$$

while the web service sticks to the reply m_2

$$\mathbf{H}(m_1(X, ws)) \rightarrow \mathbf{E}_{ws}(m_2(ws, X)).$$

Again, A^{l}LoWS provides a correct proof: g-$\mathcal{S}$CIFF gives one possible history, which is reported (feeble and strong) conformant by $\mathcal{S}$CIFF in the second phase.

Thus, in all the examples by Baldoni et al., A^{l}LoWS provides the same answer proposed in [27].

5.4.5 Web services taking early decisions

In [26] Baldoni et al. have identified also conformance issues due to particular brancnhing structures in the protocol/behavioural definitions.

In Figure 5.8 it is shown one of the possible situations. The Web Service behavioural interface states that, as soon as the message m_0 is sent out, the ws will decide to choose between the two different paths m_1, m_2 and m_1, m_3. This choice is

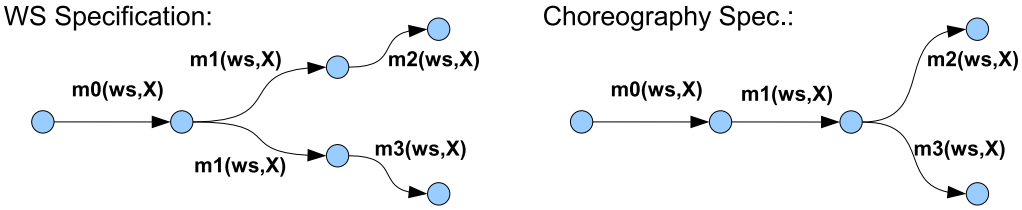


Figure 5.8: Message branching in the Web Service Specification

made (presumably) on the basis on some internal, private policy that ws does not make public. The choreography instead specify that, from the global viewpoint, the decision between m_2 and m_3 is taken after m_1. This situation is considered by the Baldoni et al. as a conformant one.

The issue not clearly stated in [26] is that the behavioural interface of ws introduces two different paths, and the choice between them is made upon some internal policy. We map this situation by introducing, in the ws specification, a further abducible, whose meaning is to map this internal choice made at a early stage (in this case the predicate p):

$$\begin{aligned}
\mathbf{H}(m_0(ws,X)) &\rightarrow \mathbf{E}_{WS}(m_1(ws,X)) \wedge p \\
&\vee \mathbf{E}_{WS}(m_1(ws,X)) \wedge not_p \\
\mathbf{H}(m_1(ws,X)) \wedge p &\rightarrow \mathbf{E}_{WS}(m_2(ws,X)) \\
\mathbf{H}(m_1(ws,X)) \wedge not_p &\rightarrow \mathbf{E}_{WS}(m_3(ws,X))
\end{aligned}$$

In the choreography specification instead, it is not needed to introduce a further predicate, since the paths can be all distinguished from each other in any moment. Hence the choreography specification will be like:

$$\begin{aligned}
\mathbf{H}(m_0(ws,X)) &\rightarrow \mathbf{E}_{chor}(m_1(ws,X)) \\
\mathbf{H}(m_1(ws,X)) &\rightarrow \mathbf{E}_{chor}(m_2(ws,X)) \\
&\vee \mathbf{E}_{chor}(m_3(ws,X))
\end{aligned}$$

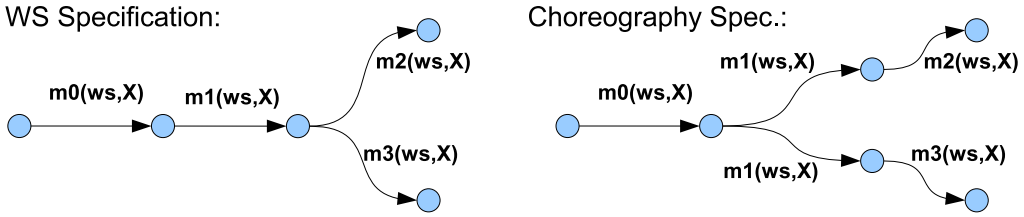


Figure 5.9: Message branching in the Choreography Specification

The g-$\mathcal{S}$CIFF generates only two histories:

$$\mathbf{HAP}_1 = \{\mathbf{H}(m_0), \mathbf{H}(m_1), p, \mathbf{H}(m_2)\}$$
$$\mathbf{HAP}_2 = \{\mathbf{H}(m_0), \mathbf{H}(m_1), \neg p, \mathbf{H}(m_3)\}$$

The $\mathcal{S}$CIFF recognize both the histories, hence the web service is considered as being strong conformant.

5.4.6 A choreography that takes an early decision

The dual case of the situation presented in Section 5.4.5 is when the choreography specification provides two different paths (possibly distinguished from each other only later), while the web services choose a different path only later. Such situation is graphically represented in Figure 5.9.

This case is not considered as conformant, since ws could choose to utter message m_3 at a later moment, while the choreography expects ws to send, for example, m_2.

As we did before, we use an abducible predicate to distinguish the different paths chosen (this time at the choreography level). ws can be specified as:

$$\begin{aligned} \mathbf{H}(m_0(ws, X)) &\rightarrow \mathbf{E}_{WS}(m_1(ws, X)) \\ \mathbf{H}(m_1(ws, X)) &\rightarrow \mathbf{E}_{WS}(m_2(ws, X)) \\ &\vee \mathbf{E}_{WS}(m_3(ws, X)) \end{aligned}$$

The choreography instead is specified as:

$$\begin{aligned}
\mathbf{H}(m_0(ws, X)) &\rightarrow \mathbf{E}_{chor}(m_1(ws, X)) \wedge p \\
&\vee \mathbf{E}_{chor}(m_1(ws, X)) \wedge not_p \\
\mathbf{H}(m_1(ws, X)) \wedge p &\rightarrow \mathbf{E}_{chor}(m_2(ws, X)) \\
\mathbf{H}(m_1(ws, X)) \wedge not_p &\rightarrow \mathbf{E}_{chor}(m_3(ws, X))
\end{aligned}$$

This time the g-$\mathcal{S}$CIFF generates four different histories:

$$\begin{aligned}
\mathbf{HAP}_1 &= \{\mathbf{H}(m_0), \mathbf{H}(m_1), \mathbf{H}(m_2), p\} \\
\mathbf{HAP}_2 &= \{\mathbf{H}(m_0), \mathbf{H}(m_1), \mathbf{H}(m_2), \neg p\} \\
\mathbf{HAP}_3 &= \{\mathbf{H}(m_0), \mathbf{H}(m_1), \mathbf{H}(m_3), p\} \\
\mathbf{HAP}_4 &= \{\mathbf{H}(m_0), \mathbf{H}(m_1), \mathbf{H}(m_3), \neg p\}
\end{aligned}$$

However, ws results to be not feeble conformant: taking, for example, the history $\mathbf{HAP}_2$, it is possible to see that (since p has been abduced), the choreography has an expectation about a message m_3 that will not fulfilled by ws.

We would like to note, however, that this situation appears quite strange and in contrast with the same meaning of choreography. In fact, the specification states that a choice is taken, without any criteria, and without any possibility of understanding (until is too late) which path is desired at a global level. In our opinion, this contradict the goal for which choreographies were introduced: i.e., to ease and properly rule the interaction between different peers.

5.4.7 Web service that decides too early to wait for a message

A similar situation to the one discussed in Section 5.4.5 is the one depicted in Figure 5.10. In this situation however, ws decides to wait for a message m_2 or m_3 when before sending the message m_1 (at a previous interaction step). Note that the only difference w.r.t. the situation presented in Section 5.4.5 is that ws waits (instead of sending) a message.

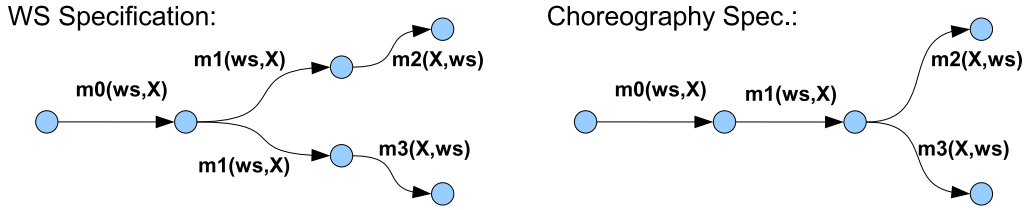


Figure 5.10: Message branching in the Web Service Specification - waiting case

Again, we use an abducible predicate to represent such situation. Hence, ws is specified as follow:

$$\begin{aligned}
\mathbf{H}(m_0(ws,X)) &\rightarrow \mathbf{E}_{WS}(m_1(ws,X)) \wedge p \\
&\vee \mathbf{E}_{WS}(m_1(ws,X)) \wedge not_p \\
\mathbf{H}(m_1(ws,X)) \wedge p &\rightarrow \mathbf{E}_{WS}(m_2(X,ws)) \\
\mathbf{H}(m_1(ws,X)) \wedge not_p &\rightarrow \mathbf{E}_{WS}(m_3(X,ws))
\end{aligned}$$

The choreography specification instead is not different form the previous one:

$$\begin{aligned}
\mathbf{H}(m_0(ws,X)) &\rightarrow \mathbf{E}_{chor}(m_1(ws,X)) \\
\mathbf{H}(m_1(ws,X)) &\rightarrow \mathbf{E}_{chor}(m_2(X,ws)) \\
&\vee \mathbf{E}_{chor}(m_3,(X,ws))
\end{aligned}$$

Although the situation is very similar to the one discussed in Section 5.4.5, in this case Baldoni et al. state that the ws is not conformant to the choreography. In fact, it might happen that the ws, following some internal policy, choose at a certain point to wait for a message m_2, while the choreography still leave this choice open. Later, the peer X is still free to send m_3 rather than m_2: if this is the choice, ws will wait for a message that no one will ever utter.

This time the g-$\mathcal{S}$CIFF generates four different histories:

$$\begin{aligned}
\mathbf{HAP}_1 &= \{\mathbf{H}(m_0), p, \mathbf{H}(m_1), \mathbf{H}(m_2)\} \\
\mathbf{HAP}_2 &= \{\mathbf{H}(m_0), \neg p, \mathbf{H}(m_1), \mathbf{H}(m_2)\} \\
\mathbf{HAP}_3 &= \{\mathbf{H}(m_0), p, \mathbf{H}(m_1), \mathbf{H}(m_3)\} \\
\mathbf{HAP}_4 &= \{\mathbf{H}(m_0), \neg p, \mathbf{H}(m_1), \mathbf{H}(m_3)\}
\end{aligned}$$

If we consider the history $\mathbf{HAP}_2$, we can note that $\neg p$ has been abduced. This means that ws will wait for message m_3. Unfortunately, this message will never be uttered. ws is not (feeble) conformant w.r.t. the choreography specification.

5.4.8 Choreography that decides early to wait for a message to be received by ws

The dual case of the situation discussed in Section 5.4.6 is shown in Figure 5.11. In this case, however, the choreography states that the path is chosen before ws sends out m_1. Note that this time m_2 and m_3 are expected to be received by ws, and note sent as in Section 5.4.6

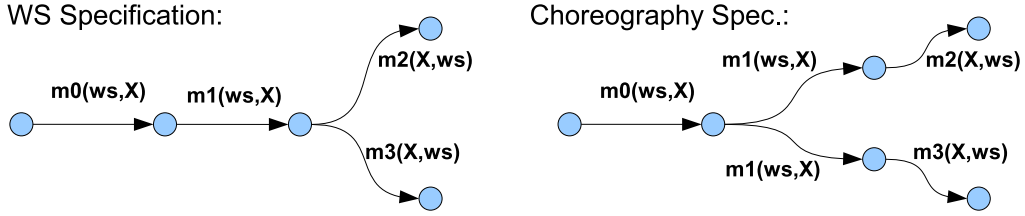


Figure 5.11: Message branching in the Choreography Specification - waiting case

The ws can be specified as:

$$\begin{aligned}
\mathbf{H}(m_0(ws, X)) &\rightarrow \mathbf{E}_{WS}(m_1(ws, X)) \\
\mathbf{H}(m_1(ws, X)) &\rightarrow \mathbf{E}_{WS}(m_2(X, ws)) \\
&\vee \mathbf{E}_{WS}(m_3(X, ws))
\end{aligned}$$

The choreography instead is specified by introducing also an abducible predicate p, used to discriminate the different path chosen in the interaction:

$$\begin{aligned}
\mathbf{H}(m_0(ws, X)) &\rightarrow \mathbf{E}_{chor}(m_1(ws, X)) \wedge p \\
&\vee \quad \mathbf{E}_{chor}(m_1(ws, X)) \wedge not_p \\
\mathbf{H}(m_1(ws, X)) \wedge p &\rightarrow \mathbf{E}_{chor}(m_2(X, ws)) \\
\mathbf{H}(m_1(ws, X)) \wedge not_p &\rightarrow \mathbf{E}_{chor}(m_3(X, ws))
\end{aligned}$$

This time the early choice taken in the choreography specification does not undermine the conformance: in fact ws is still capable to process both the message m_2 and m_3.

The g-$\mathcal{S}$CIFF generates two different histories:

$$\begin{aligned}
\mathbf{HAP}_1 &= \{\mathbf{H}(m_0), \mathbf{H}(m_1), \mathbf{H}(m_2), p\} \\
\mathbf{HAP}_2 &= \{\mathbf{H}(m_0), \mathbf{H}(m_1), \mathbf{H}(m_3), \neg p\}
\end{aligned}$$

Both $\mathbf{HAP}_1$ and $\mathbf{HAP}_2$ are compliant (by $\mathcal{S}$CIFF), hence ws is strong conformant w.r.t. the choreography specification.

5.4.9 Forbidden message

In all the previous cases, feeble and strong conformance coincide. However, there might be instances in which the choreography assumes that the interaction has finished, while the web service continues sending messages. For example, the choreography expects only one message:

$$G_{chor} = \mathbf{E}_{chor}(m_1(X, ws))$$

while ws will send back a message m_2:

$$\mathbf{H}(m_1(X, ws)) \rightarrow \mathbf{E}_{ws}(m_2(ws, X)).$$

In this case, the only possible history is $\mathbf{HAP}^* = \{\mathbf{H}(m_1), \mathbf{H}(m_2)\}$. This history does not leave any pending expectations, both from the choreography and from the web service's viewpoints, so ws is judged feeble conformant. However, the message m_2 was not expected by the choreography, and ws is not strong conformant.

5.4.10 Mutual exclusion

Many protocols include mutual exclusion between choices: for instance a choreography might prescribe that if a given condition on a message m_1 holds, a message m_2 should be exchanged, otherwise another message m_3 should be sent. In A^{l}LoWS, conditions can be expressed by means of constraints (either the ones predefined in the underlying solver, i.e., CLP(FD), or user-defined) or by means of defined predicates. As a simple example, consider the following: the choreography prescribes to reply either m_2 or m_3, depending on the content of the previous message m_1:

$$\begin{aligned} \mathbf{H}(m_1(X, ws, C)) \quad &\rightarrow \quad \mathbf{E}_{chor}(m_2(ws, X, C_2)), C > 0 \\ &\vee \quad \mathbf{E}_{chor}(m_3(ws, X, C_3)), C \leq 0 \end{aligned}$$

while the web service always replies m_2:

$$\mathbf{H}(m_1(X, ws, C)) \rightarrow \mathbf{E}_{ws}(m_2(ws, X, C_2))$$

g-$\mathcal{S}$CIFF generates two possible histories, with variables and constraints upon the variables (Fig 5.12). In both the messages m_1 and m_2 are generated, but while in the first the proof procedure assumes that C takes a value greater than 0, in the second C is non positive. In the second phase, $\mathcal{S}$CIFF takes as input both the happened events and the constraint store, and accepts as conformant the first history, while discarding as non-conformant the second.

Notice that constraints scope is not restricted only to variables in the content, but might involve all the variables in the message, including time. The choreography

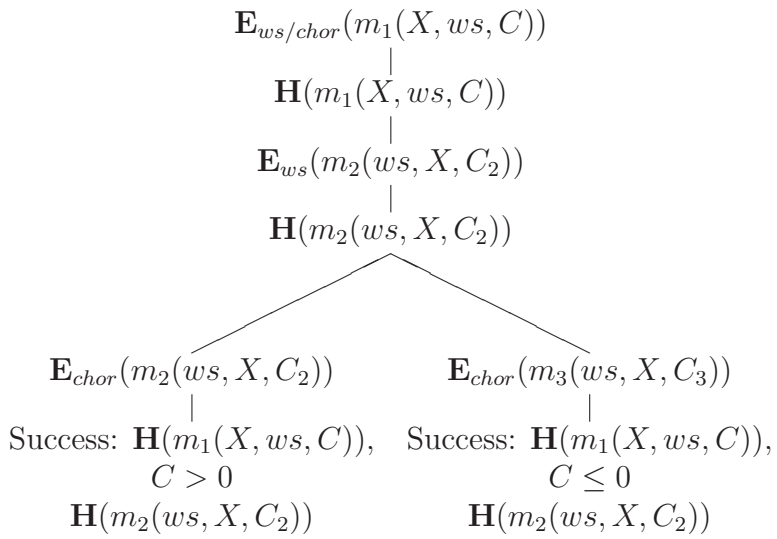


Figure 5.12: g-$\mathcal{S}$CIFF derivation for Example 5.4.10

might contain conditions on deadlines (if you receive a message within 5 minutes answer *ok*, otherwise reply *too late*)), or on participants (if the sender is authorized, etc.).

5.4.11 Deadlines

Suppose that the choreography specifies a deadline for the receipt of a given message m_2:

$$\begin{aligned} \mathbf{H}(m_1(X, ws, C_1), T_1) \quad &\rightarrow \quad \mathbf{E}_{chor}(m_2(ws, X, C_2), T_2) \\ &\wedge \quad T_2 < T_1 + \delta_{chor} \end{aligned}$$

The web service ws, however, replies within a deadline that might be different:

$$\begin{aligned} \mathbf{H}(m_1(X, ws, C_1), T_1) \quad &\rightarrow \quad \mathbf{E}_{ws}(m_2(ws, X, C_2), T_2) \\ &\wedge \quad T_2 < T_1 + \delta_{ws}. \end{aligned}$$

In this case, the only possible history is

$$\begin{aligned} \mathbf{HAP}^* \quad = \{ \quad &\mathbf{H}(m_1(X, ws, C_1), T_1), \\ &\mathbf{H}(m_2(ws, X, C_2), T_2), T_2 < T_1 + \delta_{ws} \quad \} \end{aligned}$$

Applying $\mathcal{S}$CIFF to the generated history, we get the expectation

$$\mathbf{E}_{chor}(m_2(ws, X, C_2), T_2) \wedge T_2 < T_1 + \delta_{chor}$$

; this expectation matches with the second item of the $\mathbf{HAP}^*$ history if a further condition holds: $T_2 < T_1 + \delta_{chor}$. Coherently with the philosophy of Constraint Logic Programming, $\mathcal{S}$CIFF provides this constraint in output, as a conditional answer: the web service is conformant provided that the answer arrives before the deadline in the choreography specification. Depending on the propagation performed by the adopted constraint solver, the information provided could be even more significant. For example, if the two values δ_{ws} and δ_{chor} are ground, a CLP(FD) solver would provide the conditional answer only if necessary, i.e., if $\delta_{chor} < \delta_{ws}$, (the deadline imposed by the choreography is more tight than the one the web service will meet).

5.5 A test conformance example

In this section we exemplify the proposed approach by using a very simple choreography specification, shown in Fig. 5.1. The interaction is initiated by a $User$ that asks the Flight Service FS to book a flight. If there are seats available on the plane, the FS will reply with *flightOffer*, specifying the $Price$ of booking the seat. Otherwise, the FS replies with *notAvailable*.

The $offer$ can be accepted (with *ackOffer*) or refused (with *nAckOffer*) by the $User$. If the *offer* is accepted, the flight company will book the seat. The $User$, after booking, has still the freedom to $Cancel$ the booking. Otherwise, it will issue a payment oder (*payment*) to the $Bank$, that will send the notification ($notifyPayment$) to the creditor, the FS.

When the FS has received both the booking order (*ackOffer*) and the payment (*notifyPayment*), it will normally issue the $flightTicket$ to the $User$; however, the FS retains the right to refuse the ticket and send a $flightCancelled$ message in case of problems (e.g., overbooking or other error conditions).

Fig. 5.2 shows the behavioural interface of a Flight Server web service; the specification in terms of ICs is in Spec. 5.1.2. The FS establishes that the late payment is an error condition, and will cancel the booking if the payment notification does not arrive within δ time units after the booking.

In next section, we show how the conformance of *fs* is proven in A^{l}LoWS.

5.5.1 Conformance of the Flight Service

The test of conformance of the Flight Service fs is performed by generating, through g-$\mathcal{S}$CIFF, the set of the possible histories. The g-$\mathcal{S}$CIFF derivation provides five

possible histories:

$$
\begin{aligned}
\mathbf{HAP^*}_1 = \{&\mathbf{H}(request(U, fs, F), T_r),\\
&\mathbf{H}(\mathit{offer}(fs, C, F, P), T_o),\\
&\mathbf{H}(\mathit{ackOffer}(C, fs, F, P), T_a),\\
&\mathbf{H}(payment(C, B, P, fs), T_p),\\
&\mathbf{H}(notifyPayment(B, fs, P)), T_n) \wedge T_n > T_a + \delta\\
&\mathbf{H}(\mathit{flightCancelled}(fs, C, F), T_c)\},\\
\mathbf{HAP^*}_2 = \{&\mathbf{H}(request(U, fs, F), T_r),\\
&\mathbf{H}(\mathit{offer}(fs, C, F, P), T_o),\\
&\mathbf{H}(\mathit{ackOffer}(C, fs, F, P), T_a),\\
&\mathbf{H}(payment(C, B, P, fs), T_p),\\
&\mathbf{H}(notifyPayment(B, fs, P)), T_n) \wedge T_n \leq T_a + \delta\\
&\mathbf{H}(\mathit{flightTicket}(fs, C, F), T_t)\},\\
\mathbf{HAP^*}_3 = \{&\mathbf{H}(request(U, fs, F), T_r),\\
&\mathbf{H}(\mathit{offer}(fs, C, F, P), T_o),\\
&\mathbf{H}(\mathit{ackOffer}(C, fs, F, P), T_a),\\
&\mathbf{H}(cancel(C, fs, F), T_c),\\
&\mathbf{H}(\mathit{flightCancelled}(fs, C, F))\},
\end{aligned}
$$

$$\begin{aligned}
\mathbf{HAP^*}_4 = \{&\mathbf{H}(request(U, fs, F), T_r),\\
&\mathbf{H}(\mathit{offer}(fs, C, F, P), T_o),\\
&\mathbf{H}(nAckOffer(C, fs, F, P), T_a)\},\\
\mathbf{HAP^*}_5 = \{&\mathbf{H}(request(U, fs, F), T_r),\\
&\mathbf{H}(notAvailable(fs, C, F, P), T_n)\}.
\end{aligned}$$

Two of the histories include time constraints. All the possible histories are trivially conformant: they satisfy both the expectations of the choreography, and those of the web service fs. Thus, fs is feeble conformant. Moreover, all the generated events are expected, and this shows that fs is also strong conformant.

5.5.2 Conformance of the User web service

Suppose now that the user web service has the behavioural interface in Spec. 5.5.1.

Note that the $User$ implements a policy for deciding whether to accept (*ackOffer*) or refuse (*nAckOffer*) the *offer* of the FS: if the $Price$ is less than a max quota, then the *offer* is accepted (and declined otherwise). Also, this $User$ web service does not have expectations on the $Bank$'s reply: in fact, in this choreography, the $Bank$ does not need to notify the $User$ about the payment. Finally, $User$ always expects to receive a ticket after paying.

Invoked with the goal $\mathbf{E}_{chor}(request) \wedge \mathbf{E}_{user}(request)$, g-$\mathcal{S}$CIFF generates the

Specification 5.5.1 The behavioural interface of a web service that is not conformant for the role of $User$

$$\begin{aligned}&\mathbf{H}(request(user, FS, Flight), T_r)\\ \rightarrow&\mathbf{E}_{user}(offer(FS, user, Flight, Price), T_o)\\ \vee&\mathbf{E}_{user}(notAvailable(FS, user, Flight), T_{na})\end{aligned} \tag{5.5.1}$$

$$\begin{aligned}&\mathbf{H}(offer(FS, user, Flight, Price), T_o)\\ \rightarrow&\mathbf{E}_{user}(ackOffer(user, FS, Flight, Price), T_a) \wedge Price \leq max\\ \vee&\mathbf{E}_{user}(nAckOffer(user, FS, Flight, Price), T_a) \wedge Price > max\end{aligned} \tag{5.5.2}$$

$$\begin{aligned}&\mathbf{H}(ackOffer(user, FS, Flight, Price), T_a)\\ \rightarrow&\mathbf{E}_{user}(payment(user, Bank, Price, FS), T_f)\\ \vee&\mathbf{E}_{user}(cancel(user, FS, Flight), T_f)\end{aligned} \tag{5.5.3}$$

$$\begin{aligned}&\mathbf{H}(ackOffer(user, FS, Flight, Price), T_a)\\ &\wedge \mathbf{H}(payment(user, Bank, Price, FS), T_p)\\ \rightarrow&\mathbf{E}_{user}(flightTicket(FS, user, Flight), T_f)\end{aligned} \tag{5.5.4}$$

$$\begin{aligned}&\mathbf{H}(cancel(user, FS, Flight), T_a)\\ \rightarrow&\mathbf{E}_{user}(flightCancelled(FS, user, Flight), T_f)\end{aligned} \tag{5.5.5}$$

$$\begin{aligned}&\mathbf{H}(payment(User, Bank, Price, Creditor), T_p)\\ \rightarrow&\mathbf{E}_{chor}(notifyPayment(Bank, Creditor, Price), T_n)\end{aligned} \tag{5.5.6}$$

possible histories, that in this case are:

$$\begin{aligned}\mathbf{HAP}^*{}_1 = \{&\mathbf{H}(request(U, fs, F), T_r),\\ &\mathbf{H}(offer(fs, C, F, P), T_o),\\ &\mathbf{H}(ackOffer(C, fs, F, P), T_a) \wedge P \leq max,\\ &\mathbf{H}(payment(C, B, P, fs), T_p),\\ &\mathbf{H}(notifyPayment(B, fs, P)), T_n),\\ &\mathbf{H}(flightCancelled(fs, C, F), T_c)\},\end{aligned}$$

$$\begin{aligned}
\mathbf{HAP^*}_2 = \{&\mathbf{H}(request(U, fs, F), T_r),\\
&\mathbf{H}(\mathit{offer}(fs, C, F, P), T_o),\\
&\mathbf{H}(ackOffer(C, fs, F, P), T_a) \wedge P \leq max,\\
&\mathbf{H}(payment(C, B, P, fs), T_p),\\
&\mathbf{H}(notifyPayment(B, fs, P)), T_n),\\
&\mathbf{H}(\mathit{flightTicket}(fs, C, F), T_t)\},\\
\mathbf{HAP^*}_3 = \{&\mathbf{H}(request(U, fs, F), T_r),\\
&\mathbf{H}(\mathit{offer}(fs, C, F, P), T_o),\\
&\mathbf{H}(ackOffer(C, fs, F, P), T_a) \wedge P \leq max,\\
&\mathbf{H}(cancel(C, fs, F), T_c),\\
&\mathbf{H}(\mathit{flightCancelled}(fs, C, F), T_{fc})\},\\
\mathbf{HAP^*}_4 = \{&\mathbf{H}(request(U, fs, F), T_r),\\
&\mathbf{H}(\mathit{offer}(fs, C, F, P), T_o) \wedge P \leq max,\\
&\mathbf{H}(nAckOffer(C, fs, F, P), T_a)\},\\
\mathbf{HAP^*}_5 = \{&\mathbf{H}(request(U, fs, F), T_r),\\
&\mathbf{H}(notAvailable(fs, C, F, P), T_n)\}.
\end{aligned}$$

However, in the second phase, $\mathcal{S}$CIFF applied to the history $\mathbf{HAP^*}_1$ signals that the *user*'s expectation $\mathbf{E}_{user}(\mathit{flightTicket}(FS, user, Flight), T_t)$ remains unfulfilled, proving that *user* is not (feeble) conformant. In fact, this *user* web service undermines the interoperability with other web services conformant with the same choreography. As an example, we can easily see that in case the *Bank* does not provide the *notifyPayment* within the deadline imposed by fs (Spec. 5.1.2), the two web services are unable to complete the choreography.

5.6 Related Works

A number of languages for specifying service choreographies and testing "a priori" and/or "run-time" conformance have been proposed in the literature. Two examples of these languages are represented by state machines [33] and Petri nets [63].

Our work is highly inspired by Baldoni et al. [27]. We adopt, like them, a Multi-agent Systems point of view, in defining a priori conformance in order to guarantee interoperability. As in [27], we give an interpretation of the a-priori conformance as a property that relates two formal specifications: the global one determining the conversations allowed by the protocol and the local one related to the single peer. But, while in [27] a global interaction protocol is represented as a finite state automation, we adopt the formalisms and technologies developed in $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF. For example, a difference between our work and [27] can be found in the number of parties as they can manage only 2-party protocols while we do not impose any limit. We also manage concurrency, which they do not consider at the moment.

Another similar work is described in [33]. In this work, authors focus on two-party choreographies involving each one a requester and a provider (named service conversations) and formulate some requirements for a modelling language suitable for them. The requirements include genericity, automated support, and relevance. The authors argue that state machines satisfy these requirements and sketch an architecture of a service conversation controller capable of monitoring messages exchanged between a requester and provider in order to determine whether they conform to a conversation.

An example of use of Petri nets for the formalization of choreographies is discussed in [63]. Four different viewpoints (interface behaviour, provider behaviour, choreography, and orchestration) and relations between viewpoints are identified and

formalised. These relations are used to perform (global) consistency checking of multi-viewpoint service designs thereby providing a formal foundation for incremental and collaborative approaches to service-oriented design. Our proposal is limited to a deep analysis of the relation between choreographies and behaviour interfaces but deal with both "a priori" and "run-time" conformance.

Endriss et al. [68, 70] apply a formalism based on computational logic to the a-priori conformance in the MAS field. Their formalism is similar to the one we propose, but they restrict their analysis to a particular type of protocols (named *shallow protocols*). Doing this, they address only 2-party interactions, without the possibility of expressing conditions over the content of the exchanged messages, and without considering concurrency. While the two works agree on the notion of strong/exhaustive conformance, we have dual notions of feeble/weak conformance: in [68, 70] weak conformant is an agent that does not perform forbidden actions, but we have no knowledge on its capability to perform requested actions. Dually, in this work, we call feeble conformant an agent that does execute all the required actions, but there is no knowledge on its ability to avoid forbidden actions.

The use of abduction for verification was also explored in other work. Noteworthily, Russo et al. [128] use an abductive proof procedure for analysing event-based requirements specifications. Their method uses abduction for analysing the correctness of specifications, while our system is more focussed on the check of compliance/conformance of a set of web services.

In [100], the authors tackle the problem of verifying (general and specific) properties of a Service obtained from the composition of many web services. Each web

service specification (written in BPEL4WS) is translated in a *labelled state transition system*; then, by applying a composition operator, they get the state transition system representing the composed service. Finally, model checking techniques are applied to this latter model, to the end of verifying the properties. Note that, by appropriately defining and extending the validity of the composed STS, they tackle different communication models (synchronous, ordered asynchronous and unordered asynchronous communications) that appear to be quite common in real cases.

Both our work and [100] focus on verifying interoperability, but while we concentrate on the interoperability issue of a single peer w.r.t. a global protocol, in [100] the authors address the interoperability of a group of web services, referring only to the interface behavior of each web service. However, we share the same intuition that *"the situation where some messages can be emitted without being ever consumed should not occur in valid composition."*. The authors address also the problem of proving properties about the possible interactions between a group of inter-operating web services, by means of model checking techniques. We are discuss our results in this verification type in Chapter 4.

Chapter 6

Protocol Executability: the SCIFF Agent Architecture

As already introduced in Section 1.1, the idea behind the *Executability* is to re-use the specification of an interaction protocol, in order to ease the implementation of the software peers that should interact using such a protocol.

When implementing the peers that use an interaction protocol, several problems could arises; to cite some:

- the chosen protocol might not be specified enough: e.g., the TCP protocol specification [124] does not specify a minimum time interval before a *syn* message can be retransmitted;
- the specification of the chosen protocol could contain ambiguities (due, for example, to the use of natural language);
- the implementation of a peer could contain software bugs;
- if a reference implementation is missing, it would be hard also to run any test of functionality for the implementation of new peers;

Executability is a property related to both the interaction protocol as well as to the specification language used to define such a protocol. It is related to the protocol itself since quite frequently (especially in the MAS scenario) a protocol allows a peer for several different actions (communicative actions in MAS). These actions, from the protocol viewpoint, can be preferred non-deterministically by the peer: typically, the choice of an action is made by the peer on the basis of some (possibly private and internal) policy.

The executability property is related also to the specification language, that should be able to support, somehow, the peer developing process. For example, a specification given in the natural language would be easier for a human reader, but probably useless for automatizing the development process of a peer. On the other hand, a more formal specification of the same protocol would be useful for rapid prototyping a software peer, but it might result of difficult comprehension for a human software developer.

In this chapter we present an extension of an existing agent architecture, where the single agents are programmed by directly using a protocol specification. The existing agent architecture (JADE, [31]) provides the communication facilities, as well as an entire agent infrastructure compliant with the FIPA specifications [75]. The "mind" of the agent instead is realized by means of the $\mathcal{S}$CIFF Proof Procedure; it takes as input the protocol specified in the $\mathcal{S}$CIFF Language, plus some additional information, and computes an agent behaviour.

Contributions of the author. The author contributed in a substantial way to the contents presented in this chapter. However the ideas presented and the obtained results are also consequence of the fruitful discussions with the colleagues and the

senior researchers.

Chapter organization. This section is organized as follow. In Section 6.1 we provide an overview of the agent proposed architecture, where beside the $\mathcal{S}$CIFF Agent, we have introduced the possibility of checking compliance through the *SOCS-SI* tool. In Section 6.2 we present the $\mathcal{S}$CIFF Agent and its internal architecture, showing how a protocol specification can be used as a base for programming the behaviour of a single agent. In Section 6.3 instead we discuss which syntactic restrictions, if applied to the protocol definition, can guarantee that the $\mathcal{S}$CIFF Agent enjoy the property of being conformant (w.r.t. the protocol). Finally, in Section 6.4 we present some implementation details while in Section 6.5 we discuss the related works.

6.1 The $\mathcal{S}$CIFF Agent Platform

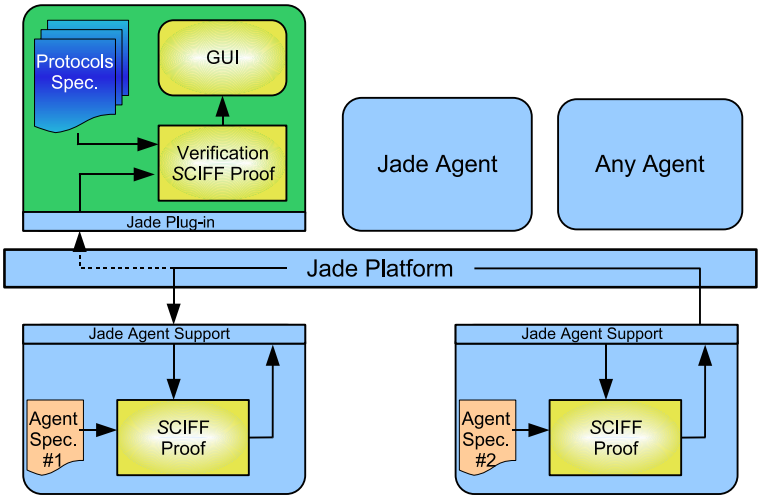


Figure 6.1: Overview of the $\mathcal{S}$CIFF agent platform architecture.

The $\mathcal{S}$CIFF agent platform, represented in Figure 6.1, has been implemented on top of the JADE agent platform [31]. All the components have been implemented as JADE agents, thus exploiting the capabilities of this powerful agent platform. The single components can be used also separately, and in conjunction with other JADE-compliant agents.

In Figure 6.1 two different components have been depicted: the *SOCS-SI* tool (see Section 3.1), that has been extended to the JADE platform, and the $\mathcal{S}$CIFF Agent: both components use the $\mathcal{S}$CIFF Proof Procedure as the reasoning engine, but they perform different tasks. *SOCS-SI* "captures" at run-time the messages exchanged by the agents through the communication platform, and verify the conformance of the dialogues w.r.t. a given protocol specification. The $\mathcal{S}$CIFF Agent instead uses the $\mathcal{S}$CIFF Proof Procedure to compute, for each step of the interaction, which is its next action (communicative action, in this case).

6.2 The SCIFF Agent

The $\mathcal{S}$CIFF agent is a generic agent whose behaviour is determined by its specification, provided by means of the $\mathcal{S}$CIFF language, as explained later. The agent communication language used by the $\mathcal{S}$CIFF agents is based on the ACL used in JADE (and defined by FIPA [75]). An important difference however is that the $\mathcal{S}$CIFF agent is not restricted to use the pre-defined set of FIPA message performatives: the agent designer can use any desired performative. If this is the case, the semantics of such performatives should be always expressed by providing, besides the agent specification, a proper protocol, thus providing a social semantics, and possibly by means of the $\mathcal{S}$CIFF language. Of course, careful considerations must be taken when introducing such new performatives, since inter-operability with other agent implementations is not guaranteed anymore, unless it is known a-priori that they support non-FIPA performatives.

6.2.1 Specification of the agent behaviour

The main purpose of the $\mathcal{S}$CIFF Agent is to provide an implementation of a peer playing a certain role in a given interaction protocol. The idea is to specify the agent behaviour by means of the same protocol specification (given using the $\mathcal{S}$CIFF Language).

The protocol specification is used in the following way: given a set of happened events, it is possible to query the $\mathcal{S}$CIFF Proof Procedure in the open modality (i.e. considering that new events can still happen). The $\mathcal{S}$CIFF Proof Procedure is used to compute all the abductive answers Δ_i, for which the given history is compliant w.r.t. the protocol. Each set Δ_i represent a possible allowed behaviour of the peers. This

behaviour is expressed in terms of expectations about the events. In this scenario, all the events are message exchanges (i.e., we restrict to the communicative acts).

The $\mathcal{S}$CIFF Agent chooses one of the Δ_i and try to behave as he is expected to. Practically, all the expectations about the agent to send a message to another agent are considered mandatory, and such messages are effectively sent out. Expectations about other agents to send messages to the $\mathcal{S}$CIFF Agent instead are considered as expectations about the future: they might be confirmed by the arrival of the messages, as well as disconfirmed because no message arrives. If the latter is the case, depending on how the protocol has been defined, the $\mathcal{S}$CIFF Agent could perform some reasoning in order to recover from such situation.

Formally, an agent specification is defined as follows:

Definition 6.2.1 (Agent specification). *An* Agent Specification $\mathcal{P}_{ag}$ *is an Abductive Logic Program*

$$\mathcal{P}_{ag} \equiv \langle KB_{ag}, \mathcal{E}_{ag}, \mathcal{IC}_{ag} \rangle$$

where:

- KB_{ag} *is the Knowledge Base of the agent,*
- $\mathcal{E}_{ag}$ *is the set of* abducible predicates*, and*
- $\mathcal{IC}_{ag}$ *is the set of Integrity Constraints of the agent.*

KB_{ag} and $\mathcal{IC}_{ag}$ are the protocol specification, given in terms of an abductive specification $\mathcal{S} = \langle KB_{ag}, \mathcal{IC}_{ag} \rangle$ (see Section 2.3).

$\mathcal{E}_{ag}$ is the set of abducible predicates: it contains both expectations (positive and negative) and normal abducibles. Positive expectations can be divided into two significant subsets:

- expectations about messages where ag is the sender (of the form $\mathbf{E}(m_x(ag, A, Content))$); these expectation are interpreted as actions that agent ag intends to do;

- expectations about messages uttered by other participants to ag (of the form $ought(m_x(A, ws, Content))$, with $A \neq ag$), which can be intended as the messages that ag is able to understand.

In Specification 6.2.1 it is shown the *query-ref* protocol [75, 77]. Intuitively, the first IC means that if agent A sends to agent B a *query_ref* message, then B is expected to reply with either an *inform* or a *refuse* message by TD time units later, where TD is defined in the Social Knowledge Base by the *qt_deadline* predicate (in the shown specification the value of TD would be 10). The second IC means that, if an agent sends an *inform* message, then it is expected not to send a *refuse* message at any time.

Note that the specification is still from the global viewpoint of both the participants. In order to make it specific for the single agent ag, three steps must be applied to the generic specification:

- the role(s) that agent ag should play must be clearly specified;

- possibly non-determinisms due to several allowed answers (for a given role) should be solved;

- possibly non-determinisms due to non-specified content of the answers should be solved.

Specification 6.2.1 Specification of the *query_ref* interaction protocol.

ICs :

$$\begin{aligned}
&\mathbf{H}(m_x(A, B, \mathit{query_ref}(\mathit{Info})), T) \ \wedge \\
&\mathit{qr_deadline}(\mathit{TD}) \\
\rightarrow\ &\mathbf{E}(m_x(B, A, \mathit{inform}(\mathit{Info}, \mathit{Answer})), \mathit{T1}) \ \wedge \\
&\mathit{T1} \ < \ T \ + \ \mathit{TD} \\
\vee\ &\mathbf{E}(m_x(B, A, \mathit{refuse}(\mathit{Info})), \mathit{T1}) \ \wedge \\
&\mathit{T1} \ < \ T \ + \ \mathit{TD}
\end{aligned}$$

$$\begin{aligned}
&\mathbf{H}(m_x(A, B, \mathit{inform}(\mathit{Info}, \mathit{Answer})), \mathit{Ti}) \\
\rightarrow\ &\mathbf{EN}(m_x(A, B, \mathit{refuse}(\mathit{Info})), \mathit{Tr})
\end{aligned}$$

$SOKB$:

$$\mathit{qr\_deadline}(10).$$

6.2.2 Role Specification

Usually, a protocol depicts the interaction rules for several players, each one playing a role within the protocol. Hence a protocol specification describes the rules for different roles, from a single and global (social) viewpoint. In order to use such a protocol specification as the agent specification, it is necessary to specify which role(s) the agent should play.

Specifying a role within a protocol specification consists, in our view, to specify in which actions the role is involved, i.e. in which actions the agent (playing that role) is the sender or the receiver of a communicative act.

Example 6.2.1 In Specification 6.2.2 it is shown how the protocol described in the Spec. 6.2.1 can conceptually instantiated for a specific role. Whenever the role is taking part to a certain protocol step (i.e. it appears to be the sender or the receiver

of a message), the keyword *me* has been inserted instead of a generic variable. In this example, the selected role is the one of the agent that provides the information.

Specification 6.2.2 *query_ref* interaction protocol, specified for a particular role.

$$
\begin{array}{l}
ICs_{ag}: \\
\quad \mathbf{H}(m_x(A, me, query\_ref(Info)), T) \ \wedge \\
\quad qr\_deadline(TD) \\
\rightarrow \ \mathbf{E}(m_x(me, A, inform(Info, Answer)), T1) \ \wedge \\
\quad T1 \ < \ T \ + \ TD \\
\vee \ \mathbf{E}(m_x(me, A, refuse(Info)), T1) \ \wedge \\
\quad T1 \ < \ T \ + \ TD \\
\\
\quad \mathbf{H}(m_x(me, B, inform(Info, Answer)), Ti) \\
\rightarrow \ \mathbf{EN}(m_x(me, B, refuse(Info)), Tr) \\
\\
SOKB_{ag}: \\
\quad qr\_deadline(10).
\end{array}
$$

Note that the process of specifying the role for an agent *ag* by substituting the variables of sender/receiver with a specific keyword (e.g., in the Specification 6.2.2 has been used the keyword *me*) is not practical, since it could require a human supervision.

To provide a simpler method for specifying the role that agent *ag* should play, in the $\mathcal{S}$CIFF Agent a predicate `my_name/1` has been reserved for this purpose. The idea is to exploit a characteristics of the JADE platform: each agent is given a name that, at runtime, is unique w.r.t. the currently running platform. Each sent/received message brings as sender/receiver identifier this unique name. As a consequence, each happened event has a sender and a receiver specified by ground terms.

Given the happened events, the $\mathcal{S}$CIFF Proof Procedure elaborates all the possible abductive explanations (the sets Δ_i). In particular, since we are using the protocol specification, each set will contain the expectations about the behaviour of all the agents involved in the interactions. The following situations could happen:

1. some expectations are about the agent to send to some other agent a certain message (i.e., the sender is set to be the agent name, as given by the JADE platform);

2. some expectations are about the agent to receive a message (i.e., the receiver is set to be the agent name);

3. some expectations are about other agents, specified by their name, or not specified at all (i.e., the sender or the receiver are still an unbounded variable).

To understand which are the expectations regarding the role that agent ag should play, it is sufficient to specify in the $SOKB_{ag}$ the agent name (by means of the predicate `my_name/1`. In this way it is possible to automatically filter and select all the expectations about ag sending certain messages.

This very practical solution permits to automatically select all the expectations regarding ag, but introduces a problem: in case of protocols with many roles, a malicious agent could involve ag to play a different role. This can be avoided by explicitly stating (through the definition of the goal in the $SOKB_{ag}$) which are the negative expectations: in such a way, the $\mathcal{S}$CIFF Proof Procedure will compute, for ag, only the expectations that ag can effectively fulfill.

Example 6.2.2 In Specification 6.2.3 it is shown how the $SOKB$ should be extended in order to specify the role that an agent ag should play, in the *query_ref* protocol.

Specification 6.2.3 *query_ref* interaction protocol, specified for a particular role.

ICs_{ag} :

$$\begin{aligned}
&\mathbf{H}(m_x(A, B, \mathit{query_ref}(\mathit{Info})), T) \ \wedge \\
&\mathit{qr_deadline}(\mathit{TD}) \\
\rightarrow\ &\mathbf{E}(m_x(B, A, \mathit{inform}(\mathit{Info}, \mathit{Answer})), \mathit{T1}) \ \wedge \\
&\mathit{T1} \ < \ T \ + \ \mathit{TD} \\
\vee\ &\mathbf{E}(m_x(B, A, \mathit{refuse}(\mathit{Info})), \mathit{T1}) \ \wedge \\
&\mathit{T1} \ < \ T \ + \ \mathit{TD}
\end{aligned}$$

$$\begin{aligned}
&\mathbf{H}(m_x(A, B, \mathit{inform}(\mathit{Info}, \mathit{Answer})), \mathit{Ti}) \\
\rightarrow\ &\mathbf{EN}(m_x(A, B, \mathit{refuse}(\mathit{Info})), \mathit{Tr})
\end{aligned}$$

$SOKB_{ag}$:

society_goal:-
    $\mathbf{EN}(m_x(A, B, \mathit{query\_ref}(\mathit{Info})), T)$.
qr_deadline(*10*).
my_name(*me*).

Note that the ICshas remained the same w.r.t. the Specification 6.2.1, and only the $SOKB$ has been extended to specify the role. In particular, it is assumed that the agent ag will have the name "*me*", and that it will not play the role of the enquirer (ag can never send a *query_ref* message).

6.2.3 Protocol Non-Determinism

Interaction protocols quite often allows for a peer to behave in several different alternative ways: e.g., in the TCP opening phase, the requested peer can answer to an initial *syn* message with a *syn/ack* message or rather with a *syn* message. Another simple example is given in the *query-ref* protocol: the queried agent can answer with *inform* or with *refuse*.

When $\mathcal{S}$CIFF Proof Procedure is applied to such protocols, it happen that different abductive answers Δ_i are generated, each one corresponding to the alternatives enlisted by the protocol. In order to execute the protocol specification, the agent programmer should provide a way for selecting one amongst the possible behaviours.

Example 6.2.3 (Protocol Non-Determinism). In the Specification 6.2.1, after a *query* message is received, an agent can answer with the *inform* or with the *refuse* message. Given the following happened events:

$$\mathbf{HAP} = \{\mathbf{H}(m_x(anAgent, me, query\_ref(trainTable(tr1234))), 5)\}$$

the abductive explanations computed by the $\mathcal{S}$CIFF Proof Procedure are the following:

$$\Delta_1 = \{\mathbf{E}(m_x(me, anAgent, inform(trainTable(tr1234), 12{:}25), T1)) \ \wedge \ T1 \ < \ 15\}$$
$$\Delta_2 = \{\mathbf{E}(m_x(me, anAgent, refuse(trainTable(tr1234))), T1) \ \wedge \ T1 \ < \ 15\}$$

In order to decide which answer should be given, a predicate symbol `select_behaviour/2` has been reserved. This predicate, that must be defined by the agent developer, receives as first parameter the list of the possible behaviours, and provide as output (in the second parameter) the selected behaviours (i.e., the list of expectations that agent will try to fulfill).

6.2.4 Messages non-Determinism

In the majority of the cases, protocols rule the type of the messages that can be sent by a certain peer, but do not specify other parameters related to the message itself. E.g., the *query-ref* protocol shown in Specification 6.2.1 does not rule what the content of the *Info* parameter should be.

In order to successfully execute the protocol specification, the agent developer should provide (by means of the $SOKB$) a way for specifying the message content. This can be done by defining the predicate `message_grounder/2` that receives as input the selected expectation containing variables, and provide as output the same expectation with all the variables substituted by ground terms.

6.3 Conformance property of the SCIFF Agent

In the research literature it is possible to find several definitions of conformance. Here we will restrict our considerations to the definitions given by Endriss et al. [68, 70], and to the definitions we provide in Section 5.2. Both the works use the same term *conformance*: however, in [68] the terms are *weak, exhaustively and robust conformance*, while in our approach (Chapter 5) we define *feeble* and *strong conformance*. The interested reader can refer to Section 5.6 for a comparison between the different definitions.

Weak, Exhaustively and Robust Conformances

The $\mathcal{S}$CIFF Agent uses the protocol specification for elaborating the allowed (and expected) messages it should send. Given that an agent is *weak conformance* to a protocol $\mathcal{P}$ *iff it never utters any illegal dialogue move*, we can state the following theorem:

Proposition 6.3.1 (Weak Conformance). *A $\mathcal{S}$CIFF Agent is always weak conformant.*

Proof. The predicates `my_name`, `select_behaviour` and `message_grounder` can select only messages that are allowed from the protocol specification.

□

Note that, while in [68] the weak conformance property is restricted to the *shallow protocols* [1] class, we do no need to restrict to such set.

[1] In [68] shallow protocols are defined as protocols that can be translated into if-then rules where a single happening event is present on the left side of each rule. Shallow protocols corresponds to deterministic finite automatas where it is possible to determine the next state of the dialogue on the sole basis of the previous event.

Unfortunately, there is no guarantee that any allowed answer will be ever utter. It might be the case that the functions (defined by the developer) for solving the non-determinism issues, does not select any allowed answer at all. However, if such hypothesis is assumed, $\mathcal{S}$CIFF Agent also enjoy the *Exhaustive Conformance* property.

Proposition 6.3.2 (Exhaustive Conformance). *Given a $\mathcal{S}$CIFF Agent, if:*

1. *for every interaction step the set of possible allowed answers Δ is not empty, and*
2. *for each possible set Δ of allowed answers, the functions* `my_name`*,* `select_behaviour` *and* `message_grounder` *always select at least one answer*

Then the $\mathcal{S}$CIFF Agent is Exhaustive Conformance*.*

Proof. Proposition 6.3.1 already states the weakly conformance. Condition 1 and 2 guarantee that at least one allowed answer will be chosen amongst all the possible answers envisaged by the protocol. Hence the agent will always utter a message. □

We end the considerations about the conformance by noting that, for what regards the *Robust conformance*, the $\mathcal{S}$CIFF Agent can not guarantee that property, unless it is the protocol specification itself that defines the behaviour of answering a default message (such as `not-understood` in response to any received and not allowed message.

Feeble and Strong Conformances

The conformance as defined in the A^{l}LoWS framework can be simply proved by applying the framework directly on the $\mathcal{S}$CIFF Agent specification, considering the

part of the protocol (related to the particular role) as the behavioural interface of the agent.

However, we would like to notice that this operation is not necessary, since it means to check a protocol specification against a subset of the specification itself. As for the previous section, the only problem arises in the selection functions, that do not guarantee that any message will be ever selected to be sent out.

Proposition 6.3.3 (Strong Conformance). *Given a $\mathcal{S}$CIFF Agent, if:*

1. *for every interaction step the set of possible allowed answers Δ is not empty, and*
2. *for each possible set Δ of allowed answers, the functions* `my_name`, `select_behaviour` *and* `message_grounder` *always select at least one answer*

Then the $\mathcal{S}$CIFF Agent is Strong Conformance.

Proof. Naively, by construction of the $\mathcal{S}$CIFF Agent. In fact every possible answer that the agent will utter, is generated by a subset of the rules defining the global protocol, and therefore such messages are always allowed. □

6.4 SCIFF Agent Implementation

The $\mathcal{S}$CIFF agent has been modeled on the basis of the Kowalsky-Sadri cycle for intelligent agents [102]:

1. Observe
2. Think
3. Act

In particular, the phases of *observe* and *act* have been implemented directly in Java, by extending a JADE agent. The *think* phase instead has been realized using the $\mathcal{S}$CIFF proof procedure, that provides instructions on the basis of the happened events.

A schematic representation of the blocks composing the $\mathcal{S}$CIFF agent is shown in Figure 6.4.

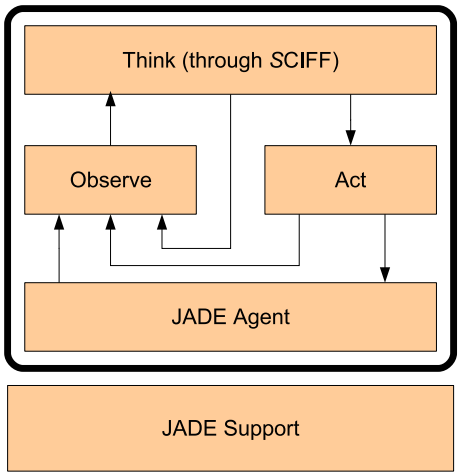


Figure 6.2: Schematic diagram of the $\mathcal{S}$CIFF Agent

More in detail, the observation step consists on analyzing all the events that happened since the observation step of the previous cycle. All the events are registered in an *Event Buffer*, a repository that keeps trace of the new events. Three types of events are considered:

i) Events corresponding to received messages.

ii) Events corresponding to sent messages.

iii) Events corresponding to internal state changes.

The "think" step consists of using the $\mathcal{S}$CIFF proof procedure to elaborate the happened events, and to generate a set of alternative expected behaviours. By applying the selection functions defined by the developer (predicates `my_name/1`, `select_behaviour/2` and `message_grounder/2`), only one behaviour is selected, and a grounding of the messages that should be sent is passed on to the execution block.

Finally, the execution step consists on interpreting the expected behaviour generated by the $\mathcal{S}$CIFF proof procedure. The expectations about the behaviour of other agents will not be considered (it will be a task of the $\mathcal{S}$CIFF procedure to understand if such expectations have been satisfied or not, possibly providing further behaviours). Instead, expectations that regards actions to be done by the agent itself, will be interpreted as orders to be executed. If the expectation, e.g., is about sending a certain message, then the execution block will send such message. Then, for each action executed, the execution block generates a corresponding event and updates the buffer of the happened events: in this way it is possible to use the $\mathcal{S}$CIFF procedure for reasoning also about the agent' act, beside the other agent acts.

6.5 Related Works

The main objective of the $\mathcal{S}$CIFF Agent is not to provide a new agent architecture, but rather to show how a protocol specification based on logic programming can be directly used to program the agent behaviour. This would introduce many advantages: in particular it would simplify the developer job of programming the agents and, as seen in Section 6.3, for particular protocol classes it would also entail a conformance property.

The protocol specification however is not enough: as discussed in Section 6.2.1, some extra information is needed. In the literature, this extra information is often referred as the *private policy* of the agent, in contrast to the public policy, ruled by the protocol.

Many agent frameworks have been proposed in the MAS research field. However, to the best of our knowledge, only Endriss et al. [68, 70] addressed the issue of using a protocol specification (given by means of computational logic) to directly specify the agent behaviour. Our work has been mainly inspired by their works: our approach and their proposal share many points: e.g., both the approaches define the protocol by means of forward rules, where the antecedent are given in terms of happened events, and the consequences in terms of expectations about the behaviours. However, our formalism is definitely more expressive and powerful, while they restrict their formalism and focus their analysis to simpler class of *shallow protocols*: such class in fact does not allow for interactions between many peers (more than 2) at the same time and, moreover, it does not allow for concurrent dialogues.

Chapter 7

Conclusions and Future Works

7.1 Summary

In this thesis we have presented the $\mathcal{S}$CIFF framework and its extensions (the g-$\mathcal{S}$CIFF the A^{l}LoWS Frameworks and the $\mathcal{S}$CIFF Agent Platform). The $\mathcal{S}$CIFF Framework allows to specify interaction protocols, by means of a formal language supported by a clear declarative semantics. Then, through the $\mathcal{S}$CIFF Proof Procedure it is possible to reason about such specifications: soundness, termination and completeness properties of this procedure have been demonstrated. The *SOCS-SI* exploits the proof procedure to test the compliance verification of peers interactions against the protocols.

The g-$\mathcal{S}$CIFF Framework extend the $\mathcal{S}$CIFF approach, and allows to statically prove protocol properties: differently by the many existing approaches, it is not focussed on a specific application domain, but it rather permits to specify any generic property. The A^{l}LoWS Framework instead permits to verify a-priori if a component, whose behaviour is described by a public behavioural interface, is complaint with a protocol specification. Finally we have discussed the $\mathcal{S}$CIFF Agent Platform, that ease the task of developing agents by directly using the protocol specification itself

as a part of the agent.

A very important advantage of our approach is that several different issues (from specification to verification and execution) are addressed within a single framework: the immediate advantage is that the same protocol specification can be used to perform the various tasks, without the need of translating it in various formalisms. Other advantages of our approach are given by the declarative flavor of the specification language, that makes it suitable to be used by human operators; at the same time, its rigorous formal semantics makes it suitable also for automated reasoning.

The language has been shown to be highly expressive, thanks also to the possibility of expressing CLP constraints over the variables; moreover the explicitly treatment of the time makes the framework suitable to perform also time-related reasoning tasks (such as deadline verification, for example, or planning). Furthermore, the presence of positive and negative expectation allows to easy represent open as well as closed interaction models.

However such expressive power, and the possibility of applying our approach to almost any application domain, have a price in terms of performances: for example, the time required to perform the properties verification task is some orders higher than the time required by any model checking approach. Anyway, it's our opinion that this is a reasonable price to be paid for such a powerful framework.

7.2 Future Works

It is our opinion that, starting from the work presented in this thesis, there are many possible research directions.

For what concern the $\mathcal{S}$CIFF Framework and the other derived frameworks, an

interesting extension could be to consider the *Run-time Responsibility Identification*, and the strictly related *Culprit Identification*. One actual limit of the framework is that, although a wrong (w.r.t. a protocol) behaviour can be detected by the presented tools, it might be difficult to identify which peer has been responsible of the violation.

Another interesting extension could regard the types of violation detected: actually, if a wrong behaviour is detected, a violation is raised. It could be worthy to investigate if different types of violation could be defined and detected. For example, in certain situation do not performing an expected action could be viewed as a "lighter" violation, w.r.t. performing a prohibited action. Moreover, it could be very interesting to allow recovery mechanisms in case of violations. Also, preferences between different expectations could be desirable.

Then, from the protocol viewpoint, it could be very interesting to investigate the protocols resulting by the composition of different protocol specifications. The method for composing such specification is already a research topic that, we believe, it is worth to be investigated.

Finally, considering the applications domain, we believe that our approach can be extended in many other application fields. For example, in the specification of business rules and business process, as well as in the workflow management systems.

Published papers

2004

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

ii Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni, *A logic based approach to interaction design in open multi-agent systems*, 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2004) (Washington, DC, USA) (Martin Fredriksson, Rune Gustavsson, Alessandro Ricci, and Andrea Omicini, eds.), IEEE Computer Society, September 2004, pp. 387–392.

2005

iii Marco Alberti and Federico Chesani, *The computational behaviour of the sciff abductive proof procedure and the socs-si system*, Intelligenza Artificiale (2005), no. Anno II No. 3, 45–51.

iv Marco Alberti, Federico Chesani, Marco Gavanelli, Alessio Guerri, Evelina Lamma, Paola Mello, and Paolo Torroni, *Expressing interaction in combinatorial auction through social integrity constraints*, Intelligenza Artificiale **II** (2005), no. 1, 22–29.

v Marco Alberti, Federico Chesani, Marco Gavanelli, Alessio Guerri, Evelina Lamma, Michela Milano, and Paolo Torroni, *Expressing interaction in combinatorial auction through social integrity constraints*, in Wolf et al. [152], pp. 53–64.

vi Marco Alberti, Federico Chesani, Marco Gavanelli, and Evelina Lamma, *The CHR-based Implementation of a System for Generation and Confirmation of Hypotheses*, in Wolf et al. [152], pp. 111–122.

vii Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni, *Security protocols verification in Abductive Logic Programming: A case study*, CILC 2005 - Convegno Italiano di Logica Computazionale (Alberto Pettorossi, Maurizio Proietti, and Valerio Senni, eds.), Università degli Studi di Roma Tor Vergata, June 21-22 2005.

viii Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni, *The SOCS computational logic approach for the specification and verification of agent societies*, Global Computing: IST/FET International Workshop, GC 2004 Rovereto, Italy, March 9-12, 2004 Revised Selected Papers (Corrado Priami and Paola Quaglia, eds.), Lecture Notes in Artificial Intelligence, vol. 3267, Springer-Verlag, 2005, pp. 324–339.

ix A. Ciampolini P. Mello M. Montali S. Storari M. Alberti, F. Chesani and P. Torroni, *Protocol specification and verification by using computational logic*, In Proceedings of Workshop dagli Oggetti agli Agenti (WOA'05), November 2005.

2006

x M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, *Compliance verification of agent interaction: a logic-based software tool*, Applied Artificial Intelligence **20** (2006), no. 2-4, 133–157.

xi Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, and Paola Mello, *A verifiable logic-based agent architecture*, Foundations of Intelligent Systems - 16th International Symposium, ISMIS 2006 Bari, Italy, September 27-29, 2006 Proceedings (Berlin Heidelberg) (Floriana Esposito, Zbigniew W. Raś, Donato Malerba, and Giovanni Semeraro, eds.), Lecture Notes in Artificial Intelligence, vol. 4203, Springer-Verlag, 2006, pp. 188–197.

xii Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Marco Montali, *A-priori verification of web services with abduction*, Proceedings of CILC 2006 (Bari, Italy) (Floriana Esposito, Donato Malerba, and Giovanni Semeraro, eds.), June 2006.

xiii Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Marco Montali, *An abductive framework for a-priori verification of web*

services, Proceedings of the Eighth Symposium on Principles and Practice of Declarative Programming, July 10-12, 2006, Venice, Italy (New York, USA) (Michael Maher, ed.), Association for Computing Machinery (ACM), Special Interest Group on Programming Languages (SIGPLAN), ACM Press, July 2006, pp. 39–50.

xiv Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, Sergio Storari, and Paolo Torroni, *Computational logic for run-time verification of web services choreographies: exploiting the SOCS-SI tool*, Web Services and Formal Methods - Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006 Proceedings (Berlin/Heidelberg) (Mario Bravetti and Gianluigi Zavattaro, eds.), Lecture Notes in Computer Science, vol. 4184, Springer-Verlag, 2006, pp. 58–72.

xv Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, and Paolo Torroni, *Policy-based reasoning for smart web service interaction*, Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS2006) (Seattle, Washington, USA) (Axel Polleres, Stefan Decker, Gopal Gupta, and Jos de Bruijn, eds.), CEUR Workshop Proceedings, vol. 196, August 10-22 2006, pp. 87–102.

xvi Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni, *Security protocols verification in abductive logic programming: a case study*, Proceedings of 6th International Workshop "Engineering Societies in the Agents' World" (ESAW'05), October 26-28, 2005 (Berlin Heidelberg) (Oğuz Dikenelli, Marie-Pierre Gleizes, and Alessandro Ricci, eds.), Lecture Notes on Artificial Intelligence, vol. 3963, Department of Computer Engineering Ege University, Springer-Verlag, 2006, pp. 106–124.

xvii Federico Chesani, Anna Ciampolini, Paola Mello, Marco Montali, and Sergio Storari, *Testing guidelines conformance by translating a graphical language to computational logic*, Workshop AI techniques in healthcare: evidence based guidelines and protocols (Peter Lucas, Silvia Miksch, and Annette ten Teije, eds.), August 2006.

xviii Federico Chesani, Marco Gavanelli, Marco Alberti, Evelina Lamma, Paola Mello, and Paolo Torroni, *Specification and verification of agent interaction using abductive reasoning*, CLIMA VI (Berlin Heidelberg) (Francesca Toni and Paolo Torroni, eds.), Lecture Notes on Artificial Intelligence, vol. 3900, Springer-Verlag, 2006, p. 243264.

xix Federico Chesani, Pietro De Matteis, Paola Mello, Marco Montali, and Sergio Storari, *A framework for defining and verifying clinical guidelines: A case*

study on cancer screening., ISMIS (Floriana Esposito, Zbigniew W. Ras, Donato Malerba, and Giovanni Semeraro, eds.), Lecture Notes in Computer Science, vol. 4203, Springer, 2006, pp. 338–343.

xx Federico Chesani, Paola Mello, Marco Montali, Marco Alberti, Marco Gavanelli, Evelina Lamma, and Sergio Storari, *Abduction for specifying and verifying web service choreographies*, 4th International Workshop on AI for Service Composition (Jose Luis Ambite, Jim Blythe, Jana Koehler, Sheila McIlraith, Marco Pistore, and Biplav Srivastava, eds.), August 2006, pp. 15–20.

2007

xxi Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni, *Verifiable agent interaction in abductive logic programming: the SCIFF framework*, ACM Transactions on Computational Logics, TO APPEAR.

xxii Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, and Paolo Torroni, *Contracting for dynamic location of web services: specification and reasoning with SCIFF*, 4th European Semantic Web Conference (Berlin/Heidelberg) (Enrico Franconi, ed.), Lecture Notes in Computer Science, European Semantic Systems Initiative (ESSI), Springer-Verlag, 2007, TO APPEAR.

Bibliography

[1] Martín Abadi and Bruno Blanchet, *Analyzing security protocols with secrecy types and logic programs*, J. ACM **52** (2005), no. 1, 102–146.

[2] Slim Abdennadher and Henning Christiansen, *An experimental CLP platform for integrity constraints and abduction*, FQAS, Flexible Query Answering Systems (H.L. Larsen, J. Kacprzyk, S. Zadrozny, T. Andreasen, and H. Christiansen, eds.), LNCS, Springer-Verlag, October 25–28 2000, pp. 141–152.

[3] *ACLP: Abductive Constraint Logic Programming*, Electronically available at `http://www.cs.ucy.ac.cy/aclp/`.

[4] A. Aggoun, D. Chan, P. Dufresne, E. Falvey, H. Grant, W. Harvey, A. Herold, G. Macartney, M. Meier, D. Miller, S. Mudambi, S. Novello, B. Perez, E. van Rossum, J. Schimpf, K. Shen, P. A. Tsahageas, and D. H. de Villeneuve, *ECL^{i}PS^e user manual, release 5.2*, IC-Parc, Imperial College, London, UK, 2001.

[5] Luigia Carlucci Aiello and Fabio Massacci, *Planning attacks to security protocols: Case studies in logic programming*, Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I (Antonis C. Kakas and Fariba Sadri, eds.), Lecture Notes in Computer Science, vol. 2407, Springer-Verlag, 2002, pp. 533–560.

[6] Marco Alberti, Federico Chesani, Marco Gavanelli, Alessio Guerri, Evelina Lamma, Paola Mello, and Paolo Torroni, *Expressing interaction in combinatorial auction through social integrity constraints*, Intelligenza Artificiale **II** (2005), no. 1, 22–29.

[7] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, Sergio Storari, and Paolo Torroni, *Computational logic for run-time verification of web services choreographies: exploiting the SOCS-SI tool*, Web Services and Formal Methods - Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006 Proceedings (Berlin/Heidelberg) (Mario Bravetti and Gianluigi Zavattaro, eds.), Lecture Notes in Computer Science, vol. 4184, Springer-Verlag, 2006, pp. 58–72.

[8] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni, *The SOCS computational logic approach for the specification and verification of agent societies*, Global Computing: IST/FET International Workshop, GC 2004 Rovereto, Italy, March 9-12, 2004 Revised Selected Papers (Corrado Priami and Paola Quaglia, eds.), Lecture Notes in Artificial Intelligence, vol. 3267, Springer-Verlag, 2005, pp. 324–339.

[9] ———, *Security protocols verification in abductive logic programming: a case study*, Proceedings of 6th International Workshop "Engineering Societies in the Agents' World" (ESAW'05), October 26-28, 2005 (Berlin Heidelberg) (Oğuz Dikenelli, Marie-Pierre Gleizes, and Alessandro Ricci, eds.), Lecture Notes on Artificial Intelligence, vol. 3963, Department of Computer Engineering Ege University, Springer-Verlag, 2006, pp. 106–124.

[10] Marco Alberti, Anna Ciampolini, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni, *A social ACL semantics by deontic constraints*, Multi-Agent Systems and Applications III. Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003 (Prague, Czech Republic) (V. Mařík, J. Müller, and M. Pěchouček, eds.), Lecture Notes in Artificial Intelligence, vol. 2691, Springer-Verlag, June 16–18 2003, pp. 204–213.

[11] Marco Alberti, D. Daolio, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni, *Specification and verification of agent interaction protocols in a logic-based system*, Proceedings of the 19th Annual ACM Symposium on Applied Computing (SAC 2004). Special Track on Agents, Interactions, Mobility,

and Systems (AIMS) (Nicosia, Cyprus) (Hisham M. Haddad, Andrea Omicini, and Roger L. Wainwright, eds.), ACM Press, March 14–17 2004, pp. 72–78.

[12] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, Giovanni Sartor, and Paolo Torroni, *Mapping deontic operators to abductive expectations*, Computational and Mathematical Organization Theory **12** (2006), no. 2–3, 205–225.

[13] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni, *An Abductive Interpretation for Open Societies*, AI*IA 2003: Advances in Artificial Intelligence, Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence, Pisa (A. Cappelli and F. Turini, eds.), Lecture Notes in Artificial Intelligence, vol. 2829, Springer-Verlag, September 23–26 2003, pp. 287–299.

[14] ———, *Specification and verification of interaction protocols: a computational logic approach based on abduction*, Technical Report CS-2003-03, Dipartimento di Ingegneria di Ferrara, Ferrara, Italy, 2003, Available at `http://www.ing.unife.it/informatica/tr/`.

[15] ———, *The $\mathcal{SCIFF}$ abductive proof-procedure*, Proceedings of the 9th National Congress on Artificial Intelligence, AI*IA 2005, Lecture Notes in Artificial Intelligence, vol. 3673, Springer-Verlag, 2005, pp. 135–147.

[16] J. Alferes, L. M. Pereira, and T. Swift, *Abduction in well-founded semantics and generalized stable models via tabled dual programs*, Theory and Practice of Logic Programming **4** (2004), 383–428.

[17] J. J. Alferes, A. Brogi, J. A. Leite, and L. M. Pereira, *Evolving logic programs*, in Flesca et al. [78], pp. 50–61.

[18] José Júlio Alferes and João Alexandre Leite (eds.), *Jelia*, Lecture Notes in Artificial Intelligence, vol. 3229, Springer-Verlag, 2004.

[19] A. Anderson, *A reduction of deontic logic to alethic modal logic*, Mind **67** (1958), 100–103.

[20] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, *Business process execution language for web services version 1.1*, 2003, Available at http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[21] Krzysztof R. Apt and Roland N. Bol, *Logic programming and negation: A survey*, Journal of Logic Programming **19/20** (1994), 9–71.

[22] K. A. Arisha, F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus, *IMPACT: a Platform for Collaborating Agents*, IEEE Intelligent Systems **14** (1999), no. 2, 64–72.

[23] Alessandro Armando, Luca Compagna, and Yuliya Lierler, *Automatic compilation of protocol insecurity problems into logic programming*, in Alferes and Leite [18], pp. 617–627.

[24] A. Artikis, J. Pitt, and M. Sergot, *Animated specifications of computational societies*, in Castelfranchi and Lewis Johnson [42], pp. 1053–1061.

[25] Michael Backes and Birgit Pfitzmann, *A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol*, FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings (Paritosh K. Pandya and Jaikumar Radhakrishnan, eds.), Lecture Notes in Computer Science, vol. 2914, Springer-Verlag, 2003, pp. 1–12.

[26] Matteo Baldoni, Cristina Baroglio, Alberto Martelli, and Viviana Patti, *A priori conformance verification for guaranteeing interoperability in open environments.*, ICSOC (Asit Dan and Winfried Lamersdorf, eds.), Lecture Notes in Computer Science, vol. 4294, Springer, 2006, pp. 339–351.

[27] Matteo Baldoni, Cristina Baroglio, Alberto Martelli, Viviana Patti, and Claudio Schifanella, *Verifying the conformance of web services to global interaction protocols: A first step*, EPEW/WS-FM (Mario Bravetti, Leïla Kloul, and Gianluigi Zavattaro, eds.), Lecture Notes in Computer Science, vol. 3670, Springer, 2005.

[28] A. Barros, M. Dumas, and P. Oaks, *A critical overview of the web services choreography description language (WS-CDL)*, BPTrends (2005).

[29] R. Barruffi, M. Milano, and R. Montanari, *Planning for security management.*, IEEE Intelligent Systems **16** (2001), no. 1, 74–80.

[30] David A. Basin, Sebastian Mödersheim, and Luca Viganò, *An on-the-fly model-checker for security protocol analysis.*, ESORICS (Einar Snekkenes and Dieter Gollmann, eds.), Lecture Notes in Computer Science, vol. 2808, Springer, 2003, pp. 253–270.

[31] Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi, *Jade - a java agent development framework*, Multi-Agent Programming: Languages, Platforms and Applications (Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, eds.), Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15, Springer-Verlag, 2005, pp. 125–147.

[32] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa, *Developing multi-agent systems with a FIPA-compliant agent framework*, Software - Practice and Experience **31** (2001), no. 2, 103–128.

[33] B. Benattallah, F. Casati, F. Toumani, and R. Hamadi, *Conceptual modeling of web service conversations*, **2681** (2003), 449–467.

[34] Bruno Blanchet, *From secrecy to authenticity in security protocols*, SAS '02: Proceedings of the 9th International Symposium on Static Analysis (London, UK), Springer-Verlag, 2002, pp. 342–359.

[35] ———, *Automatic verification of cryptographic protocols: a logic programming approach*, PPDP '03: Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming (New York, NY, USA), ACM Press, 2003, pp. 1–3.

[36] F. Bosi and M. Milano, *Enhancing CLP branch and bound techniques for scheduling problems*, Software Practice & Experience **31** (2001), no. 1, 17–42.

[37] Andrea Bracciali, Ulle Endriss, Neophytos Demetriou, Antonis C. Kakas, Wenjin Lu, and Kostas Stathis, *Crafting the mind of prosocs agents*, Applied Artificial Intelligence **20** (2006), no. 2-4, 105–131.

[38] D.T. Brock, M.D. Madigan, J.M.Martinko, and J. Parker, *Microbiologia*, Prentice-Hall International, Milano, 1995.

[39] F. Bry, M. Eckert, and P. Patranjan, *Reactivity on the web: Paradigms and applications of the language xchange*, Journal of Web Engineering **5** (2006), no. 1, 3–24.

[40] G. Caire, M. Cossentino, A. Negri, A. Poggi, and P. Turci, *Multi-agent systems implementation and testing*, Cybernetics and Systems 2004 - Volume II (Vienna, Austria) (Robert Trappl, ed.), Austrian Society for Cybernetics Studies, April 13 - 16 2004, pp. 612–617.

[41] C. Castelfranchi, *Commitments: From individual intentions to groups and organizations*, Proceedings of the First International Conference on Multiagent Systems, San Francisco, California, USA, AAAI Press, 1995, pp. 41–48.

[42] C. Castelfranchi and W. Lewis Johnson (eds.), *Proceedings of the first international joint conference on autonomous agents and multiagent systems (AAMAS-2002)*, Bologna, Italy, ACM Press, July 15–19 2002.

[43] W. Chen and D. S. Warren, *Tabled evaluation with delaying for general logic programs*, Journal of the ACM **43** (1996), no. 1, 20–74.

[44] Amit K. Chopra and Munindar P. Singh, *Producing compliant interactions: Conformance, coverage, and interoperability.*, DALT (Matteo Baldoni and Ulle Endriss, eds.), Lecture Notes in Computer Science, vol. 4327, Springer, 2006, pp. 1–15.

[45] Henning Christiansen and Verónica Dahl, *HYPROLOG: A new logic programming language with assumptions and abduction.*, Logic Programming, 21st International Conference, ICLP 2005, Sitges, Spain, October 2-5, 2005, Proceedings (Maurizio Gabbrielli and Gopal Gupta, eds.), Lecture Notes in Computer Science, vol. 3668, Springer, 2005, pp. 159–173.

[46] Anna Ciampolini, Evelina Lamma, Paola Mello, Francesca Toni, and Paolo 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** (2003), no. 1-2, 65–91.

[47] Anna Ciampolini, Evelina Lamma, Paola Mello, and Paolo Torroni, *LAILA: A language for coordinating abductive reasoning among logic agents*, Computer Languages **27** (2002), no. 4, 137–161.

[48] K. L. Clark, *Negation as Failure*, Logic and Data Bases (H. Gallaire and J. Minker, eds.), Plenum Press, 1978, pp. 293–322.

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

[50] Marco Colombetti, Nicoletta Fornara, and Mario Verdicchio, *A social approach to communication in multiagent systems*, Declarative Agent Languages and Technologies (João Alexandre Leite, Andrea Omicini, Leon Sterling, and Paolo Torroni, eds.), Lecture Notes in Artificial Intelligence, vol. 2990, Springer-Verlag, May 2004, First International Workshop, DALT 2003. Melbourne, Australia, July 2003. Revised Selected and Invited Papers, pp. 191–220.

[51] L. Console, D. Theseider Dupré, and P. Torasso, *On the relationship between abduction and deduction*, Journal of Logic and Computation **1** (1991), no. 5, 661–690.

[52] Ricardo Corin and Sandro Etalle, *An improved constraint-based system for the verification of security protocols*, Static Analysis, 9th International Symposium, SAS 2002, Madrid, Spain, September 17-20, 2002, Proceedings (Berlin, Germany) (Manuel V. Hermenegildo and German Puebla, eds.), Lecture Notes in Computer Science, vol. 2477, Springer, 2002, pp. 326–341.

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

[54] P. Davidsson, *Categories of artificial societies*, Engineering Societies in the Agents World II (A. Omicini, P. Petta, and R. Tolksdorf, eds.), Lecture Notes in Artificial Intelligence, vol. 2203, Springer-Verlag, December 2001, 2nd International Workshop (ESAW'01), Prague, Czech Republic, July 7, 2001, Revised Papers, pp. 1–9.

[55] Giorgio Delzanno, *Specifying and debugging security protocols via hereditary Harrop formulas and λProlog - a case-study -*, Functional and Logic Programming, 5th International Symposium, FLOPS 2001, Tokyo, Japan, March 7-9, 2001, Proceedings (Herbert Kuchen and Kazunori Ueda, eds.), Lecture Notes in Computer Science, vol. 2024, Springer-Verlag, 2001, pp. 123–137.

[56] Giorgio Delzanno and Sandro Etalle, *Proof theory, transformations, and logic programming for debugging security protocols*, Logic Based Program Synthesis and Transformation : 11th International Workshop, (LOPSTR 2001). Selected papers. (Paphos, Cyprus) (A. Pettorossi, ed.), Lecture Notes in Computer Science, vol. 2372, Springer Verlag, November 2001, pp. 76–90.

[57] M. Denecker and D. De Schreye, *SLDNFA: An abductive procedure for normal abductive programs*, Proceedings of the Joint International Conference and Symposium on Logic Programming, Washington, USA (Cambridge, MA) (Krzysztof R. Apt, ed.), MIT Press, November 9–13 1992, pp. 686–702.

[58] ———, *Representing Incomplete Knowledge in Abductive Logic Programming*, Logic Programming, Proceedings of the 1993 International Symposium, Vancouver, British Columbia, Canada (Cambridge, MA), MIT Press, 1993, pp. 147–163.

[59] M. Denecker and D. De Schreye, *SLDNFA: an abductive procedure for abductive logic programs*, Journal of Logic Programming **34** (1998), no. 2, 111–167.

[60] V. Dignum, J. J. Meyer, F. Dignum, and H. Weigand, *Formal specification of interaction in agent societies*, Proceedings of the Second Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS), Maryland, October 2002.

[61] V. Dignum, J. J. Meyer, and H. Weigand, *Towards an organizational model for agent societies using contracts*, in Castelfranchi and Lewis Johnson [42], pp. 694–695.

[62] V. Dignum, J. J. Meyer, H. Weigand, and F. Dignum, *An organizational-oriented model for agent societies*, Proceedings of International Workshop on Regulated Agent-Based Social Systems: Theories and Applications. AAMAS'02, Bologna, 2002.

[63] R. Dijkman and M. Dumas, *Service-oriented design: A multi-viewpoint approach*, International Journal of Cooperative Information Systems **13(4)** (2004), 337–378.

[64] M. Dincbas, P. van Hentenryck, H. Simonis, and A. Aggoun, *The constraint logic programming language chip*, Proceedings of the 2nd International Conference on Fifth Generation Computer Systems (Tokyo, Japan), December 1988, pp. 693–702.

[65] Claire Dixon, Mari-Carmen Fernández Gago, Michael Fisher, and Wiebe van der Hoek, *Using temporal logics of knowledge in the formal verification of security protocols*, Procedings of the Eleventh International Workshop on Temporal Representation and Reasoning (TIME'04), 2004.

[66] ———, *Using temporal logics of knowledge in the formal verification of security protocols*, Technical Report ULCS-03-022, University of Liverpool, Department of Computer Science, Liverpool, UK, 2004, `http://www.csc.liv.ac.uk/research/techreports/`.

[67] T. Eiter, V.S. Subrahmanian, and G. Pick, *Heterogeneous active agents, I: Semantics*, Artificial Intelligence **108** (1999), no. 1-2, 179–255.

[68] U. Endriss, N. Maudet, Fariba Sadri, and Francesca Toni, *Protocol conformance for logic-based agents*, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico (IJCAI-03) (G. Gottlob and T. Walsh, eds.), Morgan Kaufmann Publishers, August 2003.

[69] Ulle Endriss, Paolo Mancarella, Fariba Sadri, Giacomo Terreni, and Francesca Toni, *The CIFF proof procedure for abductive logic programming with constraints*, in Alferes and Leite [18], pp. 31–43.

[70] Ulle Endriss, Nicolas Maudet, Fariba Sadri, and Francesca Toni, *Logic-based agent communication protocols*, Advances in Agent Communication (F. Dignum, ed.), LNAI, vol. 2922, Springer-Verlag, 2004, pp. 91–107.

[71] K. Eshghi, *Abductive planning with the event calculus*, Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington (Cambridge, MA), MIT Press, 1988.

[72] K. Eshghi and R. A. Kowalski, *Abduction compared with negation by failure*, Proceedings of the 6th International Conference on Logic Programming (Cambridge, MA) (G. Levi and M. Martelli, eds.), MIT Press, 1989, pp. 234–255.

[73] M. Esteva, D. de la Cruz, and C. Sierra, *ISLANDER: an electronic institutions editor*, Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III (Bologna, Italy) (C. Castelfranchi and W. Lewis Johnson, eds.), ACM Press, July 15–19 2002, pp. 1045–1052.

[74] C.A. Evans and A.C. Kakas, *Hypotheticodeductive reasoning*, Proc. International Conference on Fifth Generation Computer Systems (Tokyo), 1992, pp. 546–554.

[75] *FIPA: Foundation for Intelligent Physical Agents*, Home Page: `http://www.fipa.org/`.

[76] *FIPA Communicative Act Library Specification*, August 2001, Published on August 10th, 2001, available for download from the FIPA website, `http://www.fipa.org`.

[77] *FIPA Query Interaction Protocol*, August 2001, Published on August 10th, 2001, available for download from the FIPA website, `http://www.fipa.org`.

[78] Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni (eds.), *Proceedings of the 8th european conference on logics in artificial intelligence (jelia)*, Lecture Notes in Computer Science, vol. 2424, Springer-Verlag, September 2002.

[79] N. Fornara and M. Colombetti, *Operational specification of a commitment-based agent communication language*, in Castelfranchi and Lewis Johnson [42], pp. 535–542.

[80] J. Fox, N. Johns, A. Rahmanzadeh, and R. Thomson, *Disseminating medical knowledge: the proforma approach*, Artificial Intelligence in Medicine **14** (1998), 157–181.

[81] T. Frühwirth, *Theory and practice of constraint handling rules*, Journal of Logic Programming **37** (1998), no. 1-3, 95–138.

[82] T. H. Fung and R. A. Kowalski, *The IFF proof procedure for abductive logic programming*, Journal of Logic Programming **33** (1997), no. 2, 151–165.

[83] Marco Gavanelli, Evelina Lamma, and Paola Mello, *Proof of completeness of the SCIFF proof-procedure*, Tech. Report CS-2005-02, Dipartimento di Ingegneria, Università di Ferrara, 2005, Available at `http://www.ing.unife.it/informatica/tr/CS-2005-02.pdf`.

[84] Marco Gavanelli, Evelina Lamma, Paola Mello, Michela Milano, and Paolo Torroni, *Interpreting abduction in CLP*, APPIA-GULP-PRODE Joint Conference on Declarative Programming (Reggio Calabria, Italy) (Francesco Buccafurri, ed.), Università Mediterranea di Reggio Calabria, September 3–5 2003, pp. 25–35.

[85] Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni, *SCIFF: Full proof of soundness*, Deliverable IST32530/DIFERRARA/401/D/I/b1, SOCS Consortium, Jun 2004.

[86] Marco Gavanelli, Evelina Lamma, Paolo Torroni, Paola Mello, Kostas Stathis, P. Moraïtis, A. C. Kakas, N. Demetriou, G. Terreni, Paolo Mancarella, A. Bracciali, Francesca Toni, Fariba Sadri, and U. Endriss, *Computational model for computees and societies of computees*, Tech. report, SOCS Consortium,

2003, Deliverable D8. Available electronically from the SOCS project web site: `http://lia.deis.unibo.it/research/socs/guests/publications/`.

[87] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM Journal on Computing **18** (1989), no. 1, 186–207.

[88] C. Gordon, *Practice guidelines and healthcare telematics; towards an alliance*, Health telematics for clinical guidelines and protocols (1995), 3–15.

[89] F. Guerin and J. Pitt, *Proving properties of open agent systems*, Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II (Bologna, Italy) (C. Castelfranchi and W. Lewis Johnson, eds.), ACM Press, July 15–19 2002, pp. 557–558.

[90] Chen Hao, John A. Clark, and Jeremy L. Jacob, *Synthesising efficient and effective security protocols*, Automated Reasoning for Security Protocol Analysis (ARSPA) (Cork, Ireland) (Alessandro Armando and Luca Viganò, eds.), July 2004, pp. 25–40.

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

[92] J. Jaffar, M.J. Maher, K. Marriott, and P.J. Stuckey, *The semantics of constraint logic programs*, Journal of Logic Programming **37** (1998), no. 1-3, 1–46.

[93] J. Jaffar, S. Michaylov, P.J. Stuckey, and R.H.C. Yap, *The CLP(R) language and system.*, ACM Transactions on Programming Languages and Systems **14** (1992), no. 3, 339–395.

[94] Edmund M. Clarke Jr., Orna Grumberg, and Doron A. Peled, *Model checking*, 4th edition ed., The MIT Press, 2002.

[95] A. C. Kakas, R. A. Kowalski, and Francesca Toni, *Abductive Logic Programming*, Journal of Logic and Computation **2** (1993), no. 6, 719–770.

[96] ———, *The role of abduction in logic programming*, Handbook of Logic in Artificial Intelligence and Logic Programming (D. M. Gabbay, C. J. Hogger, and J. A. Robinson, eds.), vol. 5, Oxford University Press, 1998, pp. 235–324.

[97] A. C. Kakas and Paolo Mancarella, *On the relation between Truth Maintenance and Abduction*, Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence, PRICAI-90, Nagoya, Japan (T. Fukumura, ed.), Ohmsha Ltd., 1990, pp. 438–443.

[98] A. C. Kakas, A. Michael, and C. Mourlas, *ACLP: Abductive Constraint Logic Programming*, Journal of Logic Programming **44** (2000), no. 1-3, 129–177.

[99] A. C. Kakas, B. van Nuffelen, and M. Denecker, *A-System: Problem solving through abduction*, Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, Washington, USA (IJCAI-01) (Seattle, Washington, USA) (B. Nebel, ed.), Morgan Kaufmann Publishers, August 2001, pp. 591–596.

[100] R. Kazhamiakin and M. Pistore, *A parametric communication model for the verification of bpel4ws compositions.*, EPEW/WS-FM, 2005, pp. 318–332.

[101] Kyoil Kim, Jacob A. Abraham, and Jayanta Bhadra, *Model checking of security protocols with pre-configuration*, Information Security Applications, 4th International Workshop, WISA 2003, Jeju Island, Korea, August 25-27, 2003, Revised Papers (Kijoon Chae and Moti Yung, eds.), Lecture Notes in Computer Science, vol. 2908, Springer-Verlag, 2004, pp. 1–15.

[102] R. A. Kowalski and Fariba Sadri, *From logic programming towards multi-agent systems*, Annals of Mathematics and Artificial Intelligence **25** (1999), no. 3/4, 391–419.

[103] R. A. Kowalski, Fariba Sadri, and Francesca Toni, *An agent architecture that combines backward and forward reasoning*, Proceedings of the CADE-15 Workshop on Strategies in Automated Deduction (B. Gramlich and F. Pfenning, eds.), November 1998, pp. 49–56.

[104] R. A. Kowalski and M. Sergot, *A logic-based calculus of events*, New Generation Computing **4** (1986), no. 1, 67–95.

[105] R.A. Kowalski, Francesca Toni, and G. Wetzel, *Executing suspended logic programs*, Fundamenta Informaticae **34** (1998), 203–224.

[106] Robert A. Kowalski, *The logical way to be artificially intelligent*, Computational Logic in Multi-Agent Systems, 6th International Workshop, CLIMA VI, London, UK, June 27-29, 2005, Revised Selected and Invited Papers (Paolo Torroni and Francesca Toni, eds.), Lecture Notes in Artificial Intelligence, Springer-Verlag, 2006.

[107] K. Kunen, *Negation in logic programming*, Journal of Logic Programming, vol. 4, 1987, pp. 289–308.

[108] J. W. Lloyd, *Foundations of logic programming*, 2nd extended ed., Springer-Verlag, 1987.

[109] G. Lowe, *Breaking and fixing the Needham-Shroeder public-key protocol using CSP and FDR*, Tools and Algorithms for the Construction and Analysis of Systems: Second International Workshop, TACAS'96 (T. Margaria and B. Steffen, eds.), Lecture Notes in Artificial Intelligence, vol. 1055, Springer-Verlag, 1996, pp. 147–166.

[110] Paolo Mancarella and Giacomo Terreni, *An abductive proof procedure handling active rules*, AI*IA 2003: Advances in Artificial Intelligence, Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence, Pisa (A. Cappelli and F. Turini, eds.), Lecture Notes in Artificial Intelligence, vol. 2829, Springer-Verlag, September 23–26 2003, pp. 105–117.

[111] S. Merz, *Model checking: A tutorial overview*, Modeling and Verification of Parallel Processes (F. Cassez, C. Jard, B. Rozoy, and M.D.Ryan, eds.), Lecture Notes in Computer Science, no. 2067, Springer-Verlag, 2001, pp. 3–38.

[112] J. J. Ch. Meyer, *A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic*, Notre Dame J. of Formal Logic **29(1)** (1988), 109–136.

[113] Jonathan K. Millen and Vitaly Shmatikov, *Constraint solving for bounded-process cryptographic protocol analysis*, CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, ACM press, 2001, pp. 166–175.

[114] R.M. Needham and M.D. Schroeder, *Using encryption for authentication in large networks of computers*, Communications of the ACM **21** (1978), no. 12, 993–999.

[115] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel, *Isabelle/hol - a proof assistant for higher-order logic*, Lecture Notes in Computer Science, vol. 2283, Springer-Verlag, 2002.

[116] C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede, , and H.M.W. Verbeek, *Formal semantics and analysis of control flow in ws-bpel*, Tech. Report BPM-05-15, BPMcenter.org, 2005.

[117] Albert Özkohen and Pinar Yolum, *Predicting exceptions in agent-based supply chains*, this volume, 2006.

[118] A. Dal Palù, A. Dovier, and E. Pontelli, *Heuristics, optimizations, and parallelism for protein structure prediction in CLP(FD)*, Proceedings of the 7th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (Pedro Barahona and Amy P. Felty, eds.), ACM, 230-241.

[119] Jun Pang, *Analysis of a security protocol in μCRL*, Formal Methods and Software Engineering, 4th International Conference on Formal Engineering Methods, ICFEM 2002 Shanghai, China, October 21-25, 2002, Proceedings (Chris George and Huaikou Miao, eds.), Lecture Notes in Computer Science, vol. 2495, Springer-Verlag, 2002, pp. 396–400.

[120] Lawrence C. Paulson, *The inductive approach to verifying cryptographic protocols.*, Journal of Computer Security **6** (1998), no. 1-2, 85–128.

[121] Henry Prakken and Marek Sergot, *Contrary-to-duty obligations*, Studia Logica **57** (1996), no. 1, 91–115.

[122] Franco Raimondi and Alessio Lomuscio, *Verification of multiagent systems via ordered binary decision diagrams: An algorithm and its implementation*, Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2004) (Columbia University, New York City)

(N. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, eds.), ACM Press, July 2004, pp. 630–637.

[123] R. Reiter, *On closed-word data bases*, Logic and Data Bases (H. Gallaire and J. Minker, eds.), Plenum Press, 1978, pp. 55–76.

[124] RFC793, *Transmission control protocol*, (1981), DARPA Internet Program Protocol Specification.

[125] Alessandro Ricci, Andrea Omicini, and Enrico Denti, *Objective vs. subjective coordination in agent-based systems: A case study*, Coordination Languages and Models (Farhad Arbab and Carolyn Talcott, eds.), LNCS, vol. 2315, Springer-Verlag, 2002, 5th International Conference (COORDINATION 2002), York, UK, 8–11 April 2002. Proceedings, pp. 291–299.

[126] J. S. Rosenschein and G. Zlotkin, *Rules of encounter*, MIT Press, 1994.

[127] A. Rozinat and Wil M. P. van der Aalst, *Conformance testing: Measuring the fit and appropriateness of event logs and process models*, Business Process Management Workshops (Christoph Bussler and Armin Haller, eds.), vol. 3812, 2005, pp. 163–176.

[128] A. Russo, R. Miller, B. Nuseibeh, and J. Kramer, *An abductive approach for analysing event-based requirements specifications*, Logic Programming, 18th International Conference, ICLP 2002 (Berlin Heidelberg) (P.J. Stuckey, ed.), Lecture Notes in Computer Science, vol. 2401, Springer-Verlag, 2002, pp. 22–37.

[129] Young U. Ryu and Ronald M. Lee, *Defeasible deontic reasoning: A logic programming model*, Deontic Logic in Computer Science: Normative System Specification (J.-J.Ch. Meyer and R.J. Wieringa, eds.), John Wiley & Sons Ltd, 1993, pp. 225–241.

[130] Fariba Sadri and Francesca Toni, *Abduction with negation as failure for active and reactive rules*, AI*IA'99: Advances in Artificial Intelligence, Proceedings of the 6th Congress of the Italian Association for Artificial Intelligence, Bologna (Evelina Lamma and Paola Mello, eds.), Lecture Notes in Artificial Intelligence, no. 1792, Springer-Verlag, 2000, pp. 49–60.

[131] Fariba Sadri, Francesca Toni, and Paolo Torroni, *An abductive logic programming architecture for negotiating agents*, in Flesca et al. [78], pp. 419–431.

[132] Giovanni Sartor, *Legal reasoning*, Treatise, vol. 5, Kluwer, Dordrecht, 2004.

[133] Ken Satoh, K. Inoue, K. Iwanuma, and C. Sakama, *Speculative computation by abduction under incomplete communication environments*, Proceedings of the Fourth International Conference on Multi-Agent Systems, Boston, Massachusetts, USA, IEEE Press, 2000, pp. 263–270.

[134] Ken Satoh and N. Iwayama, *A Query Evaluation Method for Abductive Logic Programming*, Proceedings of the Joint International Conference and Symposium on Logic Programming, Washington, USA (Cambridge, MA) (K. Apt, ed.), MIT Press, 1992, pp. 671–685.

[135] M. J. Sergot, *A query-the-user facility of logic programming*, Integrated Interactive Computer Systems (P. Degano and E. Sandwell, eds.), North Holland, 1983, pp. 27–41.

[136] Murray P. Shanahan, *Reinventing Shakey*, Logic-based Artificial Intelligence (J. Minker, ed.), Kluwer Int. Series In Engineering And Computer Science, vol. 597, 2000, pp. 233–253.

[137] *SICStus prolog user manual, release 3.11.0*, October 2003, `http://www.sics.se/isl/sicstus/`.

[138] M. Singh, *Agent communication language: rethinking the principles*, IEEE Computer (1998), 40–47.

[139] *Societies Of ComputeeS (SOCS): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees. IST-2001-32530*, 2002-2005, Home Page: `http://lia.deis.unibo.it/research/socs/`.

[140] Dawn Xiaodong Song, *Athena: a new efficient automatic checker for security protocol analysis*, CSFW '99: Proceedings of the 1999 IEEE Computer Security Foundations Workshop (Washington, DC, USA), IEEE Computer Society, 1999, p. 192.

[141] P.J. Stuckey, *Negation and constraint logic programming*, Information and Computation **118** (1995), no. 1, 12–33.

[142] P. Terenziani, P. Raviola, O. Bruschi, M. Torchio, M. Marzuoli, and G. Molino, *Representing knowledge levels in clinical guidelines*, Proceedings of the Join European Conference on Artificial Intelligence in Medicine and Medical Decision Making **1620** (1999), 254–258.

[143] W.M.P. van der Aalst, *Business alignment: Using process mining as a tool for delta analysis and conformance testing*, Requirements Engineering Journal **to appear** (2005).

[144] W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohed, *Life after BPEL?*, EPEW/WS-FM (Mario Bravetti, Leïla Kloul, and Gianluigi Zavattaro, eds.), LNCS, vol. 3670, Springer, 2005, pp. 35–50.

[145] L. van der Torre, *Contextual deontic logic: Normative agents, violations and independence*, Annals of Mathematics and Artificial Intelligence **37** (2003), no. 1, 33–63.

[146] Leendert W. N. van der Torre and Yao-Hua Tan, *Diagnosis and decision making in normative reasoning.*, Artificial Intelligence and Law **7** (1999), no. 1, 51–67.

[147] P. van Hentenryck and Y. Deville, *The Cardinality Operator: A new Logical Connective for Constraint Logic Programming*, Logic Programming, Proceedings of the Eigth International Conference, Paris, France (K. Furukawa, ed.), vol. 2, 1991, pp. 745–759.

[148] P. van Hentenryck, V. Saraswat, and Y. Deville, *Design, implementation, and evaluation of the constraint language cc(fd)*, Technical Report CS-93-02, Department of Computer Sciences, Brown University, January 1993.

[149] B. van Nuffelen and M. Denecker, *Problem solving in ID-logic with aggregates*, Proceedings of the 8th International Workshop on Non-Monotonic Reasoning, NMR'00, Breckenridge, CO, 2000, pp. 1–9.

[150] David von Oheimb and Volkmar Lotz, *Formal security analysis with interacting state machines*, Computer Security - ESORICS 2002, 7th European Symposium on Research in Computer Security, Zurich, Switzerland, October 14-16, 2002, Proceedings (Dieter Gollmann, Günter Karjoth, and Michael Waidner, eds.), Lecture Notes in Computer Science, vol. 2502, Springer-Verlag, 2002, pp. 212–229.

[151] W3C, *Web services choreography description language version 1.0*, Home Page: `http://www.w3.org/TR/ws-cdl-10/`.

[152] Armin Wolf, Thom Frühwirth, and Marc Meister (eds.), *19th workshop on (constraint) logic programming w(c)lp 2005*, Ulmer Informatik-Berichte, no. 2005-01, 2005.

[153] G.H. Wright, *Deontic logic*, Mind **60** (1951), 1–15.

[154] I. Xanthakos, *Semantic integration of information by abduction*, Ph.D. thesis, Imperial College London, 2003, Available at `http://www.doc.ic.ac.uk/~ix98/PhD.zip`.

[155] P. Yolum and M.P. Singh, *Flexible protocol specification and execution: applying event calculus planning using commitments*, in Castelfranchi and Lewis Johnson [42], pp. 527–534.